GumNaaM_Helpers

Virtual University

Mega File for MiD TerM

All Files in ONE FILE

(Handouts_waqar_moaz_junaid_vu toper_orange monkey)

Alhamdulillah GumNaaM_Helpers Bs..IT/CS ky Tamam Semesters ki tamam Books ki help krty hn.

Group Join krty hi description Read krna na bholyn

Semester Name	Group Link
Bs..IT/CS...smstr:1	https://chat.whatsapp.com/Ciya2X0DI0l8iQ14m0VNNj
Bs..IT/CS...smstr:2	https://chat.whatsapp.com/Kwh85sRK02z1Qk5B7aMR19
Bs..IT/CS...smstr:3	https://chat.whatsapp.com/IjIpdzi8pIA4lQVNUMmNYE
Bs..IT/CS...smstr:4	https://chat.whatsapp.com/CiGpHsSftir0oyhM47WfKh
Bs..IT/CS...smstr:5	https://chat.whatsapp.com/Hf7NWhdW06uCW7Mbt1n4Db
Bs..IT/CS...smstr:6	https://chat.whatsapp.com/DyAoT7VFkwy8DuvryDqm2L
Bs..IT/CS...smstr:7	https://chat.whatsapp.com/ENiyfIDRWatJFPbXCUGYNN
Bs..IT/CS...smstr:8	https://chat.whatsapp.com/Bnkk4n7utpdEuXazHLxRcA

...Paid Service Available for LMS Activities...

Only for those students who are so busy in office or work or job that they can't watch their LMS due to their Busyness

FOR MORE INFO. CONTACT US ONLY WHAT'S APP 0302-1500025

CS304-Handouts

© Virtual University of Pakistan

LECTURE NO.01 8

01.1. INTRODUCTION 8
01.2. WHAT IS A MODEL? 10
01.3. OO MODELS: 11
01.4. OBJECT-ORIENTATION - ADVANTAGES 12
01.5. WHAT IS AN OBJECT? 12
01.6. TANGIBLE AND INTANGIBLE OBJECTS 13
01.7. SUMMARY: 14

LECTURE NO.02 15

02.1. INFORMATION HIDING: 15
02.2. ENCAPSULATION 16
02.3. INTERFACE 17
02.4. IMPLEMENTATION 18
02.5. SEPARATION OF INTERFACE & IMPLEMENTATION 19
02.6. MESSAGES 19
02.7. SUMMARY 19

LECTURE NO.03 21

03.1. ABSTRACTION 21
03.2. CLASSES 23
03.3. INHERITANCE 25

LECTURE NO.04 29

04.1. CONCEPTS RELATED WITH INHERITANCE 29
04.2. GENERALIZATION 29
04.3. SUB-TYPING (EXTENSION) 31
04.4. SPECIALIZATION (RESTRICTION) 32
04.5. OVERRIDING 34
04.6. ABSTRACT CLASSES 36
04.7. CONCRETE CLASSES 38

LECTURE NO.05 40

05.3. SIMPLE ASSOCIATION 50
05.4. COMPOSITION 52
05.5. AGGREGATION 54

LECTURE NO.06 55

06.1. CLASS COMPATIBILITY 55
06.2. POLYMORPHISM 56
06.3. POLYMORPHISM IN OO MODEL 56
06.4. POLYMORPHISM – ADVANTAGES 57
06.5. OBJECT-ORIENTED MODELING AN EXAMPLE 57

LECTURE NO.07 65

07.1. CLASS 65
07.2. TYPE IN C++ 65
07.3. ABSTRACTION 66
07.4. DEFINING A NEW USER DEFINED TYPE 66
07.5. OBJECT AND CLASS: 68
07.6. ACCESSING MEMBERS 68
07.7. ACCESS SPECIFIERS 69

LECTURE NO.08 72

 © Virtual University of Pakistan

08.1. MEMBER FUNCTIONS 72
08.2. DEFINING MEMBER FUNCTIONS 72
08.3. INLINE FUNCTIONS 73
08.4. CONSTRUCTOR 75
08.5. CONSTRUCTOR PROPERTIES 75
08.6. DEFAULT CONSTRUCTOR 76
08.7. CONSTRUCTOR OVERLOADING 77
08.8. CONSTRUCTOR OVERLOADING 78
08.9. COPY CONSTRUCTOR 79
08.10. SHALLOW COPY 81
08.11. DEEP COPY 82

LECTURE NO.09 ..... **84**

09.1. SHALLOW COPY 85
09.2. DEEP COPY 90
09.3. IMPORTANT POINTS ABOUT COPY CONSTRUCTOR: 94
09.4. DESTRUCTOR 94
09.5. ACCESSOR FUNCTIONS 95
09.6. THIS POINTER 96

LECTURE NO.10 ..... **99**

10.1. USES OF THIS POINTER 99
10.2. SEPARATION OF INTERFACE AND IMPLEMENTATION 99
10.3. COMPLEX NUMBER 100
10.4. CONST MEMBER FUNCTIONS 102
10.5. THIS POINTER AND CONST MEMBER FUNCTION 104

LECTURE NO.11 ..... **105**

11.1. USAGE EXAMPLE OF CONSTANT MEMBER FUNCTIONS 105
11.2. DIFFERENCE BETWEEN INITIALIZATION AND ASSIGNMENT: 106
11.3. MEMBER INITIALIZER LIST 106
11.4. CONST OBJECTS 107
11.5. STATIC VARIABLES 109

LECTURE NO.12 ..... **112**

12.1. ACCESSING STATIC DATA MEMBER 113
12.2. LIFE OF STATIC DATA MEMBER 114
12.3. STATIC MEMBER FUNCTION 115
12.4. THIS POINTER AND STATIC MEMBER FUNCTIONS 116
12.5. GLOBAL VARIABLE VS. STATIC MEMBERS 116
12.6. ARRAY OF OBJECTS 116

LECTURE NO.13 ..... **118**

13.1. POINTER TO OBJECTS 118
13.2. BREAKUP OF NEW OPERATION 119
13.3. CASE STUDY 119
13.4. COMPLETE CODE OF DATE CLASS 121

LECTURE NO.14 ..... **124**

14.1. COMPOSITION 124

LECTURE NO.15 ..... **132**

15.1. AGGREGATION 135
15.2. FRIEND FUNCTIONS 138

LECTURE NO.16 ..... **142**

16.1. OPERATOR OVERLOADING 142

LECTURE NO.17 ..... **149**

17.1. OVERLOADING ASSIGNMENT OPERATOR 151

LECTURE NO.18 ..... **155**

18.1. SELF ASSIGNMENT PROBLEM: 155
18.2. OTHER BINARY OPERATORS 156
18.3. FRIEND FUNCTIONS AND OPERATOR OVERLOADING 157

LECTURE NO.19 ..... **158**

19.1. STREAM INSERTION OPERATOR 158
19.2. STREAM EXTRACTION OPERATOR 158
19.3. OVERLOADING STREAM INSERTION OPERATOR 159
19.4. OVERLOADING STREAM EXTRACTION OPERATOR: 160
19.5. OTHER BINARY OPERATORS: 161

LECTURE NO.20 ..... **163**

20.1. SUBSCRIPT [] OPERATOR 164
20.2. OVERLOADING SUBSCRIPT [] OPERATOR 164
20.3. OVERLOADING FUNCTION () OPERATOR 165
20.4. FUNCTION OPERATOR PERFORMING SUB STRING OPERATION, ... 165
20.5. UNARY OPERATORS 166

LECTURE NO.21 ..... **168**

21.1. BEHAVIOR OF ++ AND -- FOR PRE-DEFINED TYPES: 168
21.2. POST-INCREMENT OPERATOR: 169
21.3. TYPE CONVERSION 170
21.4. USER DEFINED TYPES: 173
21.5. DRAWBACKS OF TYPE CONVERSION OPERATOR: 174

LECTURE NO.22 ..... **175**

22.1. PRACTICAL IMPLEMENTATION OF INHERITANCE IN C++ 175
22.2. INHERITANCE IN CLASSES 175
22.3. UML NOTATION 175
22.4. INHERITANCE IN C++ 175
22.5. "IS A" RELATIONSHIP 176

LECTURE NO.23 ..... **183**

23.1. ACCESSING BASE CLASS MEMBER FUNCTIONS IN DERIVED CLASS: 183
23.2. "*PROTECTED*" ACCESS SPECIFIER: 185
23.3. "IS A" RELATIONSHIP 186
23.4. STATIC TYPE 189

LECTURE NO.24 ..... **191**

24.1. MODIFIED DEFAULT CONSTRUCTOR 199

LECTURE NO.25 ..... **207**

25.1. OVERLOADING VS. OVERRIDING 207
25.2. HIERARCHY OF INHERITANCE 211

LECTURE NO.26 ..... **213**

26.1. BASE INITIALIZATION 213
26.2. TYPES OF INHERITANCE 214
26.3. PRIVATE INHERITANCE 216

 © Virtual University of Pakistan

LECTURE NO.27 ..... **218**

27.1. SPECIALIZATION (RESTRICTION) 218
27.2. PROTECTED INHERITANCE 222
27.3. PROPERTIES OF PROTECTED INHERITANCE 223

LECTURE NO.28 ..... **225**

28.1. VIRTUAL FUNCTIONS 225
28.2. VIRTUAL FUNCTIONS: 230
28.3. SHAPE HIERARCHY 230
28.4. STATIC VS DYNAMIC BINDING 231

LECTURE NO.29 ..... **233**

29.1. ABSTRACT CLASSES 233
29.2. CONCRETE CLASSES 234
29.3. ABSTRACT CLASSES IN C++ 234
29.4. PURE VIRTUAL FUNCTIONS 234
29.5. SHAPE HIERARCHY 235
29.6. VIRTUAL DESTRUCTORS 236
29.7. VIRTUAL FUNCTIONS – USAGE 238
29.8. V TABLE 240
29.9. DYNAMIC DISPATCH (DYNAMIC BINDING) 242

LECTURE NO.30 ..... **243**

30.1. POLYMORPHISM – CASE STUDY: A SIMPLE PAYROLL APPLICATION 243
30.2. SHAPE HIERARCHY REVISITED: 246

LECTURE NO.31 ..... **250**

31.1. MULTIPLE INHERITANCE 250
31.2. PROBLEMS IN MULTIPLE INHERITANCE 251
31.3. VIRTUAL INHERITANCE 256

LECTURE NO.32 ..... **258**

32.1. GENERIC PROGRAMMING 259
32.2. TEMPLATES 259
32.3. FUNCTION TEMPLATES 260

LECTURE NO.33 ..... **264**

33.1. MULTIPLE TYPE ARGUMENTS 264
33.2. USER-DEFINED TYPES 264
33.3. OVERLOADING VS. TEMPLATES 265
33.4. TEMPLATE ARGUMENTS AS POLICY: 266
33.5. FIRST SOLUTION: 267
33.6. SECOND SOLUTION: 267
33.7. THIRD SOLUTION 268
33.8. DEFAULT POLICY 269

LECTURE NO.34 ..... **270**

34.1. GENERIC ALGORITHMS 270
34.2. CLASS TEMPLATES 272
34.3. EXAMPLE – CLASS TEMPLATE 273

LECTURE NO.35 ..... **276**

35.1. MEMBER TEMPLATES: 276
35.2. CLASS TEMPLATE SPECIALIZATION 278

LECTURE NO.36 ..... **282**

36.1. MEMBER TEMPLATES REVISITED 282
36.2. PARTIAL SPECIALIZATION: 283
36.3. FUNCTION TEMPLATES 284
36.4. COMPLETE SPECIALIZATION 285
36.5. USING DIFFERENT SPECIALIZATIONS 286
36.6. NON-TYPE PARAMETERS 286
36.7. EXAMPLE – TEMPLATE CLASS ARRAY 286
36.8. DEFAULT NON-TYPE PARAMETERS 288
36.9. DEFAULT TYPE PARAMETERS 288

LECTURE NO.37 288

37.1. RESOLUTION ORDER 289
37.2. FUNCTION TEMPLATE OVERLOADING 290
37.3. RESOLUTION ORDER 291
37.4. TEMPLATES AND INHERITANCE 291
37.5. DERIVATIONS IN CASE OF A GENERAL TEMPLATE CLASS 291

LECTURE NO.38 296

38.1. TEMPLATES AND FRIENDS 296
38.2. TEMPLATES AND FRIENDS – RULE 1 296
38.3. TEMPLATES AND FRIENDS – RULE 2 297
38.4. TEMPLATES AND FRIENDS – RULE 3 299
38.5. TEMPLATES AND FRIENDS – RULE 4 301

LECTURE NO.39 303

39.1. TEMPLATES & STATIC MEMBERS 303
39.2. TEMPLATES – CONCLUSION 304
39.3. GENERIC ALGORITHMS REVISITED 305
39.4. GENERIC ALGORITHMS REVISITED 306
39.5. GENERIC ALGORITHM 308
39.6. PROBLEMS 308

LECTURE NO.40 309

40.1. CURSORS 309
40.2. ITERATORS 312

LECTURE NO.41 316

41.1. STANDARD TEMPLATE LIBRARY: 316
41.2. STL CONTAINERS 316
41.3. COMMON FUNCTIONS FOR ALL CONTAINERS 322
41.4. FUNCTIONS FOR FIRST-CLASS CONTAINERS 322
41.5. CONTAINER REQUIREMENTS 323

LECTURE NO.42 325

42.1. ITERATORS 325
42.2. ITERATOR CATEGORIES 325
42.3. ITERATOR SUMMARY: 325
42.4. CONTAINER AND ITERATOR TYPES: 326
42.5. SEQUENCE CONTAINERS 326
42.6. ASSOCIATIVE CONTAINERS 327
42.7. CONTAINER ADAPTERS 327
42.8. ITERATOR OPERATIONS 327
42.9. ALGORITHMS 331

LECTURE NO.43 334

43.1. EXAMPLE – ABNORMAL TERMINATION 334

 © Virtual University of Pakistan

43.2. GRACEFUL TERMINATION 335
43.3. ERROR HANDLING 335
43.4. EXCEPTION HANDLING 338
43.5. EXCEPTION HANDLING PROCESS 338
LECTURE NO.44 ..... **342**
44.1. STACK UNWINDING 342
LECTURE NO.45 ..... **349**
45.1. RESOURCE MANAGEMENT 349

Lecture No.01

01.1. Introduction

Course Objective:
Objective of this course is to make students familiar with the concepts of object oriented programming. These concepts will be reinforced by their implementation in C++.

Course Contents:
The main topics that we will study in the 45 lectures of this course are given below,

- Object Orientation
- Objects and Classes
- Overloading
- Inheritance
- Polymorphism
- Generic Programming
- Exception Handling
- Introduction to Design Patterns

Recommended Text Book:

C++ How to Program (Deitel & Deitel **)**

Reference Books:

1. **Object-Oriented Software Engineering**
   By Jacobson, Christerson, Jonsson, Overgaard
   (For object oriented programming introductory concepts)
2. **The C++ Programming Language**
   By Bjarne Stroustrup
   (For better c++ understanding)

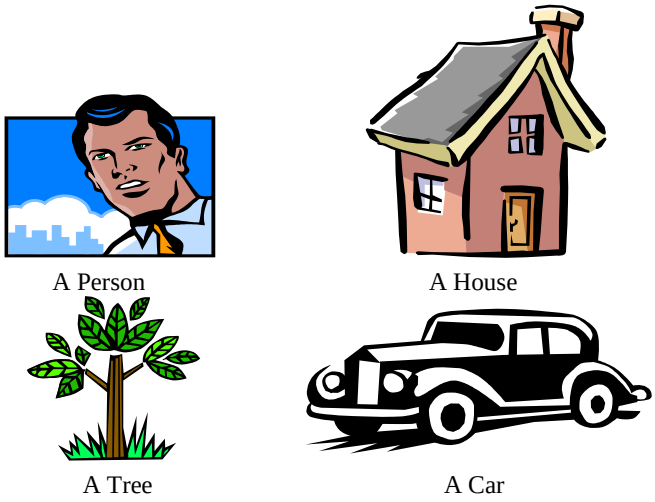
© Virtual University of Pakistan

Object-Orientation (OO)

What is Object-Orientation?

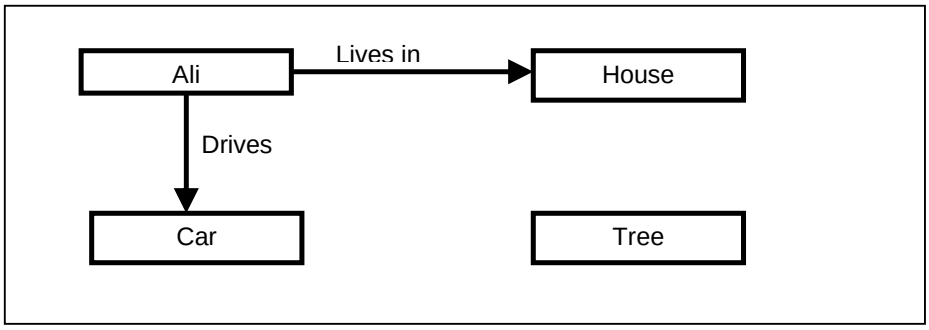
It is a technique in which we visualize our programming problems in the form of objects and their interactions as happens in real life.

Examples:
We have different objects around us in our real life that interact with each other to perform different operations for example,



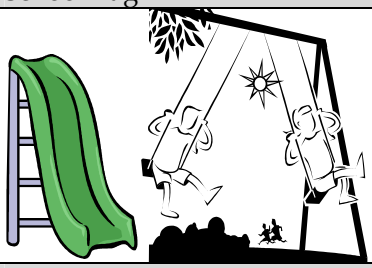
Different Objects

These objects interact with each other to perform different operations,



Take another example of a **School**; the objects in a school are **student**, **teacher**, **books, pen** ,**school bag**, **classroom, parents, playground and so on… ,**

Objects in a School

Teacher	Student	School Bag
		
Book	Pen	Playground
Parents	Classroom	Library

Suppose we want to develop a fee collection system for a school for this we will need to find out related objects and their interactions as happens in real life.
In this way we can say that **object orientation** makes it easier for us to solve our real world problems by thinking solution of the problem in terms of real world objects.

So we can say that in our daily life everything can be taken as an object that behaves in a certain way and has certain attributes.

In object orientation we move our concentration to objects in contrast to procedural paradigm in which we simply write our code in functions and call them in our main program.

01.2.What is a Model?

A model is an abstraction of something real or conceptual.
We need models to understand an aspect of reality.

© Virtual University of Pakistan

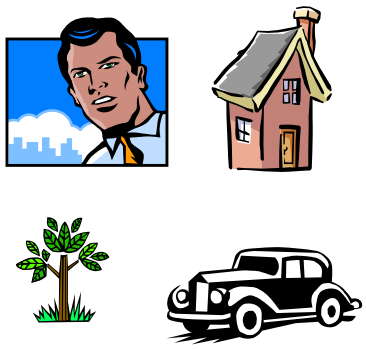
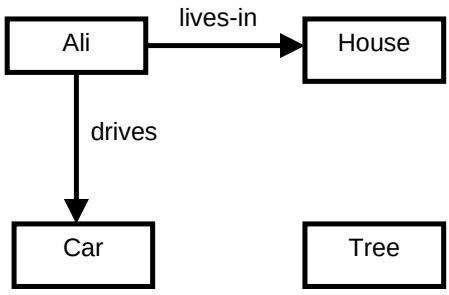
Model Examples

Highway maps
Architectural models
Mechanical models

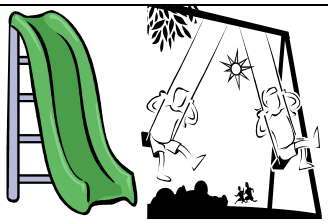
01.3.OO Models:

In the context of programming models are used to understand the problem before starting developing it.
We make Object Oriented models showing several interacting objects to understand a system given to us for implementation.

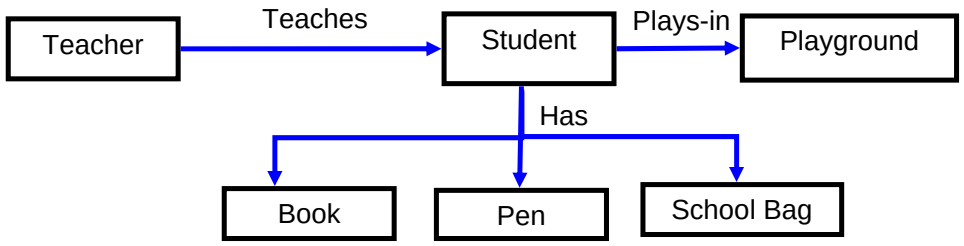
Example 1– Object Oriented Model

<table>
<tr>
<td>  </td>
<td> 
</td>
</tr>
<tr>
<td>Objects
Ali, Car, House, Tree</td>
<td>Interactions
Ali lives in the house
Ali drives the car</td>
</tr>
</table>

Example 2– Object Oriented Model (A School Model)



A School Model



Objects	**Interactions**
Teacher, Student, School Bag, Pen, Book Playground	Teacher teaches Student. Student has School Bag, Book and Pen

01.4.Object-Orientation - Advantages

As Object Oriented Models map directly to reality as we have seen in examples above therefore,

We can easily **develop** an object oriented model for a problem.
Everyone can easily **understand** an object oriented model.
We can easily implement an object oriented model for a problem using any object oriented language like c++ using its features[1] like classes, inheritance, virtual functions and so on…

01.5.What is an Object?

An object is,

1. Something tangible (Ali, School, House, Car).
2. Something conceptual (that can be apprehended intellectually for example time, date and so on…).

An object has,

1. State (attributes)
2. Well-defined behavior (operations)

[1] We will study these features in detail in this course

© Virtual University of Pakistan

3. Unique identity

01.6.Tangible and Intangible Objects

Examples of Tangible Objects:

Ali is a tangible object, having some characteristics (attributes) and behavior as given below,

Ali	
Characteristics (attributes)	**Behaviour (operations)**
Name Age	Walks Eats

We will identify Ali using his name.

Car is also a tangible object having some characteristics (attributes) and behavior given below,

Car	
State (attributes)	**Behavior (operations)**
Color Model	Accelerate Start Car Change Gear

We can identify Car using its registration number

Examples of Intangible Objects (also called as conceptual objects):

Time is an intangible (conceptual) object

Time	
State (attributes)	**Behavior (operations)**
Hours Seconds Minutes	Set/Get Hours Set/Get Seconds Set/Get Minutes

We will assign our own generated unique ID in the model for Time object

Date is also an intangible (conceptual) object

State (attributes)	**Behavior (operations)**
Year Day Month	Set/Get Year Set/Get Day Set/Get Month

We will assign our own generated unique ID in the model for Date object.

01.7.Summary:

- Model is the abstraction of some real word scenario. It helps us to understand that scenario.
- Object oriented model of any scenario (problem) describes that scenario (problem) in the form of interacting objects.
- We use Object Orientation because it helps us in mapping real world problem in a programming language.
- Object Orientation is achieved using objects and their relationships.
- Properties of an object are described using its *data members* and behavior of an object is described using its *functions.*
- Objects may be tangible (physical) or intangible (also called conceptual or virtual).
- Generally when we have given a certain problem description, **nouns** in that problem description are *candidates* for becoming objects of our system.
- There may be more than one aspects of an object
- It is not necessary that every object has a specific role in implementation of a problem there may be some objects without any role, like school parking in our school.
- It is easier to develop programs using Object Oriented Programming because it is closer to real life.

© Virtual University of Pakistan

Lecture No.02

Lecture Contents

1. Information Hiding
2. Encapsulation
3. Interface
4. Implementation
5. Separation of Interface & Implementation
6. Messages

02.1.Information Hiding:

Information hiding is one of the most important principles of OOP inspired from real life which says that all information should not be accessible to all persons. Private information should only be accessible to its owner.
By Information Hiding we mean "*Showing only those details to the outside world which are necessary for the outside world and hiding all other details from the outside world.*"

Real Life Examples of Information Hiding

1. Ali's name and other personal information is stored in his brain we can't access this information directly. For getting this information we need to ask Ali about it and it will be up to Ali how much details he would like to share with us.

2. An email server may have account information of millions of people but it will share only our account information with us if we request it to send anyone else accounts information our request will be refused.

3. A phone SIM card may store several phone numbers but we can't read the numbers directly from the SIM card rather **phone-set** reads this information for us and if the owner of this phone has not allowed others to see the numbers saved in this phone we will not be able to see those phone numbers using phone.

In object oriented programming approach we have objects with their attributes and behaviors that are hidden from other classes, so we can say that object oriented programming *follows the principle of information hiding*.

In the perspective of **Object Oriented Programming** Information Hiding is,

"Hiding the object details (state and behavior) from the users"

Here by users we mean **"an object"** of another class that is calling functions of this class using the reference of this class object or it may be some other program in which we are using this class.

Information Hiding is achieved in Object Oriented Programming using the following principles,

- All information related to an object is stored within the object
- It is hidden from the outside world
- It can only be manipulated by the object itself

Advantages of Information Hiding

Following are two major advantages of information hiding,

It simplifies our Object Oriented Model:

As we saw earlier that our object oriented model only had objects and their interactions hiding implementation details so it makes it easier for everyone to understand our object oriented model.

It is a barrier against change propagation

As implementation of functions is limited to our class and we have only given the name of functions to user along with description of parameters so if we change implementation of function it doesn't affect the object oriented model.

We can achieve information hiding using **Encapsulation** and **Abstraction**, so we see these two concepts in detail now,

02.2. Encapsulation

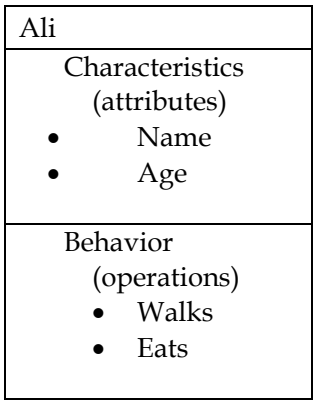
Encapsulation means *"we have enclosed all the characteristics of an object in the object itself"*
Encapsulation and information hiding are much related concepts (information hiding is achieved using Encapsulation)
We have seen in previous lecture that object characteristics include data members and behavior of the object in the form of functions.

So we can say that Data and Behavior are tightly coupled inside an object and both the information structure and implementation details of its operations are hidden from the outer world.

Examples of Encapsulation

Consider the same example of object Ali of previous lecture we described it as follows,

 © Virtual University of Pakistan



You can see that Ali stores his personal information in itself and its behavior is also implemented in it.
Now it is up to object Ali whether he wants to share that information with outside world or not. Same thing stands for its behavior if some other object in real life wants to use his behavior of walking it can not use it without the permission of Ali.

So we say that attributes and behavior of Ali are encapsulated in it.

Any other object don't know about these things unless Ali share this information with that object through an interface,

Same concept also applies to phone which has some data and behavior of showing that data to user we can only access the information stored in the phone if phone interface allow us to do so.

Advantages of Encapsulation
The following are the main advantages of Encapsulation,

a. Simplicity and clarity
As all data and functions are stored in the objects so there is no data or function around in program that is not part of any object and is this way it becomes very easy to understand the purpose of each data member and function in an object.

b. Low complexity
As data members and functions are hidden in objects and each object has a specific behavior so there is less complexity in code there will be no such situations that a functions is using some other function and that functions is using some other function.

c. Better understanding
Everyone will be able to understand whole scenario by simple looking into object diagrams without any issue as each object has specific role and specific relation with other objects.

02.3.Interface

Interface is a set of functions of an object that he wants to expose to other objects.

As we discussed previously that data and behavior of each object is hidden in that object it self so we have to use the concept of interface of the object to expose its behavior to outer word objects.

- Different objects may need different functions of an object so interface of an object may be different for different objects.
- Interfaces are necessary for object communication. Each object provides interface/s (operations) to other objects through these interfaces other objects communicate with this object.

Example – Interface of a Car

- Steer Wheels
- Accelerate
- Change Gear
- Apply Brakes
- Turn Lights On/Off

Example – Interface of a Phone

- Input Number
- Place Call
- Disconnect Call
- Add number to address book
- Remove number
- Update number

02.4.Implementation

It is actual implementation of the behavior of the object in any Object Oriented language.

It has two parts,

- Internal data structures to hold an object state that will be hidden from us it will store values for an object data members.
- Functionality in the form of member functions to provide required behavior.

Examples of Implementation

a. Gear Box in car system

Consider object Gear Box in car system it has a certain structure and functionality. When this object will be implemented it will have two things,

- Physical structure of the gear box
- Functionality implemented in this structure to change gear.

Both these things are part of implementation.

 © Virtual University of Pakistan

So it has,

- **Data Structure** in the form of Mechanical structure of gear box
- **Functionality** mechanism to change gear

b. Address Book in a Phone

Similarly take the example of contact details saved in the SIM of a phone,

In that case we can say physical structure of SIM card as **Data Structure** And Read/write operations provided by the phone as **Functionality**.

02.5. Separation of Interface & Implementation

As discussed earlier we only show interface of an object to outside world and hide actual implementation from outside world. The benefit of using this approach is that our object interface to outside word becomes independent from inside implementation of that interface.

This is achieved through the concepts of encapsulation and information hiding.

Real Life example of separation of interface and implementations

- Driver has a standard interface to drive a car and using that interface he drive can drive any car regardless of its model or type whatever engine type it has or whatever type of fuel it is using.

02.6. Messages

Objects communicate through messages they send messages (stimuli) by invoking appropriate operations on the target object. The number and kind of messages that can be sent to an object depends upon its interface

Examples – Messages

A Person sends message (stimulus) "stop" to a Car by applying brakes

A Person sends message "place call" to a Phone by pressing appropriate button

02.7. Summary

- Information hiding is achieved through encapsulation.
- Encapsulation and Information Hiding are related to each other.
- Interface of an object provides us the list of available functions.
- An object may have more than one interface.
- Interface and implementation are separated from each other to achieve Information Hiding.
- Objects communicate with each other using messages.

Useful Links:

http://www.alice.org/

A Graphical Programming Environment to teach Computer Programming.

© Virtual University of Pakistan

Lecture No.03

Lecture Contents:

- Abstraction
- Classes
- Inheritance
- Major benefits of inheritance (Reuse)

03.1. Abstraction

Real life objects have a lot of attributes and many kind of behaviors but most of the time we are interested in only that part of the objects that is related to the problem we are currently going to solve, for example in implementing a school system we don't need to take care of the personnel life of a student or a teacher as it will not effect our system in any way so we will see these objects in the perspective of school system and will ignore their other characteristics, this concept is called "*Abstraction*". Abstraction is a way to cope with complexity and it is used to simplify things.

Principle of abstraction:

"Capture only those details about an object that are relevant to current perspective"

Abstraction Example:

Suppose we want to implement abstraction for the following statement,

"Ali is a PhD student and teaches BS students"

Here object Ali has two ***perspectives*** *one is his* ***student perspective*** *and second is his* ***teacher perspective.***

We can sum up Ali's attributes as follows,

Name
Age
Student Roll No
Year of Study
CGPA
Employee ID
Designation
Salary

As you can see out of all these listed attributes some belong to Ali's student perspective(Roll No, CGPA, Year of study) and some belong to Ali's teacher perspective(Employee ID, Designation, Salary).

Similarly we can sum up Ali's behavior as follows,

Study
DevelopExam

GiveExam
TakeExam
PlaySports
Eat
DeliverLecture
Walk

As was the case with attributes of object Ali, its behavior can also be divided in Ali's student perspective as well as Ali's teacher perspective.

Student's Perspective

Attributes:

- **Name**
- **Student Roll No**
- **Year of Study**
- **CGPA**
- Employee ID
- Designation
- Salary
- Age

Behaviour:

- **Study**
- **GiveExam**
- **PlaySports**
- DeliverLecture
- DevelopExam
- TakeExam
- Eat
- Walk

Teacher's Perspective

Attributes:

- **Name**
- Student Roll No
- Year of Study
- CGPA
- **Employee ID**
- **Designation**
- **Salary**
- Age

Behaviour:

- Study
- GiveExam
- PlaySports
- **DeliverLecture**
- **DevelopExam**
- **TakeExam**
- Eat
- Walk

A cat can be viewed with different perspectives

Ordinary Perspective A pet animal with Four Legs A Tail	Surgeon's Perspective A being with A Skeleton Heart

© Virtual University of Pakistan

Two Ears Sharp Teeth	Kidney Stomach

A car can be viewed with different perspectives

 Driver's View	 Engineer's View

Abstraction – Advantages

Abstraction has following major advantages,

1. It helps us understanding and solving a problem using object oriented approach as it hides extra irrelevant details of objects.

2. Focusing on single perspective of an object provides us freedom to change implementation for other aspects of for an object later.

Similar to Encapsulation Abstraction is also used for achieving information hiding as we show only relevant details to related objects, and hide other details.

03.2.Classes

In OOP we create a general sketch for each kind of objects and then we create different instances using this sketch we call this sketch or prototype or map as "class".
All objects of same kind exhibit identical characteristics (information structure and behavior) however they have data of their own.

Class –Example 1

Consider the objects given below,

- Ali studies mathematics
- Anam studies physics
- Sohail studies chemistry

Each one is a Student so we say these objects are *instances* of the Student class.

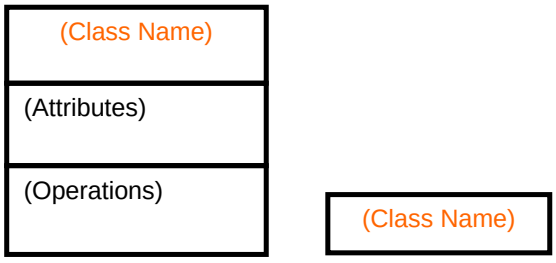
Class –Example 2

Consider the objects given below,

- Ahsan teaches mathematics
- Aamir teaches computer science
- Atif teaches physics

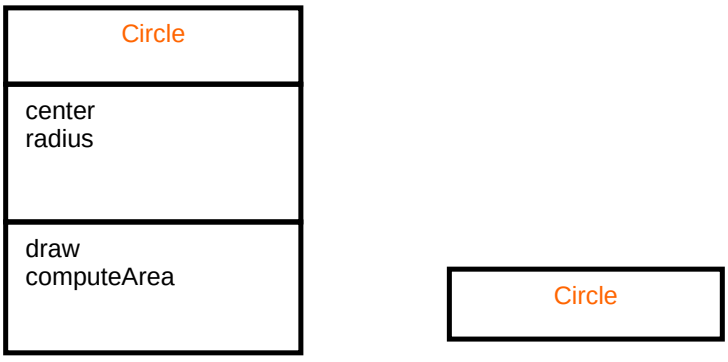
Each one is a teacher so we say these objects are *instances* of the Teacher class

Class Representation:
we can represent a class using a rectangle as follows,



Normal Form **Suppressed Form**

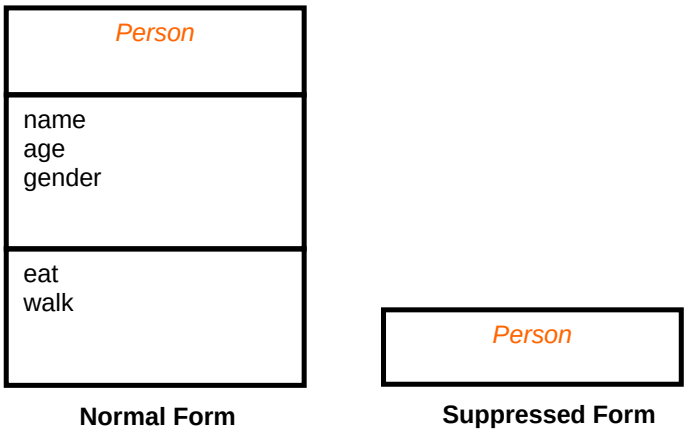
Class Example: Circle



Normal Form **Suppressed Form**

Class Example: Person

© Virtual University of Pakistan

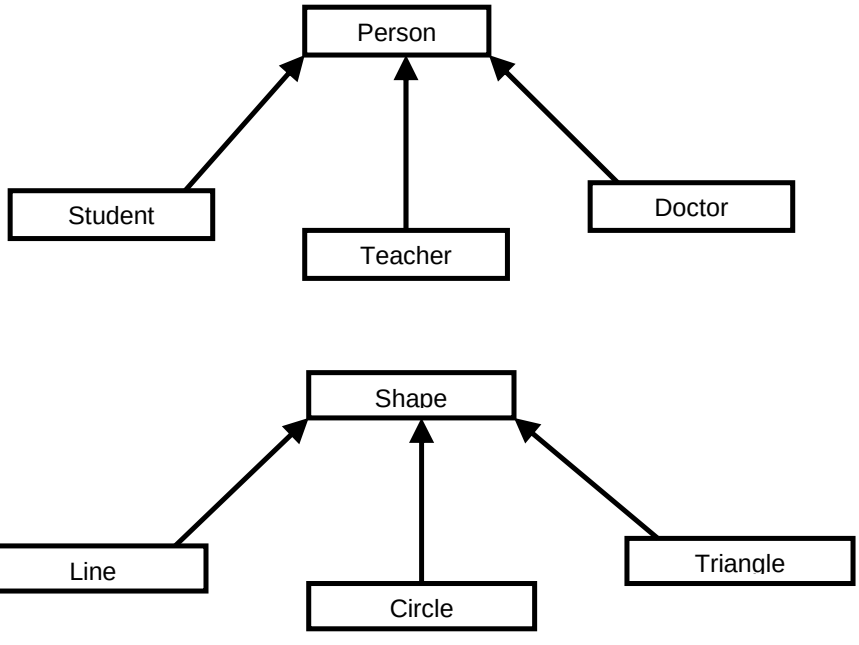


03.3.Inheritance

A child inherits characteristics of its parents, besides inherited characteristics, a child may have its own unique characteristics

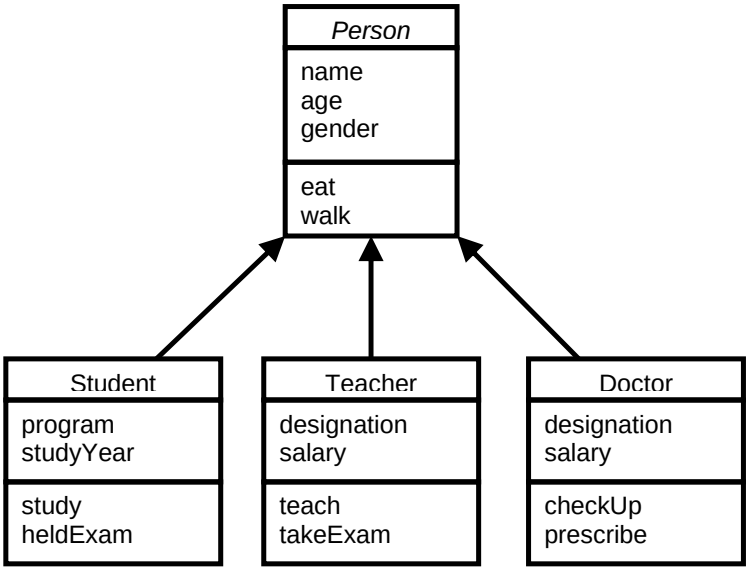
Inheritance in Classes

If a class B inherits from class A then it contains all the characteristics (information structure and behaviour) of class A
The parent class is called *base* class and the child class is called *derived* class
Besides inherited characteristics, derived class may have its own unique characteristics

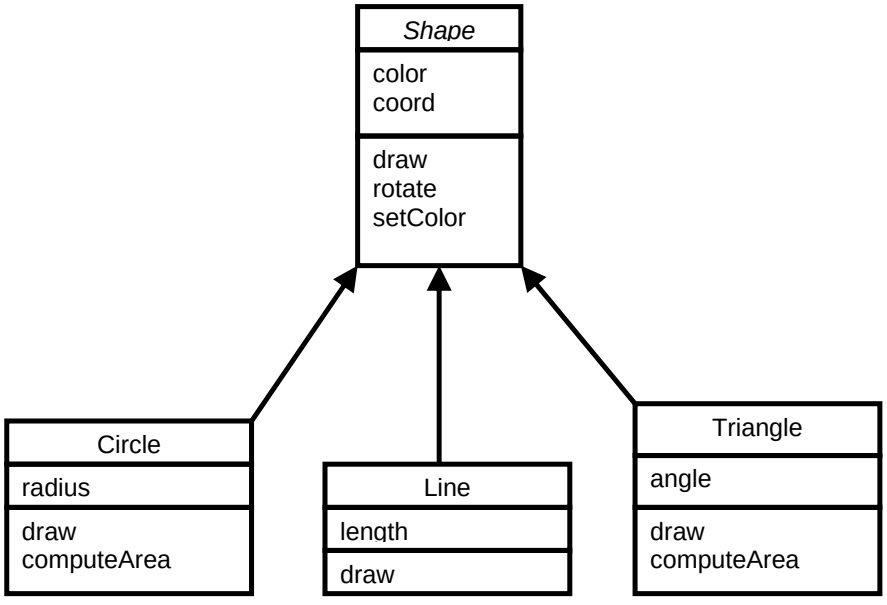


Inheritance – "IS A" or "IS A KIND OF" Relationship

Each derived class is a kind of its base class



Here,
Student IS A Person
Teacher IS A Person
Doctor IS A Person



Here,
Circle IS A Shape
Line IS A Shape
Triangle IS A Shape

© Virtual University of Pakistan

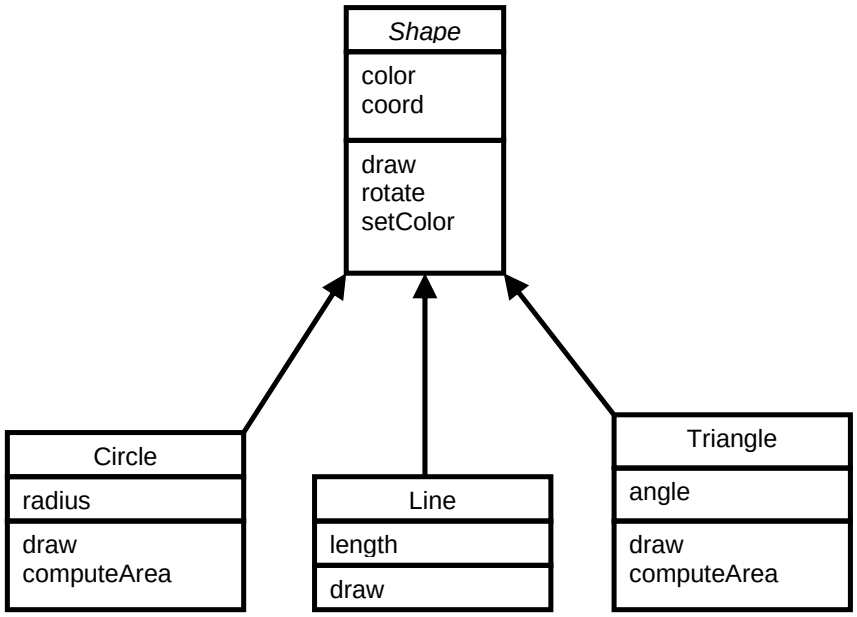
Inheritance – Advantages

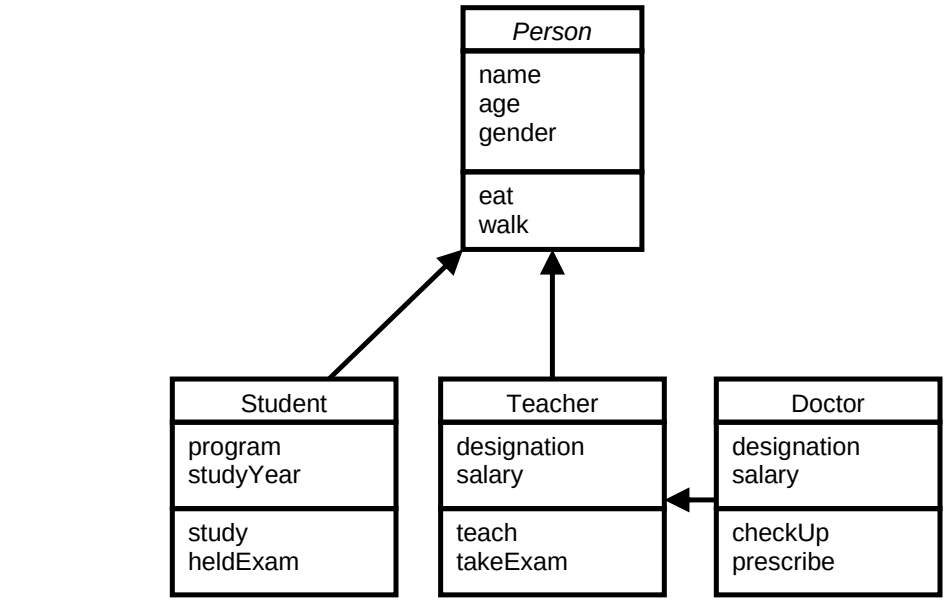
1. Reuse
2. Less redundancy
3. Increased maintainability

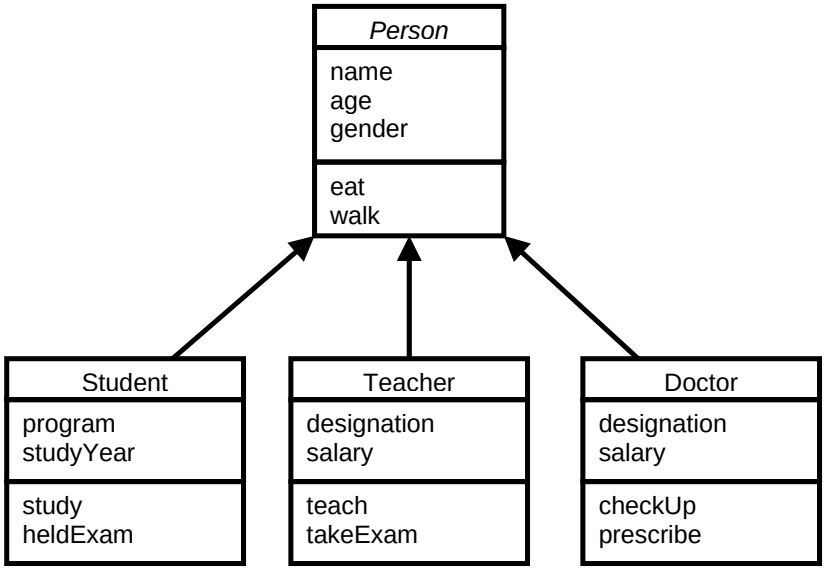
Reuse with Inheritance

Main purpose of inheritance is reuse, we can easily add new classes by inheriting from existing classes.

Select an existing class closer to the desired functionality, create a new class and inherit it from the selected class, add to and/or modify the inherited functionality







 © Virtual University of Pakistan

Lecture No.04

Lecture Contents

- Generalization
- Sub typing (extension)
- Specialization (restriction)
- Overriding
- Abstract classes
- Concrete classes

Recap – Inheritance

- Derived class inherits all the characteristics of the base class
- Besides inherited characteristics, derived class may have its own unique characteristics
- Major benefit of inheritance is reuse

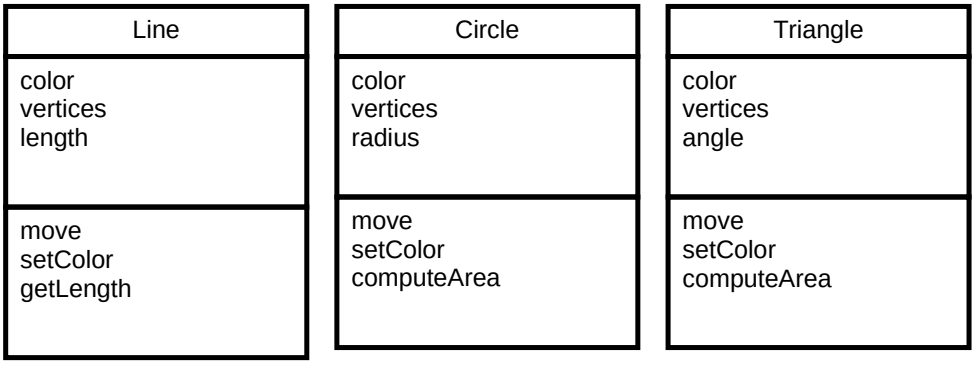
04.1.Concepts Related with Inheritance

- Generalization
- Subtyping (extension)
- Specialization (restriction)

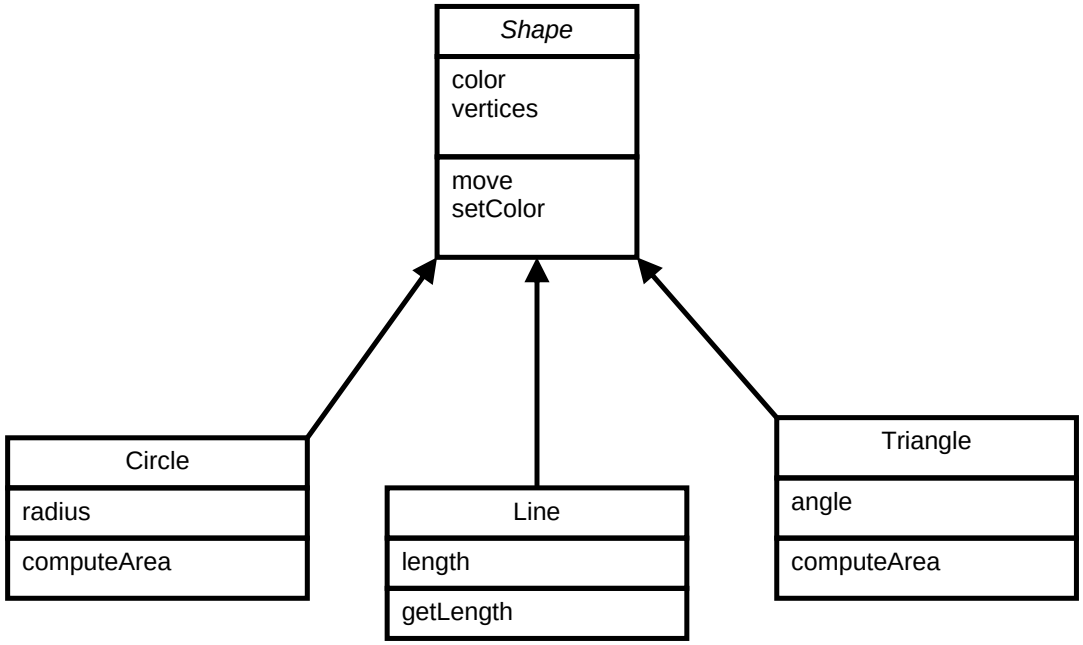
04.2.Generalization

In OO models, some classes may have common characteristics.
We extract these features into a new class and inherit original classes from this new class. There are many objects with common characteristics in object model. The common characteristics (attributes and behaviour) of all these objects are combined in a single general class. Base class encapsulates the idea of commonality of derived classes. Base class is general class representing common behaviour of all derived classes.
This concept is known as Generalization.
It reduces the redundancy and gives us reusability, using generalization our solution becomes less complex.
In generalization there should be "Is a Kind of Relationship" (also called "Is A relationship") between base and child classes.

Example: Line, Circle and Triangle



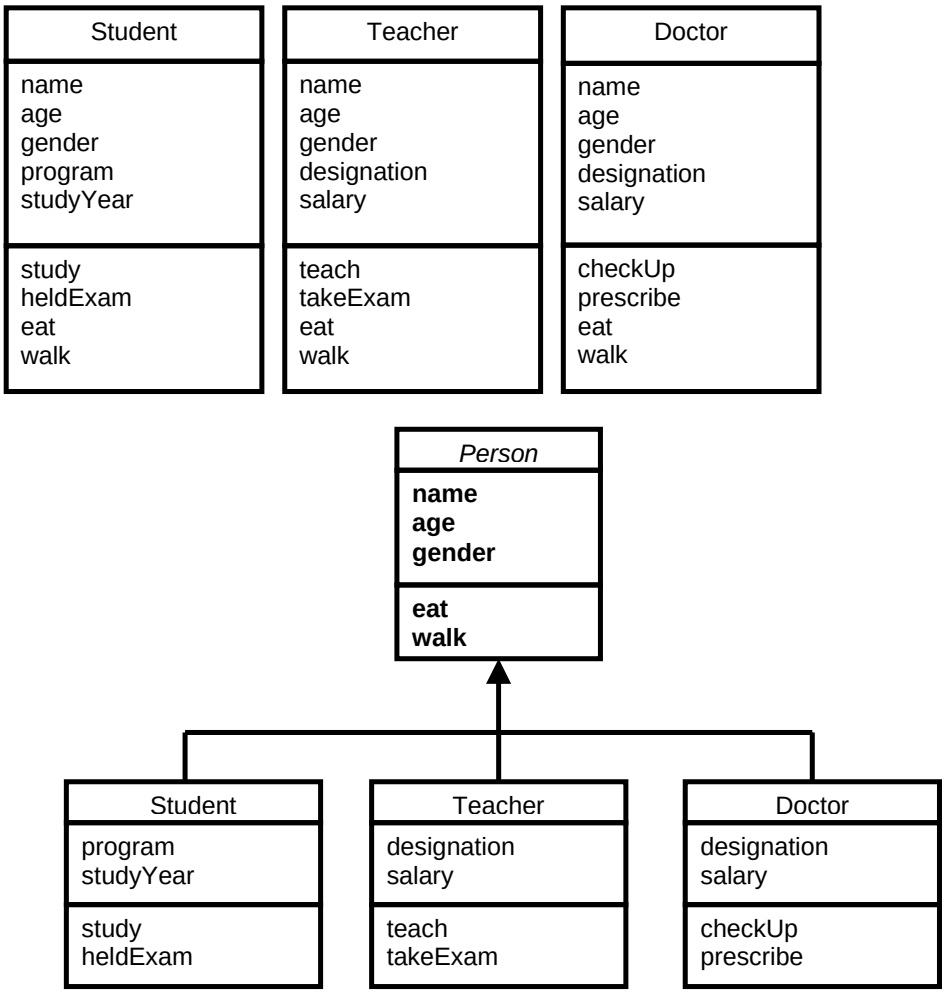
Line is shape Circle is a shape Triangle is a shape



Common attributes
Color vertices
Common behaviour
Set Color, Move

Example: Student Doctor and Teacher

 © Virtual University of Pakistan



Common attributes,
Name, age, gender

Common behaviour
Eat, Walk

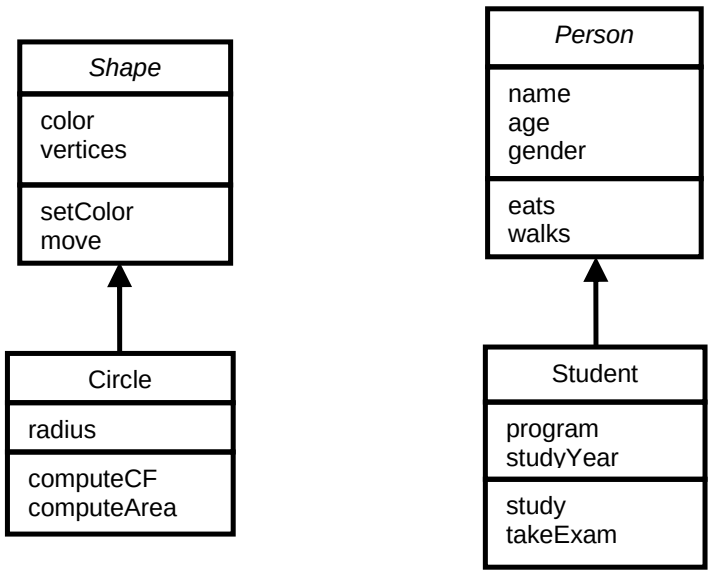
Sub-typing & Specialization

We want to add a new class to an existing model
We have developed an existing class hierarchy
Find an existing class that already implements some of the desired state and behaviour
Inherit the new class from this class and add unique behaviour to the new class

04.3.Sub-typing (Extension)

Sub-typing means that derived class is behaviourally compatible with the base class
Derived class has all the characteristics of base class plus some extra characteristics
Behaviourally compatible means that base class can be replaced by the derived class

Sub-typing (Extension) - Example



Circle is extending the behaviour of shape, it is extending attributes of shape by adding radius similarly it is extending behaviour of shape by adding compute Circumference and compute Area.

Student has two extra attributes program and studyYear
Similarly it has extended behaviour by adding study and takeExam.

Subtyping and generalization are related concepts, Subtyping (extension) and generalization is a way to look same thing in two ways.
Sub typing is looking at things from Top to bottom whereas in generalization we look at things from bottom to top.

04.4. Specialization (Restriction)

We want to add a class to existing hierarchy of classes having many similarities to already existing classes but some part of its behaviour is different or restricted. In that case we will use the concept of specialization.

Specialization means that derived class is behaviourally incompatible with the base class

Behaviourally incompatibility means that base class can't always be replaced by the derived class

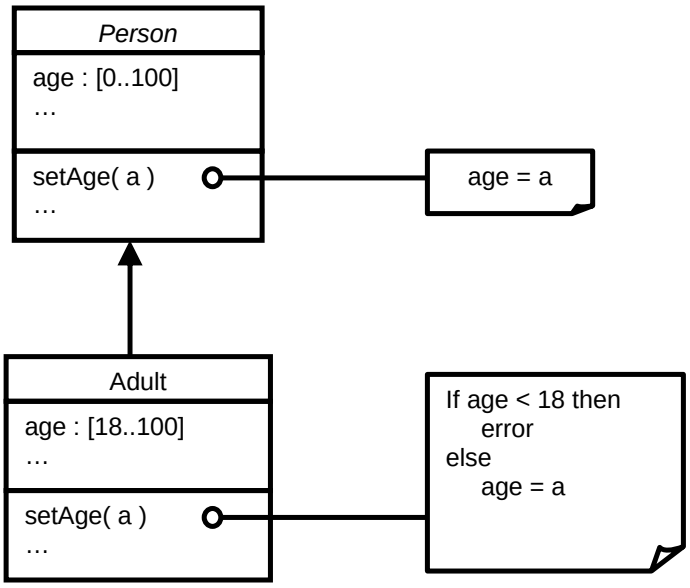
Derived class has some different of restricted characteristics than of base class.

Example – Specialization (Restriction)

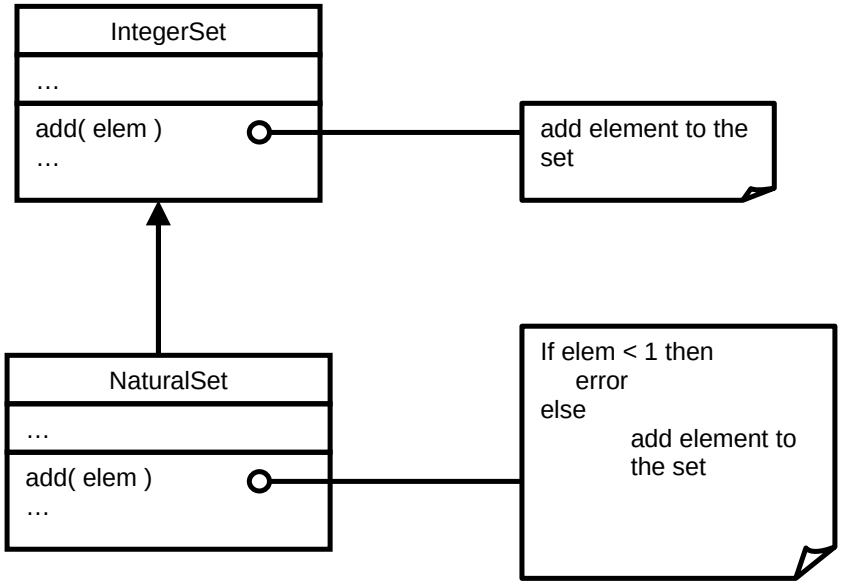
Suppose we want to add one more class of Adult for some special requirement like for ID card generation such that it is a person but its age is greater than 18 and having all other behaviour of that of person class. One solution is that we write

 © Virtual University of Pakistan

another class from beginning and write all code of person again in it with age limit, but better solution is that we derive adult class from person class and restrict age in that class as shown below in diagram,



Similarly Natural Numbers[2] are also Integers[3] with the restriction that natural numbers set can NOT contain zero or negative integers it consists of only positive integers so we can implement this relationship also as specialization,



[2] Natural numbers: positive integers only (numbers from 1 to …….onwards)

[3] Integers: all positive and negative numbers (…..-3 , -2 , -1 , 0 , 1 , 2 , 3……)

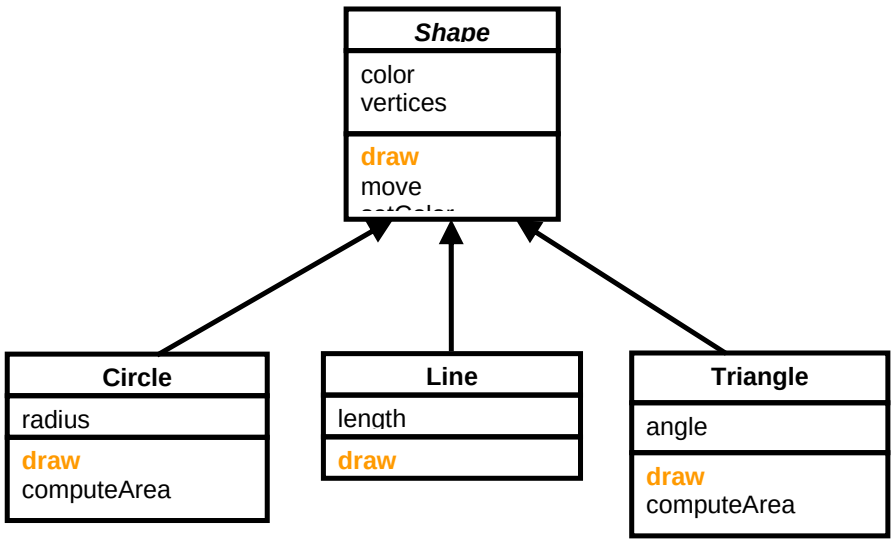
Add method behaviour is present in both base and derived classes but derived class behaviour is different in derived class. Derived class will not exhibit the behaviour of base class but it is **overriding** behaviour of base class with its own behaviour.

04.5.Overriding

A class may need to override the default behaviour provided by its base class
Derived class overrides the behaviour of its base class.
Reasons for overriding
- Provide behaviour specific to a derived class (specialization)
- Extend the default behaviour (extension)
- Restrict the default behaviour (restriction)
- Improve performance

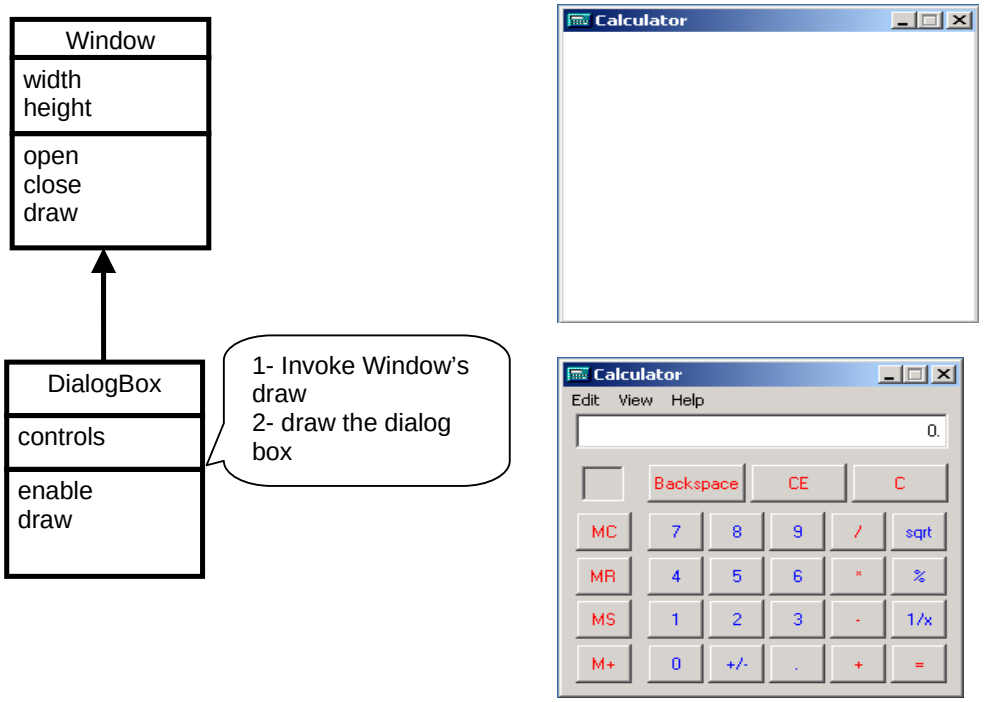
It is used for the implementation of inheritance.

Example – Specific Behaviour (Specialization)

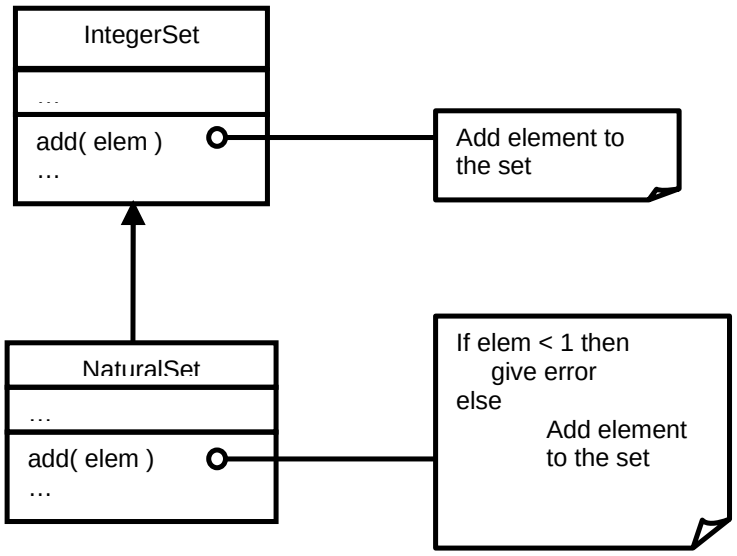


© Virtual University of Pakistan

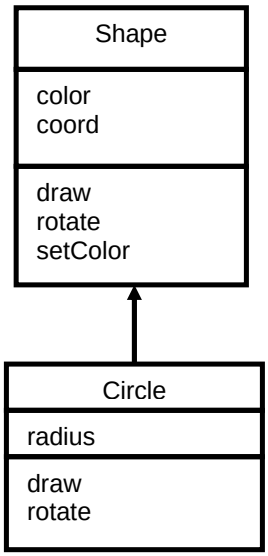
Example – Extention



Example – Restriction



Example – Improve Performance
Class Circle overrides *rotate* operation of class Shape with a Null operation.

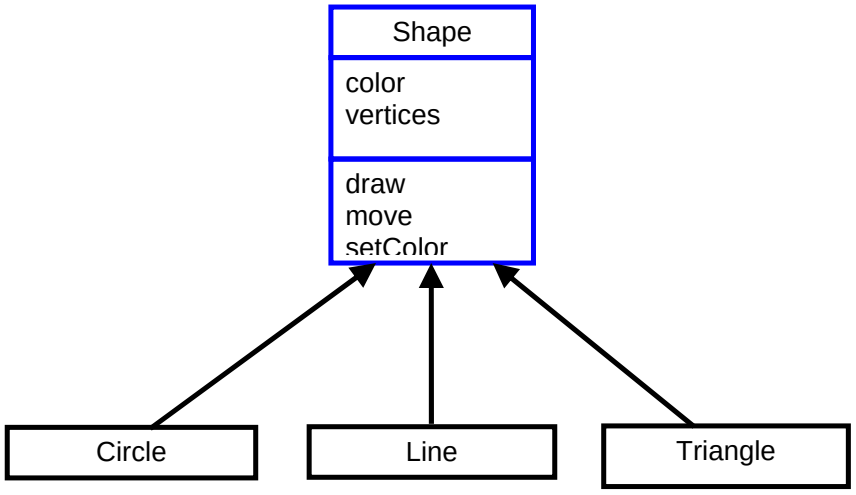


04.6. Abstract Classes

In our examples we made classes for shape and person. These are abstract concepts and the classes we make against abstract concepts are called abstract classes. They are present at or near the top in the class hierarchy to present most generalized behaviour.

An abstract class implements an abstract concept
Main purpose is to be inherited by other classes
Can't be instantiated
Promotes reuse

Abstract Classes - Example I

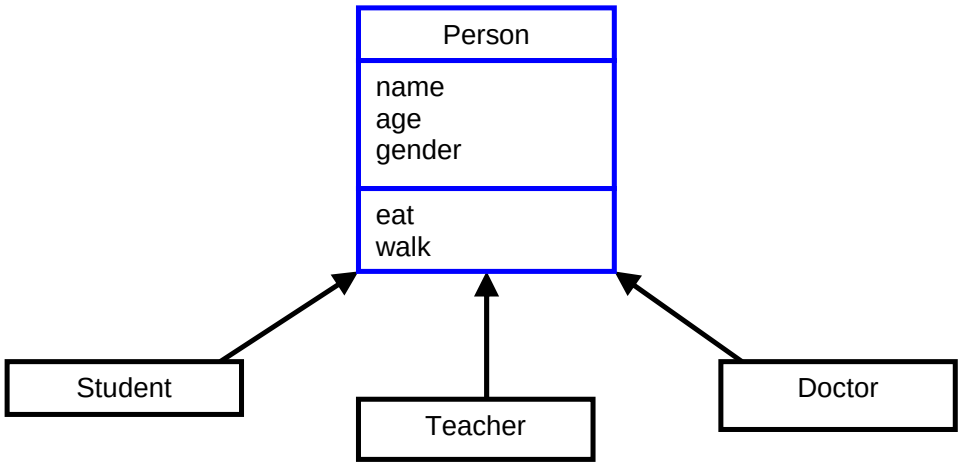


Here, Shape is an abstract class

© Virtual University of Pakistan

Abstract Class	Shape			
Concrete Classes	Circle	Line	Triangle	….

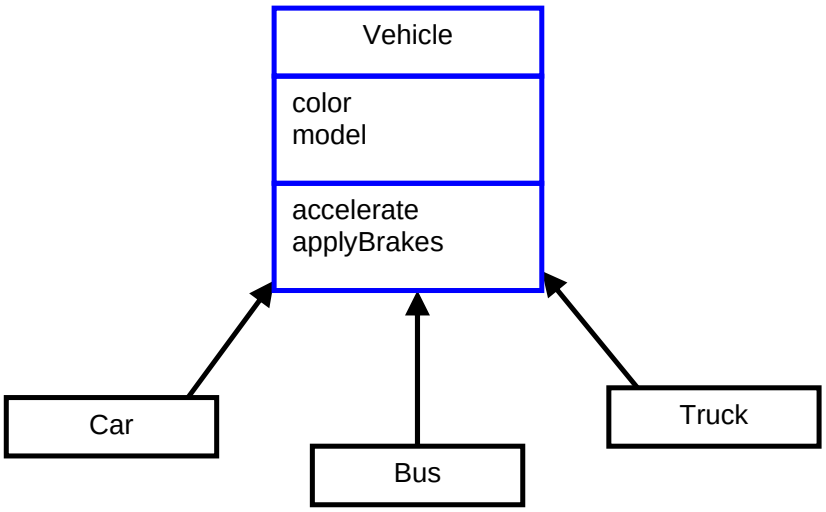
Abstract Classes - Example II



Here, Person is an abstract class

Abstract Class	Person					
Concrete Classes	Student	Teacher	Doctor	Engineer	Director	….

Abstract Classes - Example III



Here, Vehicle is an abstract class

Abstract Class	Vehicle			
Concrete Classes	Car	Bus	Truck	….

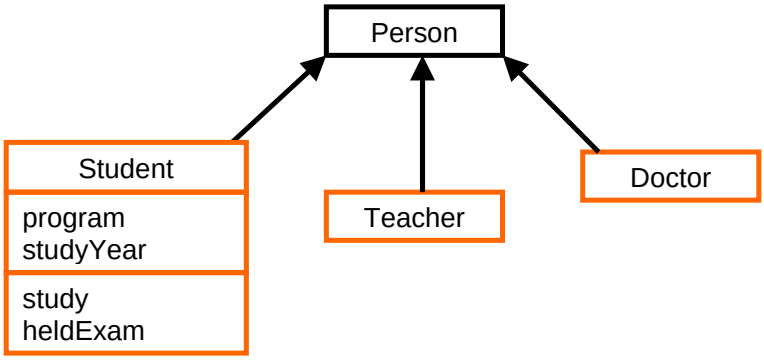
Abstract Classes can not exist standalone in an object model
While making object model we start by finding out objects in our object model and then we find out objects having common attributes and make them in the form of general classes at the top of class hierarchies.

04.7.Concrete Classes

The entities that actually we see in our real world are called concrete objects and classes made against these objects are called concrete classes.

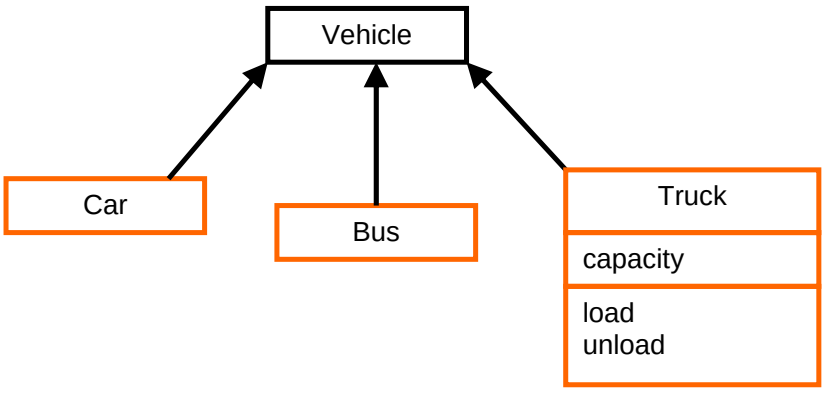
A concrete class implements a concrete concept
These are used to instantiate objects in our programs
Provides implementation details specific to the domain context

Concrete Classes - Example I



Here Student, Teacher and Doctor are concrete classes

Concrete Classes - Example II



Here Car, Bus and Truck are concrete classes

© Virtual University of Pakistan

- A concrete class may exist in an object model independently
- Concrete classes mostly lie below the top of class hierarchy in a good object model.

If there is an abstract class then hierarchy exists in the object model as there will definitely be some concrete classes as well derived from this abstract class otherwise there is no use of abstract class.

Glossary:

a. Natural numbers: numbers from 1 to …….onwards
b. Integers: all positive and negative numbers …..-3,-2,-1,0,1,2,3……
c. Whole numbers: numbers from 0 ,1 ,2, 3 ….onwards (natural no's including 0)

Some times whole numbers are also called numbers without fractional part.

Lecture No.05
Multiple Inheritance

Inheritance:
We saw inheritance purposes in last lecture

- Generalization
- Extention or sub typing
- Specialization or restriction

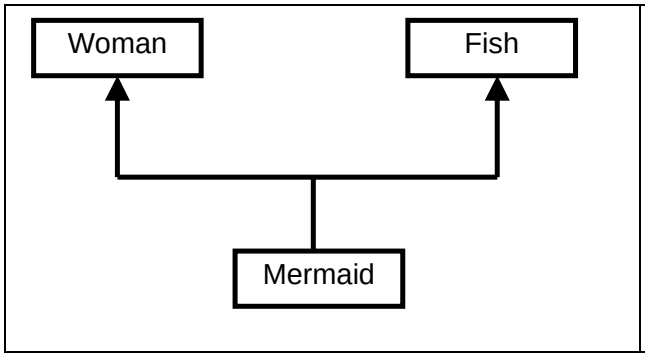
Abstract and concrete classes, former is used to represent abstract concepts later is used to represent concrete concepts.
Overriding derived classes override inherited classes (base classes) behaviour.
Overriding is used for Specialization, Extention, Restriction, and Performance.

05.1.Multiple Inheritance

Sometimes we want to reuse characteristics of more than one parent class, in that case we need to inherit a class from more than one classes.

Example 1– Multiple Inheritance

Consider the example of an imaginary specie Mermaid used in fairy tales that lives in water having features both of a women as well as of a fish, In Object Oriented programming perspective Mermaid can be derived from two classes Women and Fish.



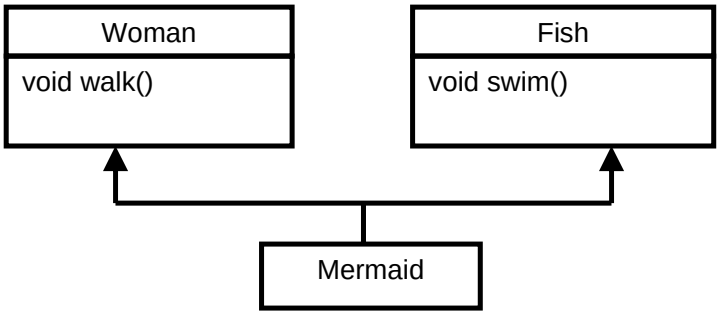
C++ Code:

```
/*Program to demonstrate simple multiple inheritance*/

class Fish {


};

class Woman {
```


© Virtual University of Pakistan

```
};

class Mermaid : public Woman , public Fish {

};
```

Our Mermaid class inherits features of both woman and fish suppose our woman class has method wald() and fish cclass has method swim then our mermaid class can use both methods i.e can walk as well as can swim.



C++ code:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

/*Program to demonstrate simple multiple inheritance*/

class Fish
{
public:
    void swim(){
    cout<<"\n In method swim";
    }

};

class Woman
{
public:
    void walk(){
    cout<<"\n In method walk"<<endl;
    }

};
```

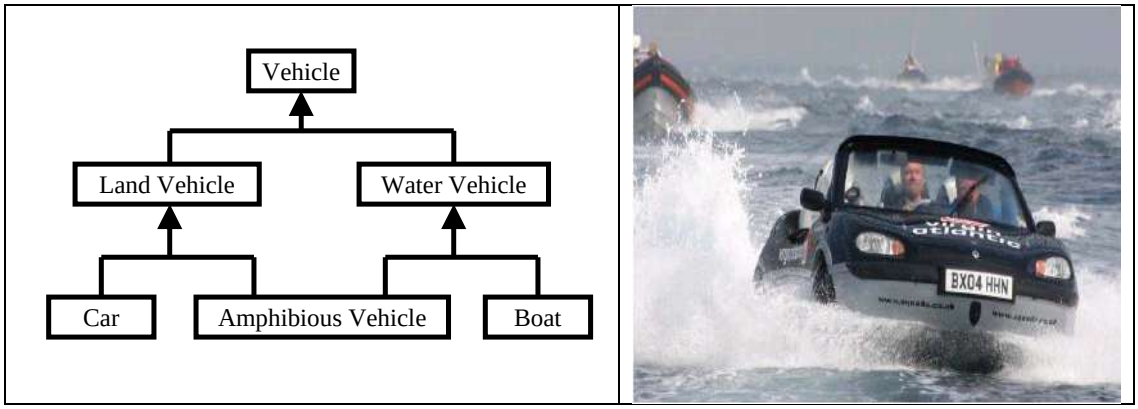
```
class Mermaid : public Woman,public Fish
{

};

int main(int argc, char *argv[])
{

Mermaid mermaid;
/*This Mermaid object will have two implicit objects one of Fish class and one of
Woman class*/
mermaid.swim();
mermaid.walk();

system("PAUSE");
return 0;

}
```

Output:

```
In method[4] swim
In method walk
```

Example 2– Multiple Inheritance

Take another example of amphibious vehicle (vehicle that can run on land as well as on water) so it has properties of both land as well as of water vehicle. The general hierarchy in this case will be,



Here we have added a general Vehicle class as well to add all common functions of Land Vehicles and Water Vehicles in that class, and specific functions of Land and

[4] class member functions are also called class methods

© Virtual University of Pakistan

Water vehicle in their respective classes then we have derived Amphibious Vehicle class from Land Vehicle and Water Vehicle classes (we can do the same in first example as well concerning Woman, Fish and Mermaid).

C++ code:

```
class Vehicle
{

};

class WaterVehicle : public Vehicle
{


};

class LandVehicle : public Vehicle
{


};

class AmphibiousVehicle : public LandVehicle,public WaterVehicle
{

};
```

Suppose we have a **changeGear** method in Vehicle class that is applicable to both water and land vehicle, we also have Float and Move methods in water and land vehicles respectively then our amphibious vehicle will have all these methods,

C++ code:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

/*Multiple Inheritance in case of Amphibious Vehicle*/

class Vehicle
{
public:
 void changeGear(){ cout<<"\nI am Vehicle changeGear() function..\n";}
};

class WaterVehicle : public Vehicle
{
```

```
public:
void Float(){ cout<<"\nI am float function of Water Vehicle";}
};


class LandVehicle : public Vehicle
{
public:
void Move(){ cout<<"\nI am move function of Land Vehicle"<<endl;}

};

class AmphibiousVehicle : public LandVehicle,public WaterVehicle
{


};

int main(int argc, char *argv[])
{

AmphibiousVehicle amphibious;

amphibious.Float();
/*Calling Float function of Water Vehicle class*/

amphibious.Move();
/*Calling Move function of Land Vehicle class*/

system("PAUSE");
return 0;


}
```

Output:

```
I am float function of Water Vehicle
I am move function of Land Vehicle
```

Advantage of Multiple Inheritance:

As was the case with simple (single) inheritance multiple inheritance also decreases redundant code as we can inherit a class from many classes and can use their functions without the need to write them again.

© Virtual University of Pakistan

However, there are more disadvantages of multiple inheritance, than its advantages.

Problems with Multiple Inheritance

Increased complexity

Amphibious vehicle hierarchy is a complicated as this class is derived from two classes that will make code more complex and less understandable however this is obvious as amphibious vehicle is a complicated vehicle. It is generic problem.

Reduced understanding

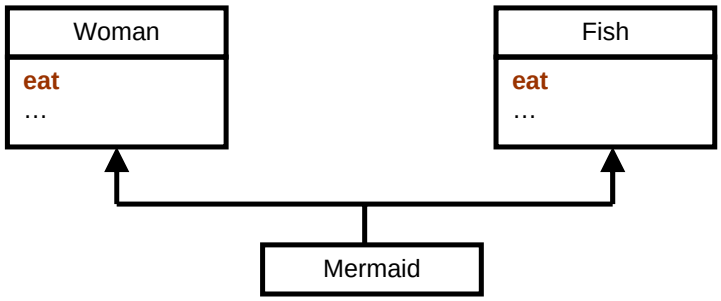
Due to increased complexity of class hierarchy the object model becomes difficult it understand especially for someone who is looking it first time.

Duplicate features

As we are deriving a single class from more than one class so there is a chance of duplication of features (same methods in both parents), following problems may arise due to duplicate features,

Problem 1: Ambiguity

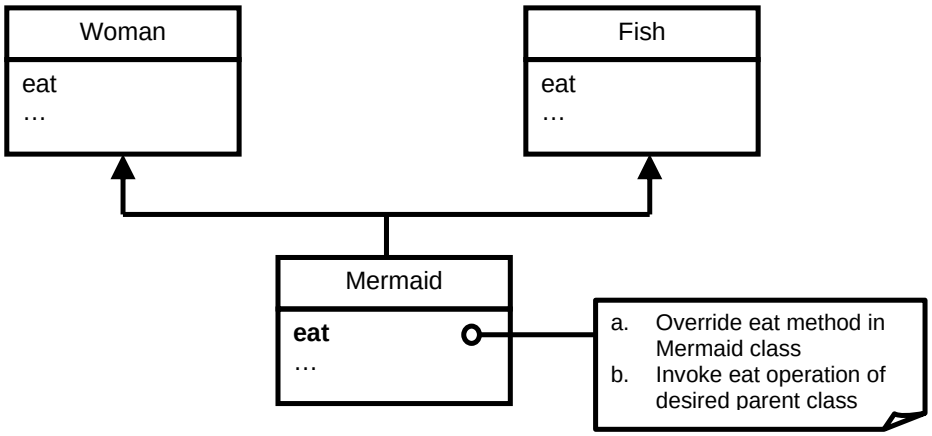
Consider the class hierarchy of Mermaid class below,



As mermaid also needs to eat and its both parents have their own methods of eating so here question arises,

Which eat operation Mermaid should inherit as both functions are available?

Solution – We can solve this problem by explicitly calling eat method from any of the parent classes in Mermaid class according to behaviour of Mermaid (i.e. if it eats like a Woman we can call eat method of Woman class and if eats like Fish we can call method of Fish class), for this we will Override the Common method in multiply inherited class and in that class overridden method we will call the appropriate base class function.



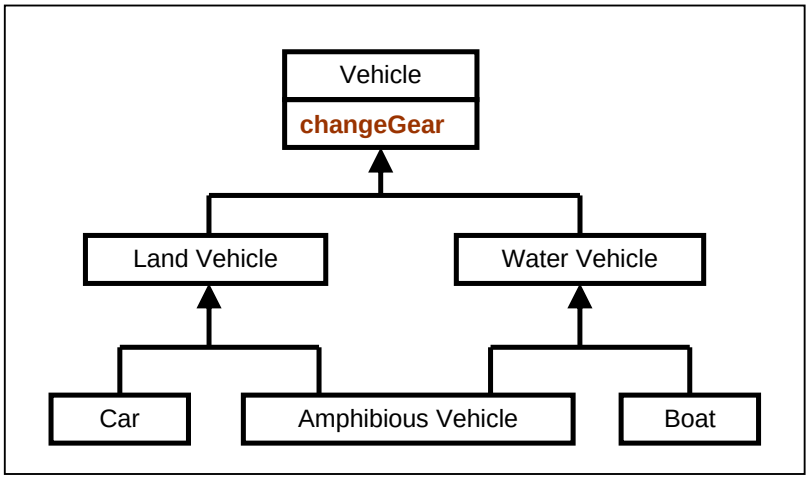
Example C++ Code

```
#include <iostream>
#include <stdlib.h>

using namespace std;

/*Program to demonstrate simple multiple inheritance*/

class Fish
{
public:
      void eat(){
      cout<<"\n In Fish eat method ";
      }

};

class Woman
{
public:
      void eat(){
      cout<<"\n In Woman eat method \n"<<endl;
      }

};

class Mermaid : public Woman,public Fish
{

     public:

      void eat(){
```

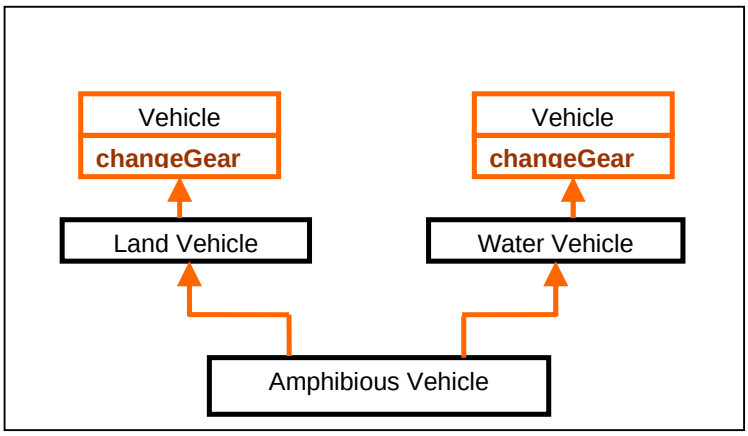
 © Virtual University of Pakistan

```
    cout<<"\n In Mermaid eat method "<<endl;
    cout<<"\n Explicity calling Woman eat method...."<<endl;

    Woman::eat();

    }

};

int main(int argc, char *argv[])
{


Mermaid mermaid;
/*This Mermaid object will have two implicit objects one of Fish class and one of
Woman class*/
mermaid.eat();
/*Calling Mermaid eat method*/

system("PAUSE");
return 0;

}
```

Problem 2: Two instances for same function (Diamond Problem)



Here Amphibious Vehicle will have two copies of changeGear function as it will have two objects of Vehicle class one with respect to Land Vehicle and one with respect to Water Vehicle as shown below,



Actual Memory Layout

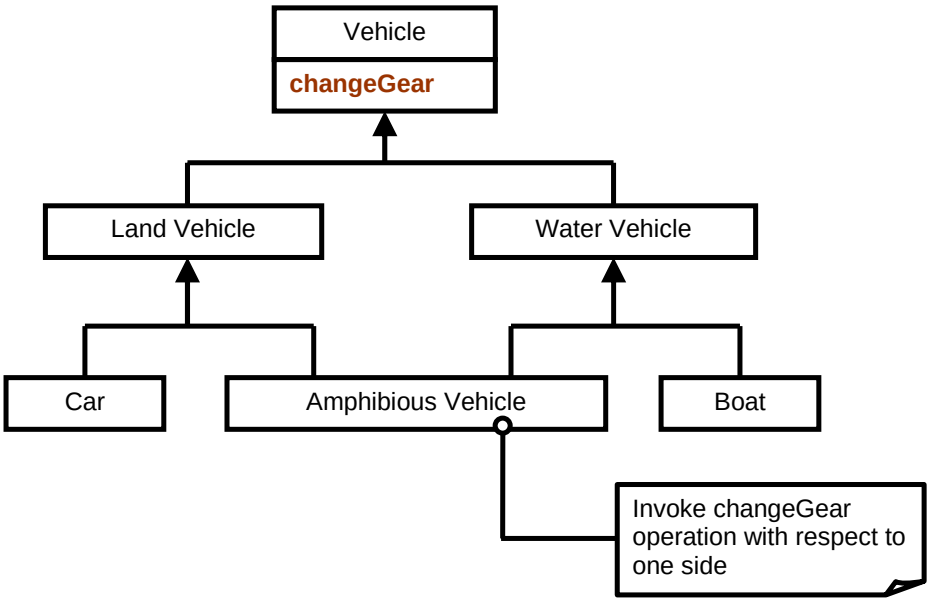
Compiler will not be able to decide which changeGear operation Amphibious Vehicle should inherit so it will generate an error as shown below (two copied of same method),

error: request for member `changeGear' is ambiguous
error: candidates are: **void Vehicle::changeGear()**
void Vehicle::changeGear()
Execution terminated

Solution to Diamond Problem

Some languages disallow diamond hierarchy
Others provide mechanism to ignore characteristics from one side. There are two cases while solving diamond problem virtual inheritance and non virtual inheritance (we will study these details in coming lectures)

© Virtual University of Pakistan



Association:

Interaction of different objects in OO model (or in problem domain) is known as association.
In object oriented model, objects interact with each other in order to perform some useful work, while modeling these objects (entities) is done using the association. Usually an object provides services to several other objects. An object keeps association with other objects to delegate tasks. This association can be represented with a line along an arrow head (⟶) or without arrow head.

05.2.Kinds of Association:
There are two main types of association which are then further subdivided i.e

1. Class Association
2. Object Association

1. Class Association

Class association is implemented in terms of Inheritance. Inheritance implements generalization/specialization relationship between objects. Inheritance is considered class association.

- In case of public inheritance it is "IS-A" relationship.
- In case of private inheritance it is "Implemented in terms of" relationship.

This relationship ensures that public members of base class are available to derived class in case of public inheritance.

When we create objects of classes in which we have implemented inheritance relationships we are forcing the inheritance relationship between created objects. In this case the derived class objects will also contain base class objects attributes and methods.

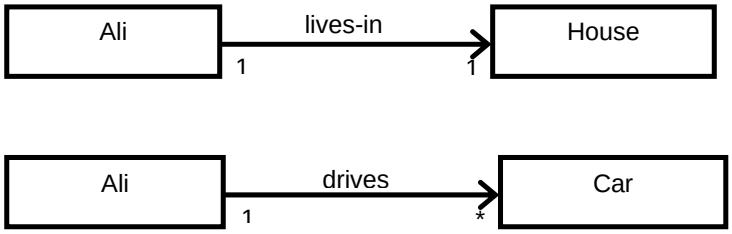
2. Object Association

It is the interaction of stand alone objects of one class with other objects of anther class.
It can be of one of the following types,

- Simple Association
- Composition
- Aggregation

05.3.Simple Association

The two interacting objects have no intrinsic relationship with other object. It is the weakest link between objects. It is a reference by which one object can interact with some other object.
Customer gets cash from cashier
Employee works for a company
Ali lives in a house
Ali drives a car



It is generally called as "**association**" instead of "**simple association**"

Kinds of Simple Association

Simple association can be categorized in two ways,

- With respect to direction (navigation)
- With respect to number of objects (cardinality)

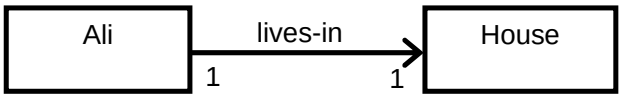
Kinds of Simple Association w.r.t Navigation

With respect to navigation association has the following types,

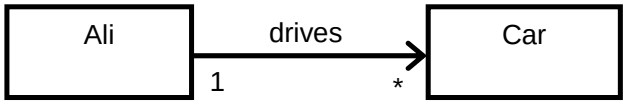
a. One-way Association
b. Two-way Association

a. One-way Association

In One way association we can navigate along a single direction only, it is denoted by an arrow towards the server object.
Examples:



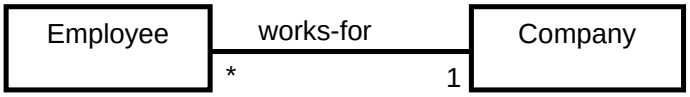
- Ali lives in a House



- Ali drives his Car

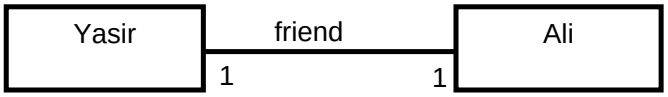
b. Two-way Association

In two way association we can navigate in both directions, it is denoted by a line between the associated objects
Examples:



Employee works for company
Company employs employees

Two-way Association - Example



Yasir is a friend of Ali
Ali is a friend of Yasir

Kinds of Simple Association w.r.t Cardinality

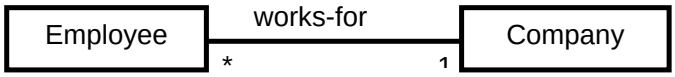
With respect to cardinality association has the following types,

a. Binary Association
b. Ternary Association
c. N-ary Association

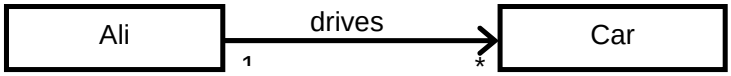
a. Binary Association

It associates objects of exactly two classes; it is denoted by a line, or an arrow between the associated objects.

Example



Association "works-for" associates objects of exactly two classes



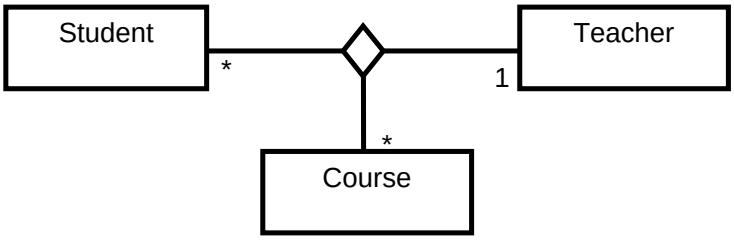
Association "drives" associates objects of exactly two classes

b. Ternary Association

It associates objects of exactly three classes; it is denoted by a diamond with lines connected to associated objects.

Example

Objects of exactly three classes are associated



c. N-ary Association

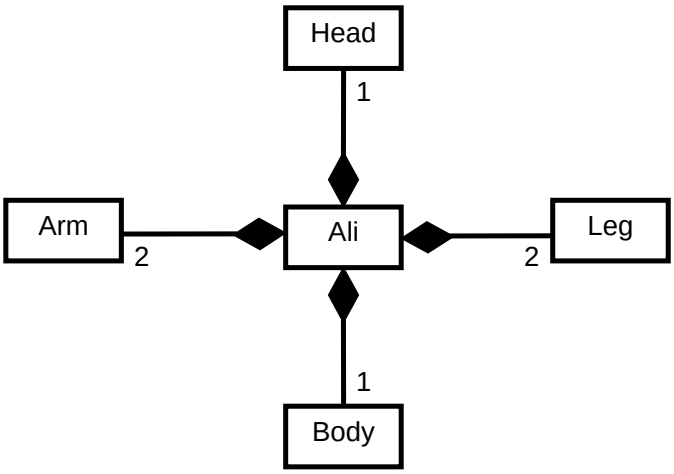
An association between 3 or more classes its practical examples are very rare.

05.4.Composition

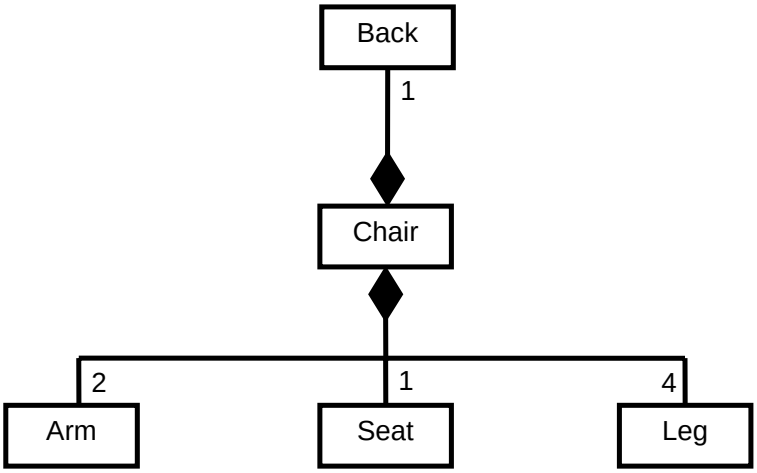
An object may be composed of other smaller objects, the relationship between the "part" objects and the "whole" object is known as Composition, Composition is represented by a line with a filled-diamond head towards the composer object

Example – Composition of Ali

 © Virtual University of Pakistan



Example – Composition of Chair



Composition is stronger relationship:
Composition is a stronger relationship, because
Composed object becomes a part of the composer
Composed object can't exist independently

Example I

Ali is made up of different body parts

They can't exist independent of Ali

Example II

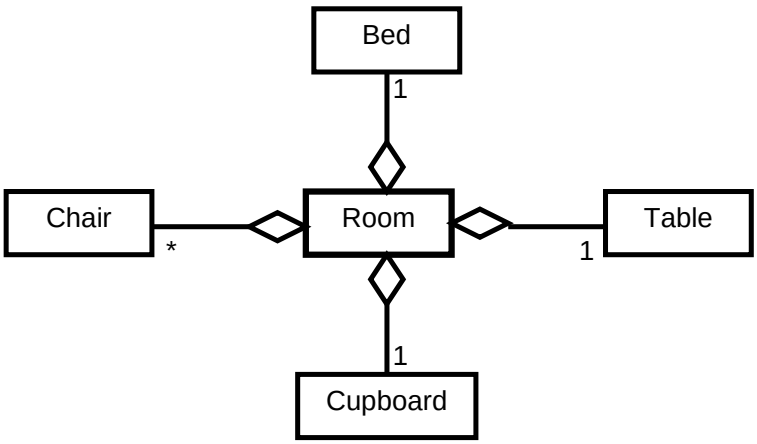
Chair's body is made up of different parts

They can't exist independently

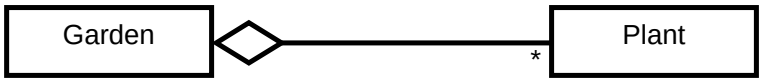
05.5. Aggregation

An object may contain a collection (aggregate) of other objects, the relationship between the container and the contained object is called aggregation, Aggregation is represented by a line with unfilled-diamond head towards the container

Example – Aggregation



Example – Aggregation



Aggregation is weaker relationship

Aggregation is weaker relationship, because

- Aggregate object is not a part of the container
- Aggregate object can exist independently

Example I
Furniture is not an intrinsic part of room
Furniture can be shifted to another room, and so can exist independent of a particular room

Example II
A plant is not an intrinsic part of a garden
It can be planted in some other garden, and so can exist independent of a particular garden

http://www.codeproject.com/KB/cpp/oopuml.aspx

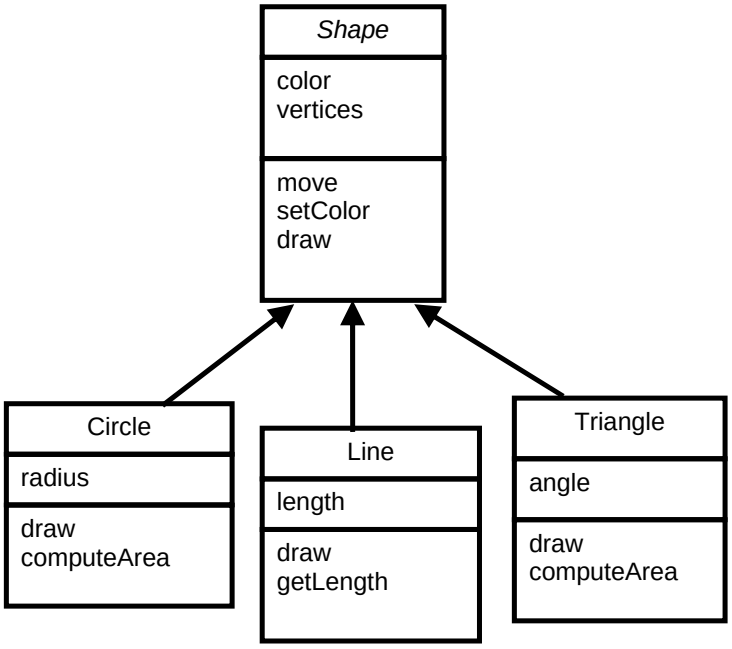
© Virtual University of Pakistan

Lecture No.06

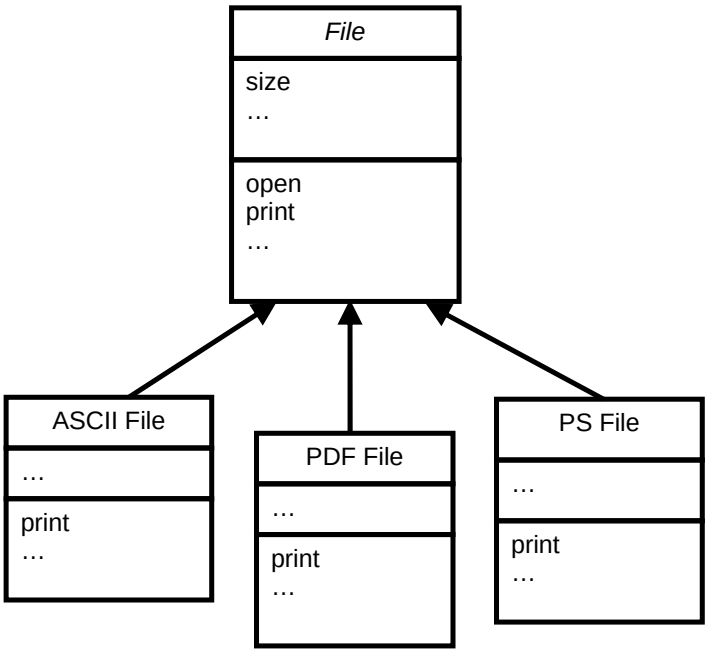
06.1.Class Compatibility

A class is behaviorally compatible with another if it supports all the operations of the other class. Such a class is called subtype. A class can be replaced by its subtype.
Derived class is usually a subtype of the base class. It can handle all the legal messages (operations) of the base class. Therefore, base class can always be replaced by the derived class.

Examples
Child class also includes characteristics of its base class.



All the three derived class are behaviourally compatible with base class.



Wherever the file class is it can be replaced by any of its child classes.

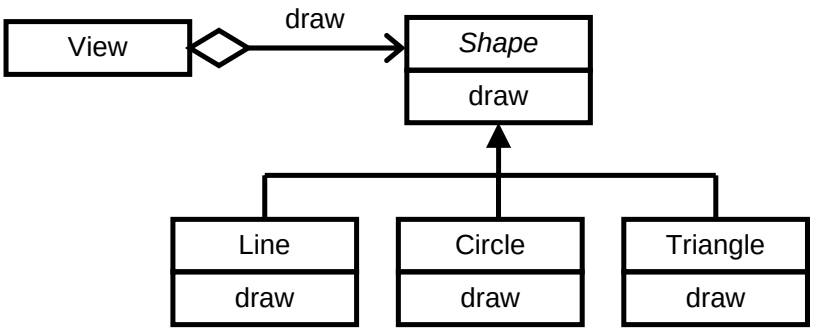
06.2.Polymorphism

It is also essential component of object oriented modeling (paradigm).
In general, polymorphism refers to existence of ***different forms*** of a single entity. For example, both Diamond and Coal are different forms of Carbon.

06.3.Polymorphism in OO Model

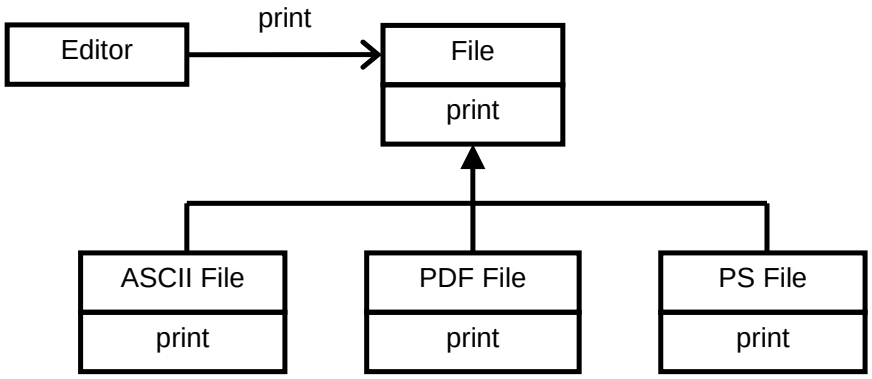
In OO model, polymorphism means that different objects can behave in different ways for the same message (stimulus). Consequently, sender of a message does not need to know exact class of the receiver.
Sender sends message to receiver and appropriate method is called on receiver side.

Example – Polymorphism



 © Virtual University of Pakistan

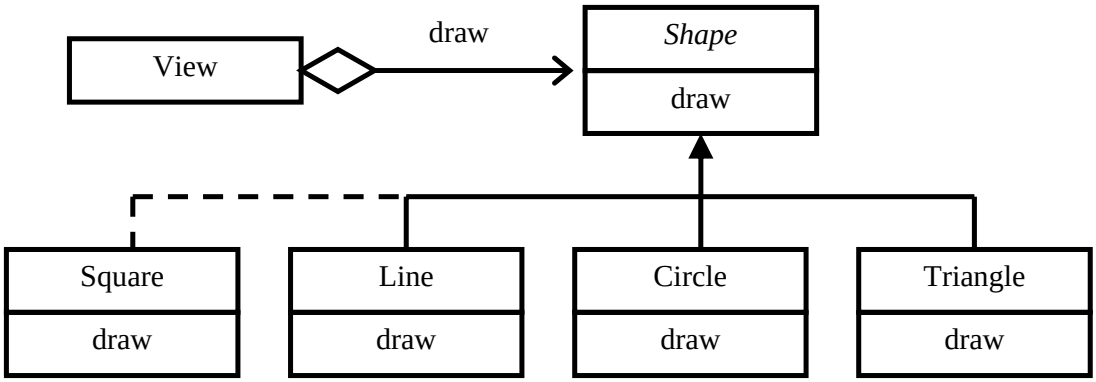
Shape class hierarchy shape is base class and there are three child classes line circle , triangle. View send draw method to shape class and draw is called according to the nature of actual object present.



Editor sends message print to file class and print is called based on the actual child object of file class message is same and appropriate execution will be done.

06.4.Polymorphism – Advantages

Messages can be interpreted in different ways depending upon the receiver class
New classes can be added without changing the existing model



In general, polymorphism is a powerful tool to develop flexible and reusable systems

06.5.Object-Oriented Modeling an Example

Problem Statement

Develop a graphic editor that can draw different geometric shapes such as line, circle and triangle. User can select, move or rotate a shape. To do so, editor provides user with a menu listing different commands. Individual shapes can be grouped together and can behave as a single shape.

Identify Classes

Extract nouns in the problem statement

Develop a graphic **editor** that can draw different geometric **shapes** such as **line**, **circle** and **triangle**. **User** can select, move or rotate a **shape**. To do so, **editor** provides **user** with a **menu** listing different **commands**. Individual **shapes** can be grouped together and can behave as a single **shape**.

Eliminate irrelevant classes

Editor – Very broad scope. But it is the name of overall system and we are going to model it so we will not make its object. For example if we are going to model computer we will not make its object but its components however if it is component of some other system then it will behave as an object. So it is marked as irrelevant.

User – Out of system boundary, it is interacting with the system from outside of the system.

Add classes by analyzing requirements

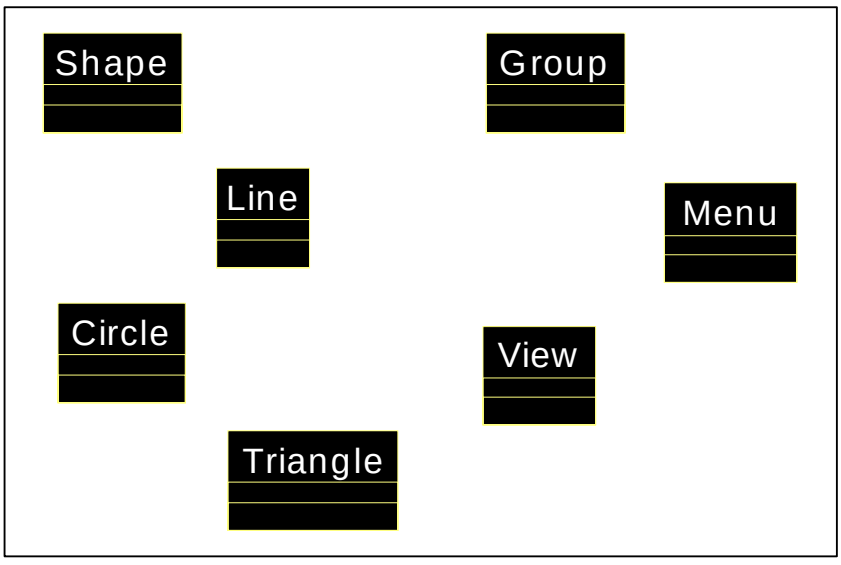
Group (of shapes) – required to behave as a shape so it should behave as an object in our system

"Individual shapes can be grouped together and can behave as a single shape"

View – graphic editor must have a display area to show the shapes. We made this object using domain knowledge.

- Shape
- Line
- Circle
- Triangle
- Menu
- Group
- View

So we have the following classes,



© Virtual University of Pakistan

Finding Associations:

Next step is to find associations between the objects.

Identify Associations

Find relationships between objects,

1. **Extract verbs connecting objects,**

*"Individual shapes can be **grouped together**"*

- Group **consists of** lines, circles, triangles
- Group can also **consists of** other groups (Composition)

Line, circle and triangle have composition relationship.

2. **Verify access paths**

   a. *View contains (draws) shapes*

      - *View contains lines*
      - *View contains circles*
      - *View contains triangles*
      - *View contains groups*

So there is **Aggregation relationship** between shapes and View.

Menu sends message to View

So there is Simple **One-Way Association relationship** between Menu and View.

Identify Attributes of the identified objects

Extract properties of the object,

a. From the problem statement
   Properties are not mentioned
b. From the domain knowledge

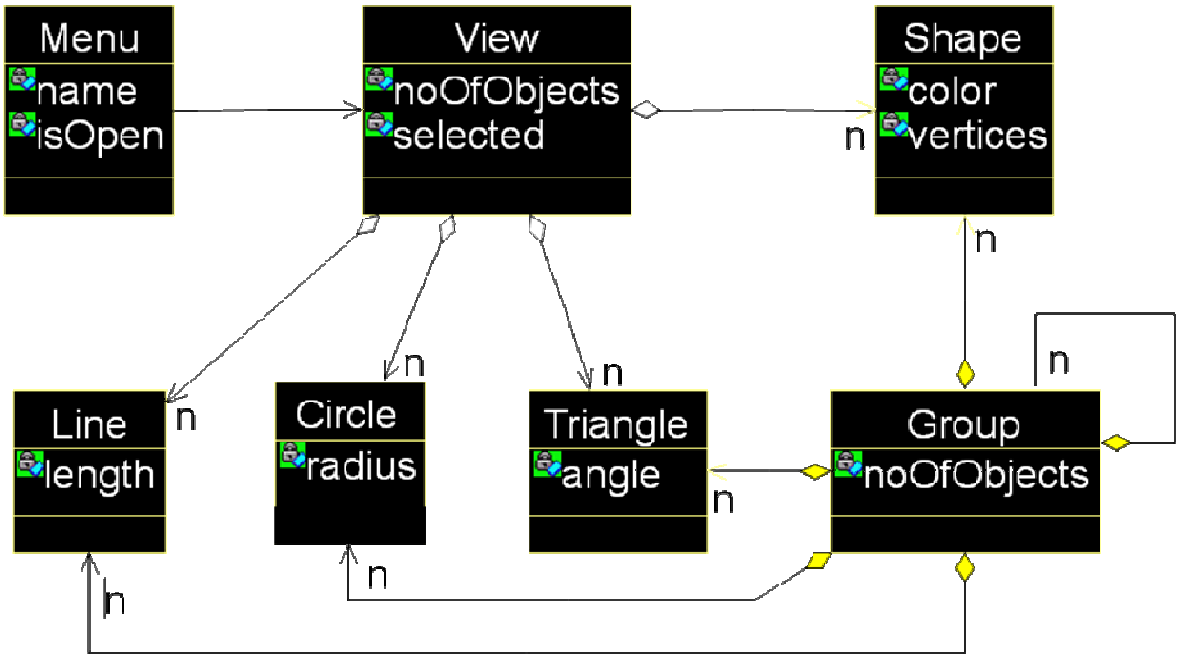
   - Line
     i. Color
     ii. Vertices
     iii. Length
   - Circle
     i. Color
     ii. Vertices
     iii. Radius
   - Triangle

i. Color
ii. Vertices
iii. Angle

- Shape
  i. Color
  ii. Vertices
- Group
  i. noOfObjects
- View
  i. noOfObjects
  ii. selected
- Menu
  i. Name
  ii. isOpen

Object Model – Graphic Editor

Object model so far is shown below,



Identify Operations

Extract verbs connected with an object

Develop a graphic editor that can **draw** different geometric shapes such as line, circle and triangle. User can **select**, **move** or **rotate** a shape. To do so, editor **provides** user with a menu listing different commands. Individual shapes can be **grouped** together and can **behave** as a single shape.

Eliminate irrelevant operations

© Virtual University of Pakistan

Develop – out of system boundary
Behave – have broad semantics

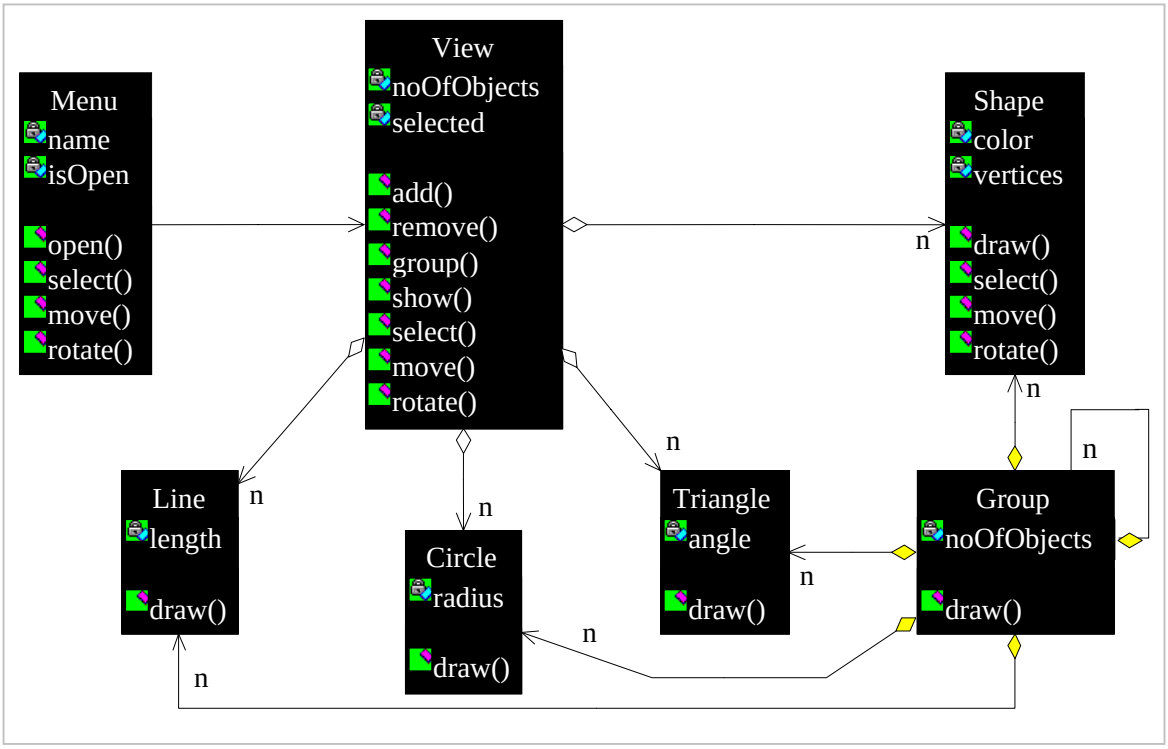
Following are selected operations:

- Line
  - Draw
  - Select
  - Move
  - Rotate
- Circle
  - Draw
  - Select
  - Move
  - Rotate
- Triangle
  - Draw
  - Select
  - Move
  - Rotate
- Shape
  - Draw
  - Select
  - Move
  - Rotate
- Group
  - Draw
  - Select
  - Move
  - Rotate
- Menu
  - Open
  - Select
  - Move
  - Rotate

Extract operations using domain knowledge

- View
  - Add
  - Remove
  - Group
  - Show
  - Select
  - Move
  - Rotate

Object Model after adding operations:



Identify Inheritance

a. Search lines like "is a kind of" by looking at keywords like "such as", "for example", etc

"…shapes such as line, circle and triangle…"

⇨ Line, Circle and Triangle inherits from Shape

By analyzing requirements

"Individual shapes can be grouped together and can behave as a single shape"

⇨ Group inherits from Shape

Refining the Object Model

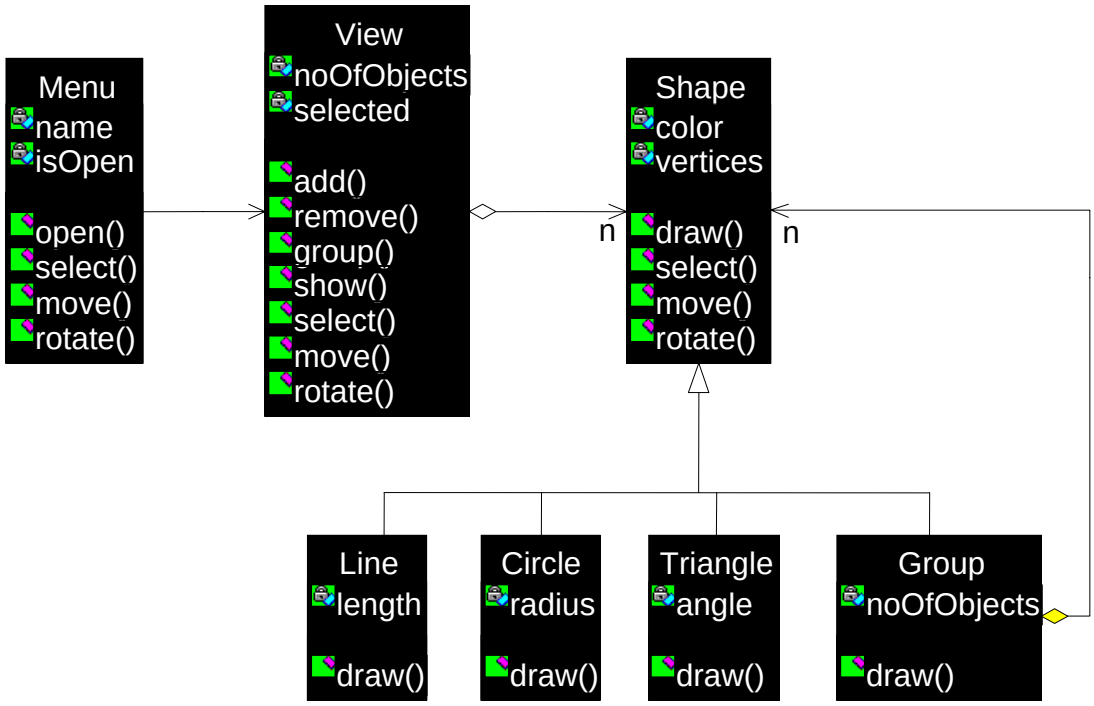
Application of inheritance demands an iteration over the whole object model
In the inheritance hierarchy,

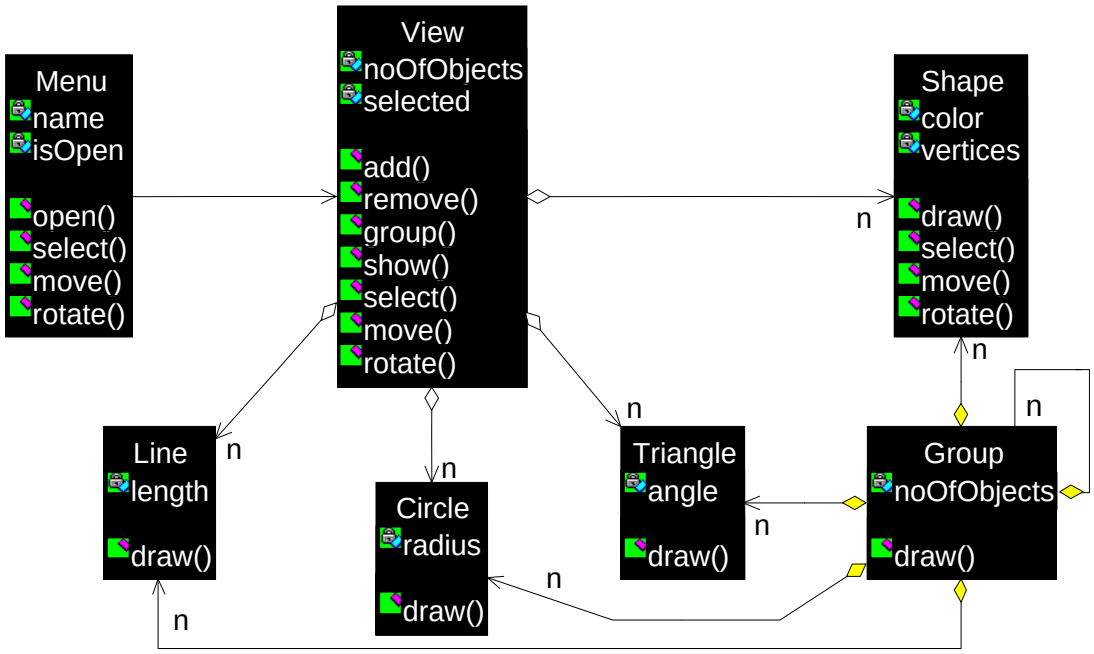
- All attributes are shared
- All associations are shared
- Some operations are shared
- Others are overridden

© Virtual University of Pakistan

Refining the Object Model

- Share associations
  - View contains all kind of shapes
  - Group consists of all kind of shapes
- Share attributes
  - Shape – Line, Circle, Triangle and Group
    - Color, vertices
- Share operations
  - Shape – Line, Circle, Triangle and Group
    - Select
    - Move
    - Rotate
- Share the interface and override implementation
  - Shape – Line, Circle, Triangle and Group
    - Draw





 © Virtual University of Pakistan

Lecture No.07

The basic concept "**Object**" of *Object Orientation (thinking in terms of objects)* is realized using classes in programming languages.

07.1.Class

It is a way (Mechanism) given by c++ to realize objects in a program. It is concrete implementation of objects in c++. We capture any object attributes and behaviour in a programming language using classes.
In other words it can be defined as facility given by c++ to create new types according to our requirement. (Class is composite data type made from basic c++ types like integers, chars and float).

Example:
Consider the examples of entity lion there are many lions but all lions will have similar attributes and behaviour.
Similarly consider student object all students have separate existence but all students have similar attributes and they exhibit similar behaviour.

When we hear word student or think about student a sketch comes in our mind for student along with its attributes and behaviour. The attributes of student comes in our mind are its name, roll no, class, degree so on. Similarly the behaviour of student comes in our mind are study, register and many more.

We need to capture the characteristic features of any object (attributes and behaviour) in the programming language. The concept of class is used for this purpose.

Now consider the scenario having many interacting objects: a **University System** having many objects like student, subject, classroom, and teacher so on…we will realize all these objects in our software using classes. These all object will use the services of each other for example student will ask teacher to teach him. This approach is closer to real life instead of having simple functions being called from main here these objects will call each other to get their services. This is the reason *we say that object oriented programming mimics real life*.

Uses

Objects are structured in terms of class so our problem becomes easier to understand in the terms c++ program.
We can implement interactions easily in terms of classes.

Student objects will interact with each other to take and give services to each other as happens in real life and mapped in object oriented programming approach.

Now we see how class mechanism helps us in implementing real life concept.

07.2. Type in C++

We implement generic concepts using types.

We have to model generic concept of Student. But there is no built in type **student** in c++ like built-in c++ type's **int** or **float**. Class is mechanism in c++ that will allow us to define student as user defined type, similarly generic concept circle will also be implemented in the same way. User define types will be,

- Student in student management system
- Circle in a drawing software

As objects have attributes and behaviour so corresponding classes will also have data members and methods as shown below,

Ali	Corresponding class
Characteristics (attributes)	/*c++ code for class Person, we can create any object like Ali from it*/
Name Age	class Person { private: /* attributes are generally made private*/ char name[]; /*char array to store name*/ int age; /*int age to store age*/
Behavior (operations)	public: /* methods are generally made public*/
Walks Eats	Person(); /*constructor used to initialize data members*/ void walks(); /* method walk */ void eats(); /*method eats*/ }

a. object **b. class code**

07.3. Abstraction

We only include those details in the system that are required for making a functional system so we will leave out irrelevant attributes and behaviour from our objects.
Take the example of student,

Student

- Name
- Address
- Sibling
- Father Business

Relevant to our problem

Not relevant to our problem

07.4. Defining a New User Defined Type

There are two ways to create user defined types for objects in c++ these are,

Structure Definition:

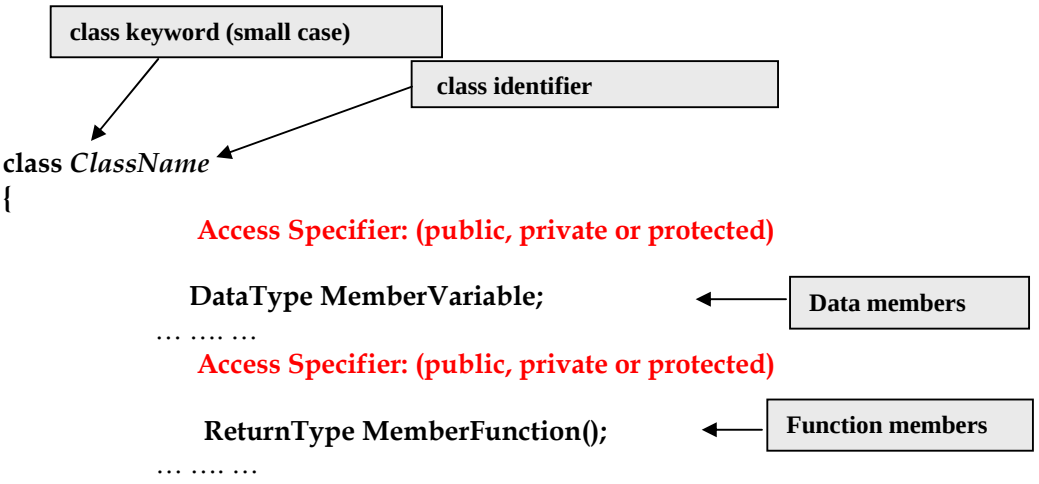
Partially we can use Structures to define an object

```
Struct{

};
```


© Virtual University of Pakistan

In c we can not define functions in a structure however in c++ we can add functions in both structure and classes.

Class Definition:



Example

```
class Student
{
    private:
        int rollNo;
        char *name;
        float CGPA;
        char *address;

    public:
        void setName(char *newName);
        void setRollNo(int newRollNo);

};
```

Member Variables

Member Functions

Why Member Functions:
They model the behaviors of an object,

Objects can make their data invisible (in accordance with the principle of data hiding). Setters and getters functions are provided by class to access the its members it also minimizes the changes to move the objects in inconsistent state as we can write checks in our setter functions for example we can check that whether the user has entered correct age value and has not entered negative value for age.
Object remains in consistent state

Example:

We can check that the entered roll number by user is positive or negative,

```
Student aStudent;
aStudent.rollNo = 514;
aStudent.rollNo = -514;          //Error
```

07.5. Object and Class:

Object is an instantiation of a user defined type or class. Once we have defined a class we can create as many objects for that class as we require.

Declaring class variables

Variables of classes (objects) are declared just like variables of structures and built-in data types as follows,

TypeName VariableName;

int var; // declaring built in int data type variable
Student aStudent; // declaring user defined class Student object

07.6. Accessing members

Members of an object can be accessed using,

a. dot operator (.) to access via the variable name
   Student aStudent; // declaring Student object
   aStudent. rollNo = 5;

b. arrow operator (->) to access via a pointer to an object
   Student * aStudent = new Student();
   // declaring and initializing Student pointer
   aStudent->rollNo = 5;

Note: it is against the principle of OOP to access the data members directly using object of class as we have done above. This code is given for example only we should write assessor functions (setters and getters) wherever we want to access the members of the class.

Member functions are accessed in the similar way using dot or arrow operator.

Example

```
class Student{
  int rollNo;
  void setRollNo(int aNo);
};

  Student aStudent;
  aStudent.setRollNo(5);
```


© Virtual University of Pakistan

```
Student *ptr_student = new Student();
ptr_student->setRollNo(5);
```

07.7. Access specifiers

These are used to enforce access restrictions to members of a class, there are three access specifiers,

1. **'public'** is used to tell that member can be accessed whenever you have access to the object
2. **'private'** is used to tell that member can only be accessed from a member function
3. **'protected'** to be discussed when we cover inheritance

Example

```
class Student{
private:
        char * name;
        int rollNo;
public:
        void setName(char *);
        void setRollNo(int);
...
};
```

Can not be accessed outside the class

Can be accessed outside the class

Example Program

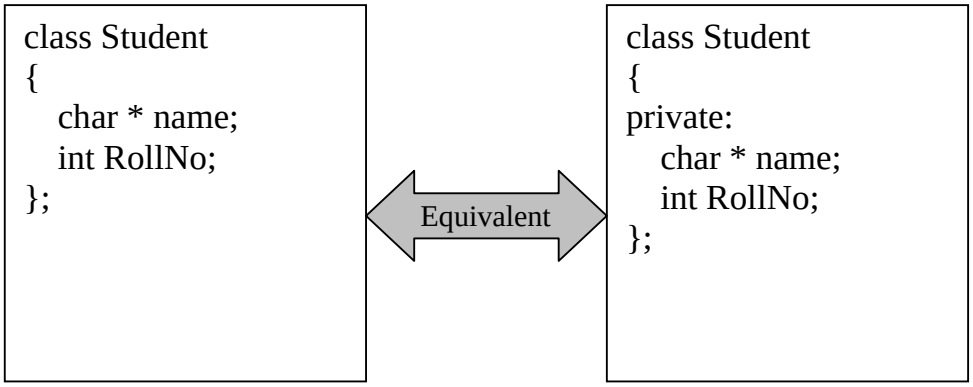
```
class Student{
char * name;
        int rollNo;
public:
        void setName(char *);
void setRollNo(int aNo);
};
void Student::setName(char * aName){
if (strlen(aName) > 0)
    {
        name = new char[strlen(aName)];
        strcpy(name,aName);
    }
}

void Student::setRollNo(int arollNo){
if(arollNo > 0)
rollNo = arollNo;
}

int main(){
        Student aStudent;
 aStudent.rollNo = 5;
```

```
/* Error: we can not access private member of the class. */
 aStudent.name  = "Ali";
/* Error: we can not access private member of the class */
        aStudent.setRollNo(1);
        aStudent.setName("Ali");
/* Correct way to access the data member using public setter functions */
}
```

Default access specifier

When no access specifier is mentioned then default access specifier is **private**.

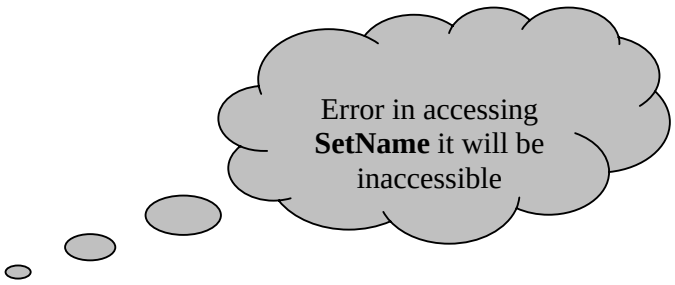
Example



Example

We should use keyword public before the methods of the class as given below of will not use public keyword they will also be treated as private ad will not be accessible outside the class as shown below,

```
class Student
{
        char * name;
        int RollNo;
        void SetName(char *);
};
Student aStudent;
aStudent.SetName(Ali);
```



Corrected code will be,

```
class Student
{
        char * name;
        int RollNo;
public:
        void setName(char *);
```


© Virtual University of Pakistan

```
};
Student aStudent;
aStudent.SetName("Ali");
```

Lecture No.08

08.1. Member Functions

- Member functions are the functions that operate on the data encapsulated in the class
- Public member functions are the interface to the class

08.2. Defining Member Functions

We can define member functions in two ways,

a. We can define member functions of the class inside the class definition when we define any class in our program.

OR

b. We declare member function inside the class definition and declare them outside the class.

In this case class definition is added in ClassName.h file and class implementation code is added in ClassName.cpp file.

Function definition inside the class:

General Syntax:

```
class ClassName {
        …
        public:
        ReturnType FunctionName() {
                …
        }
};
```

Example:

Define a class of student that has a roll number. This class should have a function that can be used to set the roll number

```
class Student{
        int rollNo;
public:
        void setRollNo(int aRollNo){
                rollNo = aRollNo;
        }
};
```

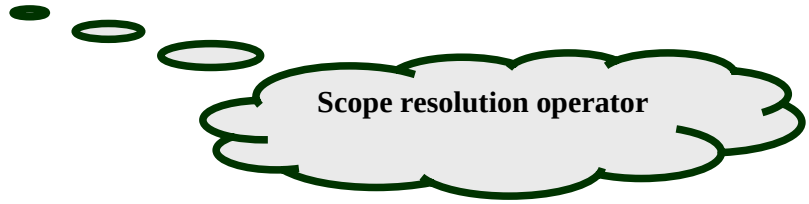
Function definition outside class

General Syntax:

```
class ClassName {
        …
```


© Virtual University of Pakistan

```
        public:
        ReturnType FunctionName();
};
ReturnType ClassName::FunctionName()
{
        …
}
```



Example

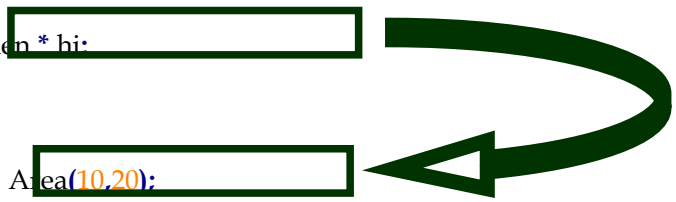
```
class Student{
        …
        int rollNo;
public:
        void setRollNo(int aRollNo);
};
void Student::setRollNo(int aRollNo){
        …
        rollNo = aRollNo;
}
```

08.3.Inline Functions

- Inline functions is a way used by compilers to improve efficiency of the program, when functions are declared inline normal process of function calling (using stack) is not followed instead function code is added by compiler at all points where these functions have been called. Basic concept behind inline functions is that they are functions in our code but in compiler generated files these functions code is added by compiler at all places where they were called in the code.
- Normally small size functions that need to be called many times during program execution are declared inline. Inline functions decrease the code execution time because program in their case doesn't involve function call overhead.
- Keyword 'inline' is used to request compiler to make a function inline.
- However using inline keyword with function doesn't guarantee that function will definitely in inlined, it depends on the compiler if it finds it can make function inline it does so otherwise it ignores the keyword inline and treat the function as normal function.

Example

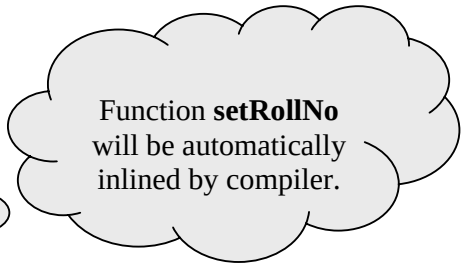
```
inline int Area(int len, int hi)
{
        return len * hi;
}
int main()
{
        cout << Area(10,20);
```



```
    return 0;
}
```

Inline Functions

The functions defined inside the class are by default inline (whether we mention keyword inline with them or not)
In case we define function outside the class then we must use the keyword 'inline' to make the function inline.
However compiler decides whether it will implement these functions code as inline or not.



Example
Inline function inside the class:

```
class Student{
        int rollNo;
public:
        void setRollNo(int aRollNo){
                …
                rollNo = aRollNo;
        }
};
```

Example
Inline function outside the class:

```
class Student{
        …
        public:
        inline void setRollNo(int aRollNo);
};
void Student::setRollNo(int aRollNo){
        …
        rollNo = aRollNo;
}
class Student{
        …
        public:
        void setRollNo(int aRollNo);
};
inline void Student::setRollNo(int     aRollNo){
        …
        rollNo = aRollNo;
}
class Student{
        …
        public:
```


© Virtual University of Pakistan

```
        inline void setRollNo(int aRollNo);
};
inline void Student::setRollNo(int
        aRollNo){
        …
        rollNo = aRollNo;
}
```

08.4. Constructor

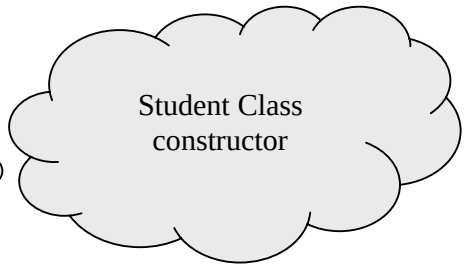
Constructor is used to **initialize** the objects of a class. Constructor is used to ensure that object is in well defined state at the time of creation.
The constructor of a class is automatically generated by compiler however we can write it by our self also.
Constructor is automatically called when the object is created. Constructors are not usually called explicitly by us.

08.5. Constructor Properties

- Constructor is a special function having same name as the class name
- Constructor does not have return type
- Constructors are commonly public members

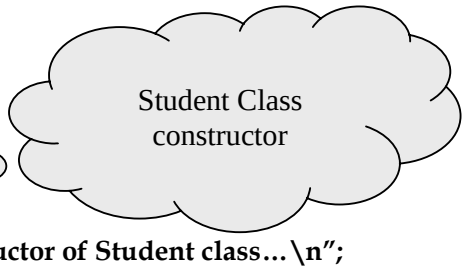
Example

```
class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;

        }
};

int main()
{
        Student aStudent;
        /*constructor is implicitly called at this point*/
}
```



We can assure that constructor is called automatically by adding cout statement in constructor as shown below,

```
#include <iostream>
using namespace std;

class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;
                cout<<"I am constructor of Student class…\n";
```



```
        }
};

int main()
{
        Student aStudent;
        /*constructor is implicitly called at this point*/
system("pause");
return 0;
}
```

08.6. Default Constructor

- Constructor without any parameter or with all parameters with default values is called default constructor
- If we do not define a default constructor the compiler will generate a default constructor
- Compiler generated default constructor is called implicit and user written default constructor is called explicit
- This compiler generated default constructor initialize the data members to their default values
- If we have given any constructor for a class whether it is
    - our own **explcit default constructor** (i.e parameterless or with parameters having default values)
      or
    - our own constructor with parameters

Then compiler will not create implicit default constructor[5].

Example

Consider the class student below it has no constructor so compiler will generate one for it,

```
class Student
{
        int rollNo;
        char *name;
        float GPA;
public:
        …               //no constructors
};
```

Code of Compiler generated implicit default constructor

```
{
```

[5] compiler generated default constructor is called implicit and user written default constructor is called explicit

© Virtual University of Pakistan

```
        rollNo = 0;
        GPA = 0.0;
        name = NULL;
}
```

08.7.Constructor Overloading

We can write constructors with parameters as well. These parameters are used to initialize the data members with user supplied data (passed as parameter). The example is shown below, here example Student class has four constructors their prototypes are,

1. **Student();** /* explicit default parameterless constructor */
2. **Student(char * aName);** /* constructor with one parameter* /
3. **Student(char * aName, int aRollNo);** /* constructor with two parameters */
4. **Student(int aRollNo, int aRollNo, float aGPA);** /* constructor with three parameters */

Example

```
class Student{
        int rollNo;
        char *name;
        float GPA;
public:
        Student();  /* explicit default constructor */
        Student(char * aName);  /* constructor with one parameter* /
        Student(char * aName, int aRollNo);  /* constructor with two parameters */
        Student(int aRollNo, int aRollNo, float aGPA);  /* constructor with three parameters */

};
Student::Student(){

        rollNo = 0;
        name = NULL; // to indicate it is pointing to nothing at this moment otherwise it can generate erroneous code.
        GPA = 0.0;

}
Student::Student(int aRollNo){

        if(aRollNo < 0){
                rollNo = 0;
        }
        else {
        rollNo = aRollNo;
        }
name = NULL;
```

```
}
Student::Student(int aRollNo,
                                    char * aName){

        if(aRollNo < 0){
                rollNo = 0;
        }
        else {
        rollNo = aRollNo;
        }

 if (strlen(aName) > 0)
                                {
        name = new char[strlen(aName)+1];
           strcpy(name,aName);
                                }
 else
       {
name = NULL;
}

}
```

We can create this Student class object using any one of these constructors as follows,

```
int main()
{
        Student student1;  // default constructor will be used
        Student student2("Name"); // one parameter constructor will be used
        Student student3("Name", 1);  // two parameter constructor will be used
        Student student4("Name",1,4.0); // three parameter constructor will be used
}
```

08.8. Constructor Overloading

We can use default parameter values to reduce the writing effort in that case we will have to write only one constructor and it will serve the purpose of all constructors as given below,

Example

```
Student::Student(       char * aName = NULL,  int aRollNo= 0,  float aGPA = 0.0) {
…

}
```

It is equivalent to all three constructors,

© Virtual University of Pakistan

Student();
Student(char * aName);
Student(char * aName, int aRollNo);
Student(char * Name, int aRollNo, float aGPA);

It will use default values if values are not passed as arguments while creating objects) it is described in code given below,

```
int main()
{
    Student student1;  /*char * aName = NULL,  int aRollNo= 0, float aGPA = 0.0*/

    Student student2( "Name" );  /*char * aName = Name, int aRollNo= 0, float aGPA = 0.0*/

    Student student3( "Name",  1 );  /*char * aName = Name, int aRollNo= 1, float aGPA = 0.0*/

    Student student4( "Name", 1 , 4.0);  /*char * aName = Name, int aRollNo= 1, float aGPA = 4.0*/


}
```

08.9.Copy Constructor
Copy constructors are used when:

- Initializing an object at the time of creation (we want to create an object with state of a pre existing object)
- When an object is passed by value to a function (As you know temporary copy of object is created on stack so we need copy constructor to create that temporary object with the state of actual object being passed).

Example

```
void func1(Student student){
…
}
```

```
int main(){

        Student studentA;
        Student studentB = studentA;
```

Copy constructor will be called to create temporary student object

```
        func1(studentA);

}
```

Copy constructor will be called as we are creating studentB in terms of studentA.

Copy Constructor (Syntax)

```
Student::Student( const Student &obj){

    /*copying values to newly created object*/
   rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;

}
```

As was the case with default constructor compiler also generates copy constructor by itself however we can override that copy constructor by writing our own copy constructor as shown in example below,

```
#include <iostream>
using namespace std;

class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;
                cout<<"I am default construct
        }

Student(const Student &obj){
         cout<<"I am copy constructor of Student class\n";
  rollNo = obj.rollNo;

}
};

int main()
{
        Student aStudent;
        /*default constructor is implicitly called at this point*/

        Student bStudent = aStudent;
        /*copy constructor is implicitly called at this point*/

system("pause");
```

Student class default constructor

Student class copy constructor

© Virtual University of Pakistan

```
return 0;
}
```

08.10. Shallow Copy

- When we initialize one object with another then the compiler copies state of one object to the other using copy constructor by assigning data member values of previous object to newly created object.

Shallow copy using default Copy Constructor (Syntax)

```
Student::Student(     const Student & obj ){

        rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;
}
```

- This kind of copying is called shallow copying

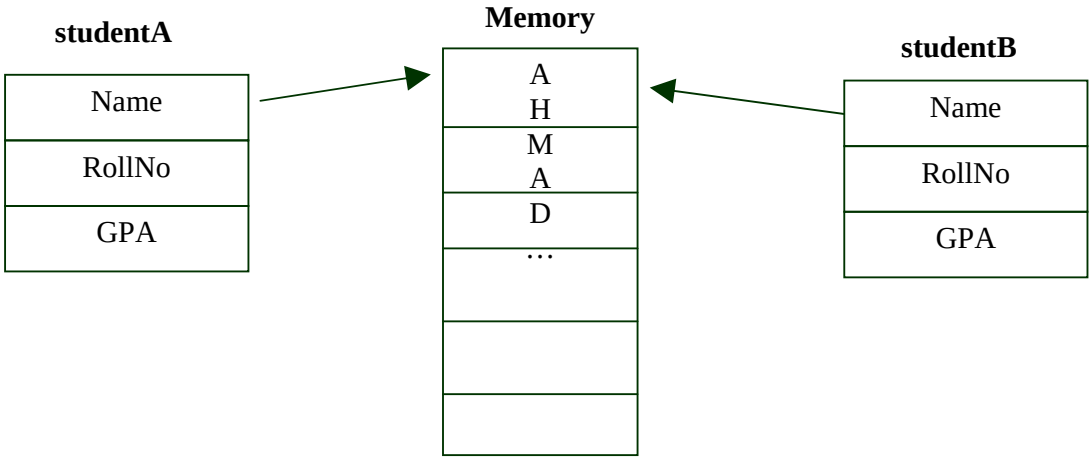
Example

Student studentA;
Student studentB = studentA; /*Shallow copy: compiler will use copy constructor to assign studentA values to newly created object studentB*/

Shallow copy works fine if our class doesn't include dynamic memory allocation but in case of dynamic memory allocation it leads to dangling pointer problem as explained below,

Problem is Shallow Copy

Student class data member name of char * type is added to store the name of student and it is using **dynamic memory** according to the length of name entered by user for student.



Object studentB is also pointing to memory allocated object studentA for Student class data member **name** of char * type, now there are two problems with this sort of copying,

Suppose we delete first object studentA for some reason then its destructor will also free memory allocated by it hence memory area containing name "AHMAD" will also be freed and will be given to some other application by operating system, but studentB member **name** is still pointing to that area so issue of *"Dangling Pointer"* will arose. [Pointer pointing to incorrect memory location]. If we will try to print the name of object studentB our program will terminate abnormally as it was pointing memory of some other applications.
Secondly if for some reason we change name of studentA the value of object studentB will also be changed as it pointing to same memory location.

We resolve these two issues using deep copy.

08.11. Deep Copy

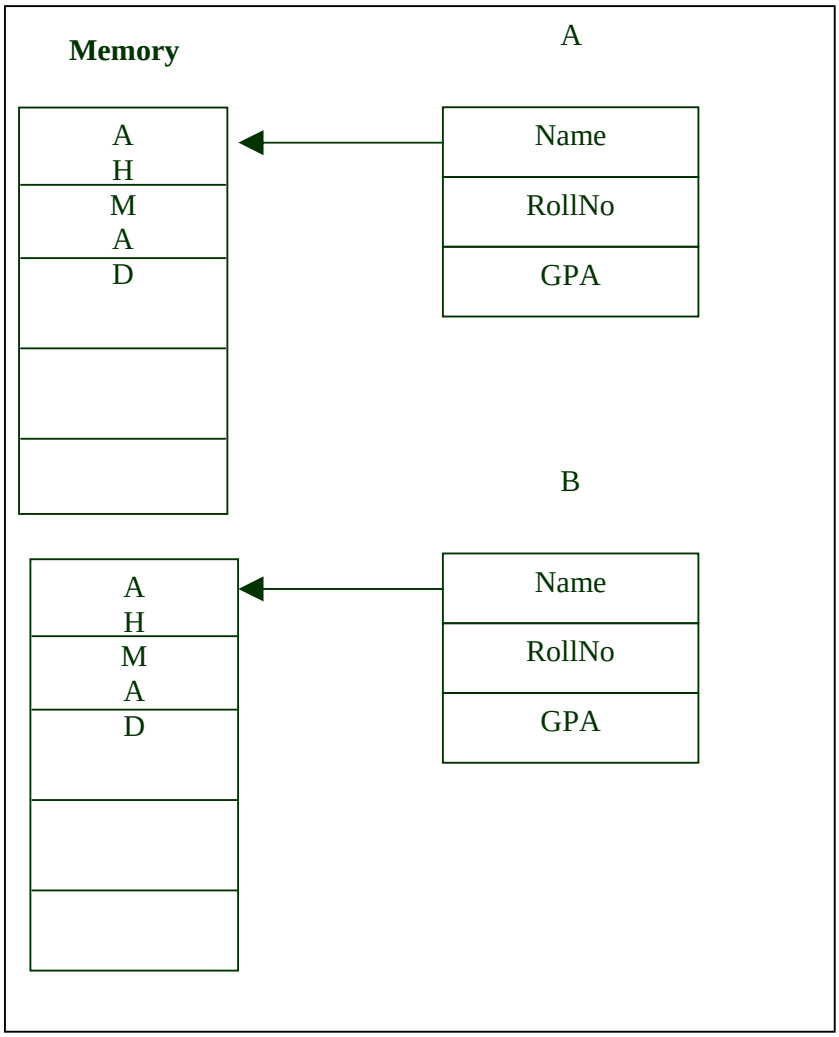
We write our own deep copy code in copy constructor so that when we create new object from an existing object using copy constructor we also allocate new dynamic memory for data members involving dynamic memory as shown below,

```
Student::Student( const Student & obj){

        int len = strlen(obj.name);
        name = new char[len+1];
        // assignming new dynamic memory to data member name of char * type for
newly created object*/
        strcpy(name, obj.name);
        …
        //copy rest of the data members in the same way
}
```


© Virtual University of Pakistan

Example

```
Student studentA;
Student studentB = studentA; // now copy constructor will perform deep copy (separate memory for both objects)
```



In case our class doesn't involve dynamic memory then default copy constructor that performs shallow copy works fine.
In case our class has any data member involving dynamic memory we have to write our own code to do deep copy.

Lecture No.09

Review

Copy Constructor

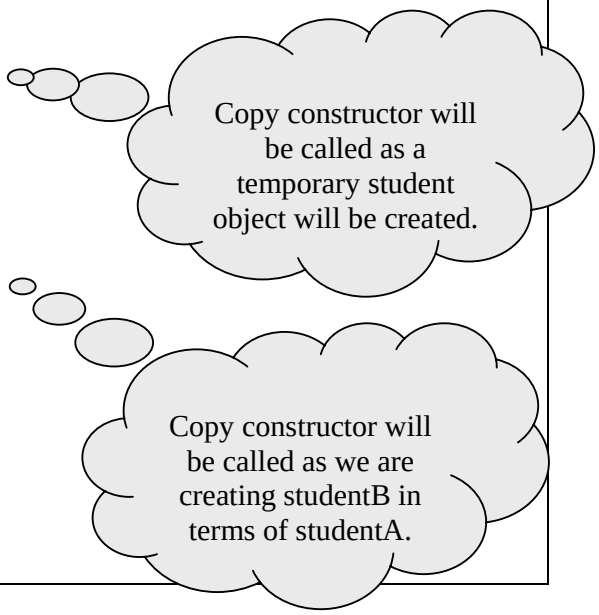
Copy constructors are used when:

Initializing an object at the time of creation (we want to create an object with state of a pre existing object)

When an object is passed by value to a function (As you know temporary copy of object is created on stack so we need copy constructor to create that temporary object with the state of actual object being passed).

Example

```
void func1(Student student){
…
}

int main(){

        Student studentA;
        Student studentB = studentA;
        func1(studentA);
        return 0;
}
```



Copy Constructor (Syntax)

```
Student::Student( const Student &obj){

   /*copying values to newly created object*/
  rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;
}
```

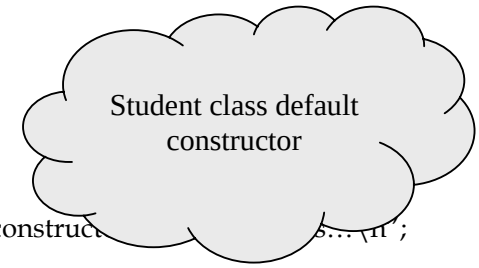
As was the case with default constructor compiler also generates copy constructor by itself however we can override that copy constructor by writing our own copy constructor as shown in example below,

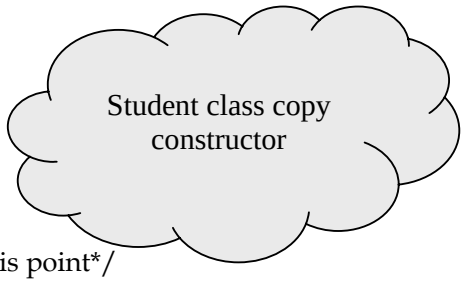
As was the case with default constructor compiler also generates copy constructor by itself however we can override that copy constructor by writing our own copy constructor as shown in example below,

```
#include <iostream>
```

 © Virtual University of Pakistan

```
using namespace std;

class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;
                cout<<"I am default construct…\n";
        }

Student(const Student &obj){
         cout<<"I am copy constructor of Student class\n";
  rollNo = obj.rollNo;

}
};

int main()
{
        Student aStudent;
        /*default constructor is implicitly called at this point*/

        Student bStudent = aStudent;
        /*copy constructor is implicitly called at this point*/

system("pause");
return 0;
}
```





09.1.Shallow Copy

When we initialize one object with another then the compiler copies state of one object to the other using copy constructor by assigning data member values of previous object to newly created object. This kind of copying is called shallow copying.

Shallow copy using default Copy Constructor (Syntax)

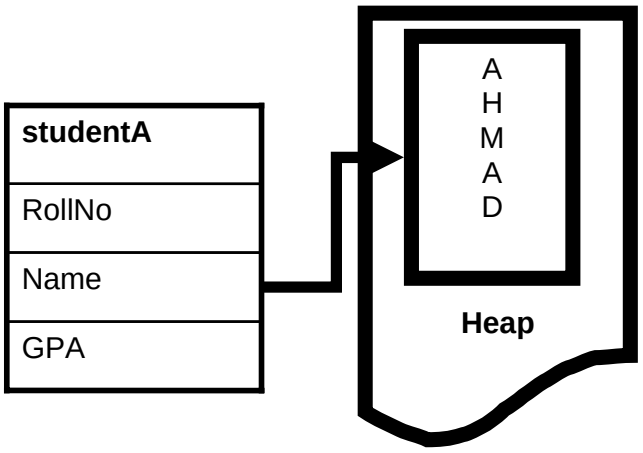
```
Student::Student(      const Student & obj ){

        rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;
}
```

This kind of copying is called shallow copying

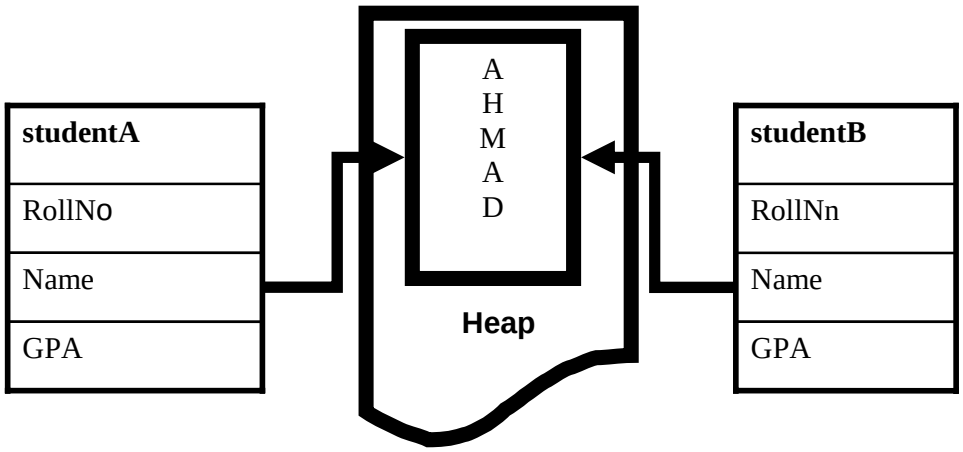
Example

Student studentA;
Student studentB = studentA; /*Shallow copy: compiler will use copy constructor to assign studentA values to newly created object studentB*/

Example
Student studentA("Ahmad");



Student studentB = studentA;




```
#include <iostream>
using namespace std;

class Student{
    char * name;
        int rollNo;
public:
        Student(char * aName, int arollNo){
      name = new char[strlen(aName)+1];
      strcpy(name,aName);
```


© Virtual University of Pakistan

```
                rollNo = arollNo;

        }
  Student(const Student &obj){
    name = obj.name;
    rollNo = obj.rollNo;


  }
  void showName(){
     cout<<name<<endl;

     }
  ~Student(){

        delete []name;

        }
};

int main()
{
        Student studentA("AHMAD",1);
        Student studentB = studentA;
        /*copy constructor is implicitly called at this point*/

        studentA.showName();
  studentB.showName();

  system("pause");
  return 0;
}
```

Shallow copy works fine if our class doesn't include dynamic memory allocation but in case of dynamic memory allocation it leads to dangling pointer problem as explained below.

Problem is Shallow Copy

Student class data member name of char * type is added to store the name of student and it is using **dynamic memory** according to the length of name entered by user for student.

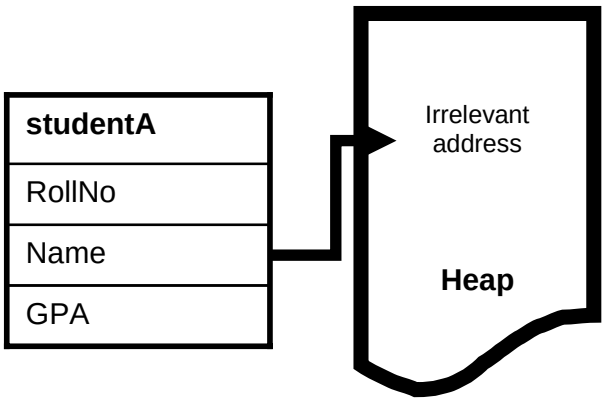
Student class data member **name** (*char **) of object studentB is also pointing to memory allocated for datamember **name** of object studentA, due to this there may be two kinds of problems.

Suppose we delete first object studentA for some reason then its destructor will also free memory allocated by it hence memory area containing name "AHMAD" will also be freed and will be given to some other application by operating system, but

studentB member **name** is still pointing to that area so issue of *"Dangling Pointer"* [Pointer pointing to incorrect memory location] will arose.
Same will happen if object studentB is deleted then studentA object data member name will become dangling pointer. This has been explained below,
Let us change code in main to make our second object studentB in a new local scope as shown below,

```
int main(){
Student studentA("Ahmad",1);

{
   Student studentB = studentA;
}
return 0;
}
```

Now if we will try to print the name of object studentA our program will not show any output as name is pointing to some irrelevant memory address,



Complete program code is given below,

```
#include <iostream>
using namespace std;

class Student{
    char * name;
        int rollNo;
public:
        Student(char * aName, int arollNo){
      name = new char[strlen(aName)+1];
      strcpy(name,aName);
                rollNo = arollNo;

        }
  Student(const Student &obj){
```


© Virtual University of Pakistan

```
    name = obj.name;
    rollNo = obj.rollNo;

  }
  void showName(){
     cout<<name<<endl;

     }
  ~Student(){
         delete []name;
         }
};

int main()
{
        Student studentA("AHMAD",1);

  {
       Student studentB = studentA;
       /*copy constructor is implicitly called at this point*/
  }

        studentA.showName();

  system("pause");
  return 0;
}
```

Secondly if for some reason we change name of object studentA the value of object studentB will also be changed as it pointing to same memory location.

```
#include <iostream>
using namespace std;

class Student{
    char * name;
        int rollNo;
public:
        Student(char * aName, int arollNo){
     name = new char[strlen(aName)+1];
     strcpy(name,aName);
                rollNo = arollNo;

        }
  Student(const Student &obj){
    name = obj.name;
    rollNo = obj.rollNo;
```

```
   }

   void setName(char * aName){
      strcpy(name,aName);

      }

   void showName(){
      cout<<name<<endl;

      }
   ~Student(){

            delete []name;

            }
};

int main()
{
         Student studentA("AHMAD",1);
         Student studentB = studentA;
         /*copy constructor is implicitly called at this point*/

         studentA.showName();
         studentB.showName();


         studentA.setName("MOEEN");
         studentA.showName();
         studentB.showName();

   system("pause");
   return 0;
}
```

We resolve these two issues using deep copy.

09.2.Deep Copy

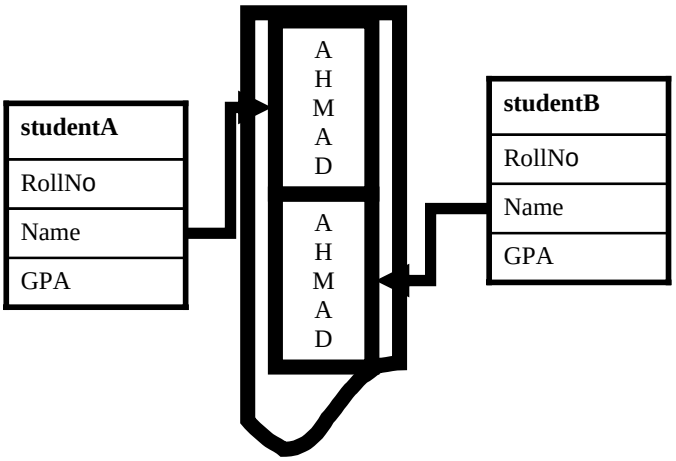
We write deep copy code in copy constructor so that when we create new object from an existing object using copy constructor we also allocate new dynamic memory for data members involving dynamic memory as shown below,

Student::Student(const Student & obj){

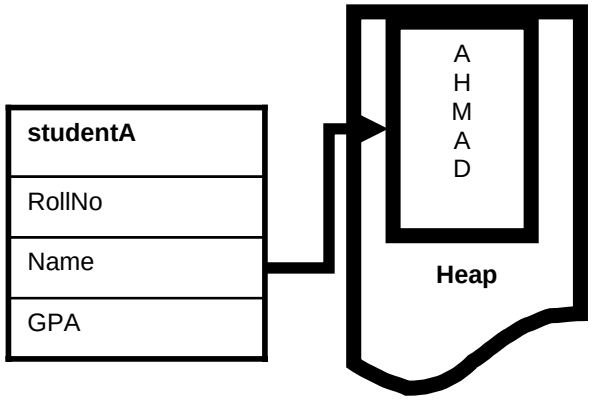
```
        int len = strlen(obj.name);
        name = new char[len+1]; // assignming new
/*dynamic memory to data member name of char * type for newly created object.*/
        strcpy(name, obj.name);
        …
        //copy rest of the data members in the same way
}
```

Now we see what happens when we created objects in main as shown below,

```
int main(){
Student studentA("Ahmad",1);

{
   Student studentB = studentA;
}
}
```



Now when we will execute code with object studentB in local scope our code still works fine and shows name for object studentA as now deletion of object studentB has no effect on object studentA as shown below,



Example

```
#include <iostream>
using namespace std;

class Student{
   char * name;
         int rollNo;
public:
         Student(char * aName, int arollNo){
      name = new char[strlen(aName)+1];
      strcpy(name,aName);
                  rollNo = arollNo;

         }
   Student(const Student &obj){

     name = new char[strlen(obj.name)+1];
     strcpy(name,obj.name);
     rollNo = obj.rollNo;


   }

  void showName(){
       cout<<name<<endl;

      }
   ~Student(){

          delete []name;

          }
};

int main()
{
```

```
        Student studentA("AHMAD",1);
   {
       Student studentB = studentA;
       /*copy constructor is implicitly called at this point*/
   }
        studentA.showName();

   system("pause");
   return 0;
}
```

09.3. Important points about copy constructor:

1. In case our class doesn't involve dynamic memory then default copy constructor that performs shallow copy works fine.
2. In case our class has any data member involving dynamic memory we have to write our own code in copy constructor to perform deep copy.
3. Copy constructor is normally used to perform deep copy
4. If we do not make a copy constructor then the compiler performs shallow copy
5. Shallow copy performs bitwise copy.

09.4. Destructor

1. Destructor is used to free memory that is allocated through dynamic allocation. We have to free memory allocated using new operator by over self in destructor otherwise it remain occupied even after our program ends.
2. Destructor is used to perform house keeping operations.
3. Destructor is a function with the same name as that of class, but preceded with a tilde '~'

Example

```
class Student
{
        …
public:
        ~Student(){
                if(name){
                        delete []name;
                }
        }
};
```

Overloading
Destructors cannot be overloaded.

Sequence of Calls
Constructors and destructors are called automatically
Constructors are called in the sequence in which object is declared
Destructors are called in reverse order

Example

```
Student::Student(char * aName){
        …
                cout << aName << "Constructor\n";
        }
        Student::~Student(){
                cout << name << "Destructor\n";
        }
};
```

Example

 © Virtual University of Pakistan

```
int main()
{
        Student studentB("Ali");
        Student studentA("Ahmad");
        return 0;
}
```

Example

Output:
Ali Constructor
Ahmad Constructor
Ahmad Destructor
Ali Destructor

09.5. Accessor Functions

In accordance to principle of information hiding data members of a class are declared as private so that outside world can not access the private data of the object only an interface is provided to outside world in the form of functions.
Accessor functions are also used to access private data of the object, we provide accessor functions to get and set private data members of the class.
We also add error checking code in accessor functions to reduce errors so that object doesn't move in illegal state.

Example – Accessing Data Member

Example - Setter

```
class Student{
        …
        int rollNo;
public:
        void setRollNo(int aRollNo){
                rollNo = aRollNo;
        }
};
```

Avoiding Error

```
void Student::setRollNo(int aRollNo){
        if(aRollNo < 0){
          rollNo = 0;
        }
        else
        {
          rollNo = aRollNo;
        }
}
```

Example - Getter

```
class Student{
```

```
        …
        int rollNo;
public:
        int getRollNo(){
                return rollNo;
        }
};
```

Good Practice:
Never return a handle to a data member from getter function because you are never sure that function accessing the reference will not change the value of the variable.

09.6.this Pointer

Consider the code of a general class given below,

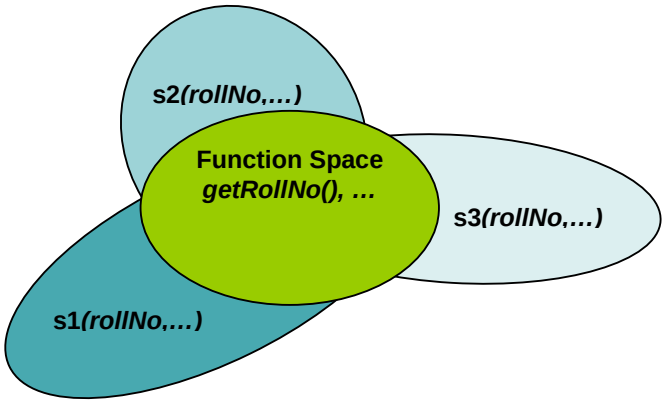
```
class Student{
        int rollNo;
        char *name;
        float GPA;
public:
        int getRollNo();
        void setRollNo(int aRollNo);
…
};
```

The compiler reserves space for the functions defined in the class

- Space for data is not allocated (since no object is yet created)

this Pointer
Student s1, s2, s3;

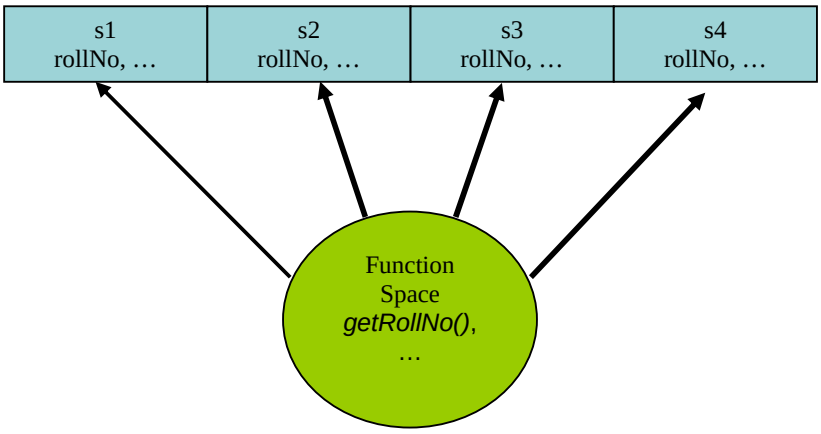
© Virtual University of Pakistan



this Pointer

- Function space is common for every variable
- Whenever a new object is created:
  - Memory is reserved for variables only
  - Previously defined functions are used over and over again

Memory layout for objects created:



this Pointer

- Address of each object is passed to the calling function.
- This address is de-referenced by the functions and hence they act on correct objects

s1 rollNo, …	s2 rollNo, …	s3 rollNo, …	s4 rollNo, …
address	*address*	*address*	*address*

Passing this Pointer

- Whenever a function is called the this pointer is passed as a parameter to that function.
- Function with n parameters is actually called with n+1 parameters

Example

void Student::setName(char *)

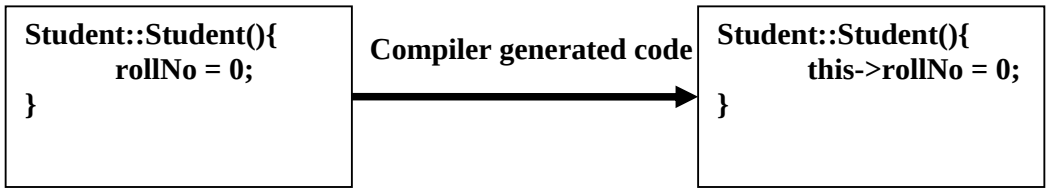
is internally represented as,

void Student::setName(char *, const Student *)

Declaration of this

DataType * const this;

Compiler Generated Code



© Virtual University of Pakistan

Lecture No.10

10.1.Uses of this Pointer

- There are situations where designer wants to return reference to current object from a function
- In such cases reference is taken from this pointer like (*this)

Example

```
Student Student::setRollNo(int aNo)
{
        …
        return *this;
}
Student Student::setName(char *aName)
{
        …
        return *this;
}
```

Usage:

```
int main()
{
        Student aStudent;
        Student bStudent;

        bStudent = aStudent.setName("Ahmad");
        …
        bStudent = aStudent.setName("Ali").setRollNo(2);

        return 0;
}
```

10.2.Separation of interface and implementation

- Public member functions exposed by a class are called interface.
- Separation of implementation from the interface is good software engineering.

Benefits of separating interface and implementation:

Consider the example of following complex no. class, this complex no. class two forms of implementations one is new and one is old implementation you can observe that if you have separated interface and implementation then we can easily change implementation without changing interface,

10.3.Complex Number

- There are two representations of complex number
  - Euler form

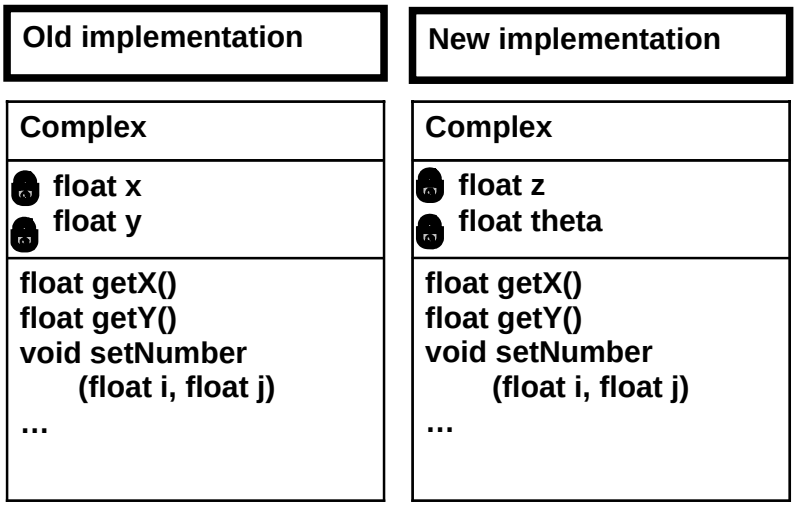
$z = x + i\,y$

  - Phasor form

$z = |z| (\cos\theta + i \sin\theta)$

z is known as the complex modulus and θ is known as the complex argument or phase

Example



Uml notation to show private data members

Example

```
class Complex{   //old
        float x;
        float y;
public:
        void setNumber(float i, float j){
                x = i;
                y = j;
        }
        …
};
```

Example

```
class Complex{ //new
        float z;
        float theta;
public:
        void setNumber(float i, float j){
```

```
            theta = arctan(j/i);
            …
        }
        …
};
```

Advantages

1. User is only concerned about ways of accessing data (interface)
2. User has no concern about the internal representation and implementation of the class

Separation of interface and implementation

In c++ generally we can relate the concept of interface of a class to its header (.h) file and and implementation of a class to its (.cpp) file. However it is not complete separation of interface and implementation.

- Usually functions are defined in implementation file (.cpp) while the class definition is given in header file (.h)
- Some authors also consider this as separation of interface and implementation

Student.h

```
class Student{
        int rollNo;
public:
        void setRollNo(int aRollNo);
        int getRollNo();
        …
};
```

Student.cpp

```
#include "student.h"

void Student::setRollNo(int aNo){
        …
}
int Student::getRollNo(){
…
}
```

We only need to include header (.h) file in main.cpp to use the Student class as shown below,

```
Main.cpp  (main file to run the program)
#include "student.h"

int main(){
        Student aStudent;
         return 0;
}
```

10.4.const Member Functions

Some functions in our programs are general purpose functions to show or access data, they are supposed to do read only tasks only however there are chances that they can change the state of data members of the class while accessing the data members due to programming mistake, c++ provides the solution of this problem using constant member functions.

We make those functions as constant who need only read only access (for example such functions that will only display data or will return the value of data members). When we make them constant compiler generates an error if these functions try to change the value of data members of the class.

const Member Functions

Keyword const is placed at the end of the parameter list to make any function as constant.

Declaration:
Inside class

```
class ClassName{
        ReturnVal Function() const;
};
```

Definition:
Outside class

```
ReturnVal ClassName::Function() const{
        …
}
```

Example

```
class Student{
public:
        int getRollNo() const {
                return rollNo;
        }
};
```

const Functions

- Constant member functions cannot modify the state of any object

 © Virtual University of Pakistan

- They are just "read-only"
- Errors due to typing are also caught at compile time

Example

Consider the function given below that is being used to check if roll no is equal to entered value if in this function we replace comparison statemtn == with assignment = statement it will compile correctly but whole code logic will change and we will get un expected result,

```
bool Student::isRollNo(int aNo){
        if(rollNo = = aNo){
                return true;
        }
        return false;
}
```

Example

```
bool Student::isRollNo(int aNo){
        /*undetected typing mistake*/
        if(rollNo = aNo){
                return true;
        }
        return false;
}
```

But if we have implemented as constant then compiler will catch this error and will produce compile time error as shown below,

Example

```
bool Student::isRollNo
                                        (int aNo)const{
        /*compiler error*/
        if(rollNo = aNo){
                return true;
        }
        return false;
}
```

const Functions

Constructors and Destructors cannot be **const because** Constructors and destructors are used to modify the object to a well defined state or to clean the memory occupied by the object.

Example

```
class Time{
public:
        Time() const {}  //error…
        ~Time() const {} //error…
};
```

const Function

- Constant member function cannot change data member
- We cannot call non constant functions in constant functions because non constant member functions may have code for changing state of the object that is not allowed in the constant functions.

Example

```
class Student{
        char * name;
public:
        char *getName();
        void setName(char * aName);
        int ConstFunc() const{
                name = getName();    //error
                setName("Ahmad");//error
        }
};
```

10.5.this Pointer and const Member Function

As we know that when a class function is called an implicit this pointer is passed to tell the function about the object it has to operate same is true for constant function with the difference that it will bbe passed as constant pointer to const data in case of constant member functions so that this pointer can not be used now to change the value of data members of the object,

```
const Student *const this;
// In case of constant member functions
```

instead of

```
Student * const this;
// In case of ordinary member functions
```

 © Virtual University of Pakistan

Lecture No.11

11.1.Usage example of Constant member functions

Problem:

Suppose we have requirement to change the class Student such that a student is given a roll number when the object is created and cannot be changed afterwards our existing class is given below,

Student Class

```
class Student{
…
        int rollNo;
public:
        Student(int aNo);
        int getRollNo();
        …
};
```

Solution of this problem:

We can do this by making rollNo constant so that cannot be changed once it is defined as shown below,

Modified Student Class

```
class Student{
…
        const int rollNo;
public:
        Student(int aNo);
        int getRollNo();
        …
};
```

Now there is only one issue of initializing this roll no with initial value but the problem is that we cannot set the value of roll no in constructor, as when code in constructor is executed the data member roll no has already been created and when we try to assign value to it in constructor compiler generates error,

Example

```
Student::Student(int aRollNo)
{
        rollNo = aRollNo;

 /*error: cannot modify a constant data member assignment statement not
```

```
initialization*/
}
```

Second solution is to write separate function but the problem remains same that we can't assign value to constant data member,

```
void Student::SetRollNo(int i)
{
        rollNo = i;
        /*error: cannot modify a constant data member again assignment statement not initialization */
}
```

We also know that we can only declare data members in structure or class but we cannot initialize them at the time of declaration in structure or class _____ because before executing constructor code, the class const member roll no has not got life it will get life along with other class members when constructor will be invoked so we can not assign any value to this constant member while declaring it.[6]

Solution:

so what is the solution of this problem as we can not initialize constant members while declaring them and we can not initialize them in constructor also because as soon as they go life they become constant to solve this problem C++ gives us new mechanism (syntax) for initialization of constant data members of the structure or class to resolve above mentioned issues,

11.2. Difference between Initialization and Assignment:

Initialization is assigning value along with creation of variable.

```
int i = 2;
```

Assignment is assigning value after creation.

```
int i;
i  = 7;
```

11.3. Member Initializer List

Member initialization list is used where we cannot modify the state of data members in the member functions of the class including constructor,

- A member initializer list is a mechanism to initialize data members
- It is given after closing parenthesis of parameter list of constructor

[6] In c++ **static const data members** can be initialized in class or structure as well.

 © Virtual University of Pakistan

- In case of more than one member use comma separated list

Example

```
class Student{
        const int rollNo;
        char *name;
        float GPA;
public:
        Student(int aRollNo) : rollNo(aRollNo), name(Null), GPA(0.0){ // initialization
                …
        }
…
};
```

Order of Initialization

- Data member are initialized in order they are declared in the class
- Order in member initializer list is not significant at all

Example

```
class ABC{
        int x;
        int y;
        int z;
public:
        ABC();
};
ABC::ABC():y(10),x(y),z(y)
{
        …
}
/*      x = Junk value
                y = 10
                z = 10 */
```

11.4.const Objects

- Objects can be declared constant with the use of const keyword
- Constant objects cannot change their state

Example

```
int main()
```

```
{
        const Student aStudent;
        return 0;
}
```

Example

```
#include <cstdlib>
#include <iostream>

using namespace std;


class Student{
        int rollNo;
public:

   Student( ) {

        }

        int getRollNo(){
                return rollNo;
        }
};


int main(){

const Student aStudent;
        int a = aStudent.getRollNo();

        //error

  system("PAUSE");
   return EXIT_SUCCESS;
}
```

```
#include <cstdlib>
#include <iostream>

using namespace std;


class Student{
        int rollNo;
public:

   Student(int aRollNo) :
rollNo(aRollNo){

        }

        int getRollNo(){
                return rollNo;
        }
};

int main(){
        const Student aStudent(5);
        int a = aStudent.getRollNo();
        //error

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

const Objects

const objects can access only const member functions so chances of change of state of const objects once they are created are eliminated.
We make getRollNo function constant so that we can access it using constant objects,

Example

```
class Student{
```


© Virtual University of Pakistan

```
…
        int rollNo;
public:
…
        int getRollNo()const{
                return rollNo;
        }
};
int main(){
        const Student aStudent;
        int a = aStudent.getRollNo();
         return 0;
}
```

Constant member functions

Make all functions that don't change the state of the object constant
This will enable constant objects to access more member functions

11.5.Static Variables

Static variables of a class are such variables which are independent of class objects.

Lifetime of static variable is throughout the program life, if static variables are not explicitly initialized then they are initialized to 0 of appropriate type.

Example

Static variable is initialized once only throughout the program, independent of how many times the function initializing it is called,

```
void func1(int i){
        static int staticInt = i;
//initialization statement will be executed once
//only as static variables are initialized once
        cout << staticInt << endl;
}
int main(){
        func1(1);
        func1(2);
        return 0;
}
```

Output:
1
1

```
void func1(int i){

 static int staticInt;
 staticInt = i;
//assignment statement will be executed with each function call
 cout << staticInt << endl;
```

```
}
int main(){
        func1(1);
        func1(2);
        return 0;
}
```

Output:
1
2

Static Data Member

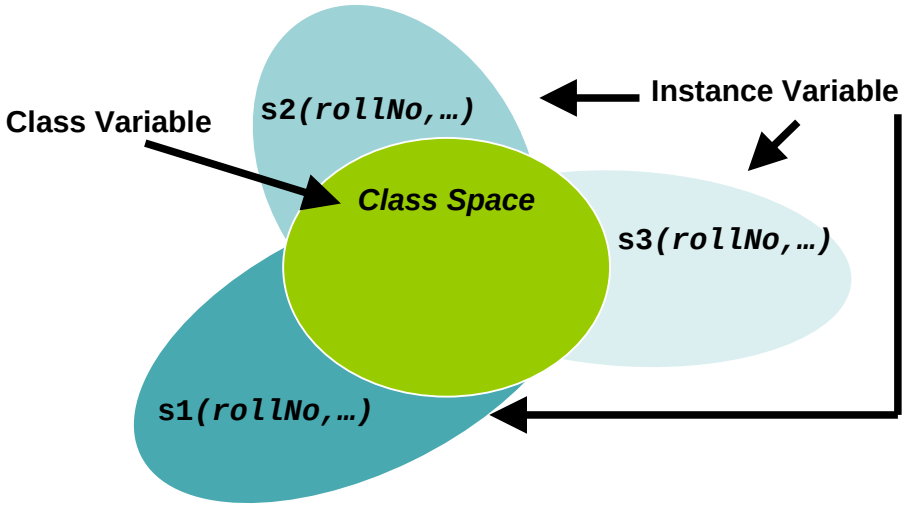
Definition
"A variable that is part of a class, yet is not part of any object of that class, is called static data member"

Static Data Member

They are shared by all instances (objects) of the class
They do not belong to any particular instance of a class

Class vs. Instance Variable

Suppose we created three objects of student class as shown below,
Student s1, s2, s3;



Static Data Member (Syntax)

Keyword static is used to make a data member static

```
class ClassName{
…
static DataType VariableName;
};
```

Defining Static Data Member

© Virtual University of Pakistan

Static data member is declared inside the class
But they are defined outside the class

Defining Static Data Member

```
class ClassName{
…
static DataType VariableName;
};

DataType ClassName::VariableName;
```

Initializing Static Data Member

Static data members should be initialized once at file scope
They are initialized at the time of definition

Example

```
class Student{
private:
static int noOfStudents;
public:
  …
};
int Student::noOfStudents = 0;
/*private static member cannot be accessed outside the class except for initialization*/
```

Initializing Static Data Member

If static data members are not explicitly initialized at the time of definition then they are initialized to 0

Example

```
int Student::noOfStudents;
```

is equivalent to

```
int Student::noOfStudents=0;
```

Lecture No.12

Review:
Static Data Member
Definition
"A variable that is part of a class, yet is not part of an object of that class, is called static data member"

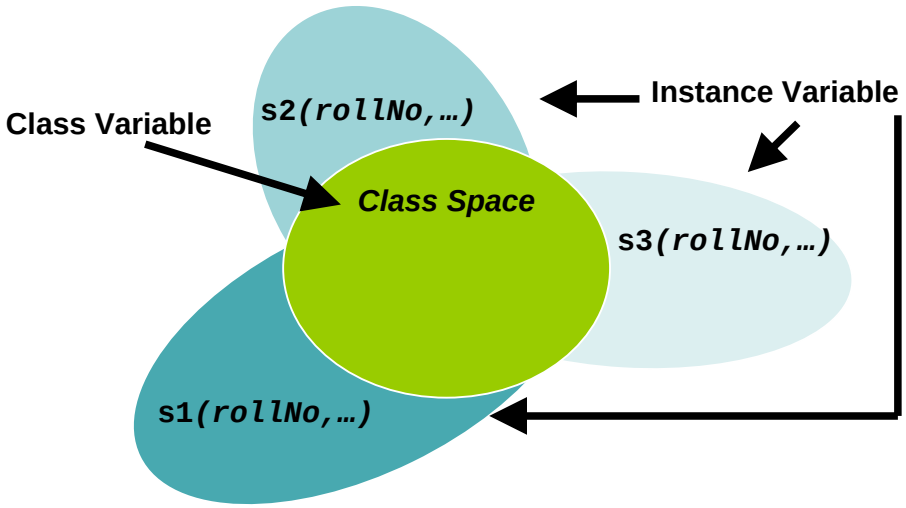
Static Data members are shared by all instances of the class and they do not belong to any particular instance of a class.

Class vs. Instance Variable

Memory for static variables in allocated in class space whereas for instance variables it is separate for each object as shown below, if we have class Student as given below,

```
class Student{
private:
static int noOfStudents;
public:
  …
};
```

When we will create objects of Student as s1, s2, s3 then memory will be allocated as given below,



Static Data Member (Syntax)
Keyword static is used to make a data member static,

```
class ClassName{
…
static DataType VariableName;
};
```


© Virtual University of Pakistan

Defining Static Data Member (allocating memory for them)

Static data member is declared inside the class
But they are defined outside the class,

```
class ClassName{
…
static DataType VariableName;
};

DataType ClassName::VariableName;
```

Initializing Static Data Member(assigning them some initial value)

Static data members should be initialized once at file scope
They are initialized at the time of definition,

Example

```
class Student{
private:
static int noOfStudents;
public:
  …
};
int Student::noOfStudents = 0;
/*private static member cannot be accessed outside the class except for initialization*/
```

Initializing Static Data Member

If static data members are not explicitly initialized at the time of definition then they are initialized to 0

Example

```
int Student::noOfStudents;
```

is equivalent to

```
int Student::noOfStudents=0;
```

12.1. Accessing Static Data Member

To access a static data member there are two ways

- Access like a normal data member (using dot operator '**.**')
- Access using a scope resolution operator '**::**'

Example

```
class Student{
public:
```

```
static int noOfStudents;
};

int Student::noOfStudents;

int main()
{
Student aStudent;
aStudent.noOfStudents = 1;
Student::noOfStudents = 1;
return 0;
}
```

12.2. Life of Static Data Member

- They are created even when there is no object of a class
- They remain in memory even when all Objects of a class are destroyed

Example

```
class Student{
public:
static int noOfStudents;
};
int Student::noOfStudents;
int main(){
Student::noOfStudents = 1;
}
```

Example

```
class Student{
public:
static int noOfStudents;
};
int Student::noOfStudents;
int main(){
{
        Student aStudent;
        aStudent.noOfStudents = 1;
}
Student::noOfStudents = 1;
return 0;
}
```

Uses

They can be used to store information that is required by all objects, like global variables

Example

Modify the class Student such that one can know the number of student created in a system

 © Virtual University of Pakistan

```
class Student{
…
public:
        static int noOfStudents;
        Student();
        ~Student();
…
};
int Student::noOfStudents = 0;
Student::Student(){
         noOfStudents++;
}
Student::~Student(){
         noOfStudents--;
}

int Student::noOfStudents = 0;
int main(){
        cout <<Student::noOfStudents <<endl;
        Student studentA;
        cout <<Student::noOfStudents <<endl;
        Student studentB;
        cout <<Student::noOfStudents <<endl;
        return 0;
}
```

Output:
0
1
2

Problem
noOfStudents is accessible outside the class
Bad design as the local data member is kept public

The solution is that we write static member function to access static members,

12.3.Static Member Function
Definition:
"The function that needs access to the members of a class, yet does not need to be invoked by a particular object, is called static member function"

- They are used to access static data members
- Access mechanism for static member functions is same as that of static data members
- They cannot access any non-static members

Example

```
class Student{
        static int noOfStudents;
        int rollNo;
public:
        static int getTotalStudent(){
                return noOfStudents;
        }
};
int main(){
        int i = Student::getTotalStudents();
        return 0;
}
```

Accessing non static data members

```
int Student::getTotalStudents(){
        return rollNo;
}
int main(){
        int i = Student::getTotalStudents();
        /*Error: There is no instance of Student, rollNo cannot be accessed*/
        return 0;
}
```

12.4.this Pointer and static member functions

- *this* pointer is passed implicitly to member functions
- *this* pointer is not passed to static member functions
- Reason is static member functions cannot access non static data members

12.5.Global Variable vs. Static Members

- Alternative to static member is to use global variable
- Global variables are accessible to all entities of the program
- User of Global variables is against the principle of information hiding.

12.6.Array of Objects

- Array of objects can only be created if an object can be created without supplying an explicit initializer
- There must always be a default constructor if we want to create array of objects

Example

```
class Test{
public:
};
int main(){
        Test array[2]; // OK
```


© Virtual University of Pakistan

```
        return 0;
}
```

```
class Test{
public:
        Test();
};
int main(){
        Test array[2]; // OK
        return 0;
}
```

```
class Test{
public:
        Test(int i);
};
int main(){
        Test array[2]; // Error
        return 0;
}
```

```
class Test{
public:
        Test(int i);
};
int main(){
        Test array[2] = {Test(0),Test(0)};
        return 0;
}
```

```
class Test{
public:
        Test(int i);
};
int main(){
        Test a(1),b(2);
        Test array[2] = {a,b};
        return 0;
}
```

Lecture No.13

13.1.Pointer to Objects

- Pointer to objects are similar as pointer to built-in types
- They can also be used to dynamically allocate objects

Example

```
class Student{
…
public:
        Student();
        Student(char * aName);
        void setRollNo(int aNo);
};
```

Example

```
int main(){
        Student obj;
        Student *ptr;
        ptr = &obj;
        ptr->setRollNo(10);
        return 0;
}
```

Allocation with new Operator

- new operator can be used to create objects at runtime

Example

```
int main(){
        Student *ptr;
        ptr = new Student;
        ptr->setRollNo(10);
        return 0;
}
```

Example

```
int main(){
        Student *ptr;
        ptr = new Student("Ali");
        ptr->setRollNo(10);
        return 0;
}
```

Example

```
int main()
{
        Student *ptr = new Student[100];
        for(int i = 0; i < 100;i++)
        {
                ptr->setRollNo(10);
        }
        return 0;
}
```


© Virtual University of Pakistan

13.2.Breakup of new Operation

new operator is decomposed as follows

- Allocating space in memory
- Calling the appropriate constructor

13.3.Case Study

Design a class date through which user must be able to perform following operations

- Get and set current day, month and year
- Increment by x number of days, months and year
- Set default date

Attributes

Attributes that can be seen in this problem statement are

- Day
- Month
- Year
- Default date

Attributes

The default date is a feature shared by all objects

- This attribute must be declared a static member

Attributes in Date.h

```
class Date
{
        int day;
        int month;
        int year;
        static Date defaultDate;
…
};
```

Interfaces

- getDay
- getMonth
- getYear
- setDay
- setMonth
- setYear
- addDay
- addMonth
- addYear
- setDefaultDate

Interfaces

As the default date is a static member the interface setDefaultDate should also be declared static

Interfaces in Date.h

```
class Date{
…
public:
        void setDay(int aDay);
        int getDay() const;
        void addDay(int x);
        …
```

```
…
};
```
Interfaces in Date.h
```
class Date{
…
public:
        static void setDefaultDate(
int aDay,int aMonth, int aYear);
…
};
```
Constructors and Destructors in Date.h
```
        Date(int aDay = 0,
  int aMonth= 0, int aYear= 0);

        ~Date(); //Destructor
};
```
Implementation of Date Class

The static member variables must be initialized

```
Date Date::defaultDate (07,3,2005);
```
Constructors
```
Date::Date(int aDay, int aMonth,
                                int aYear)   {
        if(aDay==0) {
                this->day = defaultDate.day;
        }
        else{
                setDay(aDay);
        }
        //similarly for other members
}
```
Destructor

We are not required to do any house keeping chores in destructor

```
Date::~Date
{
}
```
Getter and Setter
```
void Date::setMonth(int a){
        if(a > 0 && a <= 12){
                month = a;
}
int getMonth() const{
        return month;
}
```
addYear
```
void Date::addYear(int x){
        year += x;
        if(day == 29 && month == 2
                &&      !leapyear(year)){
                day = 1;
```

 © Virtual University of Pakistan

```
                month = 3;
        }
}
```

Helper Function

```
class Date{
…
private:
        bool leapYear(int x) const;
…
};
```

Helper Function

```
bool Date::leapYear(int x) const{
        if((x%4 == 0 && x%100 != 0)
                ||      (x%400==0)){
                return true;
        }
        return false;
}
```

setDefaultDate

```
void Date::setDefaultDate(
        int d, int m, int y){
        if(d >= 0 && d <= 31){
                day = d;
        }
        …
}
```

13.4.Complete code of Date class

```
# include<iostream.h>
# include<conio.h>
class Date{

   private:
     int day, month, year;
         static Date defaultDate;
    public:
        void setDefaultDate(int aDay,int aMonth, int aYear);
         void setDay(int aDay);
         int getDay() const;
         void addDay(int x);
         void setMonth(int aMonth);
         int getMonth() const;
         void addMonth(int x);
         void setYear(int aYear);
         int getYear() const;
         void addYear(int x);
         bool leapYear(int x)const;
         Date(int aDay , int aMonth, int aYear);
         void setDate(int aDay , int aMonth, int aYear);
```

```
        ~Date(); //Destructor
    };

    Date Date::defaultDate(07,3,2005);

Date::Date(int aDay, int aMonth, int aYear)
{
        if(aDay==0)
        {
                this->day = defaultDate.day;
        }
        else
        {
           setDay(aDay);
        }

        if(aMonth==0)
        {
                this->month = defaultDate.month;
        }
        else
        {
           setMonth(aMonth);
        }

        if(aYear==0)
        {
                this->year = defaultDate.year;
        }
        else
        {
           setYear(aYear);
        }

}

void Date::setMonth(int a) {
   if(a > 0 && a <= 12)
   {
      month = a;
   }
}

 int Date:: getMonth() const {
      return month;
   }

void Date::addYear(int x)
{
   year += x;
```

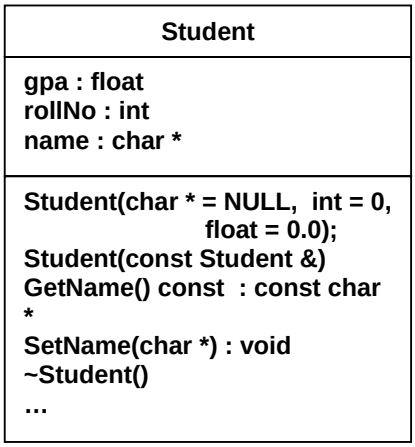

© Virtual University of Pakistan

```
    if(day == 29 && month == 2 &&  !leapYear(year))
    {
         day = 1;
         month = 3;
    }
}

bool Date::leapYear(int x) const {
    if((x%4 == 0 && x%100 != 0) || (x%400==0))
    {
          return true;
    }
    return false;
}

void Date::setYear(int aYear){
   year=aYear;
   }
void Date::setDay(int aDay){
   day=aDay;

   }

void Date::setDate(int aDay , int aMonth, int aYear){
   setDay(aDay);
   setMonth(aMonth);
   setYear(aYear);
   cout<<day<<"/"<<month<<"/"<<year<<endl;

   }


  Date::~Date(){
    cout<<"Date destructor"<<endl;
}
int main()
{
   Date aDate(0,0,0);

   aDate.setDate(20,10,2011);
   system("pause");
}
```

Lecture No.14

14.1.Composition

Consider the following implementation of the student class we discussed in previous lectures,



Composition

If one object is part of another object (relationship of part and whole) in composition lifetime of one object depends upon the other. The part objects are essential components of the whole.
For example person is **composed of** hands, eyes, feet so on.

In student class we assigning dynamic memory for variable name using new operator as shown,

```
class Student{
private:
  float gpa;
  char * name;
  int rollNumber;
public:
  Student(char * = NULL, int = 0, float = 0.0);
  Student(const Student & st);
  const char * GetName() const;
  // never return handle of private data members or private member functions
  ~Student();
  …
};

Student::Student(char * _name, int roll, float g)
{
        cout << "Constructor::Student..\n";
```

 © Virtual University of Pakistan

```
        if (!_name){
                name = new char[strlen(_name)+1];
                strcpy(name,_name);
        }
        else name = NULL;
        rollNumber = roll;
        gpa = g;
 }


Student::Student(const Student & st){
  if(str.name != NULL){
     name = new char[strlen(st.name) + 1];
     strcpy(name, st.name);
  }
  else name = NULL;
  rollNumber = st.roll;
  gpa = st.g;
}


 const char * Student::GetName(){
   return name;
 }
```

// never return handle of private data members or private member functions const ensures that private data members will not be changed

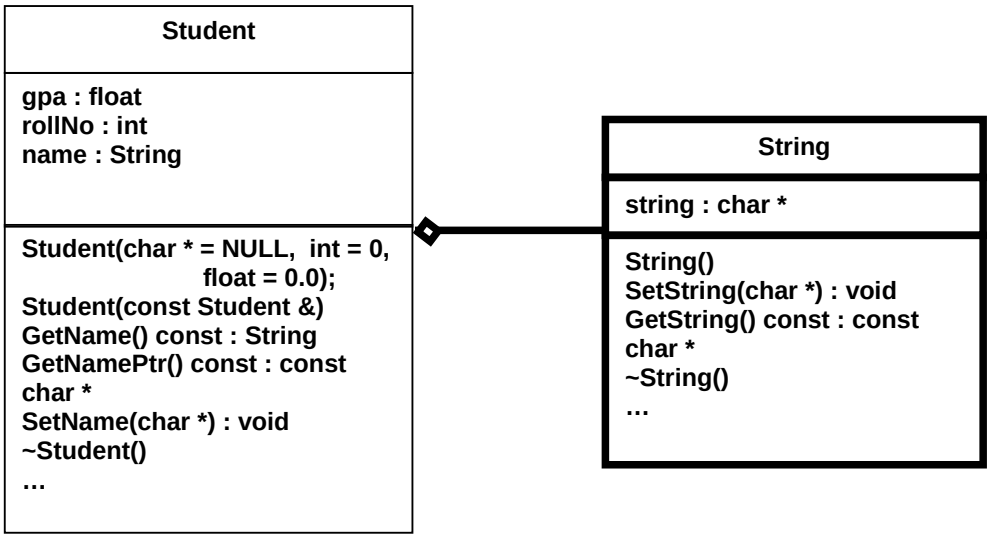
```
 Student::~Student(){
        delete [] name; // deleting name array
 }
```

In C++ *"it is all about code reuse"*
Composition is Creating objects of one class inside another class
*"**Has a**"* relationship:
Bird has a beak
Student has a name

Composition

Now we change code slightly replacing name char * by String so that it is whole object of class String as it qualifies to be an object because we have to apply many operations on it like string dynamic creation and deletion, string copy using deep copy, searching a substring and so on….

Conceptual notation:



Composition

Now we see string class code to see how it simplifies original Student object and how we have used composition here,

```
class String{
  private:
        char * ptr;
  public:
        String(); // default constructor
        String(const String &); // copy constructor
        void SetString(const char *); // setter function
        const char * GetString() const;
// getter function returning const pointer to data member ptr
        ~String()
        …
};
 String::String(){
        cout << "Constructor::String..\n";
        ptr = NULL;
 }

 String::String(const String & str){
   if(str.ptr != NULL){
         ptr = new char[strlen(str.ptr)+1];
         strcpy(ptr, str.ptr);
   }
   else ptr = NULL;
 }
 void String::SetString(const char * str){
        if(ptr != NULL){
                delete [] ptr;
```

 © Virtual University of Pakistan

```
            ptr = NULL;
        }
        if(str != NULL){
            ptr = new char[strlen(str)+1];
            strcpy(ptr, str);
        }
 }
```

issue of memory leakage (inaccessible memory)
if we simply set pointer here
memory will be outside of our object many cause problems later
user still has pointer of passed value it can itself modify it

We resolve these two issues in SetString by allocating new memory and deleting previous memory.

```
const char * String::GetString()const{
   return ptr;
}
String::~String(){
        delete [] ptr;
        cout <<"Destructor::String..\n";
}
```

Make pointer equal to NULL as well any where you delete dynamic memory.

Now consider the code of Student class again, now by adding composed string object our code has been simplified very much, (we will use methods of composed object simply by calling them where needed)

```
class Student{
private:
 float gpa;
 int rollNumber;
 String name;
public:
 Student(char* =NULL, int=0,float=0.0);
 Student(const Student &);
 void SetName(const char *);
 String GetName() const;
 const char * GetNamePtr() const;
 ~Student();
 …
};

Student ::Student(char * _name,      int roll, float g){
        cout <<"Constructor::Student..\n";
        name.SetString(_name);
```

```
        rollNumber = roll;
        gpa = g;
 }

Student::Student(const Student & s){
        name.SetString(s.name.GetString());
        gpa = s.gpa;
        rollNumber = s.rollNumber;
 }
```

Explanation:

1. **name.SetString**(s.name.GetString());
   // setting composed name of newly created object

2. name.SetString(**s.name**.GetString());
   //accessing the composed object string name of object to be copied

3. name.SetString(s.**name.GetString**());
   //accessing the value of composed object string name by calling its member function GetString

4. **name.SetString(s.name.GetString());**
   //overall result : the value of composed object string of object to be copied will be copied to newly created object composed object string.

```
const char * Student::GetNamePtr() const{
    return name.GetString();
 }

 void Student::SetName(const char * n){
        name.SetString(n);
 }

 Student::~Student(){
        cout <<"Destructor::Student..\n";
 }

void main(){
        Student *aStudent=new Student("Fakhir", 899, 3.1);
        cout << endl;
        cout << "Name:" << aStudent->GetNamePtr() << "\n";
}
```

Output:

```
Constructor::String..
Constructor::Student..
```


© Virtual University of Pakistan

```
Name:  Fakhir
Destructor::Student..
Destructor::String..
```

Important Points:

1. We can access methods of composed object in the same way as we can access methods of other objects.

   Name of composed object.MemberFunction

2. Member functions of a class can access its private data members like,

```
Student::Student(const Student & s){

        name.SetString(s.name.GetString());

        // accessing private member String name of student using its object s
        and then accessing String name member function GetString to access
        string value two methods calss in one line

            gpa = s.gpa;
        // accessing private members of student in student member function


            rollNo = s.rollNo;
        // accessing private members of student in student member function

}
```

Constructors & Composition

Constructors of the sub-objects are always executed before the constructors of the master class

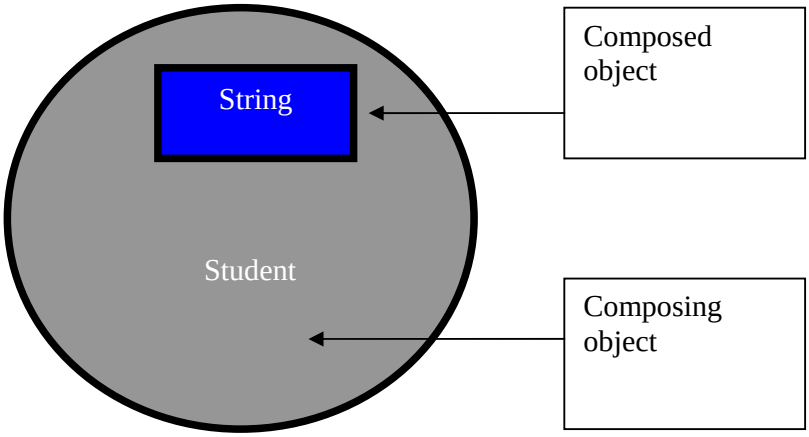
Example:

As you see the example output of program given above,

Output:

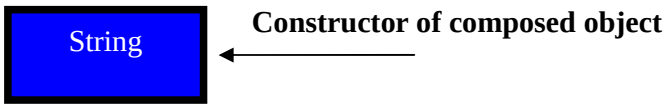
```
Constructor::String..
Constructor::Student..

Name:  Fakhir
Destructor::Student..
Destructor::String..
```

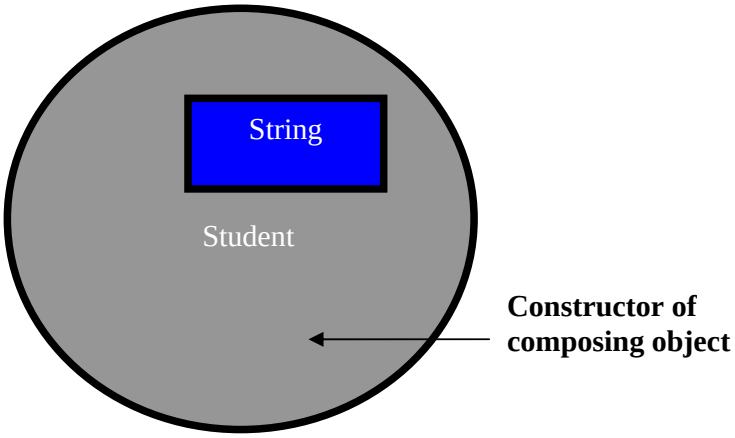
Constructor for the sub-object name is executed before the constructor of Student and destructor of sub-object is called after destructor of student. It is logical as composing object has to contain composed object so composed object should be created first and then composing object. Similarly while destructing objects we composing object is destructed first and then composed object as shown in diagram below,



Composition

Constructor calling:
Constructors are called from composed objects to composing objects.

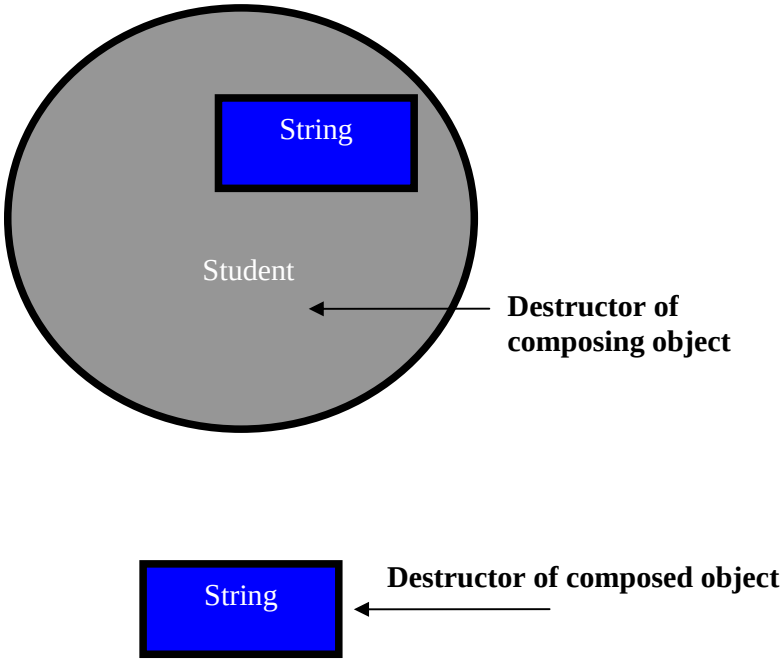




 © Virtual University of Pakistan

Destructor calling:

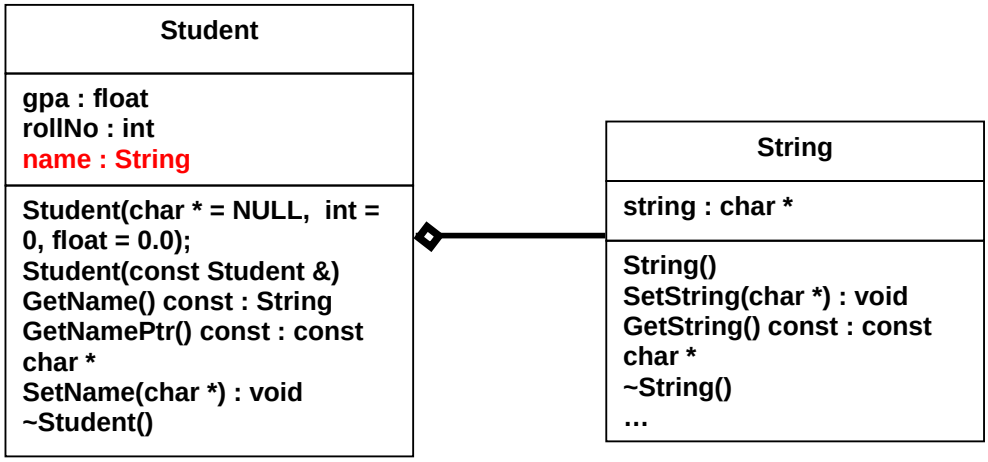
Destructors are called from composing objects to composed objects.



Lecture No.15

Composition:

We saw composition in last lecture, its Conceptual notation is given below,



We created student object in main as by passing name, rollno and gpa and then displayed the name of student using GetNamePtr member function of student class,

```
int main(){
 Student aStudent("Fakhir", 899,3.1);
 cout << endl;
 cout << "Name:" << aStudent.GetNamePtr()<< endl;
 return 0;
 }
```

Output:

The output of our code is given below,

```
Constructor::String..
Constructor::Student..
Name:  Fakhir
Destructor::Student..
Destructor::String..
```

Constructor Code:

Let us see the constructor code again,

```
Student::Student(char * n, int roll, float g){
      cout <<"Constructor::Student..\n";
      name.SetString(n);
      rollNumber = roll;
      gpa = g;
    }
```


© Virtual University of Pakistan

In this code we are setting string data member **name** of Student class using **SetString** but the problem in this approach is that we have to call SetString method explicitly to set value in string class, the reason for it is that our String class doesn't support setting its value while creating its object this is the reason we have to use the function SetString in the constructor.
This is an overhead and also not very good way to set any object value, we want to initialize our string sub-object **name** in the **student** class as we initialize other objects using constructor. For achieving this functionality we add an overloaded constructor in the **String** class that takes char string as parameter and initialize the String class object with this value using the Student constructor's "*Member initialization list*" as shown below in bold text:

```
class String{
        char *ptr;
public:
        String();
        String(char *); // constructor with char * as parameter
        String(const String &);
        void SetName(char *);
        ~String();
        …
};


String::String(char * str){
 if(str != NULL){
                ptr = new char[strlen(str)+1];
                strcpy(ptr, str);
}

else ptr = NULL;
   cout << "Overloaded Constructor::String..\n";

}
```

Now Student class constructor code is modified as follows:

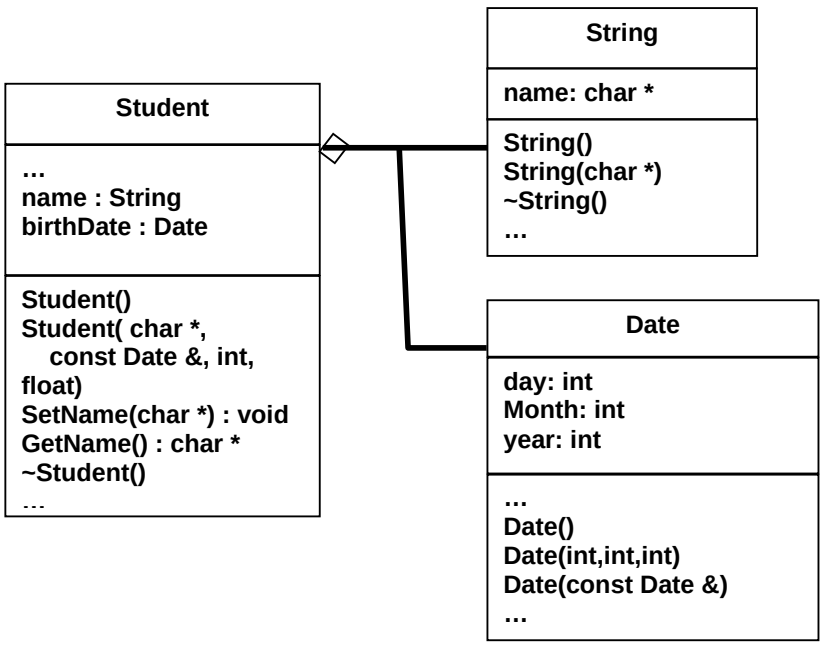
```
class Student{
private:
        float gpa;
        int rollNumber;
        String name;
public:
        …
        Student(char *=NULL, int=0, float=0.0);
};
```

```
Student::Student(char * n,int roll, float g):  name(n)  {

        cout << "Constructor::Student..\n";
        rollNumber = roll;
        gpa = g;

 }

int main(){
        Student aStudent("Fakhir", 899, 3.1);
        cout << endl;
        cout << "Name:" << aStudent.GetNamePtr()          << endl;
        return 0;
}
```

Output:

```
Overloaded Constructor::String..
Constructor::Student..

Name:  Fakhir
Destructor::Student..
Destructor::String..
```

Now suppose we want to add date object in student class to store student **Birth Date,** the conceptual diagram will be as given below,



 © Virtual University of Pakistan

Student class is modified as follows:

```
class Student{
private:
        …
        Date birthDate;
        String name;
public:
        Student(char *, const Date &, int, float);
        ~Student();
        …
};
```

Composition

```
 Student::Student(char * n, const Date & d, int roll, flaot g): name(n),birthDate(d) {
        cout << "Constructor::Student..\n";
        rollNumber = roll;
        gpa = g;
 }

 Student::~Student(){
        cout << "Destructor::Student..\n";
 }
int main(){
        Date _date(31, 12, 1982);
        Student aStudent("Fakhir",   _date,899,3.5);
        return 0;
}
```

Output:

```
        Overloaded Constructor::Date..
        Copy Constructor::Date..
        Overloaded Constructor::String..
        Constructor::Student..
        Destructor::Student..
        Destructor::String..
        Destructor::Date..
        Destructor::Date..
```

15.1. Aggregation

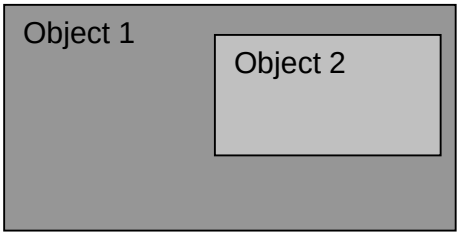
In composition we made separate object of those concepts that we think were worthy to be implemented as an object within other object to make our code simpler and to make functionality modular (divided in parts) and understandable like we made String class in Student class, but in real life most situations are such that two distinct (different) objects and one object is using services of the other one like student and teacher, student and librarian, room and chair, passenger and bus, book and

bookshelf, person and computer so on. In this case we cannot make one object as part of other object because they exist independently and only provide services to each other like in case of,

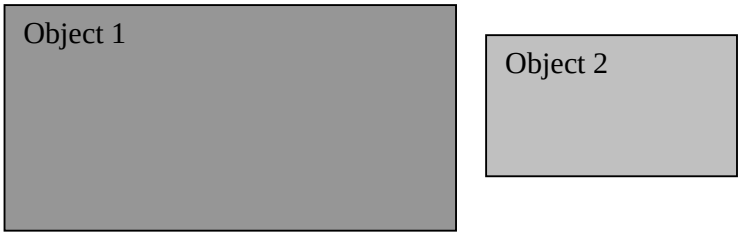
Student and Teacher: Student or Teacher object cannot be composed of other one yet they are taking services of each other.
Passenger and Bus: Passenger and Bus are taking services of each other but exist standalone also, bus includes passengers but passenger are not Part of Bus they can exist independently as well.

Composition vs. Aggregation

Aggregation is a *weak relationship* than composition because in this relationship two classes get services of each other but can exist independently as well, main difference is memory organization of two objects as shown below,



Composition

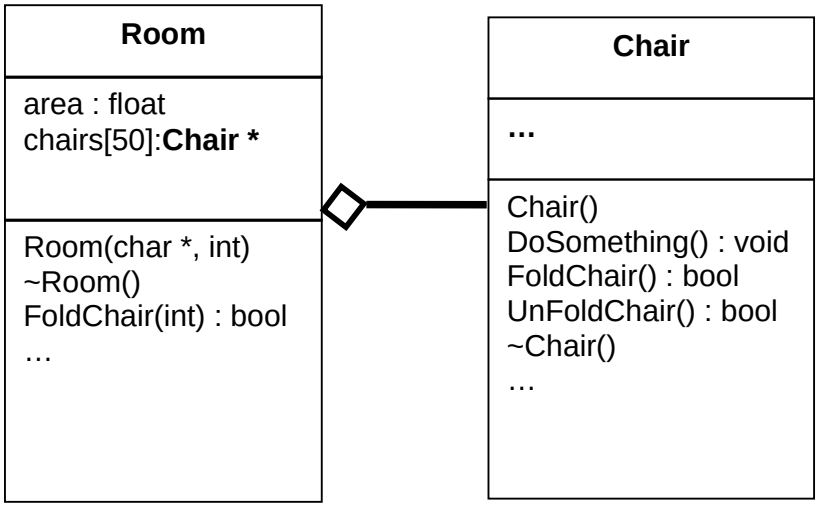


Aggregation

 © Virtual University of Pakistan

Example:

Take the example of Room and Chair as given below,



Aggregation C++ implementation:

In aggregation, a pointer or reference to an object is created inside a class. The sub-object has a life that is **NOT** dependant on the life of its master class.
e.g

- Chairs can be moved inside or outside at anytime
- When Room is destroyed, the chairs may or **may not** be destroyed

Aggregation:

```
class Room{
private:
        float area;
        Chair * chairs[50];
Public:
        Room();
        void AddChair(Chair *, int chairNo);
        Chair * GetChair(int chairNo);
        bool FoldChair(int chairNo);
        …
};

Room::Room(){
for(int i = 0; i < 50; i++)
chairs[i] = NULL;
```

```
}
void Room::AddChair(Chair * chair1, int chairNo){
        if(chairNo >= 0 && chairNo < 50)
                chairs[chairNo] = chair1;
}

Chair * Room::GetChair(int chairNo){
        if(chairNo >= 0 && chairNo < 50)
                return chairs[chairNo];
        else
                return NULL;
}

bool Room::FoldChair(int chairNo){
 if(chairNo >= 0 && chairNo < 50)
        return chairs[chairNo]->FoldChair();
 else
        return false;
}
int main(){
  Chair ch1;
  {
    Room r1;
          r1.AddChair(&ch1, 1);
          r1.FoldChair(1);
  }
   ch1.UnFoldChair(1);
  return 0;
}
```

15.2.Friend Functions

The functions which are not member functions of the class yet they can access all private members of the class are called friend functions.

Why they are needed?

They are needed in situations where we have written code for some function in one class and it need to be used by other classes as well for example,
Suppose we wrote the code to compute a complex mathematical formulae in one class but later it was required by other classes as well, in that case we will make that function friend of all other classes.

Are friend functions against the concept of Object Oriented Programming?

It can be said that friend functions are against the principle of object oriented programming because they violate the principle of encapsulation which clearly says that each object methods and functions should be encapsulated in it. But there we are making our private member accessible to other outside functions.

 © Virtual University of Pakistan

Consider the following class:

```
class X{
private:
        int a, b;
public:
        void MemberFunction();
        …
};
```

Suppose we have a global function **DoSomething** that need to access the private members of class X, when we will try to access them compiler will generate error as outside world can not access private members of a class except its member functions.

```
void DoSomething(X obj){

                obj.a = 3; //Error
                obj.b = 4; //Error
        }
```

Friend Functions

In order to access the member variables of the class, we must make function friend of that class,

```
class X{
private:
  int a, b;
public:
        …
   friend void DoSomething(X obj);
};
```

Now the function **DoSomething** can access data members of class X

```
void DoSomething(X obj){

                obj.a = 3;
                obj.b = 4;
        }
```

Friend Functions

Prototypes of friend functions appear in the class definition.
But friend functions are NOT member functions.

Friend Functions

Friend functions can be placed anywhere in the class without any effect
Access specifiers don't affect friend functions or classes

```
class X{
        ...
private:
        friend void DoSomething(X);
public:
        friend void DoAnything(X);
         ...
};
```

Friend Functions

While the definition of the friend function is:

```
void DoSomething(X obj){

        obj.a = 3;              // No Error
        obj.b = 4;              // No Error
        …
}
```

friend keyword is not given in definition.

Friend Functions

If keyword **friend** is used in the function definition, it's a syntax error

```
//Error…

friend void DoSomething(X obj){
        …
}
```

Friend Classes

Similarly, one class can also be made friend of another class:

```
class X{
        friend class Y;
        …
};
```

Member functions of class Y can access private data members of class X

```
class X{
        friend class Y;
        private:
                int x_var1, x_var2;
          ...
        };
class Y{
        private:
                int y_var1, y_var2;
```

 © Virtual University of Pakistan

```
            X objX;
        public:
            void setX(){
                objX.x_var1 = 1;
            }
        };
int main(){
            Y objY;
            objY.setX();
            return 0;
        }
```

Lecture No.16

16.1.Operator overloading

Consider the following class,

```
class Complex{
private:
        double real, img;
public:
        Complex Add(const Complex &);
        Complex Subtract(const Complex &);
        Complex Multiply(const Complex &);
        …
};
```

We want to write function to add two complex no. objects, the Add function implementation to add two complex numbers is shown below, this function is taking one complex no object and its adding the current (with reference to which it will be called) and is returning result in new object,

```
Complex Complex::Add(const Complex & c1){
        Complex t;
        t.real = real + c1.real;
        t.img  = img  + c1.img;
        return t;
};

// adds the contents of c2 to c1
// creating new object c3 and assigning it result of c1+c2
```

Now we can add two complex no. objects using the following statement,

```
Complex c3 = c1.Add(c2);
```

In this statement two operations are taking place,

1. One is addition of two objects using the function call Add and returning the result in a new object.
   Complex c3 = **c1.Add(c2);**

2. Second is copy of that temporary object to newly created object c3 using copy constructor.
   Complex c3 = c1.Add(c2);

But there are two issues with this implementation,

1. We can't add two complex no. objects by simple writing '+' as we can add basic data types like int or float as shown below,

int a = 3;

 © Virtual University of Pakistan

```
int b = 5;
c = a + b;  // correct

Complex c1(2,3), c2(4,5);
Complex c3 = c1 + c2; // error
```

Instead we have to explicitly write,

Complex c3 = c1.Add(c2)

If we give our complex no. class code some user in compiled form for use, user will need to know how we have written **Add** function (no. of parameters , return type) to add two complex no. objects so that he can call Add function correctly.

2. If we want to perform add operation on more than two objects in a single mathematical statement like:

c1+c2+c3+c4

We are unable to do it.

We have to explicitly write,

c1.Add(c2.Add(c3.Add(c4)))

Alternative way is:

```
t1 = c3.Add(c4);
t2 = c2.Add(t1);
t3 = c1.Add(t2);
```

This is also overhead, especially if the mathematical expression is large,

Converting it to C++ code will involve complicated mixture of function calls
Code will become less readable
Chances of human mistakes will become very high
Code produced will be very hard to maintain

The solution to this problem is simple that we can write normal operators like +,-,*, and so on for our user defined classes as well,

It is "***Operator overloading***"

Using operator overloading we can perform basic operations (like addition, subtraction, multiplication, division and so on…) on our own defined classes objects in the similar way as we perform them on basic built-in types(like int, float, long, double etc.).
C++ allows us to overload common operators like **+**, **-** or * etc…
With operator overloading Mathematical statements don't have to be explicitly converted into function calls as we had to do to add two complex no objects using function call **Add**.

Operator overloading

Assume that operator **+** has been overloaded then actual C++ code becomes:

c1+c2+c3+c4

The resultant code is very easy to read, write and maintain

Operator overloading

C++ automatically overloads operators for pre-defined types as these have also been implemented as classes by c++.
Example of predefined types:
int
float
double
char
long

Operator overloading

```
float x;
int y;
x = 102.02 + 0.09; // overloaded operator '+' for float type will be called by c++
Y = 50 + 47; // overloaded operator '+' for int type will be called by c++
```

The compiler probably calls the correct overloaded low level function for addition i.e:

```
// for integer addition:
Add(int a, int b)

// for float addition:
Add(float a, float b)
```

Operator overloading

Operator functions are not usually called directly, they are automatically invoked to evaluate the operations they implement by compiler.

List of operators that can be overloaded in C++:

```
new   delete     new[]      delete[]
+     -     *     /     %     ^     &     |     ~
!     =     <     >     +=    -=    *=    /=    %=
^=    &=    |=    <<    >>    >>=   <<=   ==    !=
<=    >=    &&    ||    ++    --    ,     ->*   ->
()    []
```

List of operators that can't be overloaded:

© Virtual University of Pakistan

.	.*	::	?:	#	##

Reason: They take actual current object name, rather than value in their argument as you have seen previously in the use of dot (**'.'**) operator,

Student std;
int roll = std.getRollNo() // dot operator is performing on actual function (getRollNo) of class Student that will vary from program to program.

?: is the only ternary operator in C++ and can't be overloaded.

The precedence of an operator:

The precedence of an operator is order of evaluation which operator will be evaluated first in expression.
The precedence of an operator is **NOT** affected due to overloading.
Example:

c1*c2+c3
c3+c2*c1
In both lines multiplication * will be done first and then addition.

Associativity:

Associativity is **NOT** changed due to overloading
Following arithmetic expression always is evaluated from left to right:
c1 + c2 + c3 + c4

Unary operators and assignment operator are right associative, e.g:
a=b=c is same as **a=(b=c)**
All other operators are left associative:
c1+c2+c3 is same as
(c1+c2)+c3

Important things to consider:
Always write code representing the operator for example adding subtraction code inside the + operator will create chaos.
Creating a new operator is a syntax error (whether unary, binary or ternary), you cannot create $.

Arity of Operators

Arity (no of operands it works on) of an operator is NOT affected by overloading
Example:
Division operator will take exactly two operands in any case:
b = c / d

General syntax of Operators Overloading:

In case of member functions of a class:

```
return_type class_name::operator operator_symbol( parameters ){
        /*code*/
}
```

In case of non member functions of a class (in this case we will make overloaded operator function as friend function):

```
return_type operator operator_symbol( parameters ){
        /*code*/
}
```

For example:

```
Complex& Complex::operator + (const Complex & c){
/*code*/
}

Complex& operator + (const Complex & c){
/*code*/
}
```

Binary Operators Overloading:

Binary operators act on two quantities.
Examples of binary operators:

+	-	*	/	%	^	&	|	~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	|=	<<	>>	>>=	<<=	==	!=
<=	>=	&&	||	,	->*	->		

General syntax of Binary Operators Overloading:

In case of member function of a class:

```
TYPE class_name::operator operator_symbol( TYPE rhs ){
        /*code*/
}
```

In case of non-member function of a class:

```
TYPE class_name::operator operator_symbol( TYPE rhs1, TYPE rhs2 ){
        /*code*/
}
```


© Virtual University of Pakistan

The "**operator OP**" must have at least one formal parameter of type class (user defined type)

Following is an error:
int operator + (int, int);

Examples:

Overloading + operator:

```
class Complex{
private:
        double real, img;
public:
        …
        Complex operator +(const Complex & rhs);
};

Complex Complex::operator +( const Complex & rhs){
        Complex t;
        t.real = real + rhs.real;
        t.img = img + rhs.img;
        return t;
}
```

The return type is Complex so as to facilitate complex statements like:
Complex t = c1 + c2 + c3;

The above statement is automatically converted by the compiler into appropriate function calls:
(c1.operator +(c2)).operator +(c3);

If the return type was **void**,

```
class Complex{
        ...
 public:
        void operator+(
                const Complex & rhs);
};

void Complex::operator+(const Complex & rhs){
        real = real + rhs.real;
        img = img + rhs.img;
};
```

We have to do the same operation **c1+c2+c3** as:
c1+c2
c1+c3
// final result is stored in c1

Drawbacks of void return type:

Assignments and cascaded expressions are not possible
One of the existing objects is used to store result
Code is less readable
Debugging is tough
Code is very hard to maintain

© Virtual University of Pakistan

Lecture No.17

Binary operators (cont.)

The binary operator is always called with reference to the left hand argument.

Example:
In **c1+c2**,
c1.operator+(c2)
// c1 is calling overloaded + operator and c2 is being passed as
// reference in that function.
In **c2+c1**,
c2.operator+(c1)
// c2 is calling overloaded + operator and c1 is being passed as
// reference in that function.

Adding basic data type to complex number class:

The overloading code we discussed before for complex no. class can add two complex number objects but it can not handle the following situation:

Complex c1;
c1 + 2.325 (as we are adding basic data type double to complex no.)

To do this, we have to modify the **Complex** class.

Modifying the complex class:

```
class Complex{
...
Complex operator+(const Complex & rhs);
Complex operator+(const double& rhs);
};

Complex operator + (const double& rhs){
        Complex t;
        t.real = real + rhs;
        t.img = img;
        return t;
}
```

Now we can write both forms of statements,
Complex c2, c3;
Complex c1 = c2 + c3;
Complex c4 = c2 + 235.01;
But problem arises if we do the following:

Complex c5 = 450.120 + c1;
The + operator is called with reference to **450.120**

No predefined overloaded + operator is there that takes **Complex** as an argument

Now if we write the following two functions to the class, we can add a **Complex** to a **real** or vice versa :

```
Class Complex{
…
friend Complex operator + (const Complex & lhs, const double & rhs);
friend Complex operator + (const double & lhs, const Complex & rhs);
};
```

We have made them as friend so that we can write them as non member functions and they are not called with respect to complex no. class object instead we pass both arguments (complex no. object and double value) to this function compiler invoke them according to arguments passed. Their implementation is similar,

```
Complex operator +(const Complex & lhs, const double& rhs){

        Complex t;
        t.real = lhs.real + rhs;
        t.img = lhs.img;
        return t;
}

Complex operator + (const double & lhs, const Complex & rhs){

        Complex t;
        t.real = lhs + rhs.real;
        t.img = rhs.img;
        return t;
}
```

Binary operators

So adding three overloaded versions for + operator as shown below allow us to write code to,
Add two complex objects
Add complex object and a double value.

```
Class Complex{
…
Complex operator + (const Complex &);
friend Complex operator + (const    Complex &, const double &);
friend Complex operator + (const    double &, const Complex &);
};
```

Non members which are not friend call also achieve this functionality but in that case we need extra four functions two getters and two setters each for real and imag part. Compiler searches overloaded operator code in member function first and then in non member functions.

 © Virtual University of Pakistan

Binary operators

Other binary operators are overloaded in similar to the + operator as demonstrated in the above examples
Example:

```
Complex operator * (const Complex & c1, const Complex & c2);
Complex operator / (const Complex & c1, const Complex & c2);
Complex operator - (const Complex & c1, const Complex & c2);
```

17.1. Overloading Assignment operator

As we know compiler can generate the following three functions for a class if required on its own,

- Default Constructor (in case we have not written any other constructor for a class)
- Copy Constructor
- Assignment Operator

However as we discussed previously if our class has any data member using dynamic memory then we had to write our own code for default constructor, copy constructor and similarly assignment operator as compiler generated version of these functions performs shallow copy that creates dangling pointer, and memory leakage issues in case of dynamic memory allocation.
We have already seen code of default constructor and copy constructor and the code for overloaded assignment operator is similar,

Consider the string class:

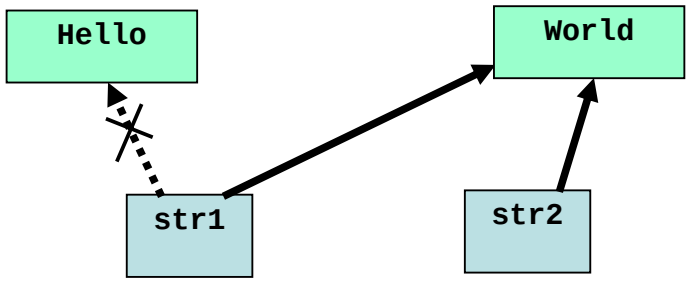
```
class String{
        int size;
        char * bufferPtr;
public:
        String(); // default constructor
        String(char *); // overloaded constructor
        String(const String &); // copy constructor
        …
};
String::String(){

                bufferPtr = NULL;
               size = 0;
}

String::String(char * ptr){
        if(ptr != NULL){
                size = strlen(ptr);
                bufferPtr = new char[size+1];
                strcpy(bufferPtr, ptr);
        }
```

```
        else{
                bufferPtr = NULL;
size = 0;
}
}

String::String(const String & rhs){
        size = rhs.size;

        if(rhs.size != 0){
                bufferPtr = new char[size+1];
                strcpy(bufferPtr, ptr);
        }
        else
                bufferPtr = NULL;

}

int main(){
        String str1("Hello");
        String str2("World");
        str1 = str2;
        return 0;
}
```

Member wise copy assignment

Assignment operator (Shallow Copy)

Result of **str1 = str2** (memory leak)



Second issue is dangling pointer issue as was in the case of copy constructor.

Modified Assignment Operator Code:

So we add overloaded assignment operator to perform deep copy as given below,

[7] Here by term member wise copy we mean copying values of members of class one by one blindly also called bitwise copy.
[8] In ANSI c++ standard term member wise copy has also been used to indicate logical copy (the deep copy)

 © Virtual University of Pakistan

```
class String{
        …
public:
        …
        void operator =(const String &);
};

void String::operator = (const String & rhs){
        size = rhs.size;
        if(rhs.size != 0){
          delete [] bufferPtr; // resolving inaccessible memory issue
          bufferPtr = new char[rhs.size+1]; // creating new dynamic memory
          strcpy(bufferPtr,rhs.bufferPtr); // deep copy
        }
        else
                bufferPtr = NULL;

}

int main(){
        String str1("ABC");
        String str2("DE"), str3("FG");
        str1 = str2;         // Valid…
        str1 = str2 = str3;  // Error…
        return 0;
}
```

The problem in statement
str1 = str2 = str3 is,

str1=str2=str3 is resolved as:
str1.operator=(str2.operator=(str3))
Assignment operator is beiing called two times one for part str2 **=** str3 and then for str1 **=** (str2 = str3) as assignment operator is right associate so first **str2=str3** will be executed, and str2 will become equal to str3, then first overloaded assignment operator execution result will be assigned to s1,
str1.operator=(**str2.operator=(str3)**)

Problem is when compiler will try to invoke second assignment operator to assign value to str1 error will be returned becuase of void return type of oveloaded assignment operator the reasn is explained below,
As we have studied before values are passed as parameters in operator overloadnig,
str2 = str3 means str2.operator = (str3)
// str3 is being called as parameter
and
str1 = str2 = str3 means str3.operator(str2.operator = (str3))
// str2.operator = (str3) is being passed as parameter

This issue can be resolved by introducing minor change in our code of copy assignment operator to make it return String object instead of void as shown below,

```
class String{
…
public:
        …
        String & operator = (const String &);
};

String & String :: operator = (const String & rhs){
        size = rhs.size;
        delete [] bufferPtr;
        if(rhs.size != 0){
          bufferPtr = new char[rhs.size+1];
          strcpy(bufferPtr,rhs.bufferPtr);
        }
        else bufferPtr = NULL;
        return *this;
}
```

Now we are returning the value by reference of the object with respect to which this overloaded assignment operator will be called. It will be str2 in the case of str2 = str3, now when part **str1 = (**str2 = str3**)** will be executed, str2 will be passed as argument, that will be assigned to str1.

© Virtual University of Pakistan

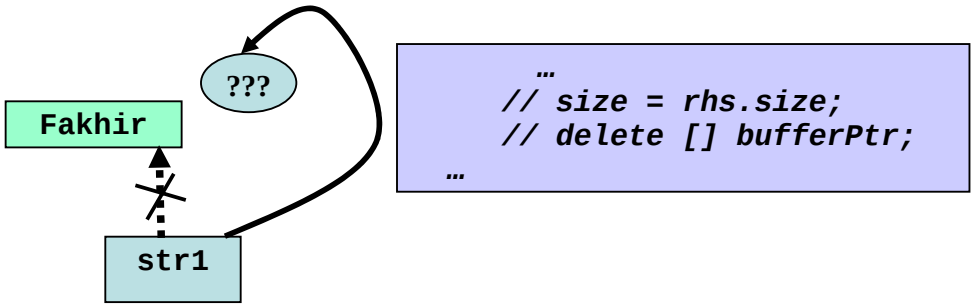
Lecture No.18

18.1.Self assignment problem:

In we assign same string to itself as done in main function below our program will produce unexpected results as source and destination operands for copying are same,

```
int main(){
        String str1("Fakhir");
        str1 = str1; // Self Assignment problem…
        return 0;
}
```

Result of str1 = str1



We can resolve this issue by adding a simple if condition to ensure that both strings are not same

```
String & String :: operator = (const String & rhs){
 if(this != &rhs){
        size = rhs.size;
        delete [] bufferPtr; // deleting memory of left hand side operand
        if(rhs.bufferPtr != NULL){
          bufferPtr = new char[rhs.size+1];
          strcpy(bufferPtr,rhs.bufferPtr);
    // memory access violation or incorrect data copy
        }
        else bufferPtr = NULL;
 }
 return *this;
}
```

Now self-assignment is properly handled:

```
int main(){
        String str1("Fakhir");
        str1 = str1;
        return 0;
}
```

We can make return type String & as constant to avoid assignment to sub expressions, like (str1 = str2) = str3

```
class String{
…
public:
        …
        const String & operator=[9]
                        (const String &);
};

int main(){
        String s1("ABC"),
                s2("DEF"),
                s3("GHI");
        // Error…
        (s1 = s2) = s3;
        return 0;
}
```

But as we can do that with primitive types so we can allow assignment to sub expressions by making return type as String & only as we have done before.

```
int main(){
        int a, b, c;
        (a = b) = c;
        return 0;
}
```

18.2.Other Binary operators

Overloading += operator:

```
class Complex{
  double real, img;
public:
Complex & operator+=(const Complex & rhs);
Complex & operator+=(const double & rhs);
...
};

Complex & Complex::operator += (const Complex & rhs){
        real = real + rhs.real;
        img = img + rhs.img;
        return * this;
}

Complex & Complex::operator += (const double & rhs){
        real = real + rhs;
        return * this;
```

[9] We have seen previously that we should not return handle to any private data member

```
}

int main(){
        Complex c1, c2, c3;
        c1 += c2;
        c3 += 0.087;
        return 0;
}
```

18.3.Friend Functions and Operator overloading

Friend functions minimize encapsulation as we can access private data of any class using friend functions,
This can result in:

- Data vulnerability
- Programming bugs
- Tough debugging

Hence, use of friend functions must be limited we can overload operators without declaring them friend functions of a class, for example the + operator can be defined as a non-member, non-friend function as shown below,
(Three versions of overloaded + operator for Complex no class to handle three kinds of statements)

- obj1 + obj2
    - + obj1
- obj1 + 3.78

```
Complex operator + (const Complex &  a, const Complex & b){
        Complex t = a; // creating temporary object t to store a+b
        return t += b; // returning t by reference
}

Complex operator + (const double & a, const Complex & b){
        Complex t = b;
        return t += a;
}

Complex operator + (const Complex & a, const double & b){
        Complex t = a;
        return t += b;
}
```

Other Binary operators

The operators
-=, /=, *=, |=, %=, &=, ^=, <<=, >>=, !=
can be overloaded in a very similar fashion.

Lecture No.19

Overloading stream insertion extraction operators

19.1. Stream Insertion operator

Often we need to display the data on the screen c++ provides us insertion operator ('<<')to put data on output stream default is console but it can be any file or network socket as well to send data on network from our program.

Example:

```
int i=1, j=2;
cout << "i= "<< i << "\n";
cout << "j= "<< j << "\n";
```

19.2. Stream Extraction operator

We also need to get data from the console or from file or network this is achieved through c++ provided stream extraction operator ('>>') that is used to get data from input stream, again default input stream is from console.

Example:

```
int i,j;
cin >> i >> j; // getting value of i and j from user
```

Explanation:

cin and **cout** are objects of istream and ostream classes used for input and output respectively, the insertion and extractions operators have been overloaded in istream and ostream classes to do these tasks.

When we write lines like,

```
int i;
cin>> i;
cout << i;
```

Actually we are using istream and ostream class objects and using these objects we are calling these classes overloaded (>> and <<) operators that have been overloaded for all basic types like integer, float , long , double and char *.
We have seen previously that actual call of overloaded operators for a class takes place by passing the objects to overloaded function as parameter like shown below,

```
cin>>i;
istream & operator >> (istream & in, int & i)
```

Here cin will be passed as istream object along with int i to be displayed and code of this functions is returning istream object by reference & to accommodate multiple input statement in a single line like,

```
int i , j;
cin>>i >> j;
```

as we did in our string class before.

© Virtual University of Pakistan

Same goes for insertion operator <<
cout << i;
ostream & operator >> (ostream & os, const int & i)

Stream insertion and extraction operator have been overloaded for basic data types but if we try to use them for user defined data types like our Complex no. class compiler will generate error as it will not find any overloaded operator code for our complex no class

```
Complex c1;
cout << c1;  // Error
cout << c1 << 2; // Error cascaded statement
```

// Compiler error: binary '<<' : no operator // defined which takes a right-hand operand of type 'class Complex'

Same error will be for stream extraction operator so will need to overload these two operators (<< and >>) for our Complex no. class.

19.3.Overloading Stream Insertion Operator

First we try to overload insertion << operator as member function as we did before,

```
class Complex{
        …
public:
        …
        void operator << (const
                                Complex & rhs);
};
```

But when we will call this overloaded function is main compiler will generate errors as shown below,

```
int main(){
        Complex c1;
        cout << c1;             // Error
        c1 << cout;
        c1 << cout << 2; // Error
        return 0;
};

class Complex{
        …
public:
        …
        void operator << (ostream &);
};

void Complex::operator << (ostream & os){
```

```
        os      << '('  << real
                << ',' << img << ')';
}
```

Now the statement c1 << cout will work but it has two limitations,
Difficult to understand and remember statement syntax (c1 << cout ;)
Cascaded statements not possible (cout << c1 << 2 ;)

Better syntax is given below to resolve these two issues,

```
class Complex{
  ...
friend ostream & operator << (ostream & os, const Complex & c);
};
```

Stream Insertion operator

```
// we want the output as: (real, img)
ostream & operator << (ostream & os, const Complex & c){
        os << '(' << c.real
          << ','
          << c.img << ')';
        return os;
}
```

ostream reference can not be const as it store the data in its buffer to insert on output stream, however Complex reference will be constant as we are only getting data from Complex object and inserting it to output stream.

```
Complex c1(1.01, 20.1), c2(0.01, 12.0);
cout << c1 << endl << c2;
```

Stream Insertion operator

Output:

```
( 1.01 , 20.1 )
( 0.01 , 12.0 )
```

Now cascading statements are also possible as given below,

```
cout << c1 << c2;
```

is equivalent to

operator<<(**operator<<(cout,c1)**,c2);

Because insertion operator is Left to right associative so first left part **cout << c1** << c2 will be executed and then the next part as opposed to copy assignment operator that will right associative.

Same thing can be done with stream extraction operator,

19.4. Overloading Stream Extraction Operator:

 © Virtual University of Pakistan

```
class Complex{
  ...
friend istream & operator >> (istream & i, Complex & c);
};
```

istream can not be cont and istream buffer will change as we will get data from it and assign it to complex reference similarly Complex object can not be const for stream extraction operator as well because we will add data to it and hence its state will change.

Stream Extraction Operator Code:

```
istream & operator << (istream & in, Complex & c){
        in >> c.real;
        in >> c.img;
        return in;
}
```

Main Program:

```
Complex c1(1.01, 20.1);
cin >> c1;
// suppose we entered    // 1.0025 for c1.real and     // 0.0241 for c1.img
cout << c1;
```

Output:

```
( 1.0025 , 0.0241 )
```

19.5.Other Binary operators:

Overloading comparison operators (Equality and Inequality operators)

```
class Complex{
public:
        bool operator == (const Complex & c);
//friend bool operator == (const //Complex & c1, const Complex & c2);
        bool operator != (const Complex & c);
//friend bool operator != (const //Complex & c1, const Complex & c2);
        …
};
```

Equality operator:

```
bool Complex::operator ==(const Complex & c){
        if((real == c.real) &&
          (img == c.img)){
```

```
                return true;
        }
        else
                return false;
}
```

As non member friend function:

```
bool operator ==(const Complex& lhs, const Complex& rhs){
        if((lhs.real == rhs.real) &&
          (lhs.img == rhs.img)){
                return true;
        }
        else
                return false;
}
```

Inequality Operator:

```
bool Complex::operator !=(const Complex & c){
        if((real != c.real) ||
          (img != c.img)){
                return true;
        }
        else
                return false;
}
```


© Virtual University of Pakistan

Lecture No.20

Modified String Class:

We have seen the following string class till now,

```
class String{
private:
        char * bufferPtr;
    int size;
public:
        String();
        String(char * ptr);
        void SetString(char * ptr);
        const char * GetString();
        ...
};

int main(){
        String str1("Test");
        String str2;
        str2.SetString("Ping");
        return 0;
}
```

What if we want to change the string from "*Ping*" to "*Pong*"?? *{ONLY 1 character to be changed…}*

Possible solution:

- Call: **str2.SetString("Pong");**
- This will delete the current buffer and allocate a new one
- Too much overhead if string is too big

Or, we can add a function which changes a character at nth location

```
class String{
        ...
public:
        void SetChar(char c, int pos);
        ...
};

 void SetChar(char c, int pos){
        if(bufferPtr != NULL){
                if(pos>0 && pos<=size)
                        bufferPtr[pos] = c;
        }
}
```

Other Binary Operators

Now we can efficiently change a single character:

```
String str1("Ping");
```

str1.SetChar('o', 2);
*// str1 is now changed to "**Pong**"*

20.1. Subscript [] Operator

There is another elegant solution present for this problem. It is subscript operator that is used on basic char [] data type as shown below,

```
char array[5] = "Ping";
array[1] = 'o';
```

We want to get same functionality for our own defined String class for this we overload the subscript **"[]"** operator.

We want function like given below for subscript operator in our String class,

```
int main(){
        String str2;
        str2.SetString("Ping");
        str[2] = 'o';
// acting as l-value (left value so that we can assign it some value)
        cout << str[2];
// acting as r-value (we are reading value using subscript operator)
        return 0;
}
```

20.2. Overloading Subscript [] Operator

Subscript operator must be overloaded as member function of the class with one parameter of integer type,

```
class String{
        ...
public:
        char & operator[](int);
        ...
};

char & String::operator[]( int pos){
        assert(pos>0 && pos<=size);
        return stringPtr[pos-1];
}
int main() {
        String s1("Ping");
        cout <<str.GetString()<< endl;
        s1[2] = 'o';
        cout << str.GetString();
        return 0;
}
```

Output:

 © Virtual University of Pakistan

Ping
Pong

20.3.Overloading Function () operator

Must be a member function
Any number of parameters can be specified
Any return type can be specified
Operator() can perform any generic operation

Function Operator

```
class String{
        ...
public:
        char & operator()(int);
        ...
};

char & String::operator()
                                (int pos){
        assert(pos>0 && pos<=size);
        return bufferPtr[pos-1];
}
int main(){
String s1("Ping");
char g = s1(2); // g = 'i'
s1(2) = 'o';
cout << g << "\n";
cout << str.GetString();
return 0;
}
```

Output:

i
Pong

20.4.Function Operator performing Sub String operation,

```
class String{
        ...
public:
        String operator()(int, int);
        ...
};

String String::operator()(int index,    int subLength){
        assert(index>0 && index+subLength-1<=size);
        char * ptr = new char[subLength+1];
```

```
        for (int i=0; i < subLength; ++i)
    ptr[i] = bufferPtr[i+index-1];
        ptr[subLength] = '\0';
        String str(ptr);
        delete [] ptr;
        return str;
}

int main(){
        String s("Hello World");
        // "<<" is overloaded
        cout << s(1, 5);
        return 0;
}
```

Function Operator
Output:

```
Hello
```

20.5. Unary Operators

Unary operators take one operand, they act on the object with reference to which they have been called as shown below,
& * + - ++ -- ! ~

Examples:

- **--x**
- **-(x++)**
- **!(*ptr ++)**

Unary Operators

Unary operators are usually prefix, except for **++** and **--**
++ and **--** both act as prefix and postfix

Example:
h++;
g-- + ++h - --i;

General syntax for unary operators
As Member Functions:
TYPE & operator OP (); // no argument the object with respect to which it is called is taken as one operand
As Non-member Functions:
Friend TYPE & operator OP (TYPE & t);
// one argument object with respect to which it is called.

 © Virtual University of Pakistan

Overloading unary '-':

```
class Complex{
...
Complex  operator - ( );
// friend Complex operator -(Complex &);
};

Complex Complex::operator -(){
        Complex temp;
        temp.real = -real;
        temp.img = -img;
        return temp;
}
```

Complex c1(1.0 , 2.0), c2;
c2 = -c1;
// c2.real = -1.0
// c2.img = -2.0

Unary '+' is overloaded in the same way.

Lecture No.21

Unary Operators

21.1. Behavior of ++ and -- for pre-defined types:

- Post-increment **++:**

Post-increment operator ++ increments the current value and then returns the previous value

- Post-decrement --: Works exactly like post ++

Example:

```
int x = 1, y = 2;
cout << y++ << endl;
cout << y;
```

Output:

2
3

Example:

```
int y = 2;
y++++;// Error
y++ = x;          // Error
```

Behavior of ++ and -- for pre-defined types:

- Pre-increment ++:

Pre-increment operator ++ increments the current value and then returns it's reference

- Pre-decrement --:

Works exactly like Pre-increment ++

Example:

```
int  y = 2;
cout << ++y << endl;
cout << y << endl;
```

Output:

3
3

Example:

```
int x = 2, y = 2;
++++y;
cout << y;
++y = x;
cout << y;
```

Output:

4
2

© Virtual University of Pakistan

Example (Pre-increment):

```
class Complex{
        double real, img;
public:
...
        Complex & operator ++ ();
    // friend Complex & operator ++(Complex &);
};

Complex & Complex::operator++(){       // member function
        real = real + 1;
        return * this;
}

Complex & operator ++ (Complex & h){  // non member function
        h.real += 1;
        return h;
}
```

Example:

```
Complex h1, h2, h3;
++h1;
```

Function **operator++()** returns a reference so that the object can be used as an *lvalue*

```
++h1 = h2 + ++h3;
```

How does a compiler know whether it is a pre-increment or a post-increment?

A post-fix unary operator is implemented using:
Member function with 1 dummy int argument
OR
Non-member function with two arguments

In post increment, current value of the object is stored in a temporary variable
Current object is incremented
Value of the temporary variable is returned

21.2.Post-increment operator:

```
class Complex{
...
        Complex operator ++ (int);
// friend Complex operator ++(const Complex &, int);
};

Complex Complex::operator ++ (int){
        complex t = *this;
```

```
        real += 1;
        return t;
}

Complex operator ++ (const
                        Complex & h, int){
        complex t = h;
        h.real += 1;
        return t;
}
```

How does a compiler know whether it is a pre-increment or a post-increment?

The dummy parameter in the operator function tells compiler that it is post-increment
Example:

```
Complex h1, h2, h3;
h1++;
h3++ = h2 + h3++; // Error…
```

The *pre* and *post* decrement operator **--** is implemented in exactly the same way

21.3.Type Conversion
The compiler automatically performs a type coercion of compatible types
e.g:

```
int f = 0.021;
double g = 34;
```

// type **float** *is automatically converted into* **int**. *Compiler only issues a warning…*

Type Conversion
The user can also explicitly convert between types:

```
int g = (int)0.0210;
double h = double(35);
```

// type **float** *is explicitly converted (casted) into* **int**. *Not even a warning*
// is issued now…

Type Conversion

For user defined classes, there are two types of conversions
From any other type to current type
From current type to any other type

We can do the following type conversion by our self,
Conversion from any other type to current type:
Requires a constructor with a single parameter
For example,

```
String str = 135;
```

Conversion from current type to any other type:
Requires an overloaded operator

 © Virtual University of Pakistan

For example,
String str;
char * ptr = str;

Conversion from other type to current type (int to String):

```
class String{
...
public:
        String(int a);
        char * GetStringPtr()const;
};
String::String(int a){
        cout << "String(int) called..." << endl;
        char array[15];
        itoa(a, array, 10);
        size = strlen(array);
        bufferPtr = new char [size + 1];
        strcpy(bufferPtr, array);
}
char * String::GetStringPtr() const{
        return bufferPtr;
}
int main(){
        String s = 345;
        cout << s.GetStringPtr() << endl;
        return 0;
}
```

Output:

```
String(int) called…
345
```

Automatic conversion like shown above using constructor has drawbacks conversion takes place transparently even if the user didn't wanted the conversion for example see the code below, in it user can write the following code to initialize the string with a single character:

```
int main(){

        String s = 'A';
// ASCII value of A that is 65 is being taken automatically it may be what the user //
wants, perhaps user intent was String s = "A" to store A in the string object
// but instead 65 is being stored

        cout    << s.GetStringPtr()<< endl << s.GetSize() << endl;
        return 0;
```

```
}
```

Output:

```
String(int) called…
65
2
```

Keyword **explicit**

There is a mechanism in C++ to restrict automatic conversions like this using constructor code it is to use keyword **explicit** with such constructors if we have used this keyword then casting must be explicitly performed by the user.
Keyword **explicit** only works with constructors.

Example:

```
class String{
        …
public:
        …
        explicit String(int);
};

int main(){
        String s;
        s = 'A';          // Error…
        return 0;
}

int main(){
        String s1, s2;
        s1 = String(101);       // valid, explicit casting…
        // OR
        s2 = (String)204;
        return 0;
}
```

Type Conversion

There is another method for type conversion:
"Operator overloading"
It is used for converting from current type (user defined) to any other basic type or user defined type.

General Syntax:

$TYPE_1$::operator $TYPE_2$();
Like,

```
String::operator char * ();  // TYPE1 = String , TYPE2 = char *
```

 © Virtual University of Pakistan

It will convert our string object to char *.

We write such functions as member function of the class.
NO return type and arguments are specified
Return type is implicitly taken to be TYPE2 by compiler

Type Conversion

Overloading pre-defined types:

```
class String{
        …
public:
        …
        operator int();
        operator char *();
};
String::operator int(){
        if(size > 0)
                return atoi(bufferPtr);
        else
                return -1;
}

String::operator char *(){
        return bufferPtr;
}


int main(){
        String s("2324");
        cout << (int)s << endl << (char *)s;
   //    int a = (int)s;
   //    int a = int (s);
        return 0;
}
```

Output:

2324
2324

21.4.User Defined types:

User-defined types can be overloaded in exactly the same way
Only prototype is shown below:

```
class String{
        …
        operator Complex();
        operator HugeInt();
        operator IntVector();
```

```
};
```

21.5. Drawbacks of Type Conversion Operator:

```
class String{
        …
public:
        …
        String(char *);
        operator int();
};

int main(){
        String s("Fakhir");
        // << is NOT overloaded
        cout << s; // compiler is automatically converting s to int
        return 0;
}
```

Output:

Junk Returned…

To avoid this problem DO NOT use *type conversion operators* instead use separate member function for such type conversion as shown below,

Modifying String class:

```
class String{
        …
public:
        …
        String(char *);
        int AsInt();
};
int String::AsInt(){
        if(size > 0)
                return atoi(bufferPtr);
        else
                return -1;
}
int main(){
        String s("434");
        // << is NOT overloaded
        cout << s; //error
        cout << s.AsInt();
        return 0;
}
```

 © Virtual University of Pakistan

Lecture No.22

22.1.Practical implementation of Inheritance in c++

Topics to be discussed in this lecture are,

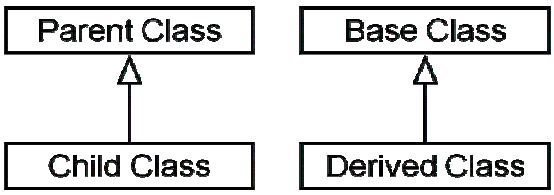
- Inheritance in Classes
- UML Notation of Inheritance
- Types of Inheritance in c++
- IS A relationship
- Accessing members
- Allocation in memory
- Constructors
- Base class initializers
- Initializing members
- Destructors
- Order of execution constructors and destructors
- Examples of C++ implementation of inheritance

22.2.Inheritance in Classes

If a class B inherits from class A, then B contains all the characteristics (information structure and behavior) of class A.
The class whose behavior is being inherited is called base class and the class who inherits the behavior of base class is called derived class. Base class is also called parent class and child class is called also derived class
Besides inherited characteristics, derived class may have its own unique characteristics

22.3.UML Notation

We use arrow from derived class to the parent class to show inheritance as shown below,



22.4.Inheritance in C++

In c++ we can inherit a class from another class in three ways,

- Public
- Private
- Protected

22.5. "IS A" Relationship

Inheritance represents "IS A" relationship for example "a student IS A person".
In general words we can say that inheritance represents,

"Derived class IS A kind of Parent class"

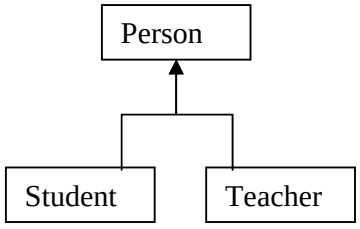
C++ Syntax of Inheritance

```
class ChildClass
        : public BaseClass{
...
};
```

Example

```
class Person{
    ...
};
class Student: public Person{
...
};
```

Accessing Members
Public members of base class become public member of derived class.
Private members of base class are not accessible from outside of base class, even in the derived class (Information Hiding)
Example
In the code given below Student and Teacher classes has been derived from single Person class,




```
class Person{
  char *name;
  int age;
  ...
public:
  const char *GetName() const;
  int GetAge() const;
  ...
};
```

 © Virtual University of Pakistan

```
class Teacher: public
  Person{
  char * dept;
  int course;
  ...
public:
  char * GetDept() const;
  int GetCourse() const;
  void Print() const;
  ...
};
```

```
class Student: public
  Person{
  int semester;
  int rollNo;
  ...
public:
  int GetSemester() const;
  int GetRollNo() const;
  void Print() const;
  ...
};
```

Example

```
void Student::Print()
```

Error

```
{
        cout << name << " is in" << " semester " << semester;
}
```

corrected Code:

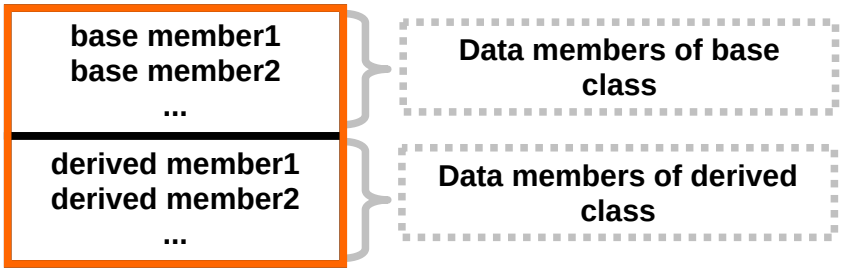
```
void Student::Print()
{
        cout << GetName() << " is in semester " << semester;
}
int main(){
        Student stdt;

        stdt.semester = 0;//error
        stdt.name = NULL; //error
        cout << stdt.GetSemester();
        cout << stdt.GetName();

        return 0;
}
```

Explanation of above code (char * data type)

In C++ char arrays (char []) are handled in two ways one way is statically using statements like,
char name[30]; // static array of length 30 characters
or dynamically as shown below,
char * name;
name = new char[30];
In dynamic creation of arrays we simply store char * in class and assign it a dynamic memory according to our need using new operator.

Allocation in Memory

The object of derived class is represented in memory as follows

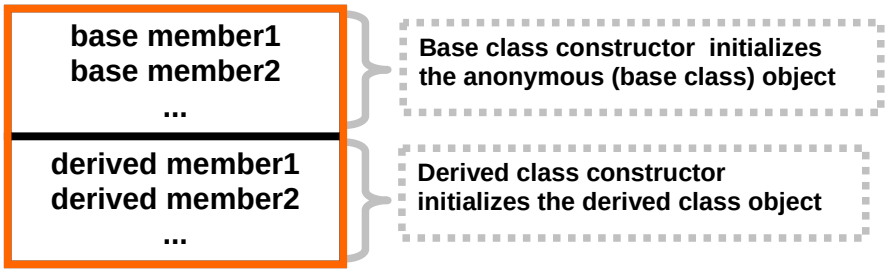


Derived Class Object

Every object of derived class has an anonymous object of base class

Constructors

- The anonymous object of base class must be initialized using constructor of base class
- When a derived class object is created the constructor of base class is executed before the constructor of derived class



Derived Class Object

Example

```
class Parent{
public:
        Parent(){ cout <<
                "Parent Constructor...";}
};
class Child : public Parent{
public:
        Child(){        cout <<
                "Child Constructor...";}
};
```

 © Virtual University of Pakistan

```
int main(){
        Child cobj;
        return 0;
}
```

Output:

```
Parent Constructor...
Child Constructor...
```

Constructor

- If default constructor of base class does not exist then the compiler will try to generate a default constructor for base class and execute it before executing constructor of derived class
- If the user has given only an overloaded constructor for base class, the compiler will not generate default constructor for base class

Example

```
class Parent{
public:
        Parent(int i){}
};
class Child : public Parent{
public:
        Child(){}
} Child_Object; //ERROR
```

Definition of Some Terms:

Default constructor: Default constructor is such constructor which either has no parameter or if it has some parameters these have default values. The benefit of default constructor is that it can be used to create class object without passing any argument.

Implicit Default constructor:

Compiler generates implicit default constructor for any class in case we have not given any constructor for the class.

Explicit Default constructor:

If user has given constructor for any class without any arguments or with all arguments with default values then it is also default constructor according to definition but it is explicit (user defined) default constructor.

Now if a base class has only **non-default constructor** (constructor with parameters without default values), then when we will create object of any class derived from this base class compiler will not be able to call base class constructor as base class has no default constructor (constructor that can be called without giving any parameters) so compiler will generate error.

We can avoid this error by calling base class non-default constructor in derived class constructor initializer list by ourself.

Base Class Initializer

- C++ has provided a mechanism to explicitly call a constructor of base class from derived class

- The syntax is similar to member initializer and is referred as base-class initialization

Example

```
class Parent{
public:
        Parent(int i){…};
};
class Child : public Parent{
public:
        Child(int i): Parent(i)
        {…}
};
```

Example

```
class Parent{
public:
        Parent(){cout <<
                "Parent Constructor...";}
        ...
};
class Child : public Parent{
public:
        Child():Parent()
        {cout << "Child Constructor...";}
        ...
};
```

Base Class Initializer

- User can provide base class initializer and member initializer simultaneously

Example

```
class Parent{
public:
        Parent(){…}
};
class Child : public Parent{
        int member;
public:
        Child():member(0), Parent()
        {…}
};
```

Base Class Initializer

- The base class initializer can be written after member initializer for derived class

 © Virtual University of Pakistan

- The base class constructor is executed before the initialization of data members of derived class.

Initializing Members

- Derived class can only initialize members of base class using overloaded constructors
    - Derived class can not initialize the public data member of base class using member initialization list

Example

```
class Person{
public:
        int age;
        char *name;
        ...
public:
        Person();
};
```

Example

```
class Student: public Person{
private:
        int semester;
...
public:
        Student(int a):age(a)
        {                                       //error
        }
};
```

Reason

- It will be an assignment not an initialization

Destructors

- Destructors are called in reverse order of constructor called
- Derived class destructor is called before the base class destructor is called

Example

```
class Parent{
public:
        Parent(){cout <<"Parent Constructor";}
        ~Parent(){cout<<"Parent Destructor";}
};

class Child : public Parent{
public:
        Child(){cout << "Child Constructor";}
        ~Child(){cout << "Child Destructo";}
};
```

Example

Output: **Parent Constructor**

```
Child Constructor
Child Destructor
Parent Destructor
```


© Virtual University of Pakistan

Cs-304 Important Mid Term Mcq's Solution 100% Correct : Solve By Vu-Topper RM!!

Question No:1(Marks:1) (Marks:1) Vu-Topper RM
A language feature that combines variables and methods in a single package is called _____
Encapsulation

Question No:2 (Marks:1) Vu-Topper RM
__is a methodology that organizes a program into a collection of interacting objects.
OOP

Question No:3 (Marks:1) Vu-Topper RM
Every object hides its state and show through behavior. this concept is known as
Information Hiding

Question No:4 (Marks:1) Vu-Topper RM
Capture only those details about an object that are relevant to current perspective
Abstraction

Question No:5 (Marks:1) Vu-Topper RM
Which is not feature of OOP in general definitions….
Duplicate/Redundant data

Question No:6 (Marks:1) Vu-Topper RM
What is the additional feature in classes that was not in structures?
Member functions

Question No:7 (Marks:1) Vu-Topper RM
Who invented OOP….
Alan Kay

Question No:8 (Marks:1) Vu-Topper RM
if we extend our model, and the rest of the model is not affected then it is called—
—
Flexibility
Class

Question No:9 (Marks:1) Vu-Topper RM
Suppose there is an object of type Person, which of the following can be considered as one of its attributes?
Name and Age

Question No:10 (Marks:1) Vu-Topper RM
Which of the following is not true about constant member function?
All

Question No:11 **(Marks:1)** **Vu-Topper RM**
Subtyping means that derived class is behaviorally _____ with the base class.
Compatible

Question No:12 **(Marks:1)** **Vu-Topper RM**
___ is represented by a line with an unfilled diamond head towards the container.
Aggregation

Question No:13 **(Marks:1)** **Vu-Topper RM**
Minimum classes required for implementing multiple inheritance:
3

Question No:14 **(Marks:1)** **Vu-Topper RM**
Destructor is a function which has the same name as that of class, but starts with a-----sign.
Tilda (~)

Question No:15 **(Marks:1)** **Vu-Topper RM**
A good model is ..... related to a real life problem.
Closely

Question No:16 **(Marks:1)** **Vu-Topper RM**
How many objects are involved in the N-ary association?
more than 3

Question No:17 **(Marks:1)** **Vu-Topper RM**
In constant member function the type of this pointer is:
Constant pointer to constant data

Question No:18 **(Marks:1)** **Vu-Topper RM**
Compiler generated default constructor is called-----and user written--- default constructor is called_____.
Implicit, explicit

Question No:19 **(Marks:1)** **Vu-Topper RM**
Which of the following is an advantage of encapsulation?
Better understanding

Question No:20 **(Marks:1)** **Vu-Topper RM**
If some of objects exhibit identical characteristics, then they belong to:
Different classes

Question No:21 **(Marks:1)** **Vu-Topper RM**
Abstraction hides the---------details.
Relevant

Question No:22 **(Marks:1)** **Vu-Topper RM**
Polymorphism makes the system:
All of the given option

Question No:23 **(Marks:1)** **Vu-Topper RM**
If class B inherits from class A then it contains all characteristics of _.
Class A

Question No:24 **(Marks:1)** **Vu-Topper RM**
How the information hidden within an object can be accessed?
Through its interface

Question No:25 **(Marks:1)** **Vu-Topper RM**
Which of the following depicts the proper definition of class?
A class is a description of a kind of object

Question No:26 **(Marks:1)** **Vu-Topper RM**
Which of the following statement best describes the Constructor?
Constructor is used to initialize the data members of a class

Question No:27 **(Marks:1)** **Vu-Topper RM**
_____ is automatically called when the object is created.
Constructor

Question No:28 **(Marks:1)** **Vu-Topper RM**
Consider the statement" room has chair" which of the following type of Association exists between whom and chair?
Aggregation

Question No:29 **(Marks:1)** **Vu-Topper RM**
The—— tells the compiler what task the function will be performing
Function definition

Question No:30 **(Marks:1)** **Vu-Topper RM**
Which of the following is a tangible entity
Car

Question No:31 **(Marks:1)** **Vu-Topper RM**

Suppose derived class is inherited from base class. What happens when a derived class object is created?

The constructor of base class is executed before the constructor of derived class.

Question No:32 (Marks:1) Vu-Topper RM

Suppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of overloaded assignment operator for the following statement to work correctly.

str1 = str2 = str3;

String& operator =(const String &);

Question No:33 (Marks:1) Vu-Topper RM

In C++, which of the following is defined as stream insertion operator?

<<

Question No:34 (Marks:1) Vu-Topper RM

Choose correct declaration of overloaded inequality (!=) operator for class String as non-member friend function.

None of the given options

Question No:35 (Marks:1) Vu-Topper RM

Relationship in which child object gets destroyed if parent object is destroyed.

Composition

Question No:36 (Marks:1) Vu-Topper RM

How the information hidden within an object can be accessed?

Through its interface

Question No:37 (Marks:1) Vu-Topper RM

Which of the following defines default constructor for class "Student"?

Student() {//...}

Question No:38 (Marks:1) Vu-Topper RM

The sub-object’s life is not dependent on the life of master class in _____.

Aggregation

Question No:39 (Marks:1) Vu-Topper RM

Hiding the implementation details makes program:

Easy to understand

Question No:40 (Marks:1) Vu-Topper RM

The functions defined inside the class are by default _____.

Inline

Question No:41 (Marks:1) Vu-Topper RM

Which of the following statement is TRUE about static function of a class?

It is used to access static data members.

Question No:42 (Marks:1) Vu-Topper RM

Deconstruct is a function which has the same name as that of class but starts with a—– sign

~

Question No:43 (Marks:1) Vu-Topper RM

'this' pointer cannot be passed implicitly to _____ functions.

Static Member

Question No:44 (Marks:1) Vu-Topper RM

Which of the following can be the behavior of an object “Usman”?

Eat

Question No:45 (Marks:1) Vu-Topper RM

In Object Oriented programming, objects communicate with each other through _____

Messages

Question No:46 (Marks:1) Vu-Topper RM

Constructor is used to_____the objects of a class.

Initialize

Question No:47 (Marks:1) Vu-Topper RM

Which of the following is NOT an access specifier in C++?

Hidden

Question No:48 (Marks:1) Vu-Topper RM

The other name of subtyping is _____.

Extension

Question No:49 (Marks:1) Vu-Topper RM

Suppose a class does not have any constructor. What will happen when an object of this class is created?

Compiler will call implicit default constructor.

Question No:50 (Marks:1) Vu-Topper RM

In case when we define the function outside the class then we must use the keyword _____ to make the function inline.

Inline

Question No:51 (Marks:1) Vu-Topper RM

In constant member function the type of this pointer is:

Constant pointer to constant data

Question No:52 (Marks:1) Vu-Topper RM

In _____, base class can be replaced by the derived class.

Extension

Question No:53 (Marks:1) Vu-Topper RM

We can get the address of a variable stored in a pointer using _____:

&symbol

Question No:54 (Marks:1) Vu-Topper RM

A child inherits characteristics from its _____.

Parent

Question No:55 (Marks:1) Vu-Topper RM

Which of the following function declaration is correct to overload the + operator as member function in Complex class?

Complex operator +(const Complex &rhs);

Question No:56 (Marks:1) Vu-Topper RM

Which of the following is not true about constant member function?

All of the given options

Question No:57 (Marks:1) Vu-Topper RM

What is the general syntax of overloading Unary Operator as member function of the class?

TYPE & operator OP ();

Question No:58 (Marks:1) Vu-Topper RM

_____ is creating objects of one class inside another class.

Composition

Question No:59 (Marks:1) Vu-Topper RM

Memory is allocated to non static members only, when
Object is created

Question No:60 (Marks:1) Vu-Topper RM

A child inherit characteristics from its ?
parents

Question No:61 (Marks:1) Vu-Topper RM

A post-fix unary operator is implemented in C++ member function with
1 dummy int arguments

Question No:62 (Marks:1) Vu-Topper RM

Which of the following is the example of objects in school?
Books , pages

Question No:63 (Marks:1) Vu-Topper RM

Suppose student is a class, which of the following constructor with one parameter for class student?
Student()

Question No:64 (Marks:1) Vu-Topper RM

If you have three classes in a C++ program A, B, and C where class A inherits from class B, then class _____ contains all the characteristics of class _____.
A, C

Question No:65 (Marks:1) Vu-Topper RM

In ____ base class can be replaced by the derived class.
Extension

Question No:66 (Marks:1) Vu-Topper RM

The concept of derived class is involved in _____
inheritance

Question No:67 (Marks:1) Vu-Topper RM

Suppose person is a class which of the following statement defines an object of class person?
Create person object

Question No:68 (Marks:1) Vu-Topper RM

The other name of subtyping is
Extension

Question No:69 (Marks:1) Vu-Topper RM

Which of the following statement best describes the constructor
constructor is used to initialize the data member of a class.

Question No:70 (Marks:1) Vu-Topper RM

Which of the following can be the behavior of an object “Usman”?
Eat

Question No:71 (Marks:1) Vu-Topper RM

Composition is ____ relationship
Part whole

Question No:72 (Marks:1) Vu-Topper RM

Which of the following is a strong relationship?
Composition

Question No:73 (Marks:1) Vu-Topper RM

This car is composed of wheels. What is the relationship between car and wheels?
Composition

Question No:74 (Marks:1) Vu-Topper RM

In ____ base class can be replaced by the derived class.
Extension

Question No:75 (Marks:1) Vu-Topper RM

___ defines the order of evaluation of an operator is an expression.
Operator precedence

Question No:76 (Marks:1) Vu-Topper RM

In composition __ are called from composing objects to composed objects.
Destructor

Question No:77 (Marks:1) Vu-Topper RM

How many objects of a given class may be constructed in an application?
As many as the application asks for

Question No:78 (Marks:1) Vu-Topper RM

Relationship in which child object gets destroyed, If parent object is destroyed.
Composition

Question No:79 (Marks:1) Vu-Topper RM

Which of the following is not true about constructor?
Constructor is a special function which can have different name than class name.

Question No:80 (Marks:1) Vu-Topper RM

Suppose test is a class, void show() is its public member function. which of the following is correct call of show() function using Test pointer ptr?
ptr->show()

Question No:81 (Marks:1) Vu-Topper RM

Which of the following features of OOP is used to deal with only relevant details?
Abstraction

Question No:82 (Marks:1) Vu-Topper RM

In C++ the subscript operator must be overloaded as a ___ of the class, with one parameter of ___ type.
Member function, int

Question No:83 (Marks:1) Vu-Topper RM

Identify the abstract class from the given statement
“vehicle class is base class. Bus, Car and truck are derived classes”
Vehicle

Question No:84 (Marks:1) Vu-Topper RM

How can we differentiate between constructor and destructors?
Destructor are preceded by a tilde(~) symbol and constructor are not preceded with any symbol.

Question No:85 (Marks:1) Vu-Topper RM

If you do not initialize static variable of int type then it is automatically initialized with ____
0

Question No:86 (Marks:1) Vu-Topper RM

___ associates the objects of the exact two classes.
Binary association

Question No:87 (Marks:1) Vu-Topper RM

Which of the following is the advantage of inheritance?

Reusability

Question No:88 (Marks:1) Vu-Topper RM

Member function defined inside a class declaration are __ by default

Public

Question No:89 (Marks:1) Vu-Topper RM

In Object Oriented programming objects communicate with each other through ____
Messages

Question No:90 (Marks:1) Vu-Topper RM

____ associates the objects of the exact three classes.
Ternary Association

Question No:91 (Marks:1) Vu-Topper RM

An objects has attributes, operations and _____
Unique identity

Question No:92 (Marks:1) Vu-Topper RM

In C++ which of the following keywords works only with constructors?
Explicit

Question No:93 (Marks:1) Vu-Topper RM

Which of the following allows to reuse characteristics of more than one parent class?
Multiple Inheritance

Question No:94 (Marks:1) Vu-Topper RM

How many objects are involved in the Binary association?
2

Question No:95 (Marks:1) Vu-Topper RM

Which of the following features of OOP is used to deal with only relevant details?
Abstraction

Question No:96 (Marks:1) Vu-Topper RM

The overloaded '-' operator for complex class will be called with reference to ___ in the following statement
Complex C3 = C1 – C2
Complex

Question No:97 (Marks:1) Vu-Topper RM

Static data member is declared __
Inside the class

Question No:98 (Marks:1) Vu-Topper RM

Which of the following will happen when a constant function tries to change the value of data members of the class?
Complex time error will occur

Question No:99 (Marks:1) Vu-Topper RM

_____ is set of functions of an objects exposed to other objects.
Interface

Question No:100 (Marks:1) Vu-Topper RM

How many objects are involved in the Ternary association?
3

Question No:101 (Marks:1) Vu-Topper RM

Mermaid is an example of
Multiple inheritance

Question No:102 (Marks:1) Vu-Topper RM

Which of the following is the correct syntax of declaring static variable 'count' of type int?
static int count

Question No:103 (Marks:1) Vu-Topper RM

Which of the following operators operate on one operand?
Unary Operators

Question No:104 (Marks:1) Vu-Topper RM

Object Data includes ___
Both static and attributes of an objects

Question No:105 (Marks:1) Vu-Topper RM

In case when we define the function outside the class then we must use the keyword ___ to make function inline.
Inline

Question No:106 (Marks:1) Vu-Topper RM

A function call is resolved at run time in a)Non virtual members function
Virtual member function

Question No:107 (Marks:1) Vu-Topper RM

If the user does not specify the type if inheritance, then the default type of inheritance is
Private inheritance

Question No:108 (Marks:1) Vu-Topper RM

In case of private inheritance, private members of base class will be in derived class?
Hidden

Question No:109 (Marks:1) Vu-Topper RM

Then int member i of base class is accessible in class, a)derived 1 only
derived 2 only

Question No:110 (Marks:1) Vu-Topper RM

In case of public inheritance, protected members of base class will be in derived class?
Protected

Question No:111 (Marks:1) Vu-Topper RM

Friend's functions of class are _ members of that class.

Private

Question No:112 (Marks:1) Vu-Topper RM

Methodologies to the development of reusable software relate to .
Generic programming

Question No:113 (Marks:1) Vu-Topper RM

Function overriding is done in context of,
Derived and base classes

Question No:114 (Marks:1) Vu-Topper RM

Which of the following is not type of inheritance in c++?
Restricted

Question No:115 (Marks:1) Vu-Topper RM

In public inheritance the public members of base class become in derived class

Public

Question No:116 (Marks:1) Vu-Topper RM

When we want to implement one class in terms of another class then we use

Public inheritance

Question No:117 (Marks:1) Vu-Topper RM

In c++ generic programming is done using

Templates

Question No:118 (Marks:1) Vu-Topper RM

Friend's functions of class are _ members of that class.

Private

Question No:119 (Marks:1) Vu-Topper RM

Methodologies to the development of reusable software relate to . a)Structure programming

Generic programming

Question No:120 (Marks:1) Vu-Topper RM

Function overriding is done in context of,

Derived and base classes

Question No:121 (Marks:1) Vu-Topper RM

the following constructors?

Implicit Default Constructor

Question No:122 (Marks:1) Vu-Topper RM

Suppose we have defined derived class copy constructor but have not defined base class copy constructor then compiler will,

Generate base class copy constructor itself

Question No:123 (Marks:1) Vu-Topper RM

Child class can call constructor of its

Direct base class

Question No:124 (Marks:1) Vu-Topper RM

A child class can call constructor ofthe parent call through

Its constructor initialization list

Question No:125 (Marks:1) Vu-Topper RM

memory allocation in or class we should use

User defined assignment operator

Question No:126 (Marks:1) Vu-Topper RM

By default, the vector data items are initialized to

0 google

Question No:127 (Marks:1) Vu-Topper RM

In Private ---- only member functions and friend classes or functions of a derived class can convert pointer or reference of derived object to that of parent object

inheritance

Question No:128 (Marks:1) Vu-Topper RM

Template functions use __ than ordinary functions.

Lesser Memory

Question No:129 (Marks:1) Vu-Topper RM

Non-Template Friend functions of a class are friends of instance/s of that class.

All google

Question No:130 (Marks:1) Vu-Topper RM

Which one of the following terms must relate to polymorphism? a)Static allocation

Dynamic binding

Question No:131 (Marks:1) Vu-Topper RM

Assume a class Derv that is privately derived from class Base. An object of class Derv locatedinmain() can access

public members of Derv

Question No:132 (Marks:1) Vu-Topper RM

A function call is resolved at run-time in a)non-virtual member function.

virtual member function.

Question No:133 (Marks:1) Vu-Topper RM

Two important STL associative containers are and

set,map

Question No:134 (Marks:1) Vu-Topper RM

An abstract class is useful when,

We do not want to instantiate its object

Question No:135 (Marks:1) Vu-Topper RM

In order to define a class template, the first line of definition must be:

template <typename T> (Page 257)

Question No:136 (Marks:1) Vu-Topper RM

Identify the correct way of declaring an object of user defined template class A for char typemembers?

A< char > obj;

Question No:137 (Marks:1) Vu-Topper RM

The user must define the operation of the copy constructor.

True

Question No:138 (Marks:1) Vu-Topper RM

The find() algorithm

takes iterators as its first two arguments. (Object-Oriented Programming in C++)

Question No:139 (Marks:1) Vu-Topper RM

Compiler performs type checking to diagnose type errors,

Static

Question No:140 (Marks:1) Vu-Topper RM

Vectors contain contiguous elements stored as a[an] . a)variable

array

Question No:141 (Marks:1) Vu-Topper RM

In a de-queue, (chose the best option)

data can be inserted or deleted at any arbitrary location, but the process is relatively slow.

Question No:142 (Marks:1) Vu-Topper RM

When a virtual function is called by referencing a specific object by name and using the dot member selection operator (e.g.,squareObject.draw()), the reference is resolved at compile time.

False

Question No:143 (Marks:1) Vu-Topper RM

Considering the resolution order in which considering the resolution order in which compiler search for functions in a program; the first priority is given to; the first priority is given to,

ordinary function

Question No:144 (Marks:1) Vu-Topper RM

One purpose of an iterator in the STL is to connect algorithms and containers.

True

Question No:145 (Marks:1) Vu-Topper RM

In , a base class can be replaced by its derived class,

Sub-typing

Question No:146 (Marks:1) Vu-Topper RM

What is a class?

A class is a description of a kind of object.

Question No:147 (Marks:1) Vu-Topper RM

Inheritance is a way to a)organize data.

add features to existing classes without rewriting them.

Question No:148 (Marks:1) Vu-Topper RM

We can use "this" pointer in the constructor in the body and even in the initialization list of any classif we are careful,

True

Question No:149 (Marks:1) Vu-Topper RM

----- and------- methods may not be declared abstract.

private,static

Question No:150 (Marks:1) Vu-Topper RM

Which of the following operator(s) take(s) one or no argument if overloaded?

++(Page 162)

Question No:151 (Marks:1) Vu-Topper RM

Which of the following operators cannot be overloaded?

Scope resolution operator (::)

Question No:152 (Marks:1) Vu-Topper RM

Virtual functions allow you to

use the same function call to execute member functions of objects from different classes

Question No:153 (Marks:1) Vu-Topper RM

The copy () algorithm returns an iterator to

the element one past the last element copied to.

Question No:154 (Marks:1) Vu-Topper RM

If you define a vector v with the default constructor, and define another vector w with aone-argument constructor to a size of 11, and insert 3 elements into each of these vectors with push_back(), then the size() member function will return for v and for w.

3 for v and 11 for w.

Question No:155 (Marks:1) Vu-Topper RM

Which of the following may not be an integral part of an object?

Protected data members

Question No:156 (Marks:1) Vu-Topper RM

A static member function can be called, even when a class is not _____.

Instantiated

Question No:157 (Marks:1) Vu-Topper RM

_____ provide the facility to access the data member.

accesser function

Question No:158 (Marks:1) Vu-Topper RM

A generalization-specialization relation between classes are implemented using

inheritance

Question No:159 (Marks:1) Vu-Topper RM

Which of the following is not an example of multiple inheritances?

Woman

Question No:160 (Marks:1) Vu-Topper RM

A generic class showing all the common attributes and a behavior of other classes represents a very important feature in oop called ---

Inheritance

Question No:161 (Marks:1) Vu-Topper RM

The parameters given in temple definition other than those used for mentioning templates types are called

Non Type parameters

Question No:162 (Marks:1) Vu-Topper RM

Which of the following is the correct way to define a template class X?

Template <typename T>classX{}

Question No:163 (Marks:1) Vu-Topper RM

In resolution order of function template, compiler searches for in the end

Generic template

Question No:164 (Marks:1) Vu-Topper RM

When we specialize a function template, it is called

Function template overloading

Question No:165 (Marks:1) Vu-Topper RM

Which of the following can be passed as type argument to template?

Both primitive type and user defined types

Question No:166 (Marks:1) Vu-Topper RM

Which of the following is known as Dereference operator inC++?

*****

Question No:167 (Marks:1) Vu-Topper RM

Which of the following represents partial specialization?

Template<class T, Class U, int>

Question No:168 (Marks:1) Vu-Topper RM

Which of the following statement is true about partial specialization?

Both class templates and function template can have partial specialization.

Question No:169 (Marks:1) Vu-Topper RM

Which of the following is correct code portion to initialize static datamember "value"of int type for class X?

Template<class T> int X < T >::value = 0;

Question No:170 (Marks:1) Vu-Topper RM

When we want to have exactly identical operations on different data type, are ____used

Function Template

Question No:171 (Marks:1) Vu-Topper RM

In case of template specialization, if compiler cannot find required complete specialization then it searches for some

Partial specialization

Question No:172 (Marks:1) Vu-Topper RM

Which of the following is a constrained version of some first-class container?

Container Adapter

Question No:173 (Marks:1) Vu-Topper RM

In resolution order, highest priority is given to in template specialization.

Complete specialization

Question No:174 (Marks:1) Vu-Topper RM

Which of the following is not a sequence container?

Map

Question No:175 (Marks:1) Vu-Topper RM

STL stands for:

Standard Template Library

Question No:176 (Marks:1) Vu-Topper RM

_____ Binding means that target function for a call is selected at compile time.

Static

Question No:177 (Marks:1) Vu-Topper RM

In C++, Composition is a relationship between _____ and _____ objects.

Parent, child

Question No:178 (Marks:1) Vu-Topper RM

A pure virtual function is a virtual function that

causes its class to be abstract.

Question No:179 (Marks:1) Vu-Topper RM

If class B inherits from class A then it contains all characteristics of _____

Polymorphism

Question No:180 (Marks:1) Vu-Topper RM

_____ is represented by a line with an unfilled diamond head towards the container.

Aggregation

Question No:181 (Marks:1) Vu-Topper RM

Subtyping means that derived class is behaviorally _____ with the base class.

Compatible

Question No:182 (Marks:1) Vu-Topper RM

In operator overloading, which of the following operator takes one or no argument.

<<

Question No:183 (Marks:1) Vu-Topper RM

In C++ generic programming is done using 148. 149. 150. 151. 152. 153.

Packages

Question No:184 (Marks:1) Vu-Topper RM

In polymorphism, messages can be interpreted in different ways depending upon the _____ class.

receiver

Question No:185 (Marks:1) Vu-Topper RM

Information hiding can be achieved through_____.

Encapsulation, Abstraction

Question No:186 (Marks:1) Vu-Topper RM

A good model is ..... related to a real life problem.

Closely

Question No:187 (Marks:1) Vu-Topper RM

Which of the following features of OOP is used to derive a class from another?

Inheritance

Question No:188 (Marks:1) Vu-Topper RM

Which of the following is a weak relationship between two objects?

Aggregation

Question No:189 (Marks:1) Vu-Topper RM

Data items in a class must be private.

False

Question No:190 (Marks:1) Vu-Topper RM

Memory is allocated to non-static members only, when:

Object is created

Question No:191 (Marks:1) Vu-Topper RM

The sub-object's life is not dependent on the life of master class in ___.

Aggregation

Question No:192 (Marks:1) Vu-Topper RM

Unary operators and assignment operator are right associative.

true

Question No:193 (Marks:1) Vu-Topper RM

The >= operator can't be overloaded.

false

Question No:194 (Marks:1) Vu-Topper RM

This pointer does not pass implicitly to _____ functions.

Static Member

Question No:195 (Marks:1) Vu-Topper RM

Operator overloading is

giving C++ operators more than they can handle.

Question No:196 (Marks:1) Vu-Topper RM

Which of the following is TRUE,

Derived class pointer can be used as Base class pointer

Question No:197 (Marks:1) Vu-Topper RM

_____ Binding means that target function for a call is selected at run time

Dynamic

Question No:198 (Marks:1) Vu-Topper RM

Consider the code below, class c1{ }; class c2 : public c1 { }; class c3 :

public c2 { }; Then c2 is

Direct base class of c3

Question No:199 (Marks:1) Vu-Topper RM

A parent class can call constructor of its child class through

Can not call the constructor of its child class

Question No:200 (Marks:1) Vu-Topper RM

_____ is automatically called when the object is created.

Constructor

Question No:201 (Marks:1) Vu-Topper RM

Inheritance is a way to

Add features to existing classes without rewriting them

Question No:202 (Marks:1) Vu-Topper RM

Through interface we access object_____.

Behaviour

Question No:203 (Marks:1) Vu-Topper RM

Which of the following is the way to extract common behaviour and attributes from the given and make a separate class of those common behaviours and attributes?

Sub-typing

Question No:204 (Marks:1) Vu-Topper RM

A class has _____ destructor

One

Question No:205 (Marks:1) Vu-Topper RM

A class has _____ constructor

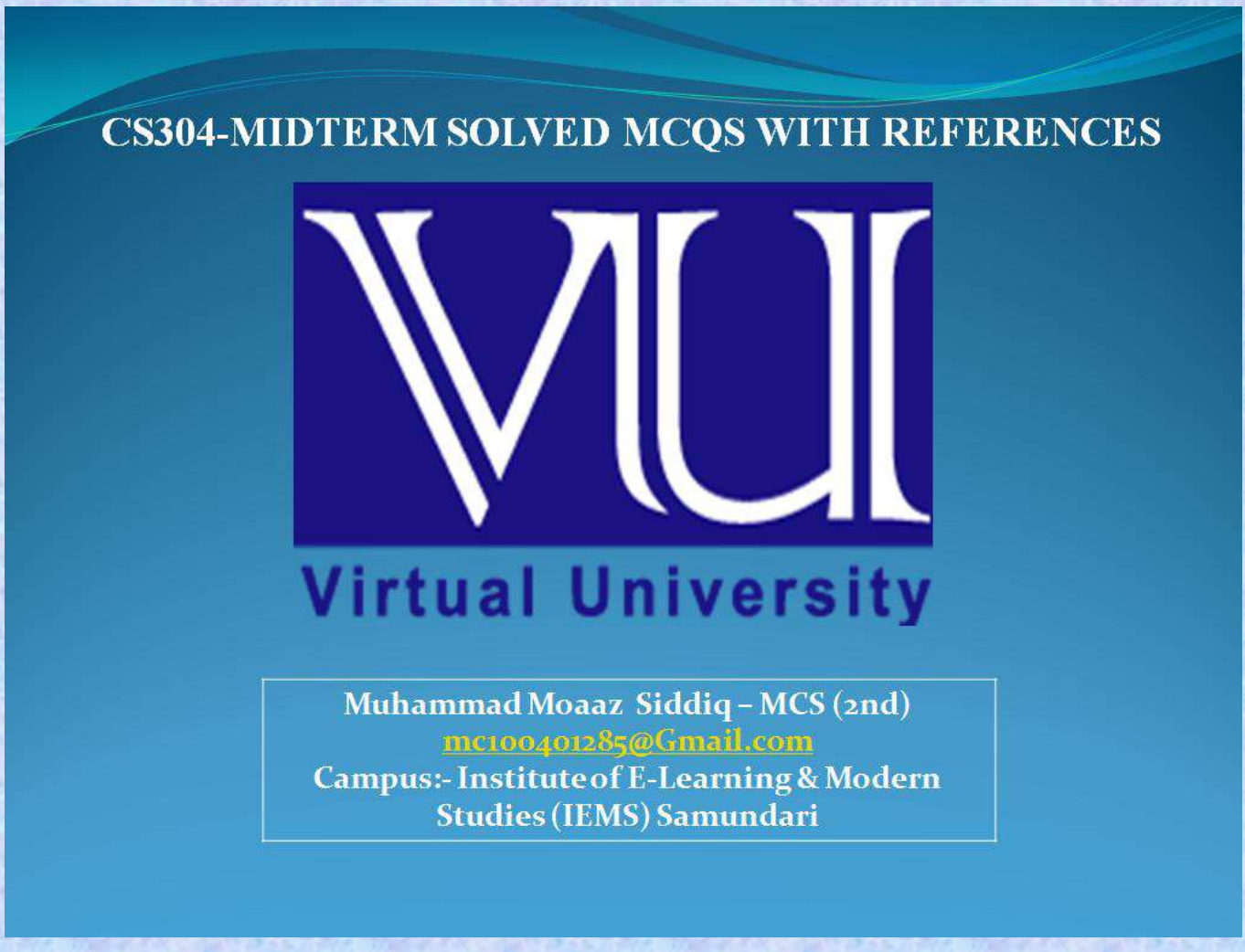
Four

Question No:206 (Marks:1) Vu-Topper RM

Behaviors of a type of objects are represented through

Operations

Proude To Be Virtulian MOAAZ SIDDIQ	**CS304- Object Oriented Programming** LATEST SOLVED MCQS FROM MIDTERM PAPERS	**Nov 07,2011**
MC100401285	Moaaz.pk@gmail.com MC100401285@gmail.com	**Latest Mcqs**



Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
Which part of an object exhibits its state?

- **Data (Page 23)**
- Operations
- Any public part
- Any private part

Question No: 2 (Marks: 1) - Please choose one
Inheritance is a way to

- organize data.
- pass arguments to objects of classes.
- **add features to existing classes without rewriting them. (Page 27)**
- improve data-hiding and encapsulation.

Question No: 3 (Marks: 1) - Please choose one
Suppose you have been given the following design,
"A person has a name, age, address and sex. You are designing a class to represent a type of person called a patient. This kind of person may be given a diagnosis, have a spouse and may be alive". Given that the person class has already been created, what of the following would be appropriate to include when you design the patient class?

- **registration date and diagnosis** Click here for Detail
- age and sex
- sex and diagnosis
- diagnosis and age

Question No: 4 (Marks: 1) - Please choose one
What problem(s) may occur when we copy objects without using deep copy constructor?

- Dangling pointer
- Memory Leakage
- **All of the given (Page 147)**
- System crash

Question No: 5 (Marks: 1) - Please choose one
this pointers are not accessible for static member functions.

- **True (Page 114)**
- False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 6 (Marks: 1) - Please choose one
A static member function cannot be declared.

- ► Static
- ► Implicit
- ► Explicit
- ► **Virtual** **Click here for detail**

Question No: 7 (Marks: 1) - Please choose one
_____ remain in memory even when all objects of a class have been destroyed.

- ► **Static variables (Page 111)**
- ► Instance variable
- ► Primitive variables
- ► None of given

Question No: 8 (Marks: 1) - Please choose one
Friend functions are _____ functions of a class.

- ► None of given
- ► object member
- ► **non-member (Page 136)**
- ► data member

Question No: 9 (Marks: 1) - Please choose one
_____, which means if A declares B as its friend it does NOT mean that A can access private data of B. It only means that B can access all data of A.

- ► **Friendship is one way only**
- ► Friendship is two way only
- ► NO Friendship between classes
- ► Any kind of friendship

Question No: 10 (Marks: 1) - Please choose one
The statement objA=objB; will cause a compiler error if the objects are of different classes.

- ► **True**
- ► False

Question No: 11 (Marks: 1) - Please choose one
Identify which of the following overloaded operator function's declaration is appropriate for the given call?

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Rational_number_1 + 2.325
Where Rational_number_1 is an object of user defined class Rational_number.

- Rational_number operator+(Rational_number & obj);
- **Rational_number operator+(double& obj);** **(Page 145)**
- Rational_number operator+(Rational_number &obj, double& num);
- operator+(double& obj);

Question No: 12 (Marks: 1) - Please choose one
Which operator can not be overloaded?

- The relation operator (>=)
- Assignment operator (=)
- Script operator ([])
- **Conditional operator (? :) (Page 141)**

Question No: 13 (Marks: 1) - Please choose one
To convert from a user-defined class to a basic type, you would most likely use

- a built-in conversion operator.
- a one-argument constructor.
- an overloaded = operator.
- **a conversion operator that's a member of the class.**

Question No: 14 (Marks: 1) - Please choose one
The technique in which we visualize our programming problems according to real life's problems is called
---

- structured programming
- **object oriented Programming** **(Page 9)**
- procedural programming
- non of the given

Question No: 15 (Marks: 1) - Please choose one
In object orientated programming, a class of objects cans _____ properties from another class of objects

- Utilize
- Borrow
- **Inherit** **Click here for detail**
- Adopt

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 16 (Marks: 1) - Please choose one
A C++ class is similar to ----------

- ► **Structure** **Click here for detial**
- ► Header File
- ► Library File
- ► None of the given

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
What problem(s) may occur when we copy objects without using deep copy constructor?

- ► Dangling pointer
- ► Memory Leakage
- ► **All of the given** **(Page 147)**
- ► System crash

Question No: 2 (Marks: 1) - Please choose one
Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

- ► The automatic assignment operator is used
- ► **The copy constructor is used** **Click here for detial**
- ► Compiler error
- ► Run-time error

Question No: 3 (Marks: 1) - Please choose one
a'A static member function can be called, even when a class is not _____.

- ► Declared
- ► Define
- ► **Instantiated** **C++ How to Program, Fifth Edition(Page 871)**
- ► Called

Question No: 4 (Marks: 1) - Please choose one
Identify which of the following overloaded operator function's declaration is appropriate for the given call?
Rational_number_1 + 2.325
Where Rational_number_1 is an object of user defined class Rational_number.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► Rational_number operator+(Rational_number & obj);
► **Rational_number operator+(double& obj); (Page 145)**
► Rational_number operator+(Rational_number &obj, double& num);
► operator+(double& obj);

Question No: 5 (Marks: 1) - Please choose one
_____ provide the facility to access the data member.

► **accesser function (Page 68)**
► private function
► inline function
► None of the given

Question No: 6 (Marks: 1) - Please choose one
Constant objects cannot change their state,

► **True (Page 105)**
► False

Question No: 7 (Marks: 1) - Please choose one
The _____ relationship indicates that an object contains other objects.

► **None of given (Page 53)**
► 'has-a'
► 'is-a'
► 'be-

Question No: 8 (Marks: 1) - Please choose one
Which one of the following features of OOP is used to derive a class from another?

► Encapsulation
► Polymorphism
► Data hiding
► **Inheritance (Page 25)**

Question No: 9 (Marks: 1) - Please choose one
_____ is a relationship

► **Inheritance (Page 25)**
► Polymarphism
► abstraction
► encapsulation

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 10 (Marks: 1) - Please choose one
_____satisfy the condition of polymorphism

- ► Carbon
- ► Diamond
- ► Coal
- ► **all of the given (Page 56)**

Question No: 11 (Marks: 1) - Please choose one
A generalization-specialization relation between classes are implemented using

- ► data hiding
- ► friend classes
- ► encapsulation
- ► **inheritance (Page 49)**

Question No: 12 (Marks: 1) - Please choose one
The >= operator can be overloaded.

- ► **True (Page 140)**
- ► False

Question No: 13 (Marks: 1) - Please choose one
In order to free the memory occupied by the object, we use ----------

- ► Constructor
- ► **Destructor (Page 92)**
- ► Shallow Copy
- ► Deep Copy

Question No: 14 (Marks: 1) - Please choose one
Which of the following is not an example of multiple inheritances?----------

- ► Mermaid
- ► **Woman (Lecture No.5)**
- ► None of the given
- ► Amphibious Vehicle

Question No: 15 (Marks: 1) - Please choose one
Static variable can be initialized more than once.

- ► **True** Click here for detail
- ► False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 16 (Marks: 1) - Please choose one
A generic class showing all the common attributes and a behavior of other classes represents a very important feature in oop called --------

- **Inheritance (Page 26)**
- Encapsulation
- Polymarphism
- Abstraction

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
We can get only one unique value which can be used by all the objects of that class by the use of,

- **static variables** Click here for detail
- dynamic variables
- instance variables
- data members

Question No: 2 (Marks: 1) - Please choose one
A member function having the same name as that of a class and a ~ sign with it is called,

- Constructor
- Getter
- Setter
- **Destructor (Page 92)**

Question No: 3 (Marks: 1) - Please choose one
Using encapsulation we can achieve

- **Information hiding (Page 16)**
- Least interdependencies among modules
- Implementation independence
- All of given options

Question No: 4 (Marks: 1) - Please choose one
Inheritance is a way to

- **make general classes into more specific classes. (Page 27)**
- pass arguments to objects of classes.
- improve data hiding and encapsulation.
- providing class growth through natural selection.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 5 (Marks: 1) - Please choose one
Static variable can be initialized more than once.
True
False (Page 107)

Question No: 6 (Marks: 1) - Please choose one
For classes with common behavior, you can save effort by placing the common behavior in a _____.

►Derived Class
►**Base class (Page 29)**
►Deprived Class
►Named class

Question No: 7 (Marks: 1) - Please choose one
Which of the following are an advantage of OOP?

►OOP makes it easy to re-use the code
►It provides an ability to create one user defined data type by extending the other
►It provides the facility of defining Abstract data types through which real world entities can be defined better.
►**All of the given options** **Click here for detail**

Question No: 8 (Marks: 1) - Please choose one
The >= operator can be overloaded.

►**True (Page 140)**
►False

Question No: 9 (Marks: 1) - Please choose one
A static member function cannot be declared.

►Static
►Implicit
►Explicit
►**Virtual** **Click here for detail**

Question No: 10 (Marks: 1) - Please choose one
Static variables act like a global variable in the context or scope of the class.

►**True** **click here for detail**
►False

Question No: 11 (Marks: 1) - Please choose one
The compiler won't object if you overload the * operator to perform division.

►**True** **Click here for detail**
►False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 12 (Marks: 1) - Please choose one
We can use "this" pointer in the constructor in the body and even in the initialization list of any class if we are careful,

- ►TRUE
- ►**False**

Question No: 13 (Marks: 1) - Please choose one
A C++ class is similar to ----------

- ►**Structure** **Click here for detail**
- ►Header File
- ►Library File
- ►None of the given

Question No: 14 (Marks: 1) - Please choose one
Which operator can not be overloaded?

- ►The relation operator (>=)
- ►Assignment operator (=)
- ►Script operator ([])
- ►**Conditional operator (? :) (Page 141)**

Question No: 15 (Marks: 1) - Please choose one
An overloaded operator always requires one less argument than its number of operands.

- ►**True (Page 896)**
- ►False

Question No: 16 (Marks: 1) - Please choose one
A generalization-specialization relation between classes are implemented using

- ►data hiding
- ►friend classes
- ►encapsulation
- ►**inheritance (Page 49)**

Question No: 17 (Marks: 1) - Please choose one
In OOP a class is an example of _____

- ►Data Type
- ►Abstract Type
- ►**User Defined Type (Page 66)**
- ►None of the given

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 18 (Marks: 1) - Please choose one
A class can be identified from a statement by ----------

►**Noun (Page 58)**
►Pronoun
►Verb
►Adverb

Question No: 19 (Marks: 1) - Please choose one
The members of a class that can be accessed without creating the object of the class is called

►Private member
►Data Member
►**Public Member (How to Program page 983)**
►Static

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
Suppose there is an object of type Person, which of the following can be considered as one of its attributes

► Name
► Age
► Work()
► **Both Name and Age (page 13)**

Question No: 2 (Marks: 1) - Please choose one
What a derived class can add?

► New data members
► New member functions and New friend functions
► New constructors and destructor
► **All of given** **Click here for Detail**

Question No: 3 (Marks: 1) - Please choose one
_____ is/are used to access information hidden within an object?

► Interface
► Private data members
► **Private member functions (Page 69)**
► Both public and private members

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 4 (Marks: 1) - Please choose one
this pointers are not accessible for static member functions.

► **True (Page 14)**
► False

Question No: 5 (Marks: 1) - Please choose one
A static member function cannot be declared.

► Static
► Implicit
► Explicit
► **Virtual** **Click here for detail**

Question No: 6 (Marks: 1) - Please choose one
C++ compiler does not allow to dynamically allocate memory for objects

► **False** **Click here for detail**
► True

Question No: 7 (Marks: 1) - Please choose one
Given the following class
class Base{
int Age=33;
}
How you can improve above class with respect to accessing the field Age?

► Define the variable Age as private
► Define the variable Age as protected
► **Define the variable Age as private and create a get method that returns it and a set method that updates it**
Click here for detail
► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 8 (Marks: 1) - Please choose one
Friend class and friend function can be used as an alternate to each other

► **True** **Click here for detail**
► False

Question No: 9 (Marks: 1) - Please choose one
Which of the following operators always takes no argument if overloaded?

► /
► -

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► +
► ++

Question No: 10 (Marks: 1) - Please choose one
Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

► The automatic assignment operator is used
► **The copy constructor is used** **Click here for detail**
► Compiler error
► Run-time error

Question No: 11 (Marks: 1) - Please choose one
Assume a class C with objects obj1, obj2, and obj3. For the statement obj3 = obj1 - obj2 to work correctly, if the overloaded - operator must

► take two arguments.
► **return a value** **Click here for detail**
► create a named temporary object.
► take four arguments

Question No: 12 (Marks: 1) - Please choose one
Which operator can not be overloaded?

► The relation operator (>=)
► Assignment operator (=)
► Script operator ([])
► **Conditional operator (? :)** **(Page 141)**

Question No: 13 (Marks: 1) - Please choose one
We achieve independence of internal implementation from its external interface through----------

► Encapsulation
► Information Hiding
► **Abstraction** **Click here for detial**
► both encapsulation and information hiding

Question No: 14 (Marks: 1) - Please choose one
Which one of the following is not an object association?

► Simple Assocation
► **Inheritance** **(Page 49)**
► Aggregation
► Composition

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Keeping in view the principle of abstraction, which of the above information the company needs to save as employee's record?

- ► 2, 4
- ► 1, 3, 5
- ► **1, 2, 3 (Lecture No.3)**
- ► 1, 2, 3, 4

Question No: 4 (Marks: 1) - Please choose one
What a derived class can add?

- ► New data members
- ► New member functions and New friend functions
- ► New constructors and destructor
- ► **All of given** **Click here for Detail**

Question No: 5 (Marks: 1) - Please choose one
The concept of derived classes is involved in,

- ► **inheritance (Page 25)**
- ► encapsulation
- ► structure
- ► array

Question No: 6 (Marks: 1) - Please choose one
Your chief Software designer has shown you a sketch of the new Computer parts system she is about to create. At the top of the hierarchy is a Class called Computer and under this are two child classes. One is called LinuxPC and one is called WindowsPC. The main difference between the two is that one runs the Linux operating System and the other runs the Windows System (of course another difference is that one needs constant re-booting and the other runs reliably). Under the WindowsPC are two Sub classes one called Server and one Called Workstation. How might you appraise your designers work?

- ► Give the goahead for further design using the current scheme
- ► **Ask for a re-design of the hierarchy with changing the Operating System to a field rather than Class type** **Click here for detail**
- ► Ask for the option of WindowsPC to be removed as it will soon be obsolete
- ► Change the hierarchy to remove the need for the superfluous Computer Class.

Question No: 7 (Marks: 1) - Please choose one
Consider the code below,
class Fred {
public:
Fred();

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

```
...
};
int main()
{
Fred a[10];
Fred* p = new Fred[10];
...
}
```

Select the best option,

► Fred a[10]; calls the default constructor 09 times
Fred* p = new Fred[10]; calls the default constructor 10 times
► Produce an error
► **Fred a[10]; calls the default constructor 11 times**
Fred* p = new Fred[10]; calls the default constructor 11 times
► Fred a[10]; calls the default constructor 10 times
Fred* p = new Fred[10]; calls the default constructor 10 times

Question No: 8 (Marks: 1) - Please choose one
Which construct is the source for the creation of an object?

► Destructor of the class
► **New operator** **Click here for detial**
► Delete operator
► Constructor of the class

Question No: 9 (Marks: 1) - Please choose one
this pointers are not accessible for static member functions.
► **True (Page 14)**
► False

Question No: 10 (Marks: 1) - Please choose one
When a variable is define as static in a class then all object of this class,

► Have different copies of this variable
► **Have same copy of this variable** **Click here for detail**
► Can not access this variable
► None of given

Question No: 11 (Marks: 1) - Please choose one
_____ remain in memory even when all objects of a class have been destroyed.
► **Static variables (Page 111)**
► Instance variable
► Primitive variables
► None of given

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 12 (Marks: 1) - Please choose one
Given the following class
class Base{
int Age=33;
}
How you can improve above class with respect to accessing the field Age?

- ► Define the variable Age as private
- ► Define the variable Age as protected
- ► **Define the variable Age as private and create a get method that returns it and a set method that updates it** **Click here for detail**
- ► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 13 (Marks: 1) - Please choose one
The life of sub object is not dependant on the life of master class in _____.

- ► Composition
- ► **Aggregation (Page 134)**
- ► Separation
- ► None of the given

Question No: 14 (Marks: 1) - Please choose one
Which one is not keyword in C++?

- ► operator
- ► **B_op** **Click here for detail**
- ► const
- ► None of given

Question No: 15 (Marks: 1) - Please choose one
The >= operator can be overloaded.

- ► **True (Page 140)**
- ► False

Question No: 16 (Marks: 1) - Please choose one
Identify which of the following overloaded operator function's declaration is appropriate for the given call?
Rational_number_1 + 2.325
Where Rational_number_1 is an object of user defined class Rational_number.

- ► Rational_number operator+(Rational_number & obj);
- ► **Rational_number operator+(double& obj);**
- ► Rational_number operator+(Rational_number &obj, double& num);
- ► operator+(double& obj);

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

MIDTERM EXAMINATION
Spring 2009

Question No: 1 (Marks: 1) - Please choose one
A template provides a convenient way to make a family of

- ► variables and data members
- ► **functions and classes** **Click here for detail**
- ► classes and exceptions
- ► programs and algorithms

Question No: 2 (Marks: 1) - Please choose one
Which one of the following terms must relate to polymorphism?

- ► Static allocation
- ► Static typing
- ► **Dynamic binding (How to program , page 1103)** **Click here for more detail**
- ► Dynamic allocation

Question No: 3 (Marks: 1) - Please choose one
What is true about function templates?

- ► The compiler generates only one copy of the function template
- ► **The compiler generates a copy of function respective to each type of data (Page 257)**
- ► The compiler can only generate copy for the int type data
- ► None of the given.

Question No: 4 (Marks: 1) - Please choose one
Which of the following is the best approach if it is required to have more than one functions having exactly same functionality and implemented on different data types?

- ► **Templates** **Click here for detial**
- ► Overloading
- ► Data hiding
- ► Encapsulation

Question No: 5 (Marks: 1) - Please choose one
template <>
class Vector<char*> { }

This is an example of partial specialization.

- ► **True (Page 281)**
- ► False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 6 (Marks: 1) - Please choose one
Classes like TwoDimensionalShape and ThreeDimensionalShape would normally be concrete, while classes like Sphere and Cube would normally be abstract.

- ► True
- ► **False** **Click here for Detail**

Question No: 7 (Marks: 1) - Please choose one
A non-virtual member function is defined in a base class and overridden in a derived class; if that function is called through a base-class pointer to a derived class object, the derived-class version is used.

- ► True
- ► **False** **Click here for Detial**

Question No: 8 (Marks: 1) - Please choose one
Assume a class Derv that is privately derived from class Base. An object of class Derv located in main() can access

- ► **public members of Derv.** **Click here for Detail**
- ► **protected members of Derv.**
- ► private members of Derv.
- ► protected members of Base.

Question No: 9 (Marks: 1) - Please choose one
In order to define a class template, the first line of definition must be:

- ► **template <typename T> (Page 281)**
- ► typename <template T>
- ► Template Class <ClassName>
- ► Class <Template T>

Question No: 10 (Marks: 1) - Please choose one
If there is a pointer p to objects of a base class, and it contains the address of an object of a derived class, and both classes contain a nonvirtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the _____ class to be executed.

- ► Base
- ► Derived
- ► Abstract
- ► **virtual** **(not sure)**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 11 (Marks: 1) - Please choose one
When the base class and the derived class have a member function with the same name, you must be more specific which function you want to call (using _____).

- ► scope resolution operator
- ► dot operator
- ► null operator
- ► **Operator overloading (Page 204) Click here for Detail**

Question No: 12 (Marks: 1) - Please choose one
Non Template Friend functions of a class are friends of _____instance/s of that class.

- ► **All Click here for detail**
- ► One specific
- ► All instances of one date type
- ► None of the given options

Question No: 13 (Marks: 1) - Please choose one
The find() algorithm

- ► finds matching sequences of elements in two containers.
- ► finds a container that matches a specified container.
- ► **takes iterators as its first two arguments. Click here for Detail**
- ► takes container elements as its first two arguments.

Question No: 14 (Marks: 1) - Please choose one
If you define a vector v with the default constructor, and define another vector w with a one-argument constructor to a size of 11, and insert 3 elements into each of these vectors with push_back(), then the size() member function will return _____ for v and _____ for w.

- ► 11 for v and 3 for w.
- ► 0 for v and 0 for w.
- ► 0 for v and 3 for w.
- ► **3 for v and 11 for w.**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 15 (Marks: 1) - Please choose one
Which of the following may not be an integral part of an object?

- ► State
- ► Behavior
- ► Protected data members
- ► **All of given (not sure)**

Question No: 16 (Marks: 1) - Please choose one
Which is not the Advantage of inheritance?

- ► providing class growth through natural selection.
- ► facilitating class libraries.
- ► **avoiding the rewriting of code.** **Click here for Detail**
- ► providing a useful conceptual framework.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

CS304 QUIZ 1 lecture 1-10

Midterm Quiz

ORANGE MONKEY TEAM

1. **Which of the following concept is more close to encapsulation**

   Information hiding…..confirm

- Information hiding is achieved through encapsulation.
- Encapsulation and Information Hiding are related to each other.

2. **_____ is represented by a line with unfilled diamond head**

   Aggregation……confirm

Aggregation

An object may contain a collection (aggregate) of other objects, the relationship between the container and the contained object is called aggregation, Aggregation is represented by a line with unfilled-diamond head towards the container

Example – Aggregation

3. **_____is represented by a line with a filled diamond head**

   Composition…..confirm

Composition

An object may be composed of other smaller objects, the relationship between the "part" objects and the "whole" object is known as Composition, Composition is represented by a line with a filled-diamond head towards the composer object

Example – Composition of Ali

ORANGE MONKEY TEAM

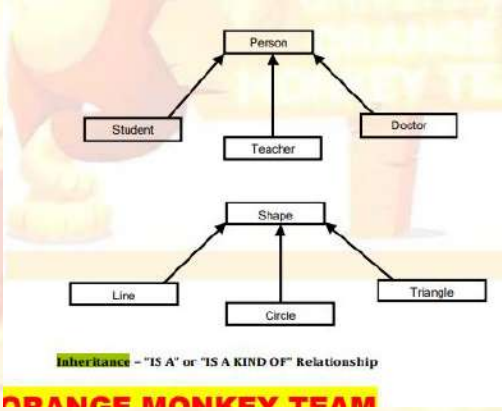
4. **if some of objects exhibit identical characteristics then they belong to**

Different classes

If some of objects exhibit identical characteristics, then they belong to:
Different classes
Multiple classes
Same class
None of the given

5. **Which of the following relationship shows the inheritance?**

Doctor is a person…..confirm



6. **Constructor have ____ return type**

No….confirm

https://www.tutorialspoint.com › Does-a-constructor-ha...
Does a constructor have a return type in Java? - Tutorialspoint
30-Jul-2019 — No, **constructor does not have any return type in Java**. Constructor looks like

7. **If class b inherits from class a which of the following is child class.**

Class A…..confirm

Inheritance in Classes

If a class B inherits from class A then it contains all the characteristics (information structure and behaviour) of class A

ORANGE MONKEY TEAM

8. **Which of the following can be the attribute of an object ali**

address…..confirm

73. Which of the following can be the attribute of an object "Ali"?

1. Eat
2. Walk
3. Sleep
4. Address

9. **A car has wheel. What is the relationship between car and wheel**

Composition

10. **In order to make any function constant, keyword const is placed _____ the parameter list.**

After

11. **Which of the following is Not true about inline function?**

Inline function does not substitute the code within the function definition on a function call….confirm

Inline Functions

- Inline functions is a way used by compilers to improve efficiency of the program, when functions are declared inline normal process of function calling (using stack) is not followed instead function code is added by compiler at all points where these functions have been called. Basic concept behind inline functions is that they are functions in our code but in compiler generated files these functions code is added by compiler at all places where they were called in the code.
- Normally small size functions that need to be called many times during program execution are declared inline. Inline functions decrease the code execution time because program in their case doesn't involve function call overhead.
- Keyword 'inline' is used to request compiler to make a function inline.
- However using inline keyword with function doesn't guarantee that function will definitely in inlined, it depends on the compiler if it finds it can make function inline it does so otherwise it ignores the keyword inline and treat the function as normal function.

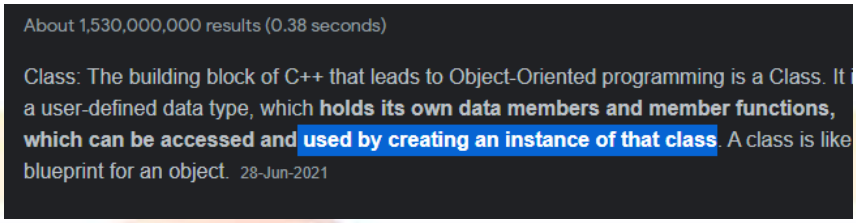
12. **Normal form of class representation includes _____.**

Class Name, Attributes, Operation…….confirm

ORANGE MONKEY TEAM

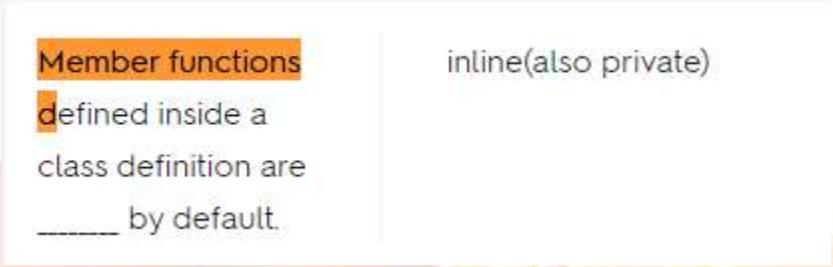
13. Which of the following is used to create user-defined data type in OOP?

Classes……confirm



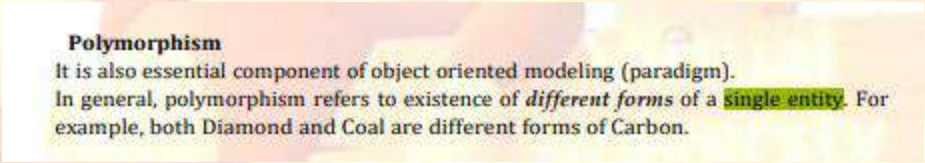
14. Member functions defined inside a class are ____ by default

Private(inline)…..confirm



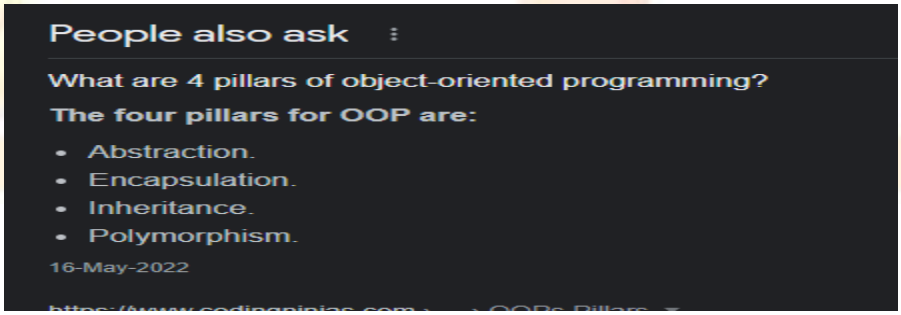
15. In which of the following different forms of a single entity exist?

Polymophism……confirm



16. Which of the following is not a pillar of object oriented programming?

Normalization….confirm



ORANGE MONKEY TEAM

17. A/an _____ class represent the abstract concept.

Abstract class….confrim

18. Which of the following statements defines destructor for class student?

~Student(){//-}…..confirm

19. Constructor is used to _____ the objects of a class

Initialize…..confirm

20. Through _____ objects communicate with outer world.

Interface….confirm

21. There is/are _____ way(s) to create user defined types for objects in c++

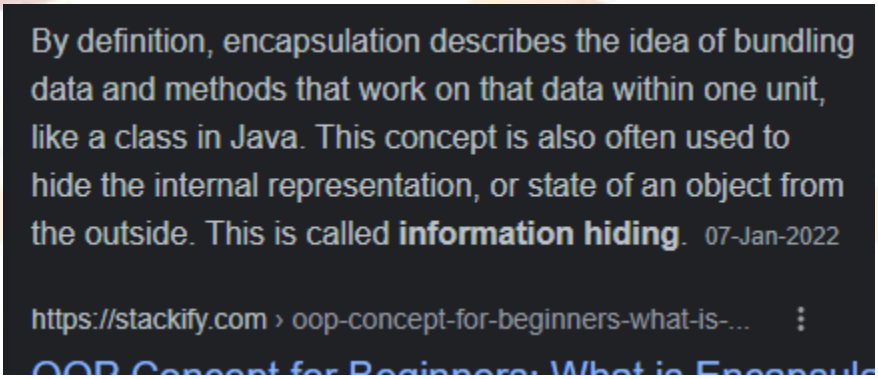
Two…..confirm

22. We can allocate dynamic memory in C++ through _____ operator

New operator…..confirm

23. Object hides its information from other this concept is known as _____.

Information hiding…..confirm



ORANGE MONKEY TEAM

24. The concept of derived class is involved in _____.

Inheritance

25. Enclosing all the characteristics of an object within the object itself is described as_____.

Encapsulation….confirm

26. Generalization is implemented through _____.

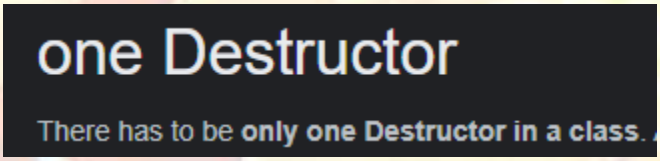
Inheritance…..confirm

27. IF we extend our object oriented model and the rest of model is not affected, this feature is called_____.

Reusability

28. A class can have only ___ destructor

One……confirm



29. Data member of a class "Student" cannot be _____

Char*name;

30. In OOP, we can achieve reusability through _____.

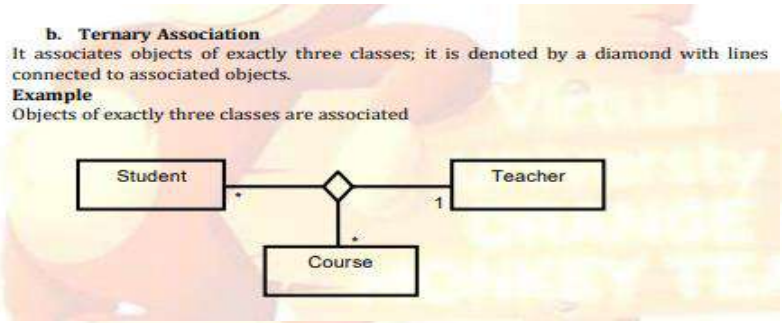
Inheritance…..confirm

31. Which of the following is not an example of multiple inheritance?

Car…..confirm

32. Student registers course and teacher teaches the course is an example of _____.

Ternary association……..confirm

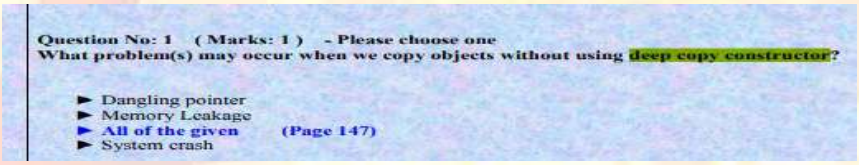


33. Which of the following method allows to reuse the characteristics of more than one parent class?

Multiple inheritance.…confirm

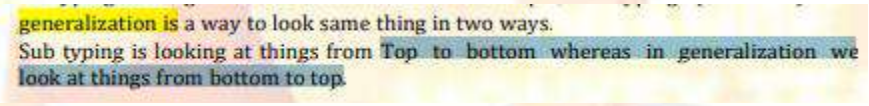
34. If our class involves dynamic memory and we do not write deep copy constructor for this class _____ problem(s) can occur.

All options…..confirm



35. Generalization is _____ approach

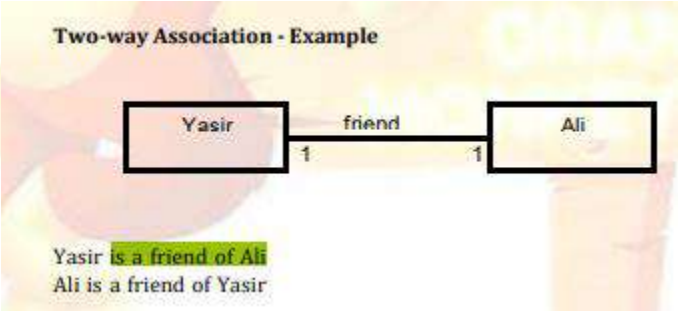
(Top-bottom, bottom to top)….both are correct…..confirm



ORANGE MONKEY TEAM

36. Statement Yasir is a friend of Ali and Ali is a friend of yasir is an example of _____.

Two way association…….confirm



37. In which of the following different forms of a single entity exist?

Polymorphism……confirm

In general, polymorphism refers to existence of *different forms* of a single entity.

38. Which of the following is True about Accessor functions?

All option……confirm

Accessor functions are also used to access private data of the object, we provide accessor functions to get and set private data members of the class.

39. Identify the common attribute of the given scenario:
Circle has color, vertices, and radius. Triangle has color, vertices and angle

Color, vertices……confirm

Common attributes
Color vertices
Common behaviour
Set Color, Move

40. In OOP, inheritance is shown in a class diagram by using _____.

Filled arrow….confirm

66	inheritance is shown by	filled arrow

ORANGE MONKEY TEAM

41. Which relationship exists between cat and animal class?

Association……not sure

42. Which of the following is not true about destructor?

Used to free memory allocated through static allocation….confirm

Destructor

1. Destructor is used to free memory that is allocated through dynamic allocation. We have to free memory allocated using new operator by over self in destructor otherwise it remain occupied even after our program ends.
2. Destructor is used to perform house keeping operations.
3. Destructor is a function with the same name as that of class, but preceded with a tilde '~'

43. _____ are best candidates to become objects in problem description.

Nouns….confirm

- Generally when we have given a certain problem description, nouns in that problem description are candidates for becoming objects of our system.

44. Abstraction hides the _____details

Irrelevant …..confirm

Abstraction – Advantages

Abstraction has following major advantages,

1. It helps us understanding and solving a problem using object oriented approach as it hides extra irrelevant details of objects.

45. _____ is the reverse process of generalization

Specialization…confirm

About 8,750,000 results (0.59 seconds)

Specialization is the reverse process of Generalization means creating new subclasses from an existing class. 05-Sept-2022

https://www.dotnettricks.com › Oops

Difference Between Generalization and Specialization

ORANGE MONKEY TEAM

CS304-Handouts

ORANGE MONKEY TEAM

LECTURE NO.01 ..... **8**

- INTRODUCTION 8
- WHAT IS A MODEL? 10
- OO MODELS 11
- OBJECT-ORIENTATION - ADVANTAGES 12
- WHAT IS AN OBJECT? 12
- TANGIBLE AND INTANGIBLE OBJECTS 13
- SUMMARY 14

LECTURE NO.02 ..... **15**

- INFORMATION HIDING 15
- ENCAPSULATION 16
- INTERFACE 17
- IMPLEMENTATION 18
- SEPARATION OF INTERFACE & IMPLEMENTATION 19
- MESSAGES 19
- SUMMARY 19

LECTURE NO.03 ..... **21**

- ABSTRACTION 21
- CLASSES 23
- INHERITANCE 25

LECTURE NO.04 ..... **29**

- CONCEPTS RELATED WITH INHERITANCE 29
- GENERALIZATION 29
- SUB-TYPING (EXTENSION) 31
- SPECIALIZATION (RESTRICTION) 32
- OVERRIDING 34
- ABSTRACT CLASSES 36
- CONCRETE CLASSES 38

LECTURE NO.05 ..... **40**

- SIMPLE ASSOCIATION 50
- COMPOSITION 52
- AGGREGATION 54

LECTURE NO.06 ..... **55**

- CLASS COMPATIBILITY 55
- POLYMORPHISM 56
- POLYMORPHISM IN OO MODEL 56
- POLYMORPHISM – ADVANTAGES 57
- OBJECT-ORIENTED MODELING AN EXAMPLE 57

LECTURE NO.07 ..... **65**

- CLASS 65
- TYPE IN C++ 65
- ABSTRACTION 66
- DEFINING A NEW USER DEFINED TYPE 66
- OBJECT AND CLASS 68
- ACCESSING MEMBERS 68
- ACCESS SPECIFIERS 69

LECTURE NO.08 ..... **72**

ORANGE MONKEY TEAM

MEMBER FUNCTIONS 72
DEFINING MEMBER FUNCTIONS 72
INLINE FUNCTIONS 73
CONSTRUCTOR 75
CONSTRUCTOR PROPERTIES 75
DEFAULT CONSTRUCTOR 76
CONSTRUCTOR OVERLOADING 77
CONSTRUCTOR OVERLOADING 78
COPY CONSTRUCTOR 79
SHALLOW COPY 81
DEEP COPY 82

LECTURE NO.09 84

SHALLOW COPY 85
DEEP COPY 90
IMPORTANT POINTS ABOUT COPY CONSTRUCTOR 94
DESTRUCTOR 94
ACCESSOR FUNCTIONS 95
THIS POINTER 96

LECTURE NO.10 99

USES OF THIS POINTER 99
SEPARATION OF INTERFACE AND IMPLEMENTATION 99
COMPLEX NUMBER 100
CONST MEMBER FUNCTIONS 102
THIS POINTER AND CONST MEMBER FUNCTION 104

LECTURE NO.11 105

USAGE EXAMPLE OF CONSTANT MEMBER FUNCTIONS 105
DIFFERENCE BETWEEN INITIALIZATION AND ASSIGNMENT 106
MEMBER INITIALIZER LIST 106
CONST OBJECTS 107
STATIC VARIABLES 109

LECTURE NO.12 112

ACCESSING STATIC DATA MEMBER 113
LIFE OF STATIC DATA MEMBER 114
STATIC MEMBER FUNCTION 115
THIS POINTER AND STATIC MEMBER FUNCTIONS 116
GLOBAL VARIABLE VS. STATIC MEMBERS 116
ARRAY OF OBJECTS 116

LECTURE NO.13 118

POINTER TO OBJECTS 118
BREAKUP OF NEW OPERATION 119
CASE STUDY 119
COMPLETE CODE OF DATE CLASS 121

LECTURE NO.14 124

14.1. COMPOSITION 124

LECTURE NO.15 132

AGGREGATION 135
FRIEND FUNCTIONS 138

LECTURE NO.16 142

ORANGE MONKEY TEAM

16.1. Operator overloading 142
LECTURE NO.17 149
17.1. Overloading Assignment operator 151
LECTURE NO.18 155
Self assignment problem 155
Other Binary operators 156
Friend Functions and Operator overloading 157
LECTURE NO.19 158
Stream Insertion operator 158
Stream Extraction operator 158
Overloading Stream Insertion Operator 159
Overloading Stream Extraction Operator 160
Other Binary operators 161
LECTURE NO.20 163
Subscript [] Operator 164
Overloading Subscript [] Operator 164
Overloading Function () operator 165
Function Operator performing Sub String operation, ... 165
Unary Operators 166
LECTURE NO.21 168
Behavior of ++ and -- for pre-defined types 168
Post-increment operator 169
Type Conversion 170
User Defined types 173
Drawbacks of Type Conversion Operator 174
LECTURE NO.22 175
Practical implementation of Inheritance in c++ 175
Inheritance in Classes 175
UML Notation 175
Inheritance in C++ 175
"IS A" Relationship 176

ORANGE MONKEY TEAM

Lecture No.01

Introduction

Course Objective:
Objective of this course is to make students familiar with the concepts of object oriented programming. These concepts will be reinforced by their implementation in C++.

Course Contents:
The main topics that we will study in the 45 lectures of this course are given below,

- Object Orientation
- Objects and Classes
- Overloading
- Inheritance
- Polymorphism
- Generic Programming
- Exception Handling
- Introduction to Design Patterns

Recommended Text Book:

C++ How to Program (Deitel & Deitel **)**

Reference Books:

1. **Object-Oriented Software Engineering**
   By Jacobson, Christerson, Jonsson, Overgaard
   (For object oriented programming introductory concepts)
2. **The C++ Programming Language**
   By Bjarne Stroustrup
   (For better c++ understanding)

ORANGE MONKEY TEAM

Object-Orientation (OO)

What is Object-Orientation?

It is a technique in which we visualize our programming problems in the form of objects and their interactions as happens in real life.

Examples:
We have different objects around us in our real life that interact with each other to perform different operations for example,

A Person

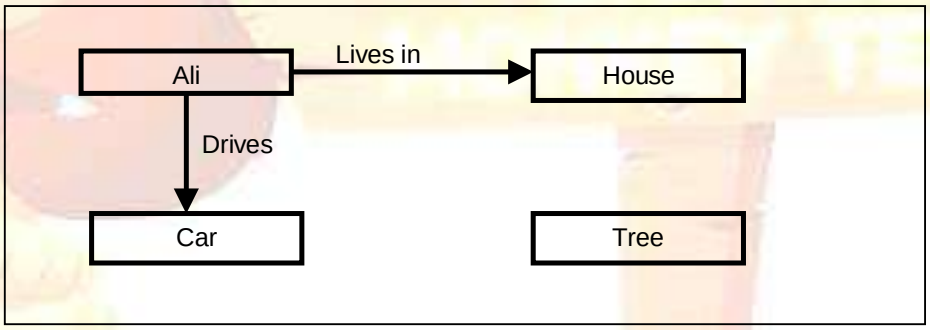
A House

A Tree

A Car

Different Objects

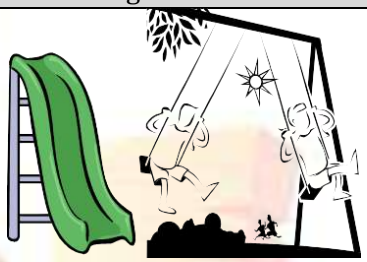
These objects interact with each other to perform different operations,



Take another example of a **School**; the objects in a school are **student**, **teacher**, **books, pen** ,**school bag**, **classroom, parents, playground and so on… ,**

ORANGE MONKEY TEAM

Objects in a School

Teacher	Student	School Bag
		
Book	Pen	Playground
Parents	Classroom	Library

Suppose we want to develop a fee collection system for a school for this we will need to find out related objects and their interactions as happens in real life.
In this way we can say that **object orientation** makes it easier for us to solve our real world problems by thinking solution of the problem in terms of real world objects.

So we can say that in our daily life everything can be taken as an object that behaves in a certain way and has certain attributes.

In object orientation we move our concentration to objects in contrast to procedural paradigm in which we simply write our code in functions and call them in our main program.

What is a Model?

A model is an abstraction of something real or conceptual.
We need models to understand an aspect of reality.

ORANGE MONKEY TEAM

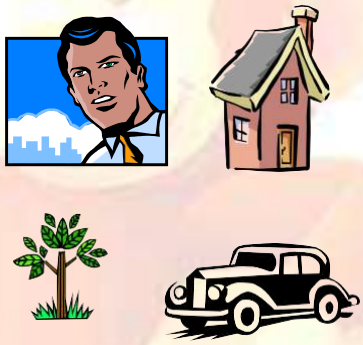
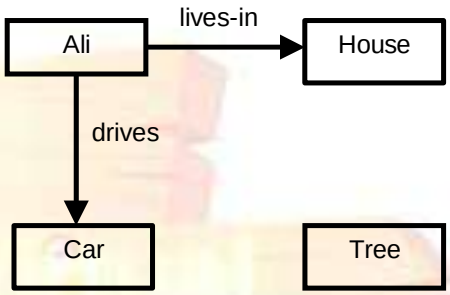
Model Examples

Highway maps
Architectural models
Mechanical models

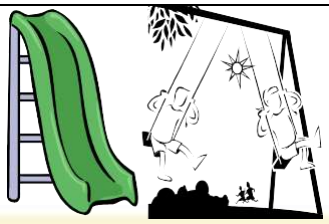
OO Models:

In the context of programming models are used to understand the problem before starting developing it.
We make Object Oriented models showing several interacting objects to understand a system given to us for implementation.

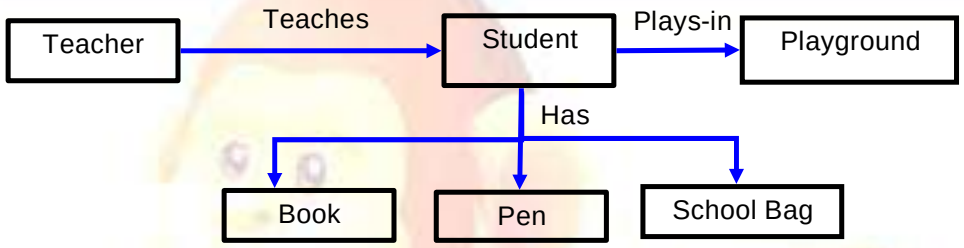
Example 1– Object Oriented Model

<table>
<tr>
<td>  </td>
<td> 
</td>
</tr>
<tr>
<td>Objects
Ali, Car, House, Tree</td>
<td>Interactions
Ali lives in the house
Ali drives the car</td>
</tr>
</table>

Example 2– Object Oriented Model (A School Model)



A School Model



Objects	**Interactions**
Teacher, Student, School Bag, Pen, Book Playground	Teacher teaches Student. Student has School Bag, Book and Pen

Object-Orientation - Advantages

As Object Oriented Models map directly to reality as we have seen in examples above therefore,

We can easily **develop** an object oriented model for a problem.
Everyone can easily **understand** an object oriented model.
We can easily implement an object oriented model for a problem using any object oriented language like c++ using its features[1] like classes, inheritance, virtual functions and so on…

What is an Object?

An object is,

1. Something tangible (Ali, School, House, Car).
2. Something conceptual (that can be apprehended intellectually for example time, date and so on…).

An object has,

1. State (attributes)
2. Well-defined behavior (operations)

[1] We will study these features in detail in this course

ORANGE MONKEY TEAM

3. Unique identity

Tangible and Intangible Objects

Examples of Tangible Objects:

Ali is a tangible object, having some characteristics (attributes) and behavior as given below,

Ali	
Characteristics (attributes)	**Behaviour (operations)**
Name Age	Walks Eats

We will identify Ali using his name.

Car is also a tangible object having some characteristics (attributes) and behavior given below,

Car	
State (attributes)	**Behavior (operations)**
Color Model	Accelerate Start Car Change Gear

We can identify Car using its registration number

Examples of Intangible Objects (also called as conceptual objects):

Time is an intangible (conceptual) object

Time	
State (attributes)	**Behavior (operations)**
Hours Seconds Minutes	Set/Get Hours Set/Get Seconds Set/Get Minutes

We will assign our own generated unique ID in the model for Time object

Date is also an intangible (conceptual) object

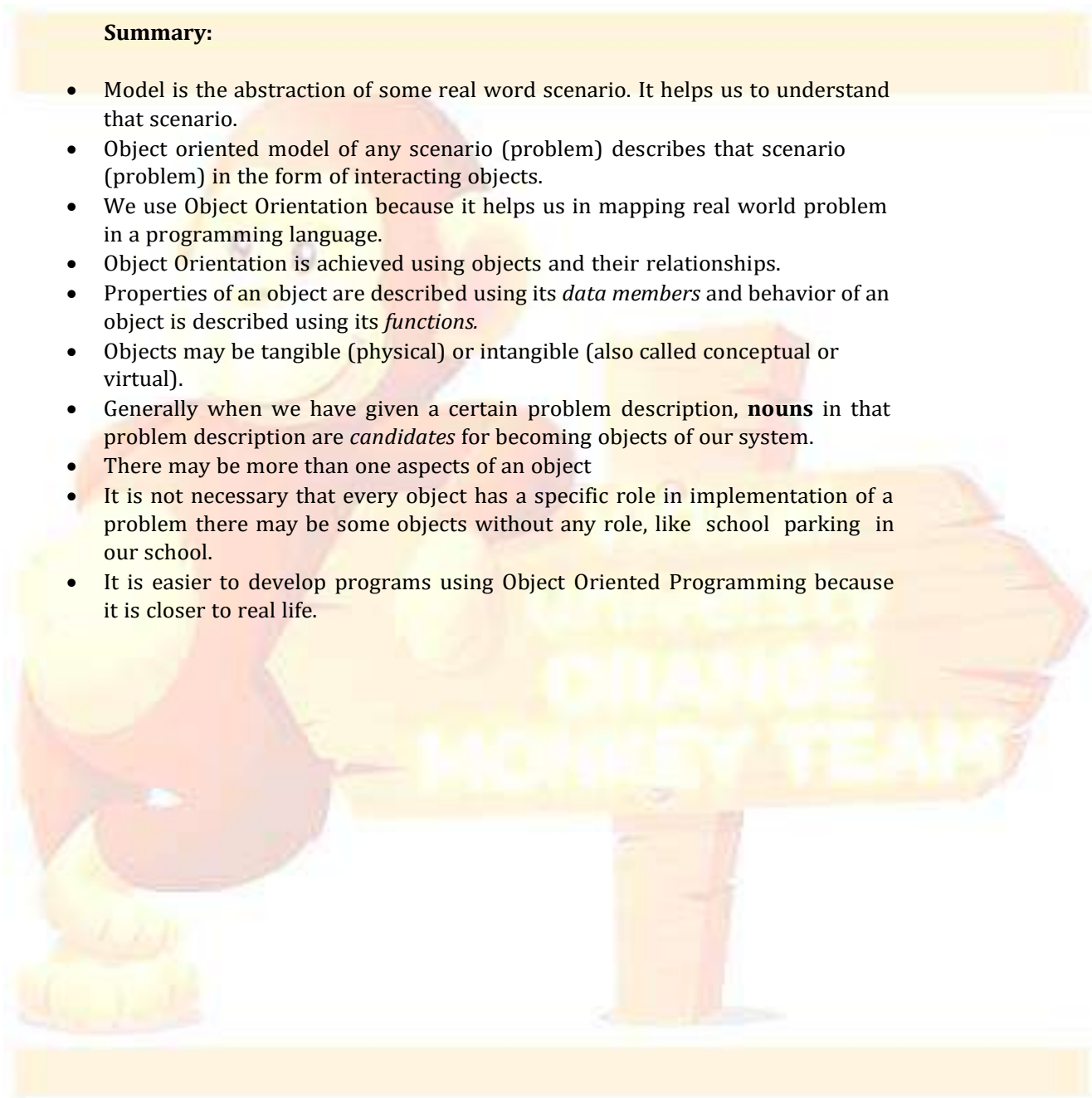
State (attributes)	**Behavior (operations)**
Year Day Month	Set/Get Year Set/Get Day Set/Get Month

ORANGE MONKEY TEAM

We will assign our own generated unique ID in the model for Date object.

Summary:

- Model is the abstraction of some real word scenario. It helps us to understand that scenario.
- Object oriented model of any scenario (problem) describes that scenario (problem) in the form of interacting objects.
- We use Object Orientation because it helps us in mapping real world problem in a programming language.
- Object Orientation is achieved using objects and their relationships.
- Properties of an object are described using its *data members* and behavior of an object is described using its *functions.*
- Objects may be tangible (physical) or intangible (also called conceptual or virtual).
- Generally when we have given a certain problem description, **nouns** in that problem description are *candidates* for becoming objects of our system.
- There may be more than one aspects of an object
- It is not necessary that every object has a specific role in implementation of a problem there may be some objects without any role, like school parking in our school.
- It is easier to develop programs using Object Oriented Programming because it is closer to real life.



Lecture No.02

Lecture Contents

1. Information Hiding
2. Encapsulation
3. Interface
4. Implementation
5. Separation of Interface & Implementation
6. Messages

Information Hiding:

Information hiding is one of the most important principles of OOP inspired from real life which says that all information should not be accessible to all persons. Private information should only be accessible to its owner.
By Information Hiding we mean "*Showing only those details to the outside world which are necessary for the outside world and hiding all other details from the outside world.*"

Real Life Examples of Information Hiding

1. Ali's name and other personal information is stored in his brain we can't access this information directly. For getting this information we need to ask Ali about it and it will be up to Ali how much details he would like to share with us.

2. An email server may have account information of millions of people but it will share only our account information with us if we request it to send anyone else accounts information our request will be refused.

3. A phone SIM card may store several phone numbers but we can't read the numbers directly from the SIM card rather **phone-set** reads this information for us and if the owner of this phone has not allowed others to see the numbers saved in this phone we will not be able to see those phone numbers using phone.

In object oriented programming approach we have objects with their attributes and behaviors that are hidden from other classes, so we can say that object oriented programming *follows the principle of information hiding*.

In the perspective of **Object Oriented Programming** Information Hiding is,

"Hiding the object details (state and behavior) from the users"

Here by users we mean **"an object"** of another class that is calling functions of this class using the reference of this class object or it may be some other program in which we are using this class.

Information Hiding is achieved in Object Oriented Programming using the following principles,

ORANGE MONKEY TEAM

- All information related to an object is stored within the object
- It is hidden from the outside world
- It can only be manipulated by the object itself

Advantages of Information Hiding

Following are two major advantages of information hiding,

It simplifies our Object Oriented Model:

As we saw earlier that our object oriented model only had objects and their interactions hiding implementation details so it makes it easier for everyone to understand our object oriented model.

It is a barrier against change propagation

As implementation of functions is limited to our class and we have only given the name of functions to user along with description of parameters so if we change implementation of function it doesn't affect the object oriented model.

We can achieve information hiding using **Encapsulation** and **Abstraction**, so we see these two concepts in detail now,

Encapsulation

Encapsulation means *"we have enclosed all the characteristics of an object in the object itself"*
Encapsulation and information hiding are much related concepts (information hiding is achieved using Encapsulation)
We have seen in previous lecture that object characteristics include data members and behavior of the object in the form of functions.

So we can say that Data and Behavior are tightly coupled inside an object and both the information structure and implementation details of its operations are hidden from the outer world.

Examples of Encapsulation

Consider the same example of object Ali of previous lecture we described it as follows,

ORANGE MONKEY TEAM

Ali
Characteristics (attributes) • Name • Age
Behavior (operations) • Walks • Eats

You can see that Ali stores his personal information in itself and its behavior is also implemented in it.
Now it is up to object Ali whether he wants to share that information with outside world or not. Same thing stands for its behavior if some other object in real life wants to use his behavior of walking it can not use it without the permission of Ali.

So we say that attributes and behavior of Ali are encapsulated in it.

Any other object don't know about these things unless Ali share this information with that object through an interface,

Same concept also applies to phone which has some data and behavior of showing that data to user we can only access the information stored in the phone if phone interface allow us to do so.

Advantages of Encapsulation
The following are the main advantages of Encapsulation,

a. Simplicity and clarity
As all data and functions are stored in the objects so there is no data or function around in program that is not part of any object and is this way it becomes very easy to understand the purpose of each data member and function in an object.

b. Low complexity
As data members and functions are hidden in objects and each object has a specific behavior so there is less complexity in code there will be no such situations that a functions is using some other function and that functions is using some other function.

c. Better understanding
Everyone will be able to understand whole scenario by simple looking into object diagrams without any issue as each object has specific role and specific relation with other objects.

Interface

ORANGE MONKEY TEAM

Interface is a set of functions of an object that he wants to expose to other objects.

As we discussed previously that data and behavior of each object is hidden in that object it self so we have to use the concept of interface of the object to expose its behavior to outer word objects.

- Different objects may need different functions of an object so interface of an object may be different for different objects.
- Interfaces are necessary for object communication. Each object provides interface/s (operations) to other objects through these interfaces other objects communicate with this object.

Example – Interface of a Car

- Steer Wheels
- Accelerate
- Change Gear
- Apply Brakes
- Turn Lights On/Off

Example – Interface of a Phone

- Input Number
- Place Call
- Disconnect Call
- Add number to address book
- Remove number
- Update number

Implementation

It is actual implementation of the behavior of the object in any Object Oriented language.

It has two parts,

- Internal data structures to hold an object state that will be hidden from us it will store values for an object data members.
- Functionality in the form of member functions to provide required behavior.

Examples of Implementation

a. **Gear Box in car system**
Consider object Gear Box in car system it has a certain structure and functionality. When this object will be implemented it will have two things,
   - Physical structure of the gear box
   - Functionality implemented in this structure to change gear.

   Both these things are part of implementation.

So it has,

- **Data Structure** in the form of Mechanical structure of gear box
- **Functionality** mechanism to change gear

b. Address Book in a Phone

Similarly take the example of contact details saved in the SIM of a phone,

In that case we can say physical structure of SIM card as **Data Structure**
And Read/write operations provided by the phone as **Functionality**.

Separation of Interface & Implementation

As discussed earlier we only show interface of an object to outside world and hide actual implementation from outside world. The benefit of using this approach is that our object interface to outside word becomes independent from inside implementation of that interface.

This is achieved through the concepts of encapsulation and information hiding.

Real Life example of separation of interface and implementations

- Driver has a standard interface to drive a car and using that interface he drive can drive any car regardless of its model or type whatever engine type it has or whatever type of fuel it is using.

Messages

Objects communicate through messages they send messages (stimuli) by invoking appropriate operations on the target object. The number and kind of messages that can be sent to an object depends upon its interface

Examples – Messages

A Person sends message (stimulus) "stop" to a Car by applying brakes

A Person sends message "place call" to a Phone by pressing appropriate button

Summary

- Information hiding is achieved through encapsulation.
- Encapsulation and Information Hiding are related to each other.
- Interface of an object provides us the list of available functions.
- An object may have more than one interface.
- Interface and implementation are separated from each other to achieve Information Hiding.
- Objects communicate with each other using messages.

ORANGE MONKEY TEAM

Useful Links:

http://www.alice.org/

A Graphical Programming Environment to teach Computer Programming.

Lecture No.03

Lecture Contents:

- Abstraction
- Classes
- Inheritance
- Major benefits of inheritance (Reuse)

Abstraction

Real life objects have a lot of attributes and many kind of behaviors but most of the time we are interested in only that part of the objects that is related to the problem we are currently going to solve, for example in implementing a school system we don't need to take care of the personnel life of a student or a teacher as it will not effect our system in any way so we will see these objects in the perspective of school system and will ignore their other characteristics, this concept is called "*Abstraction*". Abstraction is a way to cope with complexity and it is used to simplify things.

Principle of abstraction:

"Capture only those details about an object that are relevant to current perspective"

Abstraction Example:

Suppose we want to implement abstraction for the following statement,

"Ali is a PhD student and teaches BS students"

*Here object Ali has two **perspectives** one is his **student perspective** and second is his **teacher perspective.***

We can sum up Ali's attributes as follows,

Name
Age
Student Roll No
Year of Study
CGPA
Employee ID
Designation
Salary

As you can see out of all these listed attributes some belong to Ali's student perspective(Roll No, CGPA, Year of study) and some belong to Ali's teacher perspective(Employee ID, Designation, Salary).

Similarly we can sum up Ali's behavior as follows,

Study
DevelopExam

GiveExam
TakeExam
PlaySports
Eat
DeliverLecture
Walk

As was the case with attributes of object Ali, its behavior can also be divided in Ali's student perspective as well as Ali's teacher perspective.

Student's Perspective

Attributes:

- **Name**
- **Student Roll No**
- **Year of Study**
- **CGPA**
- Employee ID
- Designation
- Salary
- Age

Behaviour:

- **Study**
- **GiveExam**
- **PlaySports**
- DeliverLecture
- DevelopExam
- TakeExam
- Eat
- Walk

Teacher's Perspective

Attributes:

- **Name**
- Student Roll No
- Year of Study
- CGPA
- **Employee ID**
- **Designation**
- **Salary**
- Age

Behaviour:

- Study
- GiveExam
- PlaySports
- **DeliverLecture**
- **DevelopExam**
- **TakeExam**
- Eat
- Walk

A cat can be viewed with different perspectives

Ordinary Perspective	Surgeon's Perspective
A pet animal with Four Legs A Tail	A being with A Skeleton Heart

ORANGE MONKEY TEAM

Two Ears Sharp Teeth	Kidney Stomach

A car can be viewed with different perspectives

 Driver's View	 Engineer's View

Abstraction – Advantages

Abstraction has following major advantages,

1. It helps us understanding and solving a problem using object oriented approach as it hides extra irrelevant details of objects.

2. Focusing on single perspective of an object provides us freedom to change implementation for other aspects of for an object later.

Similar to Encapsulation Abstraction is also used for achieving information hiding as we show only relevant details to related objects, and hide other details.

Classes

In OOP we create a general sketch for each kind of objects and then we create different instances using this sketch we call this sketch or prototype or map as "class".
All objects of same kind exhibit identical characteristics (information structure and behavior) however they have data of their own.

Class –Example 1

Consider the objects given below,

- Ali studies mathematics
- Anam studies physics
- Sohail studies chemistry

Each one is a Student so we say these objects are *instances* of the Student class.

Class –Example 2

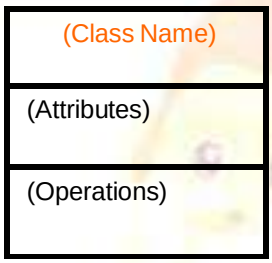
Consider the objects given below,

ORANGE MONKEY TEAM

- Ahsan teaches mathematics
- Aamir teaches computer science
- Atif teaches physics

Each one is a teacher so we say these objects are *instances* of the Teacher class

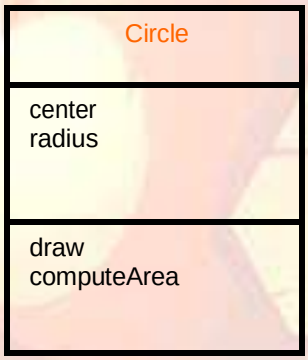
Class Representation:
we can represent a class using a rectangle as follows,



Normal Form

Suppressed Form

Class Example: Circle

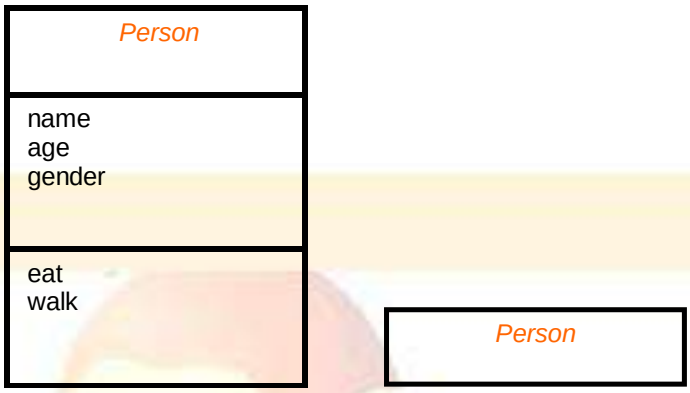


Normal Form

Suppressed Form

Class Example: Person

ORANGE MONKEY TEAM



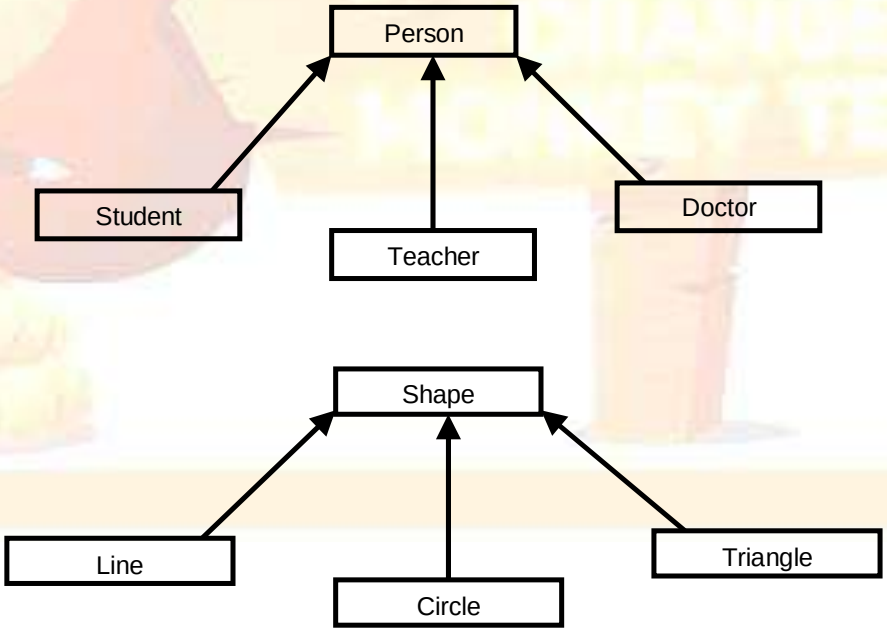
Normal Form **Suppressed Form**

Inheritance

A child inherits characteristics of its parents, besides inherited characteristics, a child may have its own unique characteristics

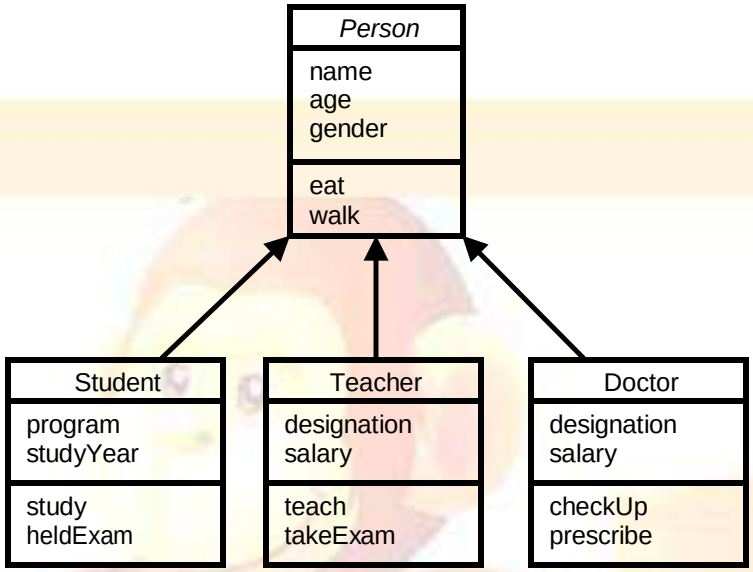
Inheritance in Classes

If a class B inherits from class A then it contains all the characteristics (information structure and behaviour) of class A
The parent class is called *base* class and the child class is called *derived* class
Besides inherited characteristics, derived class may have its own unique characteristics

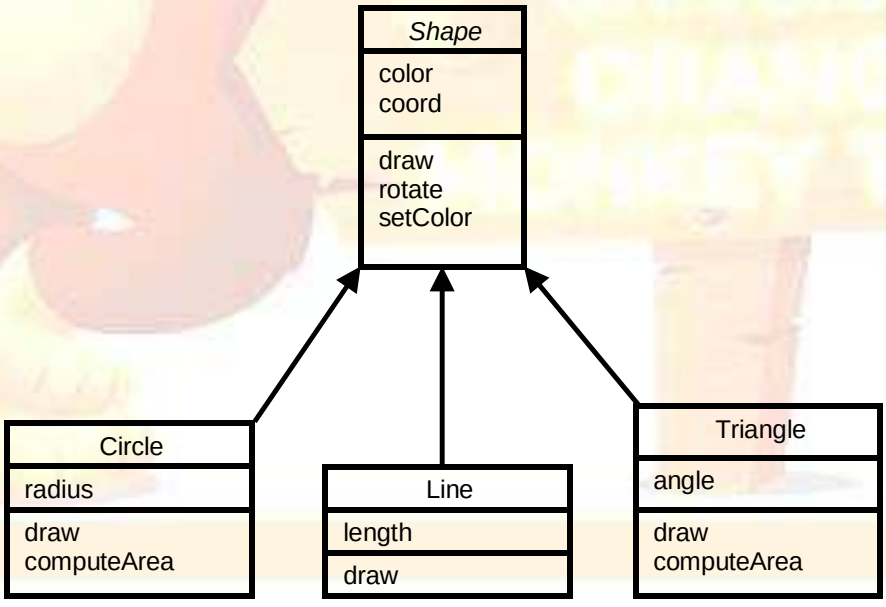


Inheritance – "IS A" or "IS A KIND OF" Relationship

Each derived class is a kind of its base class



Here,
Student IS A Person
Teacher IS A Person
Doctor IS A Person



Here,
Circle IS A Shape
Line IS A Shape
Triangle IS A Shape

ORANGE MONKEY TEAM

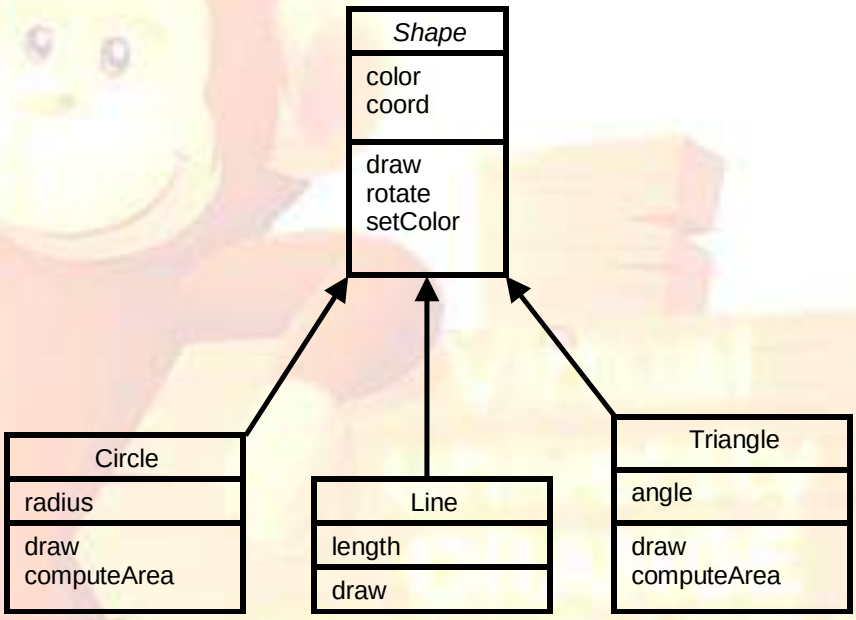
Inheritance – Advantages

1. Reuse
2. Less redundancy
3. Increased maintainability

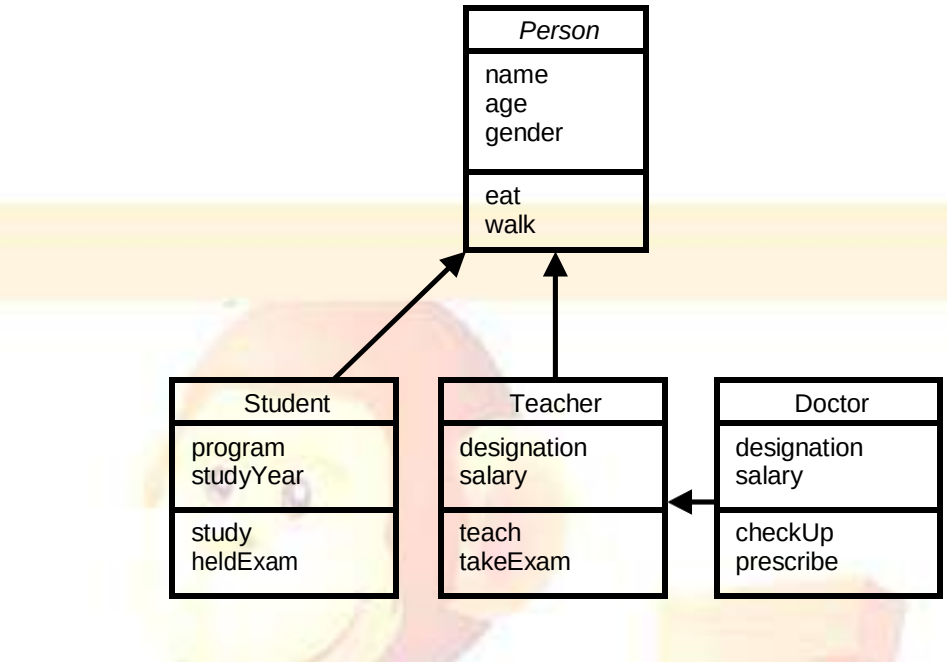
Reuse with Inheritance

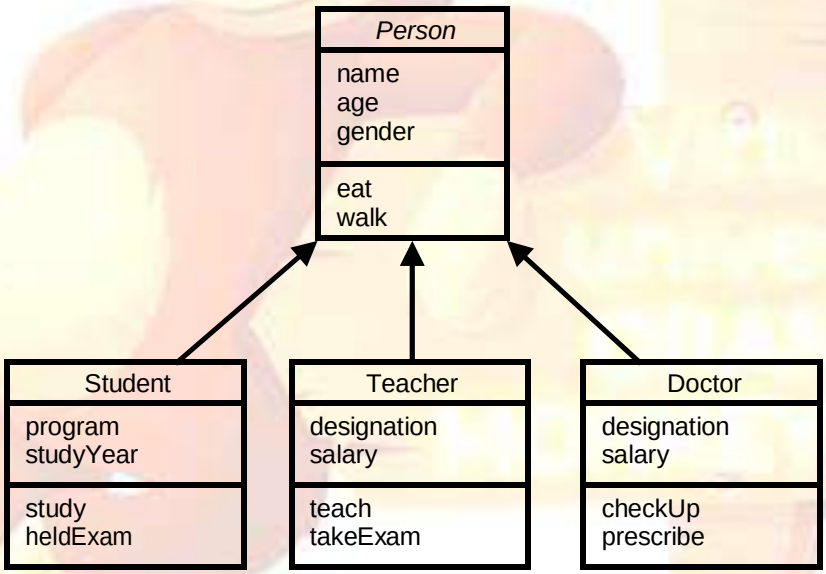
Main purpose of inheritance is reuse, we can easily add new classes by inheriting from existing classes.

Select an existing class closer to the desired functionality, create a new class and inherit it from the selected class, add to and/or modify the inherited functionality



ORANGE MONKEY TEAM





ORANGE MONKEY TEAM

Lecture No.04

Lecture Contents

- Generalization
- Sub typing (extension)
- Specialization (restriction)
- Overriding
- Abstract classes
- Concrete classes

Recap – Inheritance

- Derived class inherits all the characteristics of the base class
- Besides inherited characteristics, derived class may have its own unique characteristics
- Major benefit of inheritance is reuse

Concepts Related with Inheritance

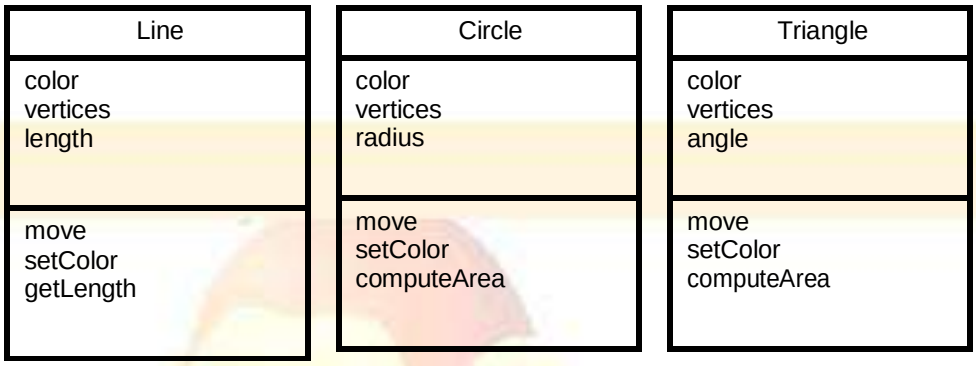
- Generalization
- Subtyping (extension)
- Specialization (restriction)

Generalization

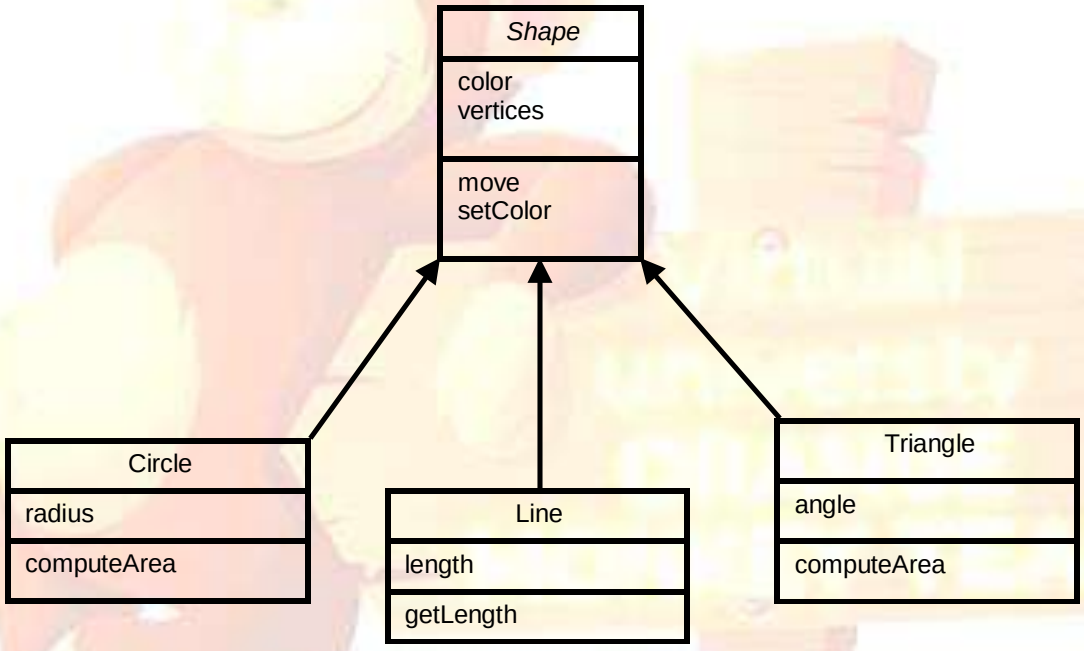
In OO models, some classes may have common characteristics.
We extract these features into a new class and inherit original classes from this new class. There are many objects with common characteristics in object model. The common characteristics (attributes and behaviour) of all these objects are combined in a single general class. Base class encapsulates the idea of commonality of derived classes. Base class is general class representing common behaviour of all derived classes.
This concept is known as Generalization.
It reduces the redundancy and gives us reusability, using generalization our solution becomes less complex.
In generalization there should be "Is a Kind of Relationship" (also called "Is A relationship") between base and child classes.

ORANGE MONKEY TEAM

Example: Line, Circle and Triangle



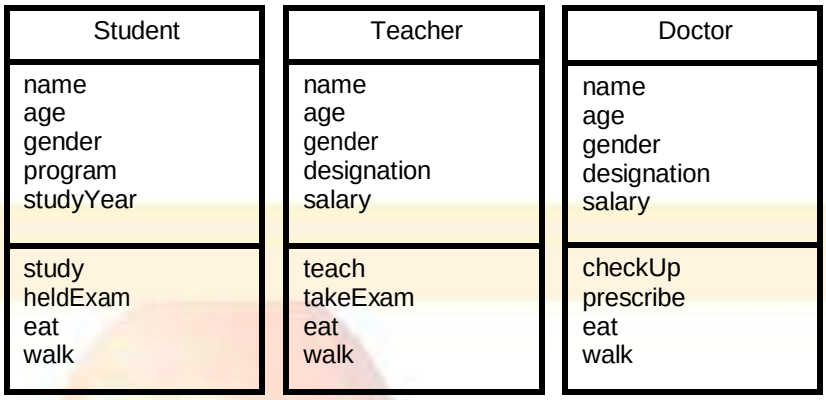
Line is shape Circle is a shape Triangle is a shape

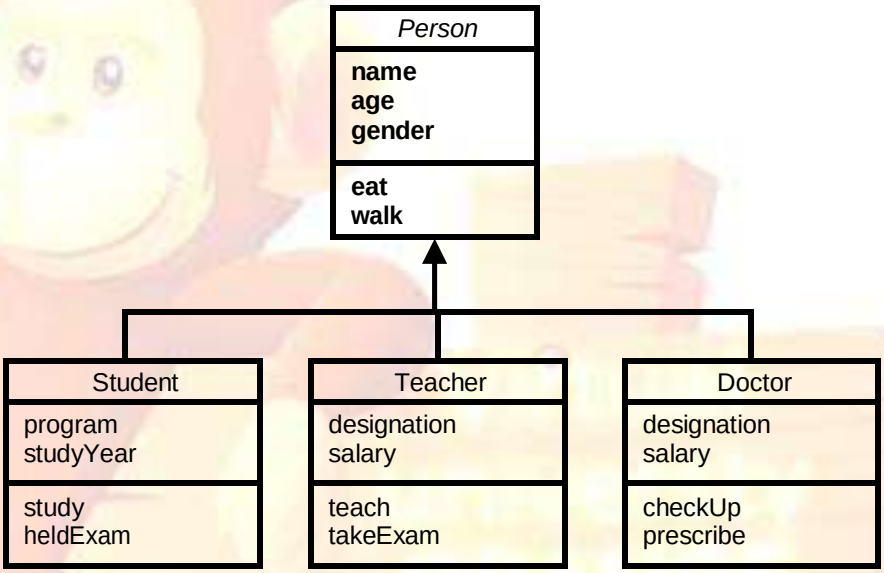


Common attributes
Color vertices
Common behaviour
Set Color, Move

Example: Student Doctor and Teacher

ORANGE MONKEY TEAM





Common attributes,
Name, age, gender

Common behaviour
Eat, Walk

Sub-typing & Specialization

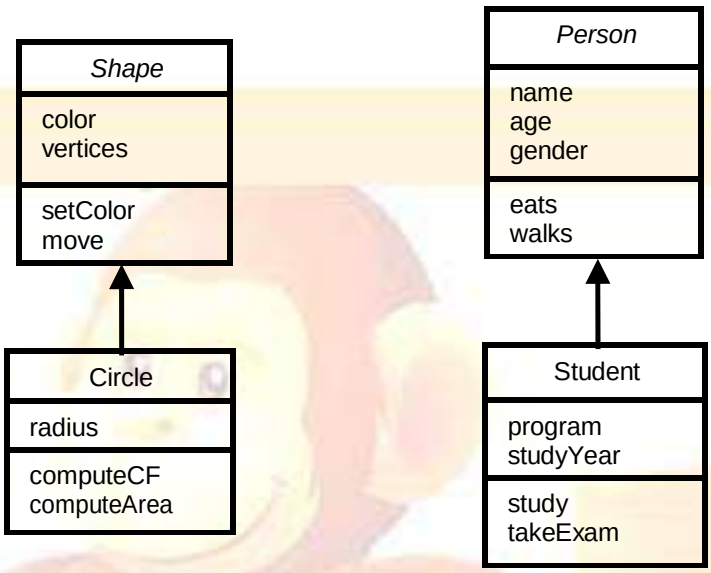
We want to add a new class to an existing model
We have developed an existing class hierarchy
Find an existing class that already implements some of the desired state and behaviour
Inherit the new class from this class and add unique behaviour to the new class

Sub-typing (Extension)

Sub-typing means that derived class is behaviourally compatible with the base class
Derived class has all the characteristics of base class plus some extra characteristics
Behaviourally compatible means that base class can be replaced by the derived class

ORANGE MONKEY TEAM

Sub-typing (Extension) - Example



Circle is extending the behaviour of shape, it is extending attributes of shape by adding radius similarly it is extending behaviour of shape by adding compute Circumference and compute Area.

Student has two extra attributes program and studyYear
Similarly it has extended behaviour by adding study and takeExam.

Subtyping and generalization are related concepts, Subtyping (extension) and generalization is a way to look same thing in two ways.
Sub typing is looking at things from Top to bottom whereas in generalization we look at things from bottom to top.

Specialization (Restriction)

We want to add a class to existing hierarchy of classes having many similarities to already existing classes but some part of its behaviour is different or restricted. In that case we will use the concept of specialization.

Specialization means that derived class is behaviourally incompatible with the base class

Behaviourally incompatibility means that base class can't always be replaced by the derived class

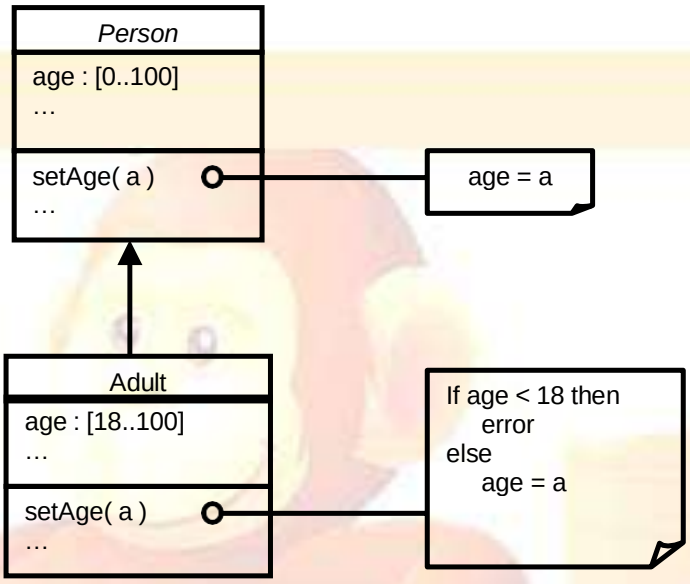
Derived class has some different of restricted characteristics than of base class.

Example – Specialization (Restriction)

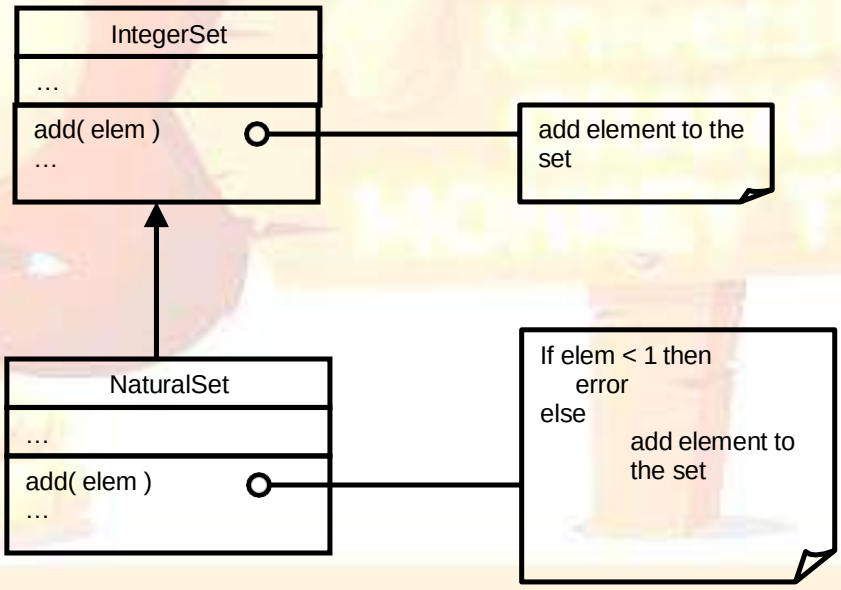
Suppose we want to add one more class of Adult for some special requirement like for ID card generation such that it is a person but its age is greater than 18 and having all other behaviour of that of person class. One solution is that we write

ORANGE MONKEY TEAM

another class from beginning and write all code of person again in it with age limit, but better solution is that we derive adult class from person class and restrict age in that class as shown below in diagram,



Similarly Natural Numbers[2] are also Integers[3] with the restriction that natural numbers set can NOT contain zero or negative integers it consists of only positive integers so we can implement this relationship also as specialization,



[2] Natural numbers: positive integers only (numbers from 1 to …….onwards)

[3] Integers: all positive and negative numbers (…..-3 , -2 , -1 , 0 , 1 , 2 , 3……..)

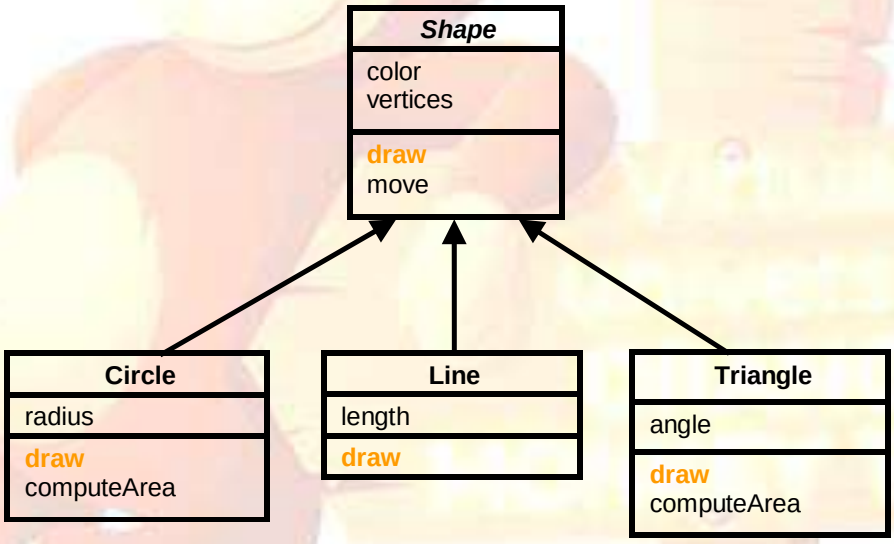
Add method behaviour is present in both base and derived classes but derived class behaviour is different in derived class. Derived class will not exhibit the behaviour of base class but it is **overriding** behaviour of base class with its own behaviour.

Overriding

A class may need to override the default behaviour provided by its base class
Derived class overrides the behaviour of its base class.
Reasons for overriding
- Provide behaviour specific to a derived class (specialization)
- Extend the default behaviour (extension)
- Restrict the default behaviour (restriction)
- Improve performance

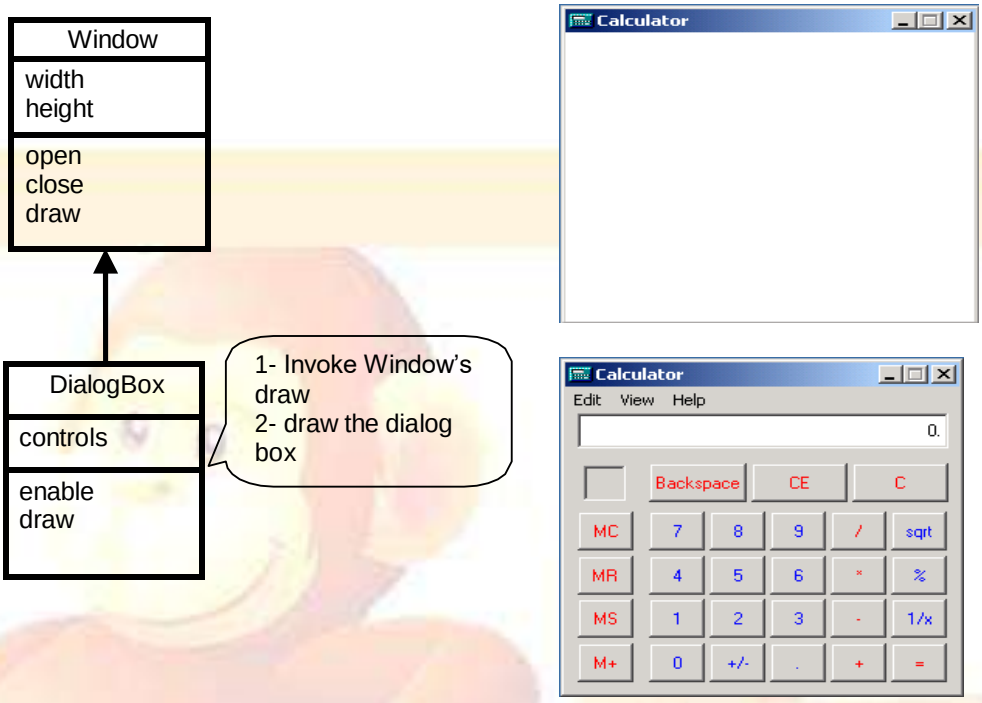
It is used for the implementation of inheritance.

Example – Specific Behaviour (Specialization)

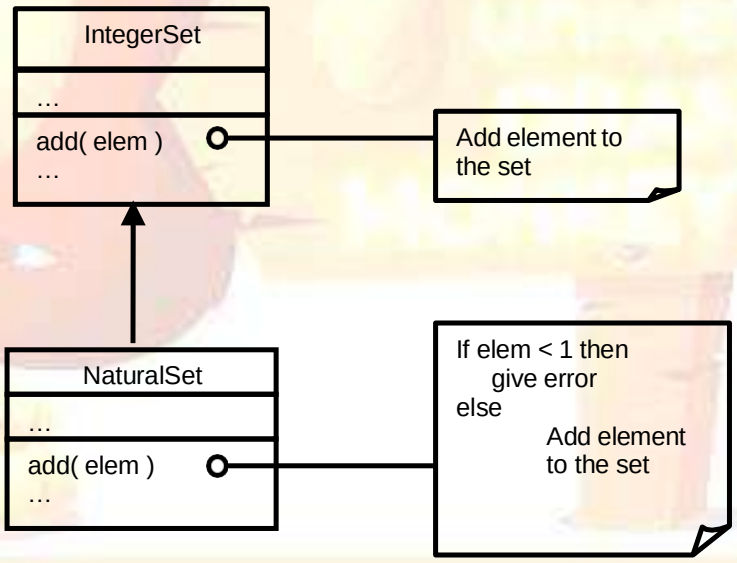


ORANGE MONKEY TEAM

Example – Extention

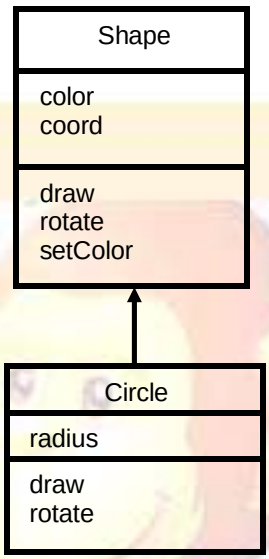


Example – Restriction



Example – Improve Performance

Class Circle overrides *rotate* operation of class Shape with a Null operation.

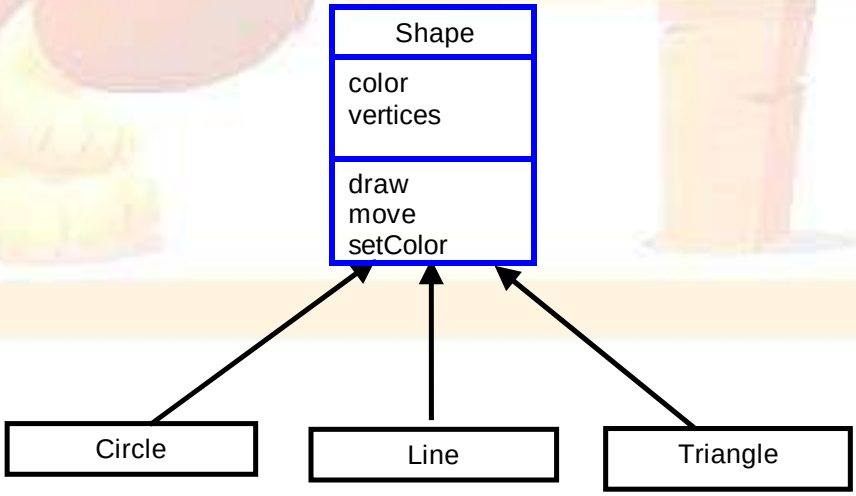


Abstract Classes

In our examples we made classes for shape and person. These are abstract concepts and the classes we make against abstract concepts are called abstract classes. They are present at or near the top in the class hierarchy to present most generalized behaviour.

An abstract class implements an abstract concept
Main purpose is to be inherited by other classes
Can't be instantiated
Promotes reuse

Abstract Classes - Example I

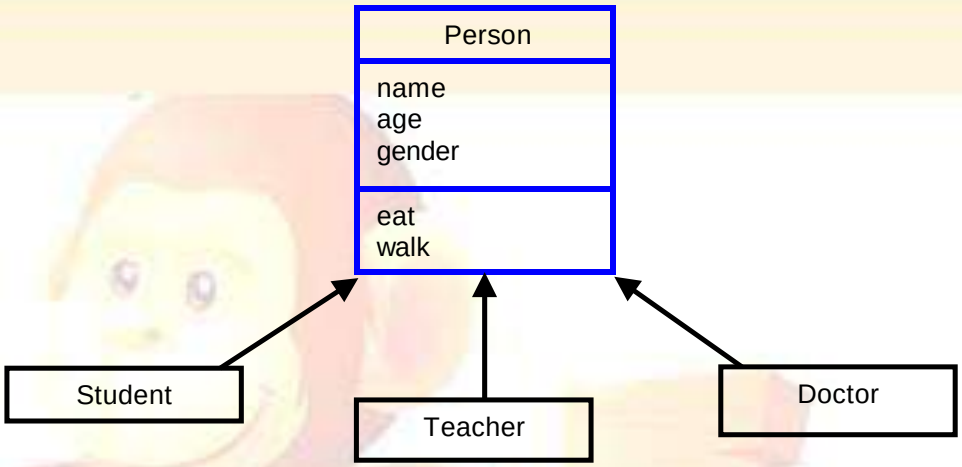


Here, Shape is an abstract class

ORANGE MONKEY TEAM

Abstract Class	Shape			
Concrete Classes	Circle	Line	Triangle	....

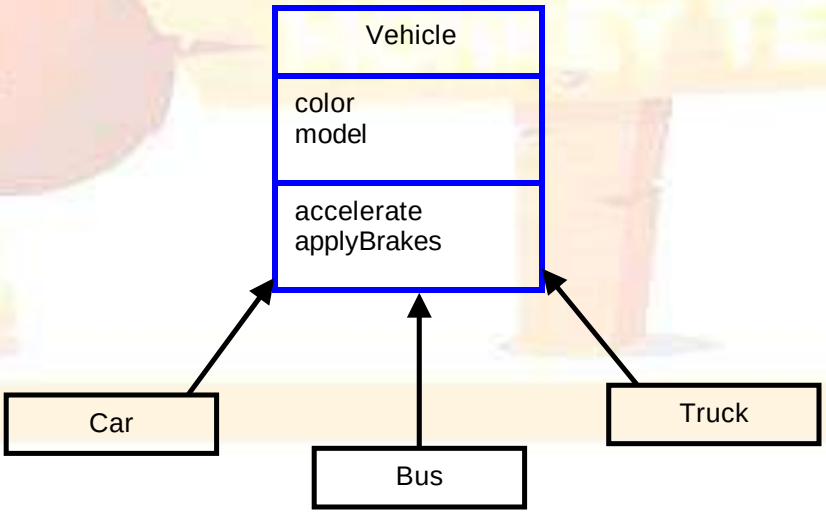
Abstract Classes - Example II



Here, Person is an abstract class

Abstract Class	Person					
Concrete Classes	Student	Teacher	Doctor	Engineer	Director	....

Abstract Classes - Example III



Here, Vehicle is an abstract class

ORANGE MONKEY TEAM

Abstract Class	Vehicle			
Concrete Classes	Car	Bus	Truck	….

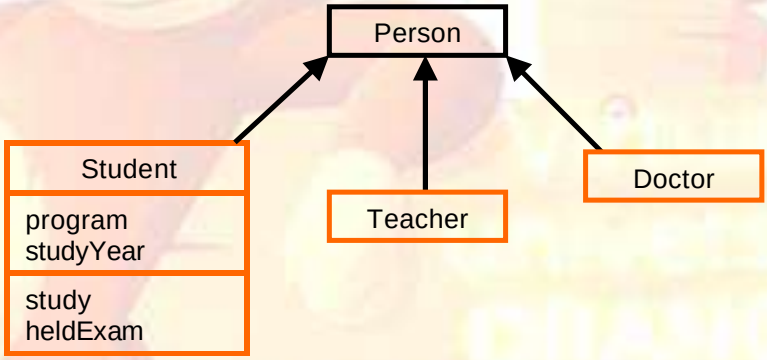
Abstract Classes can not exist standalone in an object model
While making object model we start by finding out objects in our object model and then we find out objects having common attributes and make them in the form of general classes at the top of class hierarchies.

Concrete Classes

The entities that actually we see in our real world are called concrete objects and classes made against these objects are called concrete classes.

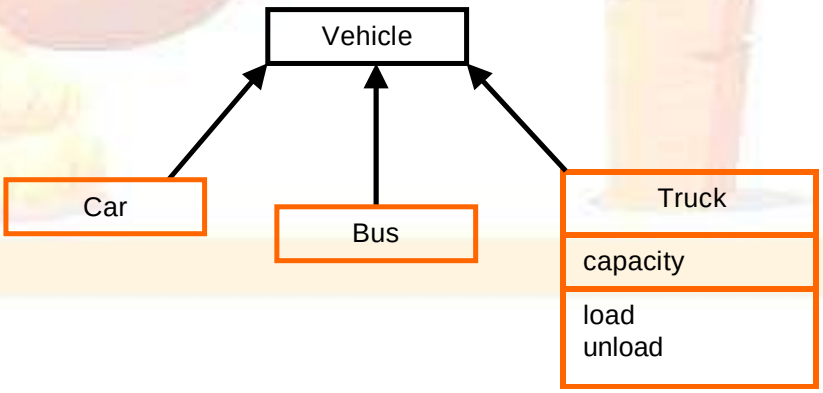
A concrete class implements a concrete concept
These are used to instantiate objects in our programs
Provides implementation details specific to the domain context

Concrete Classes - Example I



Here Student, Teacher and Doctor are concrete classes

Concrete Classes - Example II



Here Car, Bus and Truck are concrete classes

ORANGE MONKEY TEAM

- A concrete class may exist in an object model independently
- Concrete classes mostly lie below the top of class hierarchy in a good object model.

If there is an abstract class then hierarchy exists in the object model as there will definitely be some concrete classes as well derived from this abstract class otherwise there is no use of abstract class.

Glossary:

a. Natural numbers: numbers from 1 to …….onwards
b. Integers: all positive and negative numbers …..-3,-2,-1,0,1,2,3…….
c. Whole numbers: numbers from 0 ,1 ,2, 3 ….onwards (natural no's including 0)

Some times whole numbers are also called numbers without fractional part.

Lecture No.05
Multiple Inheritance

Inheritance:
We saw inheritance purposes in last lecture

- Generalization
- Extention or sub typing
- Specialization or restriction

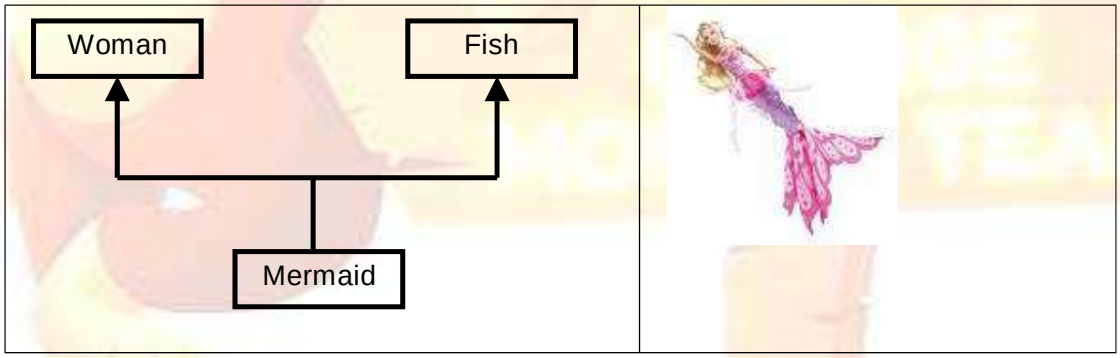
Abstract and concrete classes, former is used to represent abstract concepts later is used to represent concrete concepts.
Overriding derived classes override inherited classes (base classes) behaviour.
Overriding is used for Specialization, Extention, Restriction, and Performance.

Multiple Inheritance

Sometimes we want to reuse characteristics of more than one parent class, in that case we need to inherit a class from more than one classes.

Example 1– Multiple Inheritance

Consider the example of an imaginary specie Mermaid used in fairy tales that lives in water having features both of a women as well as of a fish, In Object Oriented programming perspective Mermaid can be derived from two classes Women and Fish.



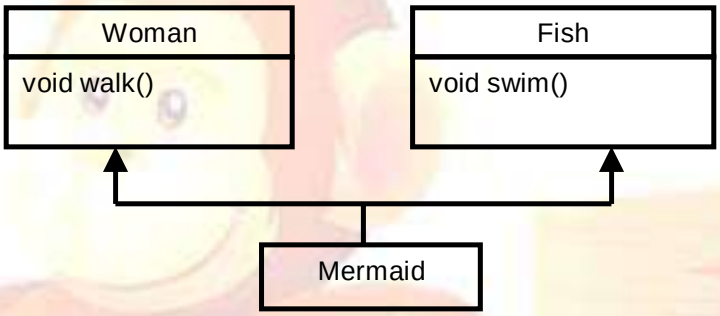
C++ Code:

```
/*Program to demonstrate simple multiple inheritance*/

class Fish {


};

class Woman {
```

ORANGE MONKEY TEAM

```
};

class Mermaid : public Woman , public Fish {

};
```

Our Mermaid class inherits features of both woman and fish suppose our woman class has method wald() and fish cclass has method swim then our mermaid class can use both methods i.e can walk as well as can swim.



C++ code:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

/*Program to demonstrate simple multiple inheritance*/

class Fish
{
public:
    void swim(){
    cout<<"\n In method swim";
    }

};

class Woman
{
public:
    void walk(){
    cout<<"\n In method walk"<<endl;
    }

};
```

ORANGE MONKEY TEAM

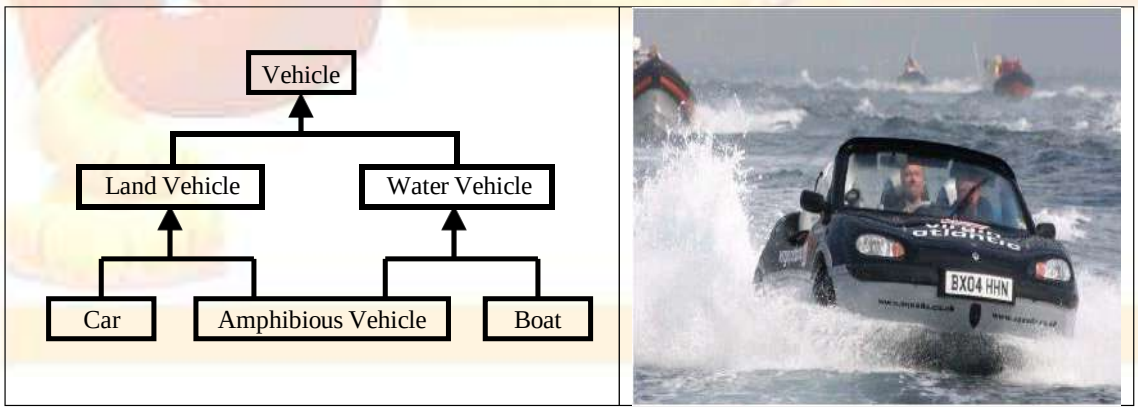
```
class Mermaid : public Woman,public Fish
{

};

int main(int argc, char *argv[])
{

Mermaid mermaid;
/*This Mermaid object will have two implicit objects one of Fish class and one of
Woman class*/
mermaid.swim();
mermaid.walk();

system("PAUSE");
return 0;

}
```

Output:

```
In method[4] swim
In method walk
```

Example 2– Multiple Inheritance

Take another example of amphibious vehicle (vehicle that can run on land as well as on water) so it has properties of both land as well as of water vehicle. The general hierarchy in this case will be,



Here we have added a general Vehicle class as well to add all common functions of Land Vehicles and Water Vehicles in that class, and specific functions of Land and

[4] class member functions are also called class methods

Water vehicle in their respective classes then we have derived Amphibious Vehicle class from Land Vehicle and Water Vehicle classes (we can do the same in first example as well concerning Woman, Fish and Mermaid).

C++ code:

```
class Vehicle
{

};

class WaterVehicle : public Vehicle
{


};

class LandVehicle : public Vehicle
{


};

class AmphibiousVehicle : public LandVehicle,public WaterVehicle
{

};
```

Suppose we have a **changeGear** method in Vehicle class that is applicable to both water and land vehicle, we also have Float and Move methods in water and land vehicles respectively then our amphibious vehicle will have all these methods,

C++ code:

```
#include <iostream>
#include <stdlib.h>

using namespace std;

/*Multiple Inheritance in case of Amphibious Vehicle*/

class Vehicle
{
public:
 void changeGear(){ cout<<"\nI am Vehicle changeGear() function..\n";}
};

class WaterVehicle : public Vehicle
{
```

ORANGE MONKEY TEAM

```
public:
void Float(){ cout<<"\nI am float function of Water Vehicle";}
};


class LandVehicle : public Vehicle
{
public:
void Move(){ cout<<"\nI am move function of Land Vehicle"<<endl;}

};

class AmphibiousVehicle : public LandVehicle,public WaterVehicle
{


};

int main(int argc, char *argv[])
{

AmphibiousVehicle amphibious;

amphibious.Float();
/*Calling Float function of Water Vehicle class*/

amphibious.Move();
/*Calling Move function of Land Vehicle class*/

system("PAUSE");
return 0;


}
```

Output:

I am float function of Water Vehicle
I am move function of Land Vehicle

Advantage of Multiple Inheritance:

As was the case with simple (single) inheritance multiple inheritance also decreases redundant code as we can inherit a class from many classes and can use their functions without the need to write them again.

However, there are more disadvantages of multiple inheritance, than its advantages.

Problems with Multiple Inheritance

Increased complexity

Amphibious vehicle hierarchy is a complicated as this class is derived from two classes that will make code more complex and less understandable however this is obvious as amphibious vehicle is a complicated vehicle. It is generic problem.

Reduced understanding

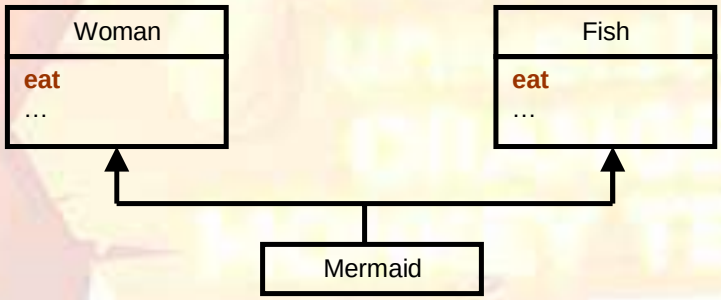
Due to increased complexity of class hierarchy the object model becomes difficult it understand especially for someone who is looking it first time.

Duplicate features

As we are deriving a single class from more than one class so there is a chance of duplication of features (same methods in both parents), following problems may arise due to duplicate features,

Problem 1: Ambiguity

Consider the class hierarchy of Mermaid class below,

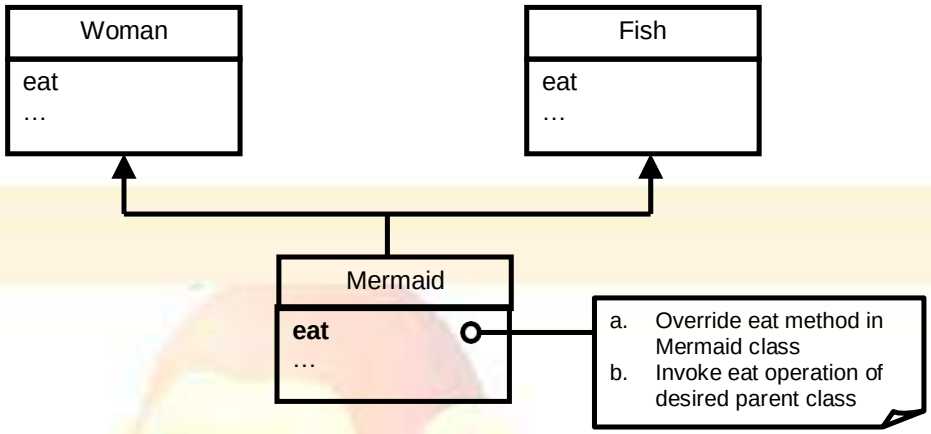


As mermaid also needs to eat and its both parents have their own methods of eating so here question arises,

Which eat operation Mermaid should inherit as both functions are available?

Solution – We can solve this problem by explicitly calling eat method from any of the parent classes in Mermaid class according to behaviour of Mermaid (i.e. if it eats like a Woman we can call eat method of Woman class and if eats like Fish we can call method of Fish class), for this we will Override the Common method in multiply inherited class and in that class overridden method we will call the appropriate base class function.

ORANGE MONKEY TEAM



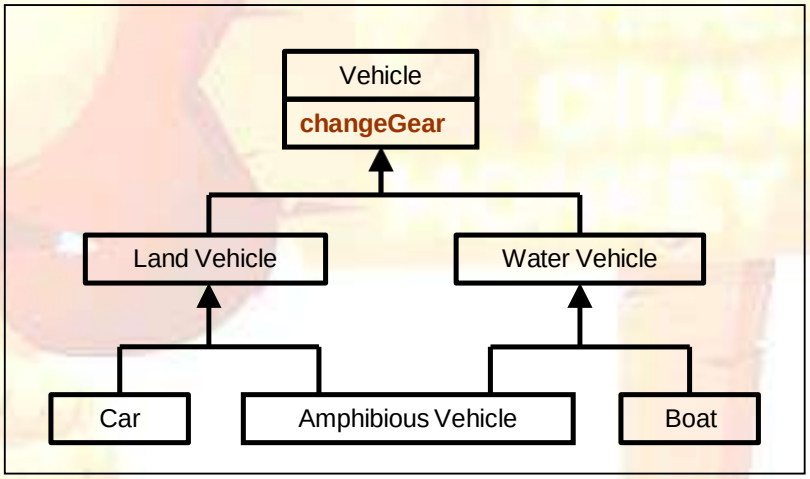
Example C++ Code

```
#include <iostream>
#include <stdlib.h>

using namespace std;

/*Program to demonstrate simple multiple inheritance*/

class Fish
{
public:
    void eat(){
    cout<<"\n In Fish eat method ";
    }

};

class Woman
{
public:
    void eat(){
    cout<<"\n In Woman eat method \n"<<endl;
    }

};

class Mermaid : public Woman,public Fish
{

    public:

     void eat(){
```

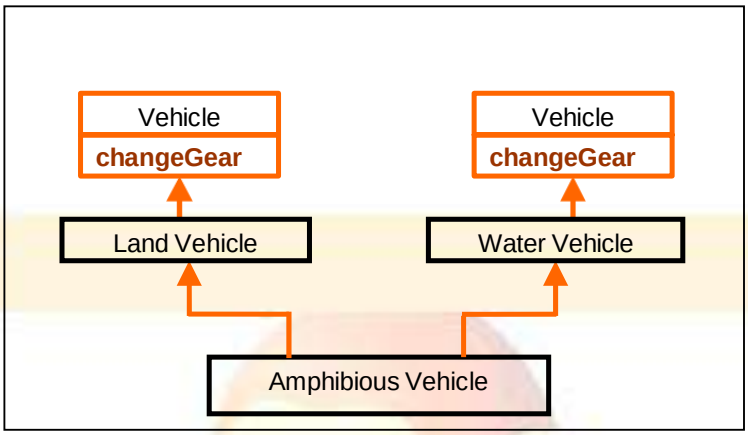
ORANGE MONKEY TEAM

```
        cout<<"\n In Mermaid eat method "<<endl;
        cout<<"\n Explicity calling Woman eat method. ...."<<endl;

        Woman::eat();

        }

};

int main(int argc, char *argv[])
{


Mermaid mermaid;
/*This Mermaid object will have two implicit objects one of Fish class and one of
Woman class*/
mermaid.eat();
/*Calling Mermaid eat method*/

system("PAUSE");
return 0;

}
```

Problem 2: Two instances for same function (Diamond Problem)



Here Amphibious Vehicle will have two copies of changeGear function as it will have two objects of Vehicle class one with respect to Land Vehicle and one with respect to Water Vehicle as shown below,



Actual Memory Layout

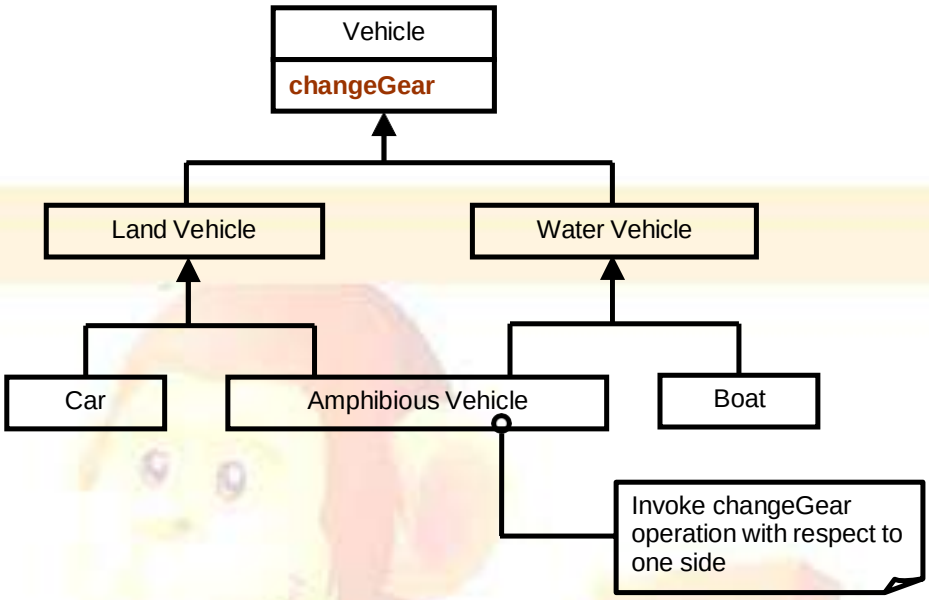
Compiler will not be able to decide which changeGear operation Amphibious Vehicle should inherit so it will generate an error as shown below (two copied of same method),

error: request for member `changeGear' is ambiguous
error: candidates are: **void Vehicle::changeGear()**
void Vehicle::changeGear()
Execution terminated

Solution to Diamond Problem

Some languages disallow diamond hierarchy
Others provide mechanism to ignore characteristics from one side. There are two cases while solving diamond problem virtual inheritance and non virtual inheritance (we will study these details in coming lectures)

ORANGE MONKEY TEAM



Association:

Interaction of different objects in OO model (or in problem domain) is known as association.
In object oriented model, objects interact with each other in order to perform some useful work, while modeling these objects (entities) is done using the association. Usually an object provides services to several other objects. An object keeps association with other objects to delegate tasks. This association can be represented with a line along an arrow head (⟶) or without arrow head.

Kinds of Association:

There are two main types of association which are then further subdivided i.e

1. Class Association
2. Object Association

1. Class Association

Class association is implemented in terms of Inheritance. Inheritance implements generalization/specialization relationship between objects. Inheritance is considered class association.

- In case of public inheritance it is "IS-A" relationship.
- In case of private inheritance it is "Implemented in terms of" relationship.

This relationship ensures that public members of base class are available to derived class in case of public inheritance.

When we create objects of classes in which we have implemented inheritance relationships we are forcing the inheritance relationship between created objects. In this case the derived class objects will also contain base class objects attributes and methods.

2. Object Association

ORANGE MONKEY TEAM

It is the interaction of stand alone objects of one class with other objects of anther class.
It can be of one of the following types,

- Simple Association
- Composition
- Aggregation

Simple Association

The two interacting objects have no intrinsic relationship with other object. It is the weakest link between objects. It is a reference by which one object can interact with some other object.
Customer gets cash from cashier
Employee works for a company
Ali lives in a house
Ali drives a car

It is generally called as "**association**" instead of "**simple association**"

Kinds of Simple Association

Simple association can be categorized in two ways,

- With respect to direction (navigation)
- With respect to number of objects (cardinality)

Kinds of Simple Association w.r.t Navigation

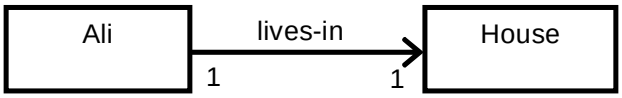
With respect to navigation association has the following types,

a. One-way Association
b. Two-way Association

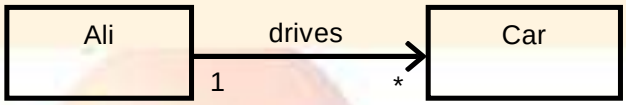
a. One-way Association

In One way association we can navigate along a single direction only, it is denoted by an arrow towards the server object.
Examples:

ORANGE MONKEY TEAM



- Ali lives in a House

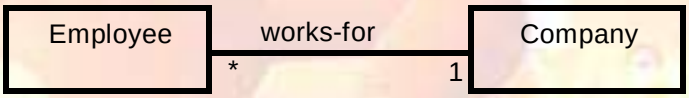


- Ali drives his Car

b. Two-way Association

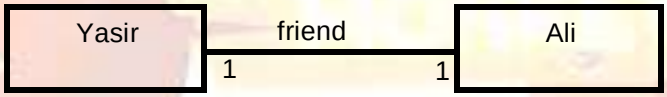
In two way association we can navigate in both directions, it is denoted by a line between the associated objects

Examples:



Employee works for company
Company employs employees

Two-way Association - Example



Yasir is a friend of Ali
Ali is a friend of Yasir

Kinds of Simple Association w.r.t Cardinality

With respect to cardinality association has the following types,

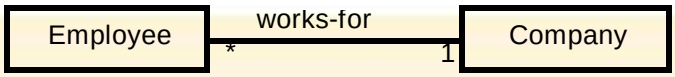
a. Binary Association
b. Ternary Association
c. N-ary Association

a. Binary Association

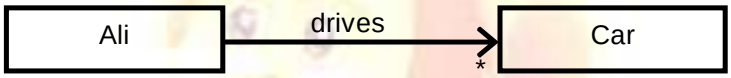
ORANGE MONKEY TEAM

It associates objects of exactly two classes; it is denoted by a line, or an arrow between the associated objects.

Example



Association "works-for" associates objects of exactly two classes



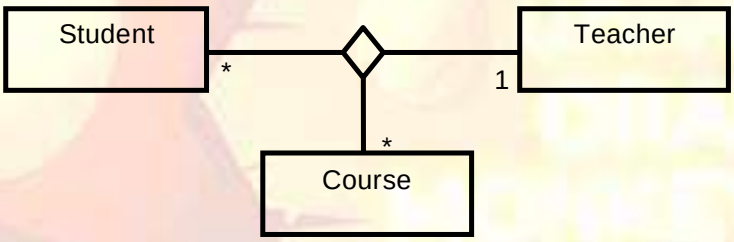
Association "drives" associates objects of exactly two classes

b. Ternary Association

It associates objects of exactly three classes; it is denoted by a diamond with lines connected to associated objects.

Example

Objects of exactly three classes are associated



c. N-ary Association

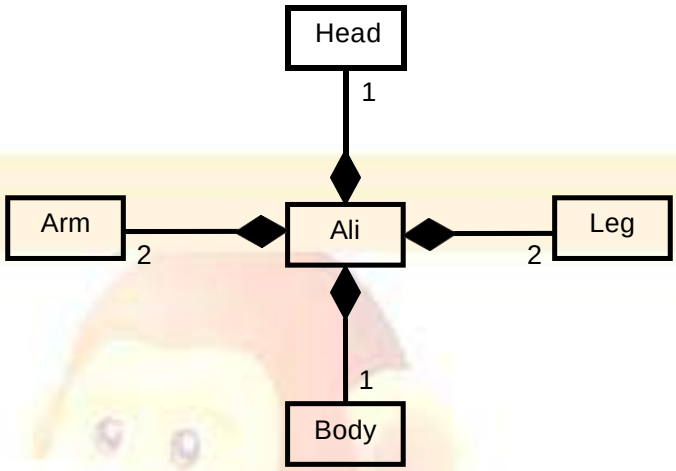
An association between 3 or more classes its practical examples are very rare.

Composition

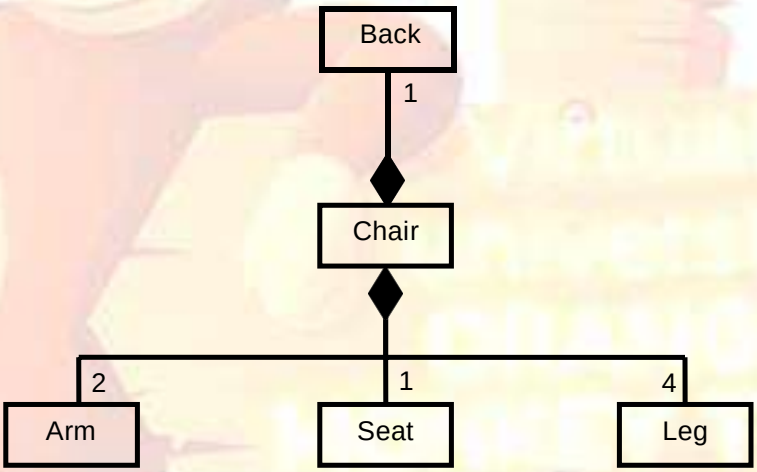
An object may be composed of other smaller objects, the relationship between the "part" objects and the "whole" object is known as Composition, Composition is represented by a line with a filled-diamond head towards the composer object

Example – Composition of Ali

ORANGE MONKEY TEAM



Example – Composition of Chair



Composition is stronger relationship:
Composition is a stronger relationship, because
Composed object becomes a part of the composer
Composed object can't exist independently

Example I

Ali is made up of different body parts

They can't exist independent of Ali

Example II

Chair's body is made up of different parts

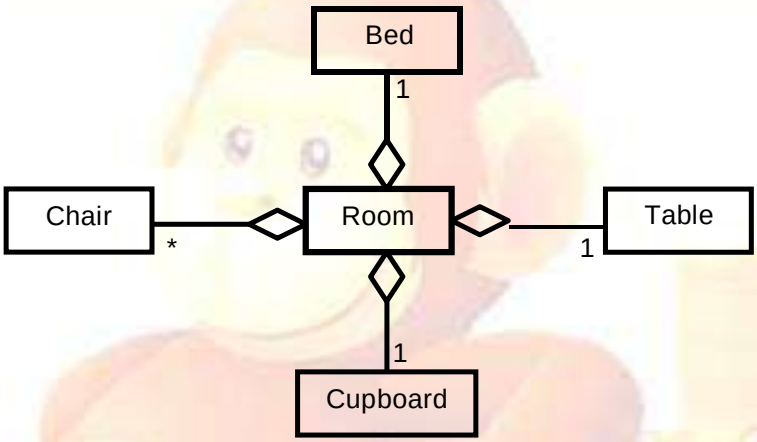
ORANGE MONKEY TEAM

They can't exist independently

Aggregation

An object may contain a collection (aggregate) of other objects, the relationship between the container and the contained object is called aggregation, Aggregation is represented by a line with unfilled-diamond head towards the container

Example – Aggregation



Example – Aggregation

Aggregation is weaker relationship

Aggregation is weaker relationship, because

- Aggregate object is not a part of the container
- Aggregate object can exist independently

Example I
Furniture is not an intrinsic part of room
Furniture can be shifted to another room, and so can exist independent of a particular room

Example II
A plant is not an intrinsic part of a garden
It can be planted in some other garden, and so can exist independent of a particular garden

http://www.codeproject.com/KB/cpp/oopuml.aspx

ORANGE MONKEY TEAM

Lecture No.06

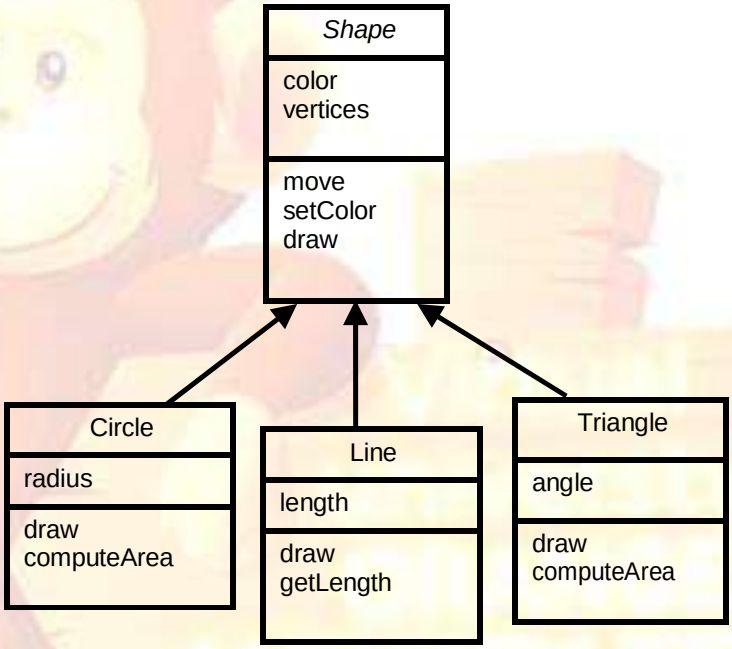
Class Compatibility

A class is behaviorally compatible with another if it supports all the operations of the other class. Such a class is called subtype. A class can be replaced by its subtype.

Derived class is usually a subtype of the base class. It can handle all the legal messages (operations) of the base class. Therefore, base class can always be replaced by the derived class.

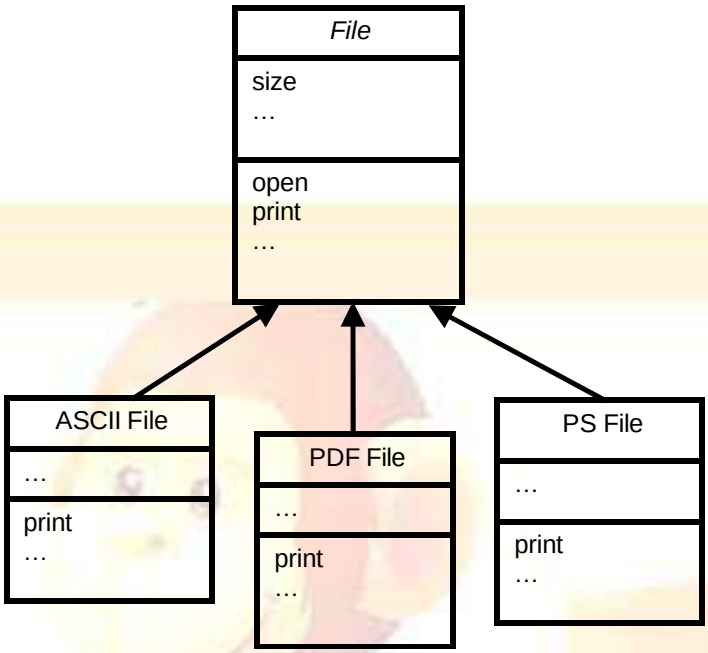
Examples

Child class also includes characteristics of its base class.



All the three derived class are behaviourally compatible with base class.

ORANGE MONKEY TEAM



Wherever the file class is it can be replaced by any of its child classes.

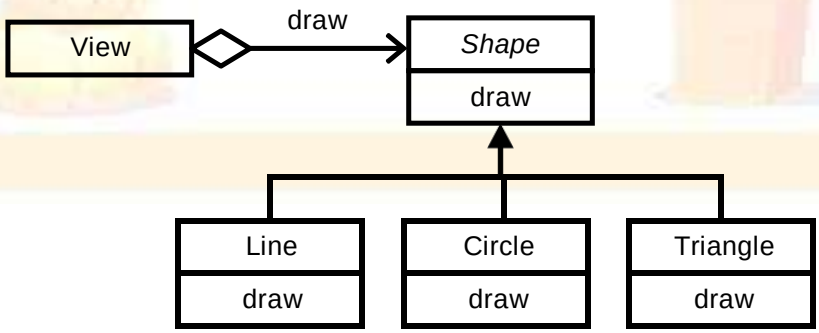
Polymorphism

It is also essential component of object oriented modeling (paradigm).
In general, polymorphism refers to existence of ***different forms*** of a single entity. For example, both Diamond and Coal are different forms of Carbon.

Polymorphism in OO Model

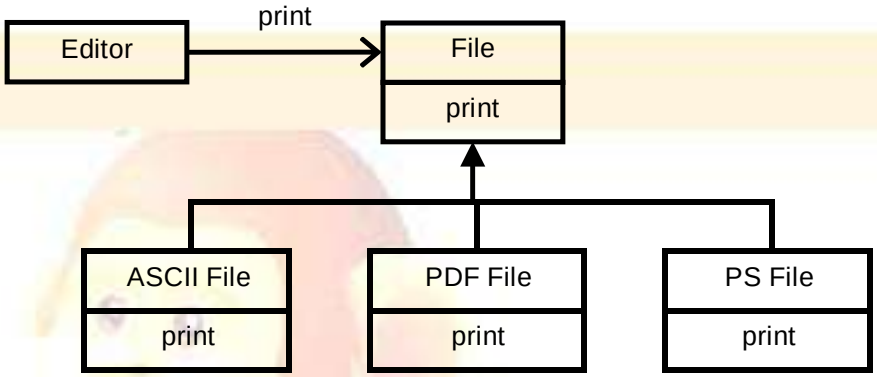
In OO model, polymorphism means that different objects can behave in different ways for the same message (stimulus). Consequently, sender of a message does not need to know exact class of the receiver.
Sender sends message to receiver and appropriate method is called on receiver side.

Example – Polymorphism



ORANGE MONKEY TEAM

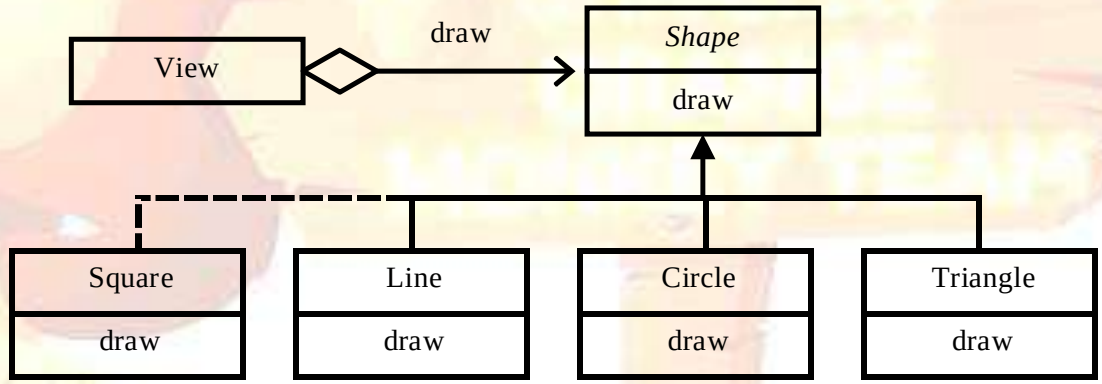
Shape class hierarchy shape is base class and there are three child classes line circle , triangle. View send draw method to shape class and draw is called according to the nature of actual object present.



Editor sends message print to file class and print is called based on the actual child object of file class message is same and appropriate execution will be done.

Polymorphism – Advantages

Messages can be interpreted in different ways depending upon the receiver class
New classes can be added without changing the existing model



In general, polymorphism is a powerful tool to develop flexible and reusable systems

Object-Oriented Modeling an Example

Problem Statement

Develop a graphic editor that can draw different geometric shapes such as line, circle and triangle. User can select, move or rotate a shape. To do so, editor provides user with a menu listing different commands. Individual shapes can be grouped together and can behave as a single shape.

ORANGE MONKEY TEAM

Identify Classes

Extract nouns in the problem statement

Develop a graphic **editor** that can draw different geometric **shapes** such as **line**, **circle** and **triangle**. **User** can select, move or rotate a **shape**. To do so, **editor** provides **user** with a **menu** listing different **commands**. Individual **shapes** can be grouped together and can behave as a single **shape**.

Eliminate irrelevant classes

Editor – Very broad scope. But it is the name of overall system and we are going to model it so we will not make its object. For example if we are going to model computer we will not make its object but its components however if it is component of some other system then it will behave as an object. So it is marked as irrelevant.

User – Out of system boundary, it is interacting with the system from outside of the system.

Add classes by analyzing requirements

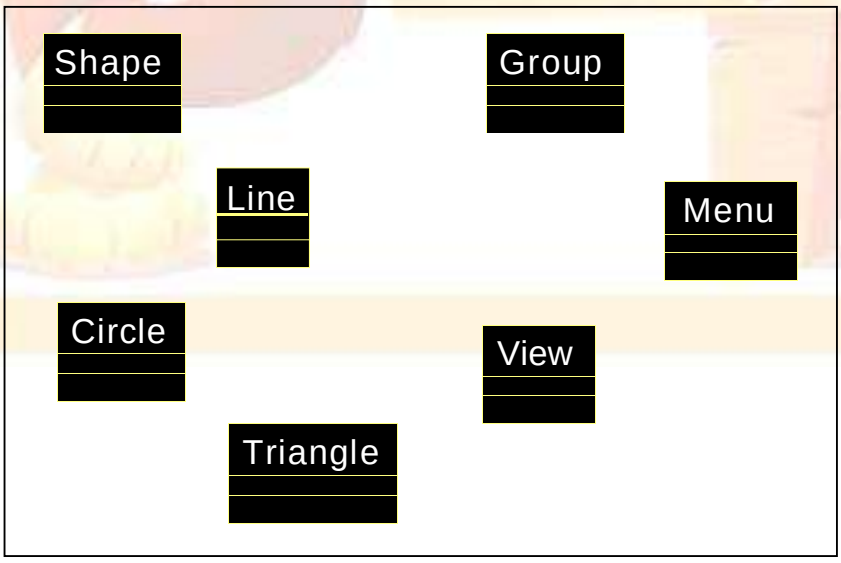
Group (of shapes) – required to behave as a shape so it should behave as an object in our system

"Individual shapes can be grouped together and can behave as a single shape"

View – graphic editor must have a display area to show the shapes. We made this object using domain knowledge.

- Shape
- Line
- Circle
- Triangle
- Menu
- Group
- View

So we have the following classes,



Finding Associations:

Next step is to find associations between the objects.

Identify Associations

Find relationships between objects,

1. **Extract verbs connecting objects,**

*"Individual shapes can be **grouped together**"*

- Group **consists of** lines, circles, triangles
- Group can also **consists of** other groups (Composition)

Line, circle and triangle have composition relationship.

2. **Verify access paths**

   a. *View contains (draws) shapes*

      - *View contains lines*
      - *View contains circles*
      - *View contains triangles*
      - *View contains groups*

So there is **Aggregation relationship** between shapes and View.

Menu sends message to View

So there is Simple **One-Way Association relationship** between Menu and View.

Identify Attributes of the identified objects

Extract properties of the object,

a. From the problem statement
   Properties are not mentioned
b. From the domain knowledge

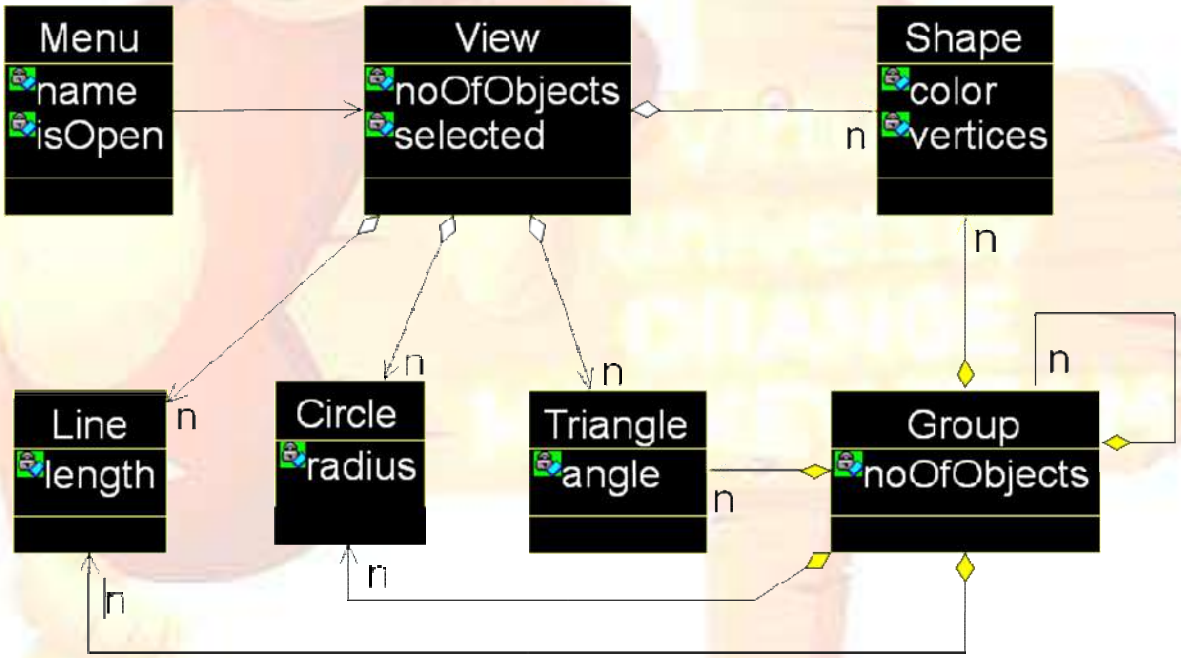
   - Line
     i. Color
     ii. Vertices
     iii. Length
   - Circle
     i. Color
     ii. Vertices
     iii. Radius
   - Triangle

i. Color
ii. Vertices
iii. Angle

- Shape
  i. Color
  ii. Vertices
- Group
  i. noOfObjects
- View
  i. noOfObjects
  ii. selected
- Menu
  i. Name
  ii. isOpen

Object Model – Graphic Editor

Object model so far is shown below,



Identify Operations

Extract verbs connected with an object

Develop a graphic editor that can **draw** different geometric shapes such as line, circle and triangle. User can **select**, **move** or **rotate** a shape. To do so, editor **provides** user with a menu listing different commands. Individual shapes can be **grouped** together and can **behave** as a single shape.

Eliminate irrelevant operations

ORANGE MONKEY TEAM

Develop – out of system boundary
Behave – have broad semantics

Following are selected operations:

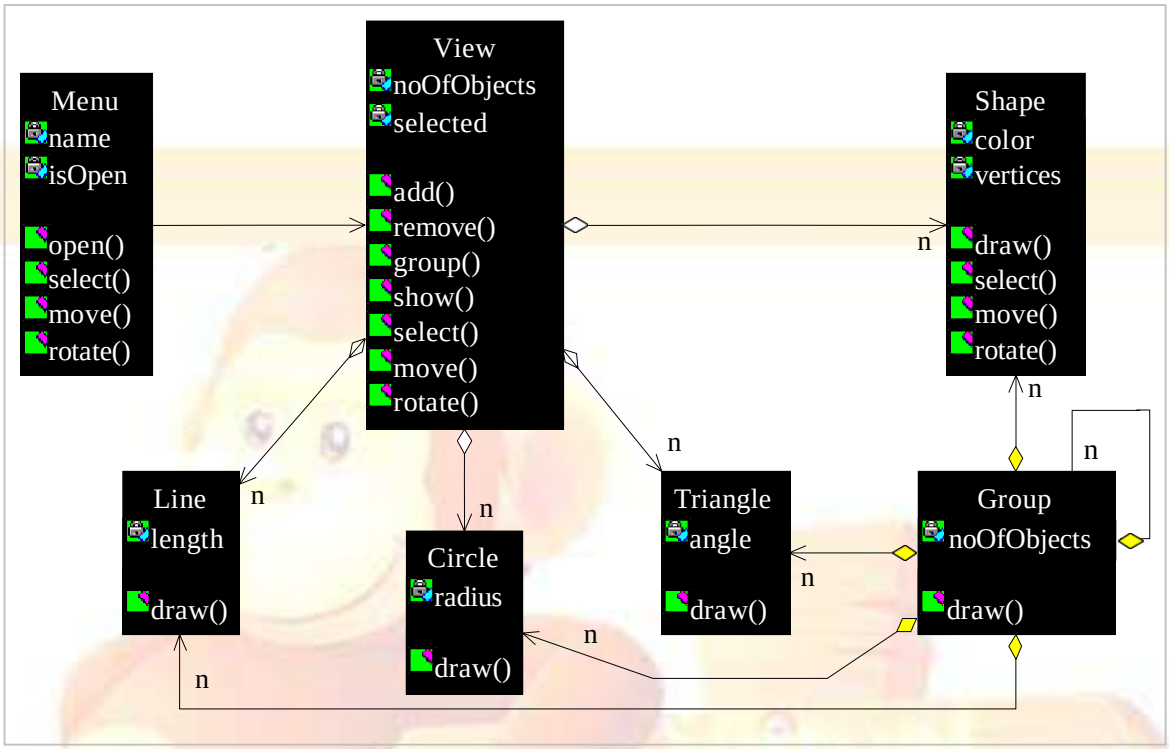
- Line
  - Draw
  - Select
  - Move
  - Rotate
- Circle
  - Draw
  - Select
  - Move
  - Rotate
- Triangle
  - Draw
  - Select
  - Move
  - Rotate
- Shape
  - Draw
  - Select
  - Move
  - Rotate
- Group
  - Draw
  - Select
  - Move
  - Rotate
- Menu
  - Open
  - Select
  - Move
  - Rotate

Extract operations using domain knowledge

- View
  - Add
  - Remove
  - Group
  - Show
  - Select
  - Move
  - Rotate

ORANGE MONKEY TEAM

Object Model after adding operations:



Identify Inheritance

a. Search lines like "is a kind of" by looking at keywords like "such as", "for example", etc

"...shapes such as line, circle and triangle..."

⇨ Line, Circle and Triangle inherits from Shape

By analyzing requirements

"Individual shapes can be grouped together and can behave as a single shape"

⇨ Group inherits from Shape

Refining the Object Model

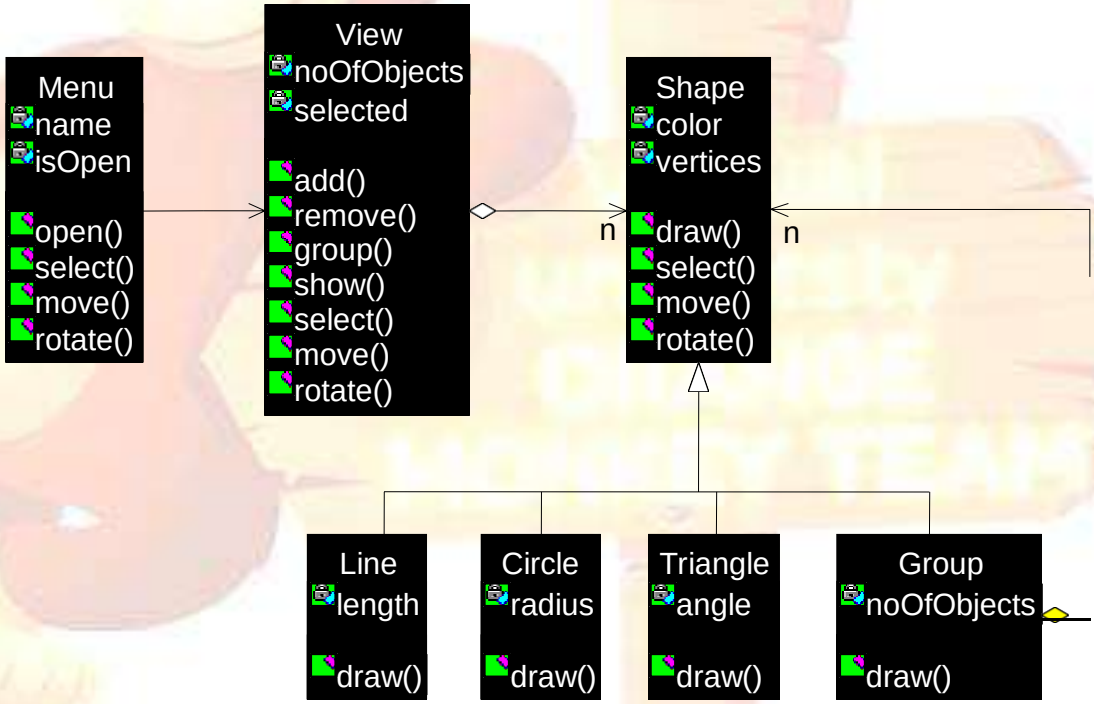
Application of inheritance demands an iteration over the whole object model
In the inheritance hierarchy,

- All attributes are shared
- All associations are shared
- Some operations are shared
- Others are overridden

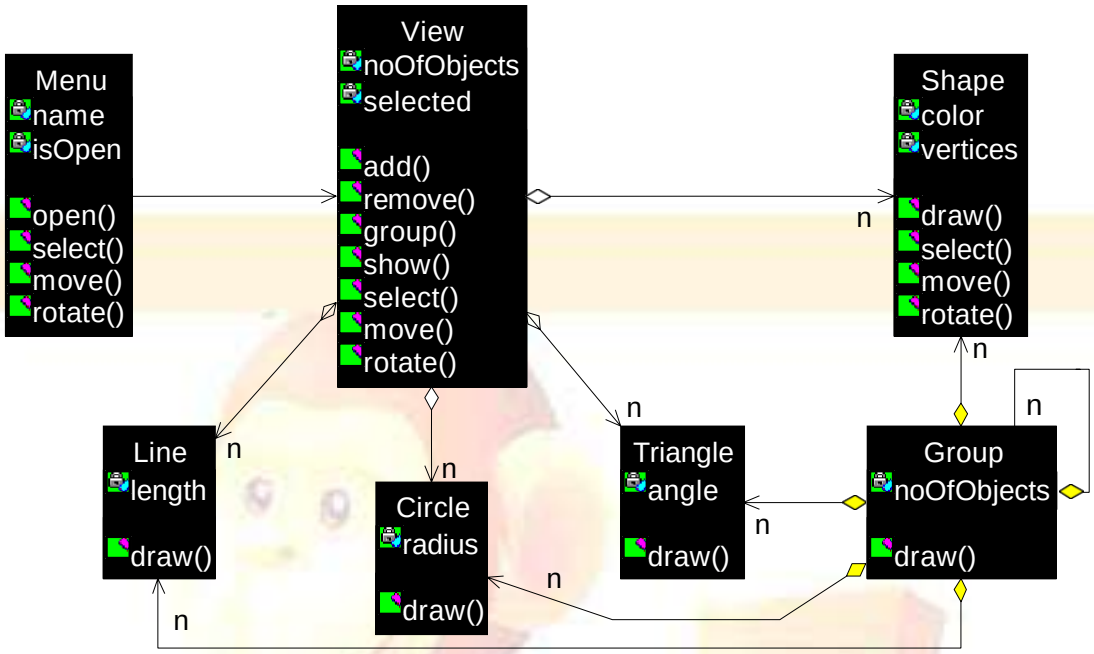
ORANGE MONKEY TEAM

Refining the Object Model

- Share associations
  - View contains all kind of shapes
  - Group consists of all kind of shapes
- Share attributes
  - Shape – Line, Circle, Triangle and Group
    - Color, vertices
- Share operations
  - Shape – Line, Circle, Triangle and Group
    - Select
    - Move
    - Rotate
- Share the interface and override implementation
  - Shape – Line, Circle, Triangle and Group
    - Draw



ORANGE MONKEY TEAM



ORANGE MONKEY TEAM

Lecture No.07

The basic concept "**Object**" of *Object Orientation (thinking in terms of objects)* is realized using classes in programming languages.

Class

It is a way (Mechanism) given by c++ to realize objects in a program. It is concrete implementation of objects in c++. We capture any object attributes and behaviour in a programming language using classes.

In other words it can be defined as facility given by c++ to create new types according to our requirement. (Class is composite data type made from basic c++ types like integers, chars and float).

Example:

Consider the examples of entity lion there are many lions but all lions will have similar attributes and behaviour.

Similarly consider student object all students have separate existence but all students have similar attributes and they exhibit similar behaviour.

When we hear word student or think about student a sketch comes in our mind for student along with its attributes and behaviour. The attributes of student comes in our mind are its name, roll no, class, degree so on. Similarly the behaviour of student comes in our mind are study, register and many more.

We need to capture the characteristic features of any object (attributes and behaviour) in the programming language. The concept of class is used for this purpose.

Now consider the scenario having many interacting objects: a **University System** having many objects like student, subject, classroom, and teacher so on…we will realize all these objects in our software using classes. These all object will use the services of each other for example student will ask teacher to teach him. This approach is closer to real life instead of having simple functions being called from main here these objects will call each other to get their services. This is the reason *we say that object oriented programming mimics real life*.

Uses

Objects are structured in terms of class so our problem becomes easier to understand in the terms c++ program.

We can implement interactions easily in terms of classes.

Student objects will interact with each other to take and give services to each other as happens in real life and mapped in object oriented programming approach.

Now we see how class mechanism helps us in implementing real life concept.

Type in C++

We implement generic concepts using types.

ORANGE MONKEY TEAM

We have to model generic concept of Student. But there is no built in type **student** in c++ like built-in c++ type's **int** or **float**. Class is mechanism in c++ that will allow us to define student as user defined type, similarly generic concept circle will also be implemented in the same way. User define types will be,

- Student in student management system
- Circle in a drawing software

As objects have attributes and behaviour so corresponding classes will also have data members and methods as shown below,

Ali	**Corresponding class**

Characteristics (attributes)	/*c++ code for class Person, we can create any object like Ali from it*/
	class Person {
	private: /* attributes are generally made private*/
	char name[]; /*char array to store name*/
	int age; /*int age to store age*/
	public: /* methods are generally made public*/
	Person(); /*constructor used to initialize data members*/
	void walks(); /* method walk */
	void eats(); /*method eats*/
	}

Name Age	
Behavior (operations)	
Walks Eats	

a. object **b. class code**

Abstraction

We only include those details in the system that are required for making a functional system so we will leave out irrelevant attributes and behaviour from our objects.
Take the example of student,

Student

- Name
- Address
- Sibling
- Father Business

Relevant to our problem

Not relevant to our problem

Defining a New User Defined Type

There are two ways to create user defined types for objects in c++ these are,

Structure Definition:

Partially we can use Structures to define an object

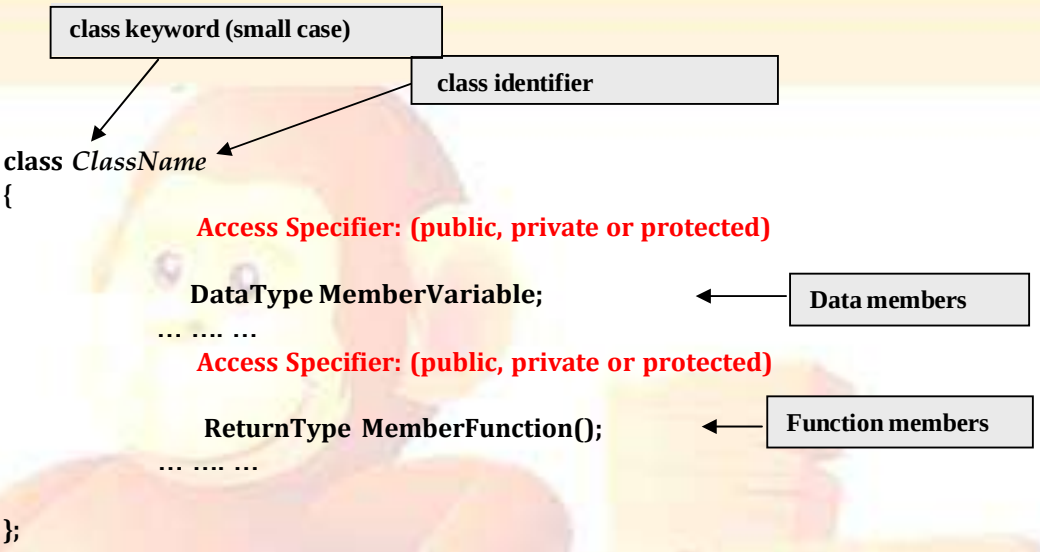
Struct{

};

ORANGE MONKEY TEAM

In c we can not define functions in a structure however in c++ we can add functions in both structure and classes.

Class Definition:



Example

```
class Student
{
    private:
        int rollNo;
        char *name;
        float CGPA;
        char *address;

    public:
        void setName(char *newName);
        void setRollNo(int newRollNo);

};
```

Member Variables

Member Functions

Why Member Functions:
They model the behaviors of an object,

Objects can make their data invisible (in accordance with the principle of data hiding). Setters and getters functions are provided by class to access the its members it also minimizes the changes to move the objects in inconsistent state as we can write checks in our setter functions for example we can check that whether the user has entered correct age value and has not entered negative value for age.
Object remains in consistent state

Example:

We can check that the entered roll number by user is positive or negative,

```
Student aStudent;
aStudent.rollNo = 514;
aStudent.rollNo = -514;        //Error
```

Object and Class:

Object is an instantiation of a user defined type or class. Once we have defined a class we can create as many objects for that class as we require.

Declaring class variables

Variables of classes (objects) are declared just like variables of structures and built-in data types as follows,

TypeName VariableName;

int var; // declaring built in int data type variable
Student aStudent; // declaring user defined class Student object

Accessing members

Members of an object can be accessed using,

a. dot operator (.) to access via the variable name

```
Student aStudent;  // declaring Student object
aStudent. rollNo = 5;
```

b. arrow operator (->) to access via a pointer to an object

```
Student * aStudent = new Student();
// declaring and initializing Student pointer
aStudent->rollNo = 5;
```

Note: it is against the principle of OOP to access the data members directly using object of class as we have done above. This code is given for example only we should write assessor functions (setters and getters) wherever we want to access the members of the class.

Member functions are accessed in the similar way using dot or arrow operator.

Example

```
class Student{
  int rollNo;
  void setRollNo(int aNo);
};

  Student aStudent;
  aStudent.setRollNo(5);
```

ORANGE MONKEY TEAM

```
Student *ptr_student = new Student();
ptr_student->setRollNo(5);
```

Access specifiers

These are used to enforce access restrictions to members of a class, there are three access specifiers,

1. **'public'** is used to tell that member can be accessed whenever you have access to the object
2. **'private'** is used to tell that member can only be accessed from a member function
3. **'protected'** to be discussed when we cover inheritance

Example

```
class Student{
private:
        char * name;
        int rollNo;
public:
        void setName(char *);
        void setRollNo(int);
...
};
```

Can not be accessed outside the class

Can be accessed outside the class

Example Program

```
class Student{
char * name;
        int rollNo;
public:
        void setName(char *);
void setRollNo(int aNo);
};
void Student::setName(char * aName){
if (strlen(aName) > 0)
    {
        name = new char[strlen(aName)];
        strcpy(name,aName);
    }
}

void Student::setRollNo(int arollNo){
if(arollNo > 0)
rollNo = arollNo;
}

int main(){
        Student aStudent;
 aStudent.rollNo = 5;
```

ORANGE MONKEY TEAM

```
/* Error: we can not access private member of the class. */
  aStudent.name = "Ali";
/* Error: we can not access private member of the class */
        aStudent.setRollNo(1);
        aStudent.setName("Ali");
/* Correct way to access the data member using public setter functions */
}
```

Default access specifier

When no access specifier is mentioned then default access specifier is **private**.

Example

```
class Student
{
   char * name;
   int RollNo;
};
```

Equivalent

```
class Student
{
private:
   char * name;
   int RollNo;
};
```

Example

We should use keyword public before the methods of the class as given below of will not use public keyword they will also be treated as private ad will not be accessible outside the class as shown below,

```
class Student
{
        char * name;
        int RollNo;
        void SetName(char *);
};
Student aStudent;
aStudent.SetName(Ali);
```

Error in accessing **SetName** it will be inaccessible

Corrected code will be,

```
class Student
{
        char * name;
        int RollNo;
public:
        void setName(char *);
```

ORANGE MONKEY TEAM

```
};
Student aStudent;
aStudent.SetName("Ali");
```

Lecture No.08

Member Functions

- Member functions are the functions that operate on the data encapsulated in the class
- Public member functions are the interface to the class

Defining Member Functions

We can define member functions in two ways,

a. We can define member functions of the class inside the class definition when we define any class in our program.

OR

b. We declare member function inside the class definition and declare them outside the class.

In this case class definition is added in ClassName.h file and class implementation code is added in ClassName.cpp file.

Function definition inside the class:

General Syntax:

```
class ClassName {
        …
        public:
        ReturnType  FunctionName() {
                …
        }
};
```

Example:

Define a class of student that has a roll number. This class should have a function that can be used to set the roll number

```
class Student{
        int rollNo;
public:
        void setRollNo(int aRollNo){
                rollNo = aRollNo;
        }
};
```

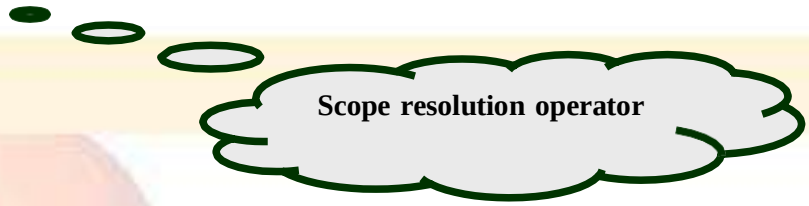
Function definition outside class

General Syntax:

```
class ClassName {
        …
```

ORANGE MONKEY TEAM

```
        public:
        ReturnType FunctionName();
};
ReturnType ClassName::FunctionName()
{
        …
}
```



Example

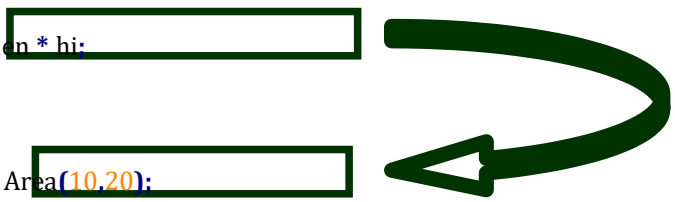
```
class Student{
        …
        int rollNo;
public:
        void setRollNo(int aRollNo);
};
void Student::setRollNo(int aRollNo){
        …
        rollNo = aRollNo;
}
```

Inline Functions

- Inline functions is a way used by compilers to improve efficiency of the program, when functions are declared inline normal process of function calling (using stack) is not followed instead function code is added by compiler at all points where these functions have been called. Basic concept behind inline functions is that they are functions in our code but in compiler generated files these functions code is added by compiler at all places where they were called in the code.
- Normally small size functions that need to be called many times during program execution are declared inline. Inline functions decrease the code execution time because program in their case doesn't involve function call overhead.
- Keyword 'inline' is used to request compiler to make a function inline.
- However using inline keyword with function doesn't guarantee that function will definitely in inlined, it depends on the compiler if it finds it can make function inline it does so otherwise it ignores the keyword inline and treat the function as normal function.

Example

```
inline int Area(int len, int hi)
{
        return len * hi;
}
int main()
{
        cout << Area(10,20);
```



ORANGE MONKEY TEAM

```
    return 0;
}
```

Inline Functions

The functions defined inside the class are by default inline (whether we mention keyword inline with them or not)
In case we define function outside the class then we must use the keyword 'inline' to make the function inline.
However compiler decides whether it will implement these functions code as inline or not.

Example
Inline function inside the class:

```
class Student{
        int rollNo;
public:
        void setRollNo(int aRollNo){
                ...
                rollNo = aRollNo;
        }
};
```

Example
Inline function outside the class:

```
class Student{
        ...
        public:
        inline void setRollNo(int aRollNo);
};
void Student::setRollNo(int aRollNo){
        ...
        rollNo = aRollNo;
}
class Student{
        ...
        public:
        void setRollNo(int aRollNo);
};
inline void Student::setRollNo(int      aRollNo){
        ...
        rollNo = aRollNo;
}
class Student{
        ...
        public:
```

ORANGE MONKEY TEAM

```
        inline void setRollNo(int aRollNo);
};
inline void Student::setRollNo(int
        aRollNo){
        …
        rollNo = aRollNo;
}
```

Constructor

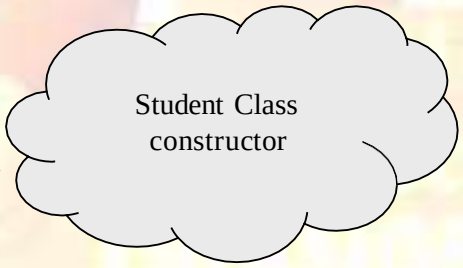
Constructor is used to **initialize** the objects of a class. Constructor is used to ensure that object is in well defined state at the time of creation.
The constructor of a class is automatically generated by compiler however we can write it by our self also.
Constructor is automatically called when the object is created. Constructors are not usually called explicitly by us.

Constructor Properties

- Constructor is a special function having same name as the class name
- Constructor does not have return type
- Constructors are commonly public members

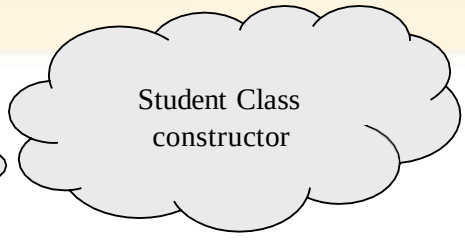
Example

```
class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;

        }
};

int main()
{
        Student aStudent;
        /*constructor is implicitly called at this point*/
}
```



We can assure that constructor is called automatically by adding cout statement in constructor as shown below,

```
#include <iostream>
using namespace std;

class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;
                cout<<"I am constructor of Student class…\n";
```



ORANGE MONKEY TEAM

```
        }
};

int main()
{
        Student aStudent;
        /*constructor is implicitly called at this point*/
system("pause");
return 0;
}
```

Default Constructor

- Constructor without any parameter or with all parameters with default values is called default constructor
- If we do not define a default constructor the compiler will generate a default constructor
- Compiler generated default constructor is called implicit and user written default constructor is called explicit
- This compiler generated default constructor initialize the data members to their default values
- If we have given any constructor for a class whether it is
    - our own **explcit default constructor** (i.e parameterless or with parameters having default values)
      or
    - our own constructor with parameters

Then compiler will not create implicit default constructor[5].

Example

Consider the class student below it has no constructor so compiler will generate one for it,

```
class Student
{
        int rollNo;
        char *name;
        float GPA;
public:
        ...             //no constructors
};
```

Code of Compiler generated implicit default constructor

```
{
```

[5] compiler generated default constructor is called implicit and user written default constructor is called explicit

ORANGE MONKEY TEAM

```
        rollNo = 0;
        GPA = 0.0;
        name = NULL;
}
```

Constructor Overloading

We can write constructors with parameters as well. These parameters are used to initialize the data members with user supplied data (passed as parameter). The example is shown below, here example Student class has four constructors their prototypes are,

1. **Student();** /* explicit default parameterless constructor */
2. **Student(char * aName);** /* constructor with one parameter* /
3. **Student(char * aName, int aRollNo);** /* constructor with two parameters */
4. **Student(int aRollNo, int aRollNo, float aGPA);** /* constructor with three parameters */

Example

```
class Student{
        int rollNo;
        char *name;
        float GPA;
public:
        Student(); /* explicit default constructor */
        Student(char * aName); /* constructor with one parameter* /
        Student(char * aName, int aRollNo); /* constructor with two parameters */
        Student(int aRollNo, int aRollNo, float aGPA);    /* constructor with three parameters */

};
Student::Student(){

        rollNo = 0;
        name = NULL; // to indicate it is pointing to nothing at this moment otherwise it can generate erroneous code.
        GPA = 0.0;

}
Student::Student(int aRollNo){

        if(aRollNo < 0){
                rollNo = 0;
        }
        else {
        rollNo = aRollNo;
        }
name = NULL;
```

ORANGE MONKEY TEAM

```
}
Student::Student(int  aRollNo,
                                    char * aName){

        if(aRollNo < 0){
                rollNo = 0;
        }
        else {
        rollNo = aRollNo;
        }

 if (strlen(aName) > 0)
                        {
        name = new char[strlen(aName)+1];
           strcpy(name,aName);
                        }
 else
       {
name = NULL;
}

}
```

We can create this Student class object using any one of these constructors as follows,

```
int main()
{
        Student student1;  // default constructor will be used
        Student student2("Name"); // one parameter constructor will be used
        Student student3("Name", 1); // two parameter constructor will be used
        Student student4("Name",1,4.0); // three parameter constructor will be used
}
```

Constructor Overloading

We can use default parameter values to reduce the writing effort in that case we will have to write only one constructor and it will serve the purpose of all constructors as given below,

Example

```
Student::Student(       char * aName = NULL,  int aRollNo= 0,  float aGPA = 0.0) {
…

}
```

It is equivalent to all three constructors,

Student();
Student(char * aName);
Student(char * aName, int aRollNo);
Student(char * Name, int aRollNo, float aGPA);

It will use default values if values are not passed as arguments while creating objects) it is described in code given below,

```
int main()
{
    Student student1;  /*char * aName = NULL, int aRollNo= 0, float aGPA = 0.0*/

    Student student2( "Name" ); /*char * aName = Name, int aRollNo= 0, float aGPA = 0.0*/

    Student student3( "Name", 1 ); /*char * aName = Name, int aRollNo= 1, float aGPA = 0.0*/

    Student student4( "Name", 1 , 4.0); /*char * aName = Name, int aRollNo= 1, float aGPA = 4.0*/


}
```

Copy Constructor

Copy constructors are used when:

- Initializing an object at the time of creation (we want to create an object with state of a pre existing object)
- When an object is passed by value to a function (As you know temporary copy of object is created on stack so we need copy constructor to create that temporary object with the state of actual object being passed).

Example

```
void func1(Student student){
...
}
```

Copy constructor will be called to create temporary student object

```
int main(){

    Student studentA;
    Student studentB = studentA;
```

ORANGE MONKEY TEAM

```
        func1(studentA);

}
```

Copy constructor will be called as we are creating studentB in terms of studentA.

Copy Constructor (Syntax)

```
Student::Student( const Student &obj){

    /*copying values to newly created object*/
   rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;

}
```

As was the case with default constructor compiler also generates copy constructor by itself however we can override that copy constructor by writing our own copy constructor as shown in example below,

```
#include <iostream>
using namespace std;

class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;
                cout<<"I am default constructor of Student class...\n";
        }

Student(const Student &obj){
         cout<<"I am copy constructor of Student class\n";
  rollNo = obj.rollNo;

}
};

int main()
{
        Student aStudent;
        /*default constructor is implicitly called at this point*/

        Student bStudent = aStudent;
        /*copy constructor is implicitly called at this point*/

system("pause");
```

Student class default constructor

Student class copy constructor

```
return 0;
}
```

Shallow Copy

- When we initialize one object with another then the compiler copies state of one object to the other using copy constructor by assigning data member values of previous object to newly created object.

Shallow copy using default Copy Constructor (Syntax)

```
Student::Student(     const Student & obj ){

        rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;
}
```

- This kind of copying is called shallow copying

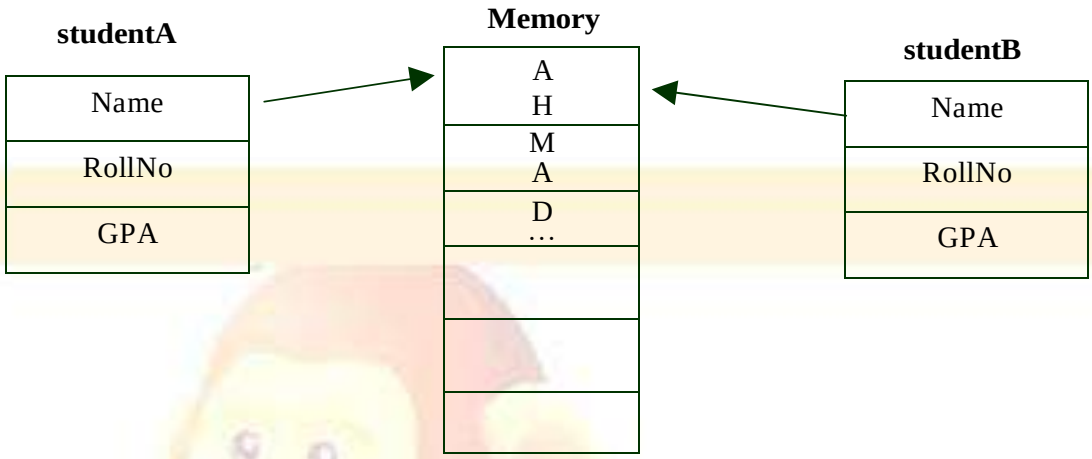
Example

Student studentA;
Student studentB = studentA; /*Shallow copy: compiler will use copy constructor to assign studentA values to newly created object studentB*/

Shallow copy works fine if our class doesn't include dynamic memory allocation but in case of dynamic memory allocation it leads to dangling pointer problem as explained below,

Problem is Shallow Copy

Student class data member name of char * type is added to store the name of student and it is using **dynamic memory** according to the length of name entered by user for student.



Object studentB is also pointing to memory allocated object studentA for Student class data member **name** of char * type, now there are two problems with this sort of copying,

Suppose we delete first object studentA for some reason then its destructor will also free memory allocated by it hence memory area containing name "AHMAD" will also be freed and will be given to some other application by operating system, but studentB member **name** is still pointing to that area so issue of *"Dangling Pointer"* will arose. [Pointer pointing to incorrect memory location]. If we will try to print the name of object studentB our program will terminate abnormally as it was pointing memory of some other applications.
Secondly if for some reason we change name of studentA the value of object studentB will also be changed as it pointing to same memory location.

We resolve these two issues using deep copy.

Deep Copy

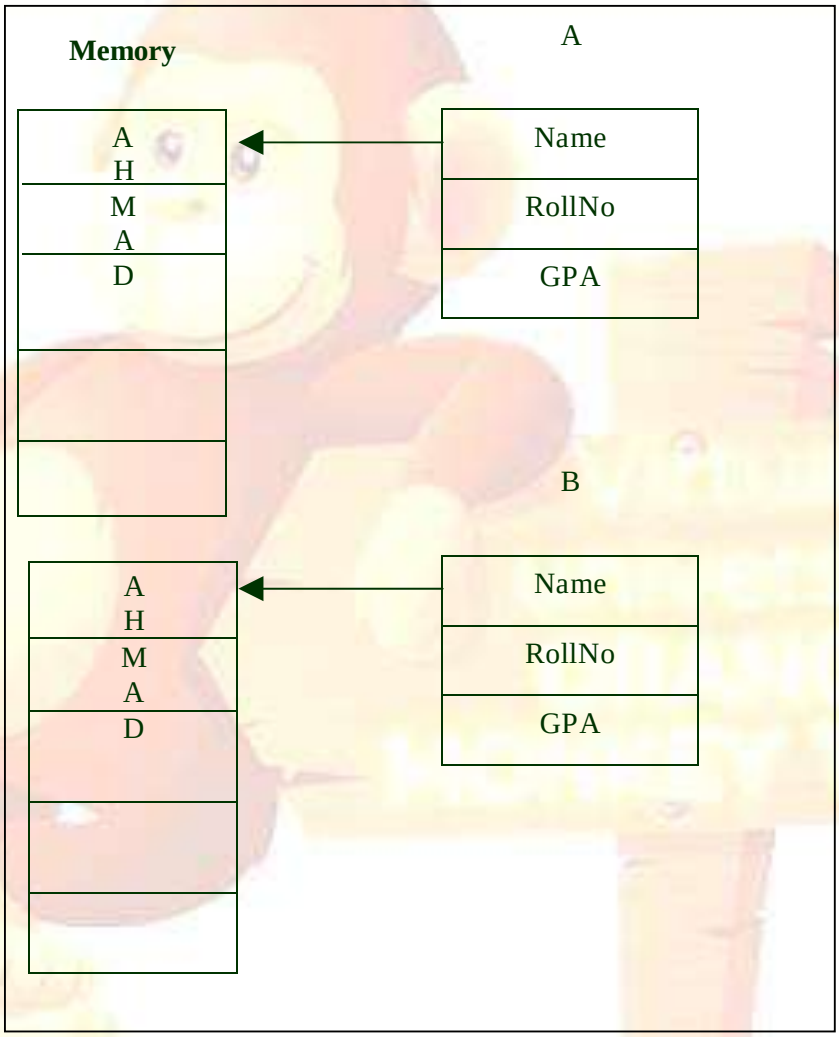
We write our own deep copy code in copy constructor so that when we create new object from an existing object using copy constructor we also allocate new dynamic memory for data members involving dynamic memory as shown below,

```
Student::Student( const Student & obj){

        int len = strlen(obj.name);
        name = new char[len+1];
        // assignming new dynamic memory to data member name of char * type for
newly created object*/
        strcpy(name, obj.name);
        …
        //copy rest of the data members in the same way
}
```

ORANGE MONKEY TEAM

Example

```
Student studentA;
Student studentB = studentA; // now copy constructor will  perform  deep
copy (separate memory for both objects)
```



In case our class doesn't involve dynamic memory then default copy constructor that performs shallow copy works fine.
In case our class has any data member involving dynamic memory we have to write our own code to do deep copy.

ORANGE MONKEY TEAM

Lecture No.09

Review

Copy Constructor

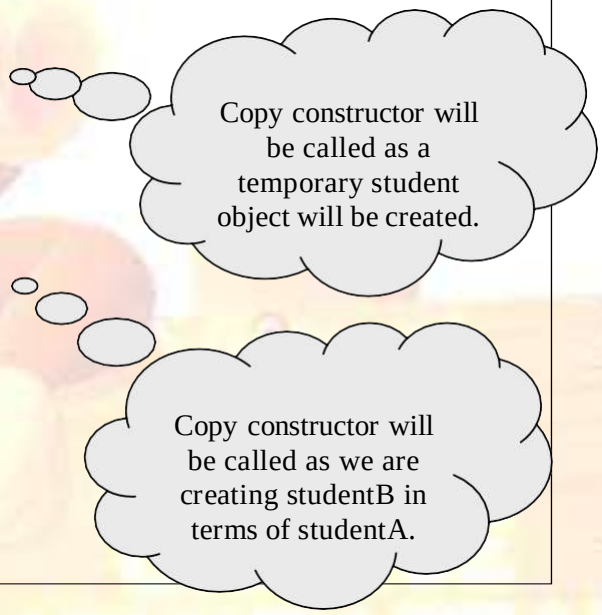
Copy constructors are used when:

Initializing an object at the time of creation (we want to create an object with state of a pre existing object)

When an object is passed by value to a function (As you know temporary copy of object is created on stack so we need copy constructor to create that temporary object with the state of actual object being passed).

Example

```
void func1(Student student){
...
}

int main(){

        Student studentA;
        Student studentB = studentA;
        func1(studentA);
        return 0;
}
```



Copy Constructor (Syntax)

```
Student::Student( const Student &obj){

   /*copying values to newly created object*/
  rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;
}
```

As was the case with default constructor compiler also generates copy constructor by itself however we can override that copy constructor by writing our own copy constructor as shown in example below,

As was the case with default constructor compiler also generates copy constructor by itself however we can override that copy constructor by writing our own copy constructor as shown in example below,

```
#include <iostream>
```

ORANGE MONKEY TEAM

```
using namespace std;

class Student{
        int rollNo;
public:
        Student(){
                rollNo = 0;
                cout<<"I am default constructor of Student class...\n";
        }

Student(const Student &obj){
         cout<<"I am copy constructor of Student class\n";
  rollNo = obj.rollNo;

}
};

int main()
{
        Student aStudent;
        /*default constructor is implicitly called at this point*/

        Student bStudent = aStudent;
        /*copy constructor is implicitly called at this point*/

system("pause");
return 0;
}
```

Student class default constructor

Student class copy constructor

Shallow Copy

When we initialize one object with another then the compiler copies state of one object to the other using copy constructor by assigning data member values of previous object to newly created object. This kind of copying is called shallow copying.

Shallow copy using default Copy Constructor (Syntax)

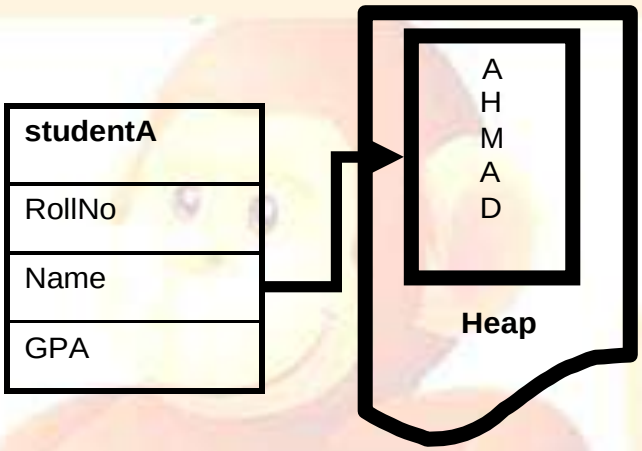
```
Student::Student(      const Student & obj ){

        rollNo = obj.rollNo;
        name = obj.name;
        GPA = obj.GPA;
}
```

This kind of copying is called shallow copying

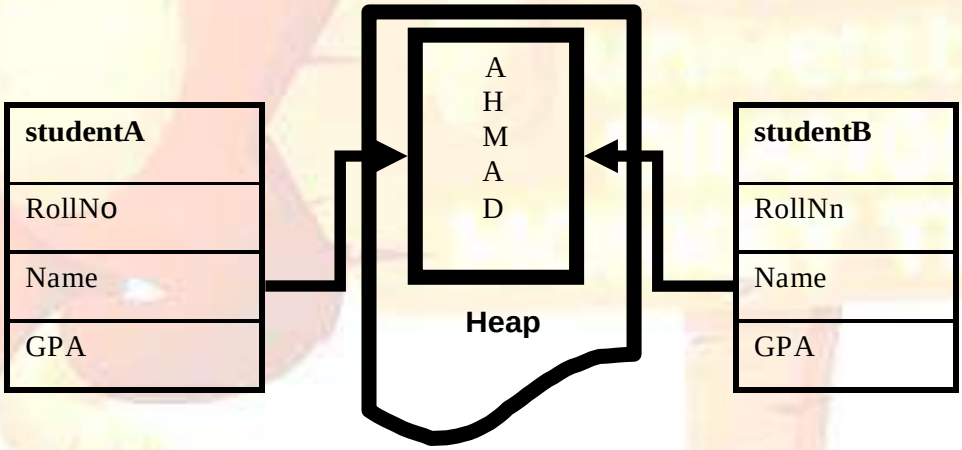
Example

Student studentA;
Student studentB = studentA; /*Shallow copy: compiler will use copy constructor to assign studentA values to newly created object studentB*/

Example
Student studentA("Ahmad");



Student studentB = studentA;




```
#include <iostream>
using namespace std;

class Student{
    char * name;
        int rollNo;
public:
        Student(char * aName, int arollNo){
      name = new char[strlen(aName)+1];
      strcpy(name,aName);
```

ORANGE MONKEY TEAM

```
                rollNo = arollNo;

        }
    Student(const Student &obj){
      name = obj.name;
      rollNo = obj.rollNo;


    }
    void showName(){
       cout<<name<<endl;

       }
    ~Student(){

         delete []name;

         }
};

int main()
{
        Student studentA("AHMAD",1);
        Student studentB = studentA;
        /*copy constructor is implicitly called at this point*/

        studentA.showName();
    studentB.showName();

    system("pause");
    return 0;
}
```

Shallow copy works fine if our class doesn't include dynamic memory allocation but in case of dynamic memory allocation it leads to dangling pointer problem as explained below.

Problem is Shallow Copy

Student class data member name of char * type is added to store the name of student and it is using **dynamic memory** according to the length of name entered by user for student.

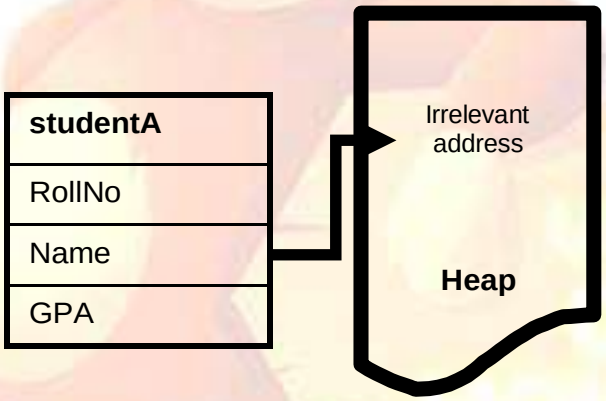
Student class data member **name** (*char **) of object studentB is also pointing to memory allocated for datamember **name** of object studentA, due to this there may be two kinds of problems.

Suppose we delete first object studentA for some reason then its destructor will also free memory allocated by it hence memory area containing name "AHMAD" will also be freed and will be given to some other application by operating system, but

studentB member **name** is still pointing to that area so issue of *"Dangling Pointer"* [Pointer pointing to incorrect memory location] will arose.
Same will happen if object studentB is deleted then studentA object data member name will become dangling pointer. This has been explained below,
Let us change code in main to make our second object studentB in a new local scope as shown below,

```
int main(){
Student studentA("Ahmad",1);

{
    Student studentB = studentA;
}
return 0;
}
```

Now if we will try to print the name of object studentA our program will not show any output as name is pointing to some irrelevant memory address,



Complete program code is given below,

```
#include <iostream>
using namespace std;

class Student{
     char * name;
          int rollNo;
public:
          Student(char * aName, int arollNo){
       name = new char[strlen(aName)+1];
       strcpy(name,aName);
                    rollNo = arollNo;

          }
   Student(const Student &obj){
```

ORANGE MONKEY TEAM

```
    name = obj.name;
    rollNo = obj.rollNo;

  }
  void showName(){
     cout<<name<<endl;

    }
  ~Student(){
          delete []name;
          }
};

int main()
{
        Student studentA("AHMAD",1);

  {
       Student studentB = studentA;
       /*copy constructor is implicitly called at this point*/
  }

        studentA.showName();

  system("pause");
  return 0;
}
```

Secondly if for some reason we change name of object studentA the value of object studentB will also be changed as it pointing to same memory location.

```
#include <iostream>
using namespace std;

class Student{
    char * name;
        int rollNo;
public:
        Student(char * aName, int arollNo){
     name = new char[strlen(aName)+1];
     strcpy(name,aName);
                rollNo = arollNo;

        }
  Student(const Student &obj){
    name = obj.name;
    rollNo = obj.rollNo;
```

```
    }

    void setName(char * aName){
        strcpy(name,aName);

        }

    void showName(){
        cout<<name<<endl;

        }
    ~Student(){

            delete []name;

            }
};

int main()
{
        Student studentA("AHMAD",1);
        Student studentB = studentA;
        /*copy constructor is implicitly called at this point*/

        studentA.showName();
        studentB.showName();


        studentA.setName("MOEEN");
        studentA.showName();
        studentB.showName();

    system("pause");
    return 0;
}
```

We resolve these two issues using deep copy.

Deep Copy

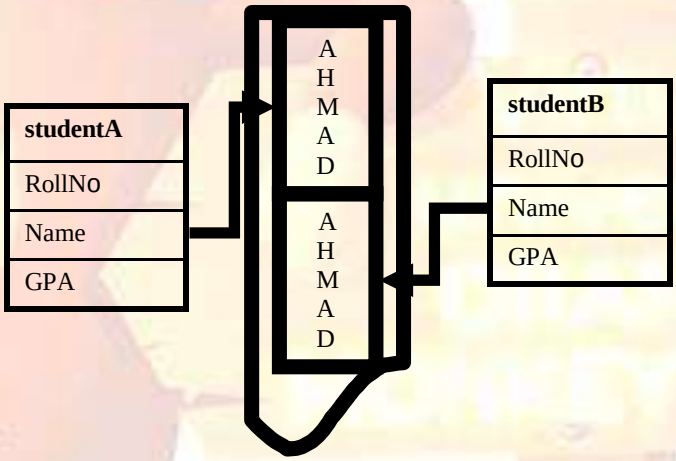
We write deep copy code in copy constructor so that when we create new object from an existing object using copy constructor we also allocate new dynamic memory for data members involving dynamic memory as shown below,

Student::Student(const Student & obj){

```
        int len = strlen(obj.name);
        name = new char[len+1]; // assignming new
/*dynamic memory to data member name of char * type for newly created object.*/
        strcpy(name, obj.name);
        …
        //copy rest of the data members in the same way
}
```

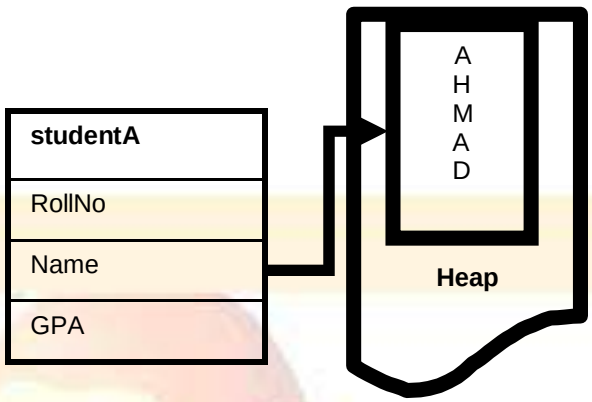
Now we see what happens when we created objects in main as shown below,

```
int main(){
Student studentA("Ahmad",1);

{
   Student studentB = studentA;
}
}
```



Now when we will execute code with object studentB in local scope our code still works fine and shows name for object studentA as now deletion of object studentB has no effect on object studentA as shown below,

ORANGE MONKEY TEAM



Example

```
#include <iostream>
using namespace std;

class Student{
   char * name;
         int rollNo;
public:
         Student(char * aName, int arollNo){
       name = new char[strlen(aName)+1];
       strcpy(name,aName);
                    rollNo = arollNo;

          }
   Student(const Student &obj){

      name = new char[strlen(obj.name)+1];
      strcpy(name,obj.name);
      rollNo = obj.rollNo;


   }

  void showName(){
       cout<<name<<endl;

       }
   ~Student(){

            delete []name;

            }
};

int main()
{
```

ORANGE MONKEY TEAM

```
        Student studentA("AHMAD",1);
    {
        Student studentB = studentA;
        /*copy constructor is implicitly called at this point*/
    }
        studentA.showName();

    system("pause");
    return 0;
}
```

ORANGE MONKEY TEAM

Important points about copy constructor:

1. In case our class doesn't involve dynamic memory then default copy constructor that performs shallow copy works fine.
2. In case our class has any data member involving dynamic memory we have to write our own code in copy constructor to perform deep copy.
3. Copy constructor is normally used to perform deep copy
4. If we do not make a copy constructor then the compiler performs shallow copy
5. Shallow copy performs bitwise copy.

Destructor

1. Destructor is used to free memory that is allocated through dynamic allocation. We have to free memory allocated using new operator by over self in destructor otherwise it remain occupied even after our program ends.
2. Destructor is used to perform house keeping operations.
3. Destructor is a function with the same name as that of class, but preceded with a tilde '~'

Example

```
class Student
{
        ...
public:
        ~Student(){
                if(name){
                        delete []name;
                }
        }
};
```

Overloading

Destructors cannot be overloaded.

Sequence of Calls

Constructors and destructors are called automatically
Constructors are called in the sequence in which object is declared
Destructors are called in reverse order

Example

```
Student::Student(char * aName){
        ...
                cout << aName << "Constructor\n";
        }
        Student::~Student(){
                cout << name << "Destructor\n";
        }
};
```

Example

ORANGE MONKEY TEAM

```
int main()
{
        Student studentB("Ali");
        Student studentA("Ahmad");
        return 0;
}
```

Example

Output:
Ali Constructor
Ahmad Constructor
Ahmad Destructor
Ali Destructor

Accessor Functions

In accordance to principle of information hiding data members of a class are declared as private so that outside world can not access the private data of the object only an interface is provided to outside world in the form of functions.
Accessor functions are also used to access private data of the object, we provide accessor functions to get and set private data members of the class.
We also add error checking code in accessor functions to reduce errors so that object doesn't move in illegal state.

Example – Accessing Data Member

Example - Setter

```
class Student{
        ...
        int rollNo;
public:
        void setRollNo(int aRollNo){
                rollNo = aRollNo;
        }
};
```

Avoiding Error

```
void Student::setRollNo(int aRollNo){
        if(aRollNo < 0){
          rollNo = 0;
        }
        else
        {
          rollNo = aRollNo;
        }
}
```

Example - Getter

```
class Student{
```

ORANGE MONKEY TEAM

```
        ...
        int rollNo;
public:
        int getRollNo(){
                return rollNo;
        }
};
```

Good Practice:
Never return a handle to a data member from getter function because you are never sure that function accessing the reference will not change the value of the variable.

this Pointer

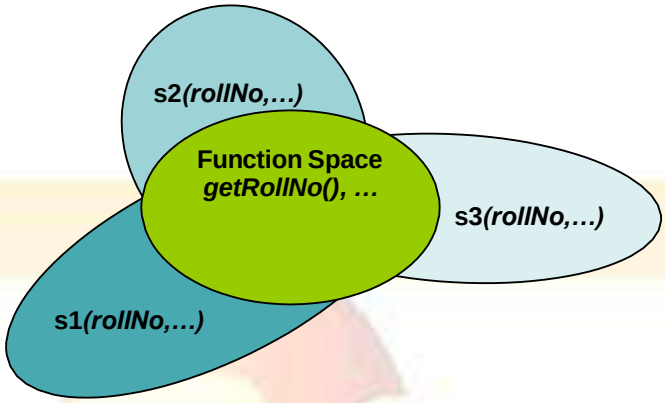
Consider the code of a general class given below,

```
class Student{
        int rollNo;
        char *name;
        float GPA;
public:
        int getRollNo();
        void setRollNo(int aRollNo);
...
};
```

The compiler reserves space for the functions defined in the class

- Space for data is not allocated (since no object is yet created)

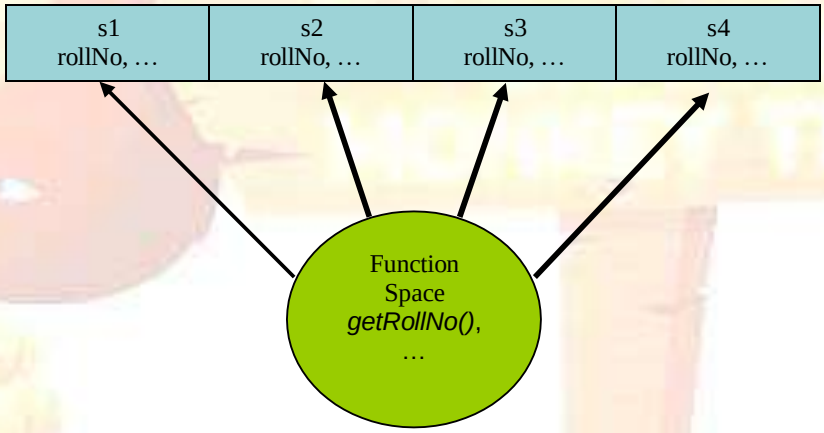
this Pointer
Student s1, s2, s3;



this Pointer

- Function space is common for every variable
- Whenever a new object is created:
  - Memory is reserved for variables only
  - Previously defined functions are used over and over again

Memory layout for objects created:



this Pointer

- Address of each object is passed to the calling function.
- This address is de-referenced by the functions and hence they act on correct objects

ORANGE MONKEY TEAM

s1 rollNo, …	s2 rollNo, …	s3 rollNo, …	s4 rollNo, …
address	*address*	*address*	*address*

Passing this Pointer

- Whenever a function is called the this pointer is passed as a parameter to that function.
- Function with n parameters is actually called with n+1 parameters

Example

void Student::setName(char *)

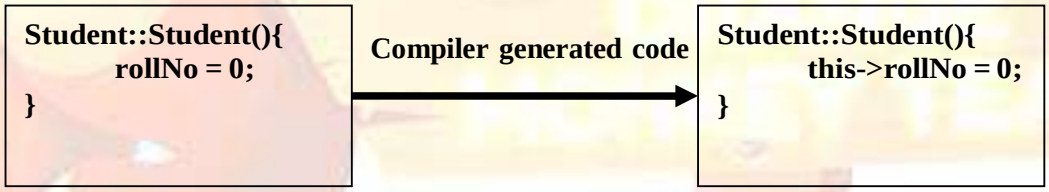
is internally represented as,

void Student::setName(char *, const Student *)

Declaration of this

DataType * const this;

Compiler Generated Code



ORANGE MONKEY TEAM

Lecture No.10

Uses of this Pointer

- There are situations where designer wants to return reference to current object from a function
- In such cases reference is taken from this pointer like (*this)

Example

```
Student Student::setRollNo(int aNo)
{
        ...
        return *this;
}
Student Student::setName(char *aName)
{
        ...
        return *this;
}
```

Usage:

```
int main()
{
        Student  aStudent;
        Student bStudent;

        bStudent = aStudent.setName("Ahmad");
        ...
        bStudent = aStudent.setName("Ali").setRollNo(2);

        return 0;
}
```

Separation of interface and implementation

- Public member functions exposed by a class are called interface.
- Separation of implementation from the interface is good software engineering.

Benefits of separating interface and implementation:

Consider the example of following complex no. class, this complex no. class two forms of implementations one is new and one is old implementation you can observe that if you have separated interface and implementation then we can easily change implementation without changing interface,

ORANGE MONKEY TEAM

Complex Number

- There are two representations of complex number
  - Euler form

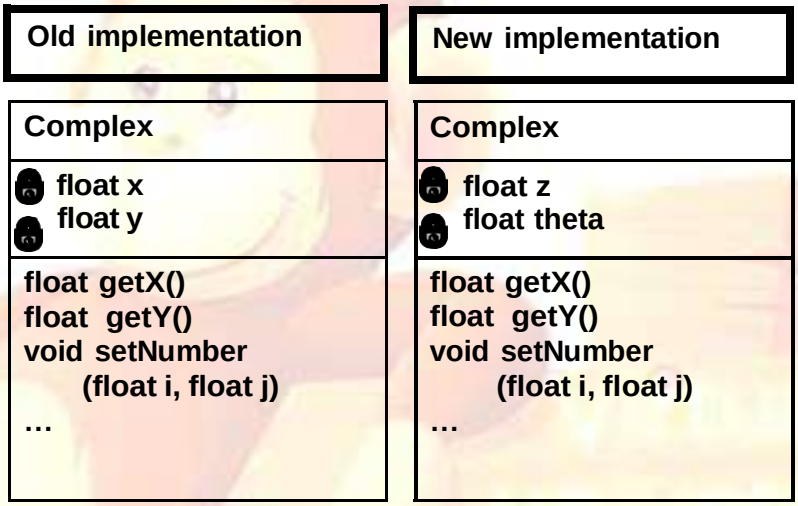
$z = x + iy$

  - Phasor form

$z = |z| (\cos\theta + i \sin\theta)$

z is known as the complex modulus and θ is known as the complex argument or phase

Example



- Uml notation to show private data members

Example

```
class Complex{    //old
        float x;
        float y;
public:
        void setNumber(float i, float j){
                x = i;
                y = j;
        }
        …
};
```

Example

```
class Complex{ //new
        float z;
        float theta;
public:
        void setNumber(float i, float j){
```

ORANGE MONKEY TEAM

```
            theta = arctan(j/i);
            …
        }
        …
};
```

Advantages

1. User is only concerned about ways of accessing data (interface)
2. User has no concern about the internal representation and implementation of the class

Separation of interface and implementation

In c++ generally we can relate the concept of interface of a class to its header (.h) file and and implementation of a class to its (.cpp) file. However it is not complete separation of interface and implementation.

- Usually functions are defined in implementation file (.cpp) while the class definition is given in header file (.h)
- Some authors also consider this as separation of interface and implementation

Student.h

```
class Student{
        int rollNo;
public:
        void setRollNo(int aRollNo);
        int getRollNo();
        …
};
```

Student.cpp

```
#include "student.h"

void Student::setRollNo(int aNo){
        …
}
int Student::getRollNo(){
…
}
```

We only need to include header (.h) file in main.cpp to use the Student class as shown below,

Main.cpp (main file to run the program)

```
#include "student.h"

int main(){
        Student aStudent;
         return 0;
}
```

const Member Functions

Some functions in our programs are general purpose functions to show or access data, they are supposed to do read only tasks only however there are chances that they can change the state of data members of the class while accessing the data members due to programming mistake, c++ provides the solution of this problem using constant member functions.

We make those functions as constant who need only read only access (for example such functions that will only display data or will return the value of data members). When we make them constant compiler generates an error if these functions try to change the value of data members of the class.

const Member Functions

Keyword const is placed at the end of the parameter list to make any function as constant.

Declaration:

```
Inside class
class ClassName{
        ReturnVal Function() const;
};
```

Definition:

```
Outside class
ReturnVal ClassName::Function() const{
        …
}
```

Example

```
class Student{
public:
        int getRollNo() const {
                return rollNo;
        }
};
```

const Functions

- Constant member functions cannot modify the state of any object

ORANGE MONKEY TEAM

- They are just "read-only"
- Errors due to typing are also caught at compile time

Example

Consider the function given below that is being used to check if roll no is equal to entered value if in this function we replace comparison statemtn == with assignment = statement it will compile correctly but whole code logic will change and we will get un expected result,

```
bool Student::isRollNo(int aNo){
        if(rollNo = = aNo){
                return true;
        }
        return false;
}
```

Example

```
bool Student::isRollNo(int aNo){
        /*undetected typing mistake*/
        if(rollNo = aNo){
                return true;
        }
        return false;
}
```

But if we have implemented as constant then compiler will catch this error and will produce compile time error as shown below,

Example

```
bool Student::isRollNo
                                (int aNo)const{
        /*compiler error*/
        if(rollNo = aNo){
                return true;
        }
        return false;
}
```

const Functions

Constructors and Destructors cannot be **const because** Constructors and destructors are used to modify the object to a well defined state or to clean the memory occupied by the object.

Example

```
class Time{
public:
        Time() const {}  //error…
        ~Time() const {} //error…
};
```

const Function

- Constant member function cannot change data member
- We cannot call non constant functions in constant functions because non constant member functions may have code for changing state of the object that is not allowed in the constant functions.

Example

```
class Student{
        char * name;
public:
        char *getName();
        void setName(char * aName);
        int ConstFunc() const{
                name = getName();     //error
                setName("Ahmad");//error
        }
};
```

this Pointer and const Member Function

As we know that when a class function is called an implicit this pointer is passed to tell the function about the object it has to operate same is true for constant function with the difference that it will bbe passed as constant pointer to const data in case of constant member functions so that this pointer can not be used now to change the value of data members of the object,

```
const Student *const this;
// In case of constant member functions
```

instead of

```
Student * const this;
// In case of ordinary member functions
```

ORANGE MONKEY TEAM

Lecture No.11

Usage example of Constant member functions

Problem:

Suppose we have requirement to change the class Student such that a student is given a roll number when the object is created and cannot be changed afterwards our existing class is given below,

Student Class

```
class Student{
…
        int rollNo;
public:
        Student(int aNo);
        int getRollNo();
        …
};
```

Solution of this problem:

We can do this by making rollNo constant so that cannot be changed once it is defined as shown below,

Modified Student Class

```
class Student{
…
        const int rollNo;
public:
        Student(int aNo);
        int getRollNo();
        …
};
```

Now there is only one issue of initializing this roll no with initial value but the problem is that we cannot set the value of roll no in constructor, as when code in constructor is executed the data member roll no has already been created and when we try to assign value to it in constructor compiler generates error,

Example

```
Student::Student(int aRollNo)
{
        rollNo = aRollNo;

 /*error: cannot modify a constant data member assignment statement not
```

ORANGE MONKEY TEAM

```
initialization*/
}
```

Second solution is to write separate function but the problem remains same that we can't assign value to constant data member,

```
void Student::SetRollNo(int i)
{
        rollNo = i;
        /*error: cannot modify a constant data member again assignment
statement not initialization */
}
```

We also know that we can only declare data members in structure or class but we cannot initialize them at the time of declaration in structure or class_____because before executing constructor code, the class const member roll no has not got life it will get life along with other class members when constructor will be invoked so we can not assign any value to this constant member while declaring it.[6]

Solution:

so what is the solution of this problem as we can not initialize constant members while declaring them and we can not initialize them in constructor also because as soon as they go life they become constant to solve this problem C++ gives us new mechanism (syntax) for initialization of constant data members of the structure or class to resolve above mentioned issues,

Difference between Initialization and Assignment:

Initialization is assigning value along with creation of variable.

```
int i = 2;
```

Assignment is assigning value after creation.

```
int i;
i = 7;
```

Member Initializer List

Member initialization list is used where we cannot modify the state of data members in the member functions of the class including constructor,

- A member initializer list is a mechanism to initialize data members
- It is given after closing parenthesis of parameter list of constructor

[6] In c++ **static const data members** can be initialized in class or structure as well.

- In case of more than one member use comma separated list

Example

```
class Student{
        const int rollNo;
        char *name;
        float GPA;
public:
        Student(int aRollNo) : rollNo(aRollNo), name(Null), GPA(0.0){ //
initialization
                ...
        }
...
};
```

Order of Initialization

- Data member are initialized in order they are declared in the class
- Order in member initializer list is not significant at all

Example

```
class ABC{
        int x;
        int y;
        int z;
public:
        ABC();
};
ABC::ABC():y(10),x(y),z(y)
{
        ...
}
/*      x = Junk value
                y = 10
                z = 10 */
```

const Objects

- Objects can be declared constant with the use of const keyword
- Constant objects cannot change their state

Example

```
int main()
```

ORANGE MONKEY TEAM

```
{
        const Student aStudent;
        return 0;
}
```

Example

```
#include <cstdlib>
#include <iostream>

using namespace std;


class Student{
        int rollNo;
public:

   Student( ) {

        }

        int getRollNo(){
                return rollNo;
        }
};


int main(){

const Student aStudent;
        int a = aStudent.getRollNo();

        //error

  system("PAUSE");
    return  EXIT_SUCCESS;
}
```

```
#include <cstdlib>
#include <iostream>

using namespace std;


class Student{
        int rollNo;
public:

   Student(int aRollNo) :
rollNo(aRollNo){

        }

        int getRollNo(){
                return rollNo;
        }
};

int main(){
        const Student aStudent(5);
        int a = aStudent.getRollNo();
        //error

   system("PAUSE");
   return  EXIT_SUCCESS;
}
```

const Objects

const objects can access only const member functions so chances of change of state of const objects once they are created are eliminated.
We make getRollNo function constant so that we can access it using constant objects,

Example

```
class Student{
```

ORANGE MONKEY TEAM

```
…
        int rollNo;
public:
…
        int getRollNo()const{
                return rollNo;
        }
};
int main(){
        const Student aStudent;
        int a = aStudent.getRollNo();
         return 0;
}
```

Constant member functions

Make all functions that don't change the state of the object constant
This will enable constant objects to access more member functions

Static Variables

Static variables of a class are such variables which are independent of class objects.

Lifetime of static variable is throughout the program life, if static variables are not explicitly initialized then they are initialized to 0 of appropriate type.

Example

Static variable is initialized once only throughout the program, independent of how many times the function initializing it is called,

```
void func1(int i){
        static int staticInt = i;
//initialization statement will be executed once
//only as static variables are initialized once
        cout << staticInt << endl;
}
int main(){
        func1(1);
        func1(2);
        return 0;
}
```

Output:
1
1

```
void func1(int i){

 static int staticInt;
 staticInt = i;
//assignment statement will be executed with each function call
 cout << staticInt << endl;
```

ORANGE MONKEY TEAM

```
}
int main(){
        func1(1);
        func1(2);
        return 0;
}
```

Output:
1
2

Static Data Member

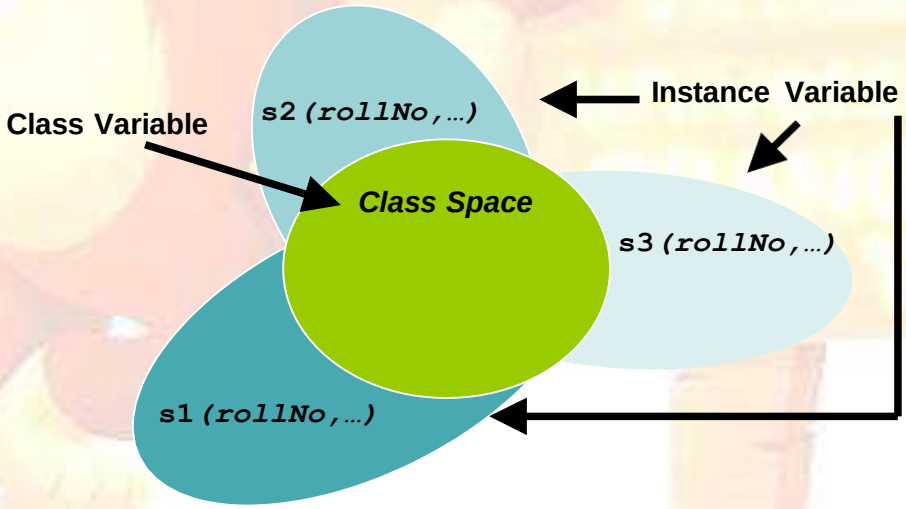
Definition
"A variable that is part of a class, yet is not part of any object of that class, is called static data member"

Static Data Member

They are shared by all instances (objects) of the class
They do not belong to any particular instance of a class

Class vs. Instance Variable

Suppose we created three objects of student class as shown below,
Student s1, s2, s3;



Static Data Member (Syntax)

Keyword static is used to make a data member static

```
class ClassName{
…
static DataType VariableName;
};
```

Defining Static Data Member

ORANGE MONKEY TEAM

Static data member is declared inside the class
But they are defined outside the class

Defining Static Data Member

```
class ClassName{
...
static DataType VariableName;
};

DataType ClassName::VariableName;
```

Initializing Static Data Member

Static data members should be initialized once at file scope
They are initialized at the time of definition

Example

```
class Student{
private:
static int noOfStudents;
public:
  ...
};
int Student::noOfStudents = 0;
/*private static member cannot be accessed outside the class except for initialization*/
```

Initializing Static Data Member

If static data members are not explicitly initialized at the time of definition then they are initialized to 0

Example

```
int Student::noOfStudents;
```

is equivalent to

```
int Student::noOfStudents=0;
```

ORANGE MONKEY TEAM

Lecture No.12

Review:
Static Data Member
Definition
"A variable that is part of a class, yet is not part of an object of that class, is called static data member"

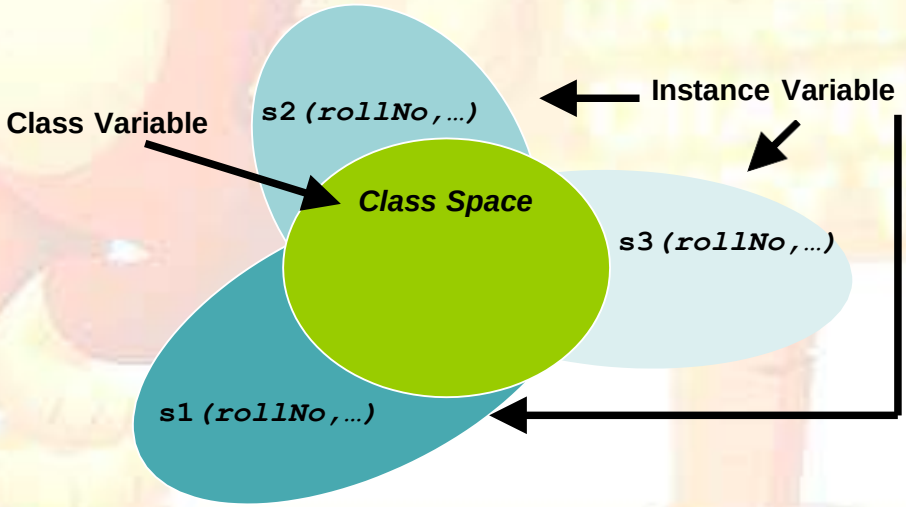
Static Data members are shared by all instances of the class and they do not belong to any particular instance of a class.

Class vs. Instance Variable

Memory for static variables in allocated in class space whereas for instance variables it is separate for each object as shown below, if we have class Student as given below,

```
class Student{
private:
static int noOfStudents;
public:
  ...
};
```

When we will create objects of Student as s1, s2, s3 then memory will be allocated as given below,



Static Data Member (Syntax)
Keyword static is used to make a data member static,

```
class ClassName{
...
static DataType VariableName;
};
```

ORANGE MONKEY TEAM

Defining Static Data Member (allocating memory for them)

Static data member is declared inside the class
But they are defined outside the class,

```
class ClassName{
...
static DataType VariableName;
};

DataType ClassName::VariableName;
```

Initializing Static Data Member(assigning them some initial value)

Static data members should be initialized once at file scope
They are initialized at the time of definition,

Example

```
class Student{
private:
static int noOfStudents;
public:
  ...
};
int Student::noOfStudents = 0;
/*private static member cannot be accessed outside the class except for initialization*/
```

Initializing Static Data Member

If static data members are not explicitly initialized at the time of definition then they are initialized to 0

Example

```
int Student::noOfStudents;
```

is equivalent to

```
int Student::noOfStudents=0;
```

Accessing Static Data Member

To access a static data member there are two ways

- Access like a normal data member (using dot operator '.')
- Access using a scope resolution operator '::'

Example

```
class Student{
public:
```

```
static int noOfStudents;
};

int Student::noOfStudents;

int main()
{
Student aStudent;
aStudent.noOfStudents = 1;
Student::noOfStudents = 1;
return 0;
}
```

Life of Static Data Member

- They are created even when there is no object of a class
- They remain in memory even when all Objects of a class are destroyed

Example

```
class Student{
public:
static int noOfStudents;
};
int Student::noOfStudents;
int main(){
Student::noOfStudents = 1;
}
```

Example

```
class Student{
public:
static int noOfStudents;
};
int Student::noOfStudents;
int main(){
{
        Student aStudent;
        aStudent.noOfStudents = 1;
}
Student::noOfStudents = 1;
return 0;
}
```

Uses

They can be used to store information that is required by all objects, like global variables

Example

Modify the class Student such that one can know the number of student created in a system

```
class Student{
...
public:
        static int noOfStudents;
        Student();
        ~Student();
...
};
int Student::noOfStudents = 0;
Student::Student(){
        noOfStudents++;
}
Student::~Student(){
        noOfStudents--;
}

int Student::noOfStudents = 0;
int main(){
        cout <<Student::noOfStudents <<endl;
        Student studentA;
        cout <<Student::noOfStudents <<endl;
        Student studentB;
        cout <<Student::noOfStudents <<endl;
        return 0;
}
```

Output:
0
1
2

Problem
noOfStudents is accessible outside the class
Bad design as the local data member is kept public

The solution is that we write static member function to access static members,

Static Member Function
Definition:
"The function that needs access to the members of a class, yet does not need to be invoked by a particular object, is called static member function"

- They are used to access static data members
- Access mechanism for static member functions is same as that of static data members
- They cannot access any non-static members

Example

```
class Student{
        static int noOfStudents;
        int rollNo;
public:
        static int getTotalStudent(){
                return noOfStudents;
        }
};
int main(){
        int i = Student::getTotalStudents();
        return 0;
}
```

Accessing non static data members

```
int Student::getTotalStudents(){
        return rollNo;
}
int main(){
        int i = Student::getTotalStudents();
        /*Error: There is no instance of Student, rollNo cannot be accessed*/
        return 0;
}
```

this Pointer and static member functions

- *this* pointer is passed implicitly to member functions
- *this* pointer is not passed to static member functions
- Reason is static member functions cannot access non static data members

Global Variable vs. Static Members

- Alternative to static member is to use global variable
- Global variables are accessible to all entities of the program
- User of Global variables is against the principle of information hiding.

Array of Objects

- Array of objects can only be created if an object can be created without supplying an explicit initializer
- There must always be a default constructor if we want to create array of objects

Example

```
class Test{
public:
};
int main(){
        Test array[2]; // OK
```

ORANGE MONKEY TEAM

```
        return 0;
}
```

```
class Test{
public:
        Test();
};
int main(){
        Test array[2]; // OK
        return 0;
}
```

```
class Test{
public:
        Test(int i);
};
int main(){
        Test array[2]; // Error
        return 0;
}
```

```
class Test{
public:
        Test(int i);
};
int main(){
        Test array[2] = {Test(0),Test(0)};
        return 0;
}
```

```
class Test{
public:
        Test(int i);
};
int main(){
        Test a(1),b(2);
        Test array[2] = {a,b};
        return 0;
}
```

ORANGE MONKEY TEAM

Lecture No.13

Pointer to Objects

- Pointer to objects are similar as pointer to built-in types
- They can also be used to dynamically allocate objects

Example

```
class Student{
...
public:
        Student();
        Student(char * aName);
        void setRollNo(int aNo);
};
```

Example

```
int main(){
        Student  obj;
        Student *ptr;
        ptr = &obj;
        ptr->setRollNo(10);
        return 0;
}
```

Allocation with new Operator

- new operator can be used to create objects at runtime

Example

```
int main(){
        Student *ptr;
        ptr = new Student;
        ptr->setRollNo(10);
        return 0;
}
```

Example

```
int main(){
        Student *ptr;
        ptr = new Student("Ali");
        ptr->setRollNo(10);
        return 0;
}
```

Example

```
int main()
{
        Student *ptr = new Student[100];
        for(int i = 0; i < 100;i++)
        {
                ptr->setRollNo(10);
        }
        return 0;
}
```

ORANGE MONKEY TEAM

Breakup of new Operation

new operator is decomposed as follows

- Allocating space in memory
- Calling the appropriate constructor

Case Study

Design a class date through which user must be able to perform following operations

- Get and set current day, month and year
- Increment by x number of days, months and year
- Set default date

Attributes

Attributes that can be seen in this problem statement are

- Day
- Month
- Year
- Default date

Attributes

The default date is a feature shared by all objects

- This attribute must be declared a static member

Attributes in Date.h

```
class Date
{
        int  day;
        int month;
        int year;
        static Date defaultDate;
...
};
```

Interfaces

- getDay
- getMonth
- getYear
- setDay
- setMonth
- setYear
- addDay
- addMonth
- addYear
- setDefaultDate

Interfaces

As the default date is a static member the interface setDefaultDate should also be declared static

Interfaces in Date.h

```
class Date{
...
public:
        void setDay(int aDay);
        int getDay()  const;
        void addDay(int x);
        ...
```

ORANGE MONKEY TEAM

```
...
};
Interfaces in Date.h
class Date{
...
public:
        static void setDefaultDate(
int aDay,int aMonth, int aYear);
...
};
Constructors and Destructors in Date.h
        Date(int aDay = 0,
  int aMonth= 0, int aYear= 0);

        ~Date(); //Destructor
};
```

Implementation of Date Class

The static member variables must be initialized

```
Date Date::defaultDate (07,3,2005);
```

Constructors

```
Date::Date(int aDay, int aMonth,
                                int aYear)    {
        if(aDay==0) {
                this->day = defaultDate.day;
        }
        else{
                setDay(aDay);
        }
        //similarly for other members
}
```

Destructor

We are not required to do any house keeping chores in destructor

```
Date::~Date
{
}
```

Getter and Setter

```
void Date::setMonth(int a){
        if(a > 0 && a <= 12){
                month = a;
}
int getMonth() const{
        return month;
}
```

addYear

```
void Date::addYear(int x){
        year += x;
        if(day == 29 && month == 2
                &&      !leapyear(year)){
                day = 1;
```

ORANGE MONKEY TEAM

```
                month = 3;
        }
}
```

Helper Function

```
class Date{
...
private:
        bool leapYear(int x) const;
...
};
```

Helper Function

```
bool Date::leapYear(int x) const{
        if((x%4 == 0 && x%100 != 0)
                ||      (x%400==0)){
                return true;
        }
        return false;
}
```

setDefaultDate

```
void  Date::setDefaultDate(
        int d, int  m,  int  y){
        if(d >= 0 && d <= 31){
                day = d;
        }
        ...
}
```

Complete code of Date class

```
# include<iostream.h>
# include<conio.h>
class Date{

   private:
      int day,  month,  year;
          static Date defaultDate;
     public:
         void setDefaultDate(int aDay,int aMonth, int aYear);
          void setDay(int aDay);
          int getDay() const;
          void addDay(int x);
          void setMonth(int aMonth);
          int getMonth() const;
          void addMonth(int x);
          void setYear(int aYear);
          int getYear()  const;
          void addYear(int x);
          bool leapYear(int x)const;
          Date(int aDay , int aMonth, int aYear);
          void setDate(int aDay , int aMonth, int aYear);
```

```
    ~Date(); //Destructor
};

Date Date::defaultDate(07,3,2005);

Date::Date(int aDay, int aMonth, int aYear)
{
        if(aDay==0)
        {
                this->day = defaultDate.day;
        }
        else
        {
           setDay(aDay);
        }

        if(aMonth==0)
        {
                this->month = defaultDate.month;
        }
        else
        {
           setMonth(aMonth);
        }

        if(aYear==0)
        {
                this->year = defaultDate.year;
        }
        else
        {
           setYear(aYear);
        }

}

void Date::setMonth(int a) {
   if(a > 0 && a <= 12)
   {
      month = a;
   }
 }

 int Date:: getMonth() const {
        return month;
   }

void Date::addYear(int x)
{
   year += x;
```

```
    if(day == 29 && month == 2 && !leapYear(year))
    {
        day = 1;
        month = 3;
    }
}

bool Date::leapYear(int x) const {
    if((x%4 == 0 && x%100 != 0) || (x%400==0))
    {
        return true;
    }
    return false;
}

void Date::setYear(int aYear){
   year=aYear;
   }
void Date::setDay(int aDay){
   day=aDay;

   }

void Date::setDate(int aDay , int aMonth, int aYear){
   setDay(aDay);
   setMonth(aMonth);
   setYear(aYear);
   cout<<day<<"/"<<month<<"/"<<year<<endl;

   }


  Date::~Date(){
   cout<<"Date destructor"<<endl;
}
int main()
{
   Date aDate(0,0,0);

   aDate.setDate(20,10,2011);
   system("pause");
}
```

Lecture No.14

Composition

Consider the following implementation of the student class we discussed in previous lectures,

Student
gpa : float rollNo : int name : char *
Student(char * = NULL, int = 0, float = 0.0); Student(const Student &) GetName() const : const char * SetName(char *) : void ~Student() …

Composition

If one object is part of another object (relationship of part and whole) in composition lifetime of one object depends upon the other. The part objects are essential components of the whole.
For example person is **composed of** hands, eyes, feet so on.

In student class we assigning dynamic memory for variable name using new operator as shown,

```
class Student{
private:
  float gpa;
  char * name;
int rollNumber;
public:
  Student(char * = NULL, int = 0, float = 0.0);
  Student(const Student & st);
  const char * GetName() const;
  // never return handle of private data members or private member functions
  ~Student();
  …
};

Student::Student(char * _name, int roll, float g)
{
        cout << "Constructor::Student..\n";
```

ORANGE MONKEY TEAM

```
        if (!_name){
                name = new char[strlen(_name)+1];
                strcpy(name,_name);
        }
        else name = NULL;
        rollNumber = roll;
        gpa = g;
 }


Student::Student(const Student & st){
  if(str.name != NULL){
     name = new char[strlen(st.name) + 1];
     strcpy(name, st.name);
  }
  else name = NULL;
  rollNumber = st.roll;
  gpa = st.g;
}


 const char * Student::GetName(){
   return name;
 }
```

// never return handle of private data members or private member functions const ensures that private data members will not be changed

```
 Student::~Student(){
        delete [] name; // deleting name array
 }
```

In C++ "*it is all about code reuse*"

Composition is Creating objects of one class inside another class

"Has a" relationship:

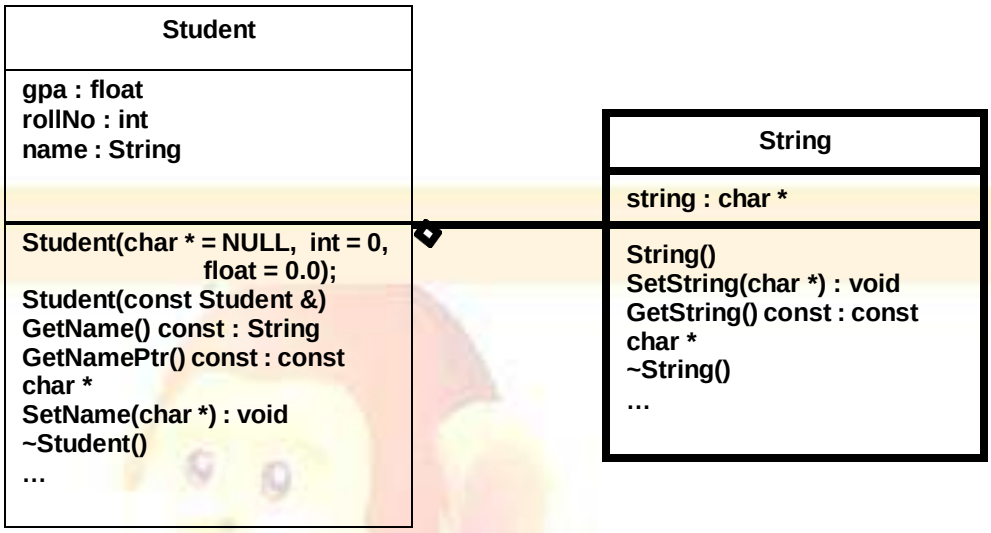
Bird has a beak

Student has a name

Composition

Now we change code slightly replacing name char * by String so that it is whole object of class String as it qualifies to be an object because we have to apply many operations on it like string dynamic creation and deletion, string copy using deep copy, searching a substring and so on….

Conceptual notation:



Composition

Now we see string class code to see how it simplifies original Student object and how we have used composition here,

```
class String{
  private:
          char * ptr;
  public:
          String(); // default constructor
          String(const String &); // copy constructor
          void SetString(const char *); // setter function
          const char * GetString() const;
// getter function returning const pointer to data member ptr
          ~String()
          ...
};
  String::String(){
          cout << "Constructor::String..\n";
          ptr = NULL;
  }

  String::String(const String & str){
    if(str.ptr != NULL){
           ptr = new char[strlen(str.ptr)+1];
           strcpy(ptr, str.ptr);
    }
    else ptr = NULL;
  }
  void String::SetString(const char * str){
          if(ptr != NULL){
                  delete [] ptr;
```

ORANGE MONKEY TEAM

```
            ptr = NULL;
        }
        if(str != NULL){
            ptr = new char[strlen(str)+1];
            strcpy(ptr, str);
        }
    }
```

issue of memory leakage (inaccessible memory)
if we simply set pointer here
memory will be outside of our object many cause problems later
user still has pointer of passed value it can itself modify it

We resolve these two issues in SetString by allocating new memory and deleting previous memory.

```
const char * String::GetString()const{
    return ptr;
}
String::~String(){
        delete [] ptr;
        cout <<"Destructor::String..\n";
}
```

Make pointer equal to NULL as well any where you delete dynamic memory.

Now consider the code of Student class again, now by adding composed string object our code has been simplified very much, (we will use methods of composed object simply by calling them where needed)

```
class Student{
private:
  float gpa;
  int rollNumber;
String name;
public:
  Student(char* =NULL, int=0,float=0.0);
  Student(const Student &);
  void SetName(const char *);
  String GetName() const;
  const char * GetNamePtr() const;
  ~Student();
  ...
};

Student ::Student(char * _name,      int roll, float g){
        cout <<"Constructor::Student..\n";
        name.SetString(_name);
```

ORANGE MONKEY TEAM

```
        rollNumber = roll;
        gpa = g;
}

Student::Student(const Student & s){
        name.SetString(s.name.GetString());
        gpa = s.gpa;
        rollNumber = s.rollNumber;
}
```

Explanation:

1. **name.SetString**(s.name.GetString());
   // setting composed name of newly created object

2. name.SetString(**s.name**.GetString());
   //accessing the composed object string name of object to be copied

3. name.SetString(s.**name.GetString**());
   //accessing the value of composed object string name by calling its member function GetString

4. **name.SetString(s.name.GetString());**
   //overall result : the value of composed object string of object to be copied will be copied to newly created object composed object string.

```
const char * Student::GetNamePtr() const{
    return name.GetString();
}

void Student::SetName(const char * n){
        name.SetString(n);
}

Student::~Student(){
        cout <<"Destructor::Student..\n";
}

void main(){
        Student *aStudent=new Student("Fakhir", 899, 3.1);
        cout << endl;
        cout << "Name:" << aStudent->GetNamePtr() << "\n";
}
```

Output:

```
Constructor::String..
Constructor::Student..
```

ORANGE MONKEY TEAM

```
Name: Fakhir
Destructor::Student..
Destructor::String..
```

Important Points:

1. We can access methods of composed object in the same way as we can access methods of other objects.

   Name of composed object.MemberFunction

2. Member functions of a class can access its private data members like,

```
Student::Student(const Student & s){

        name.SetString(s.name.GetString());

        // accessing private member String name of student using its object s
        and then accessing String name member function GetString to access
        string value two methods calss in one line

            gpa = s.gpa;
        // accessing private members of student in student member function


            rollNo = s.rollNo;
        // accessing private members of student in student member function

 }
```

Constructors & Composition

Constructors of the sub-objects are always executed before the constructors of the master class

Example:

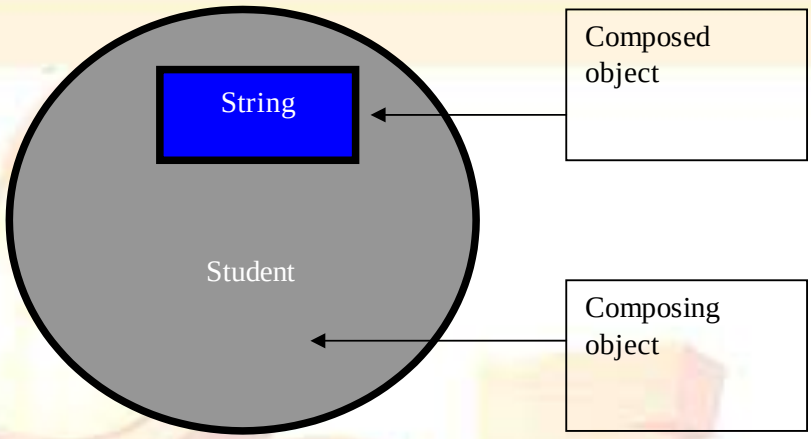
As you see the example output of program given above,

Output:

```
Constructor::String..
Constructor::Student..

Name: Fakhir
Destructor::Student..
Destructor::String..
```

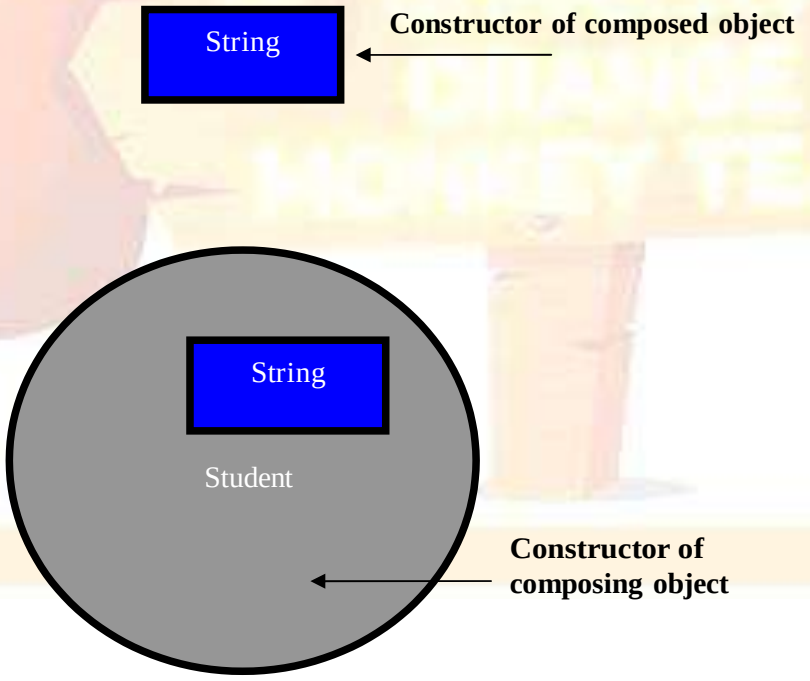
ORANGE MONKEY TEAM

Constructor for the sub-object name is executed before the constructor of Student and destructor of sub-object is called after destructor of student. It is logical as composing object has to contain composed object so composed object should be created first and then composing object. Similarly while destructing objects we composing object is destructed first and then composed object as shown in diagram below,



Composition

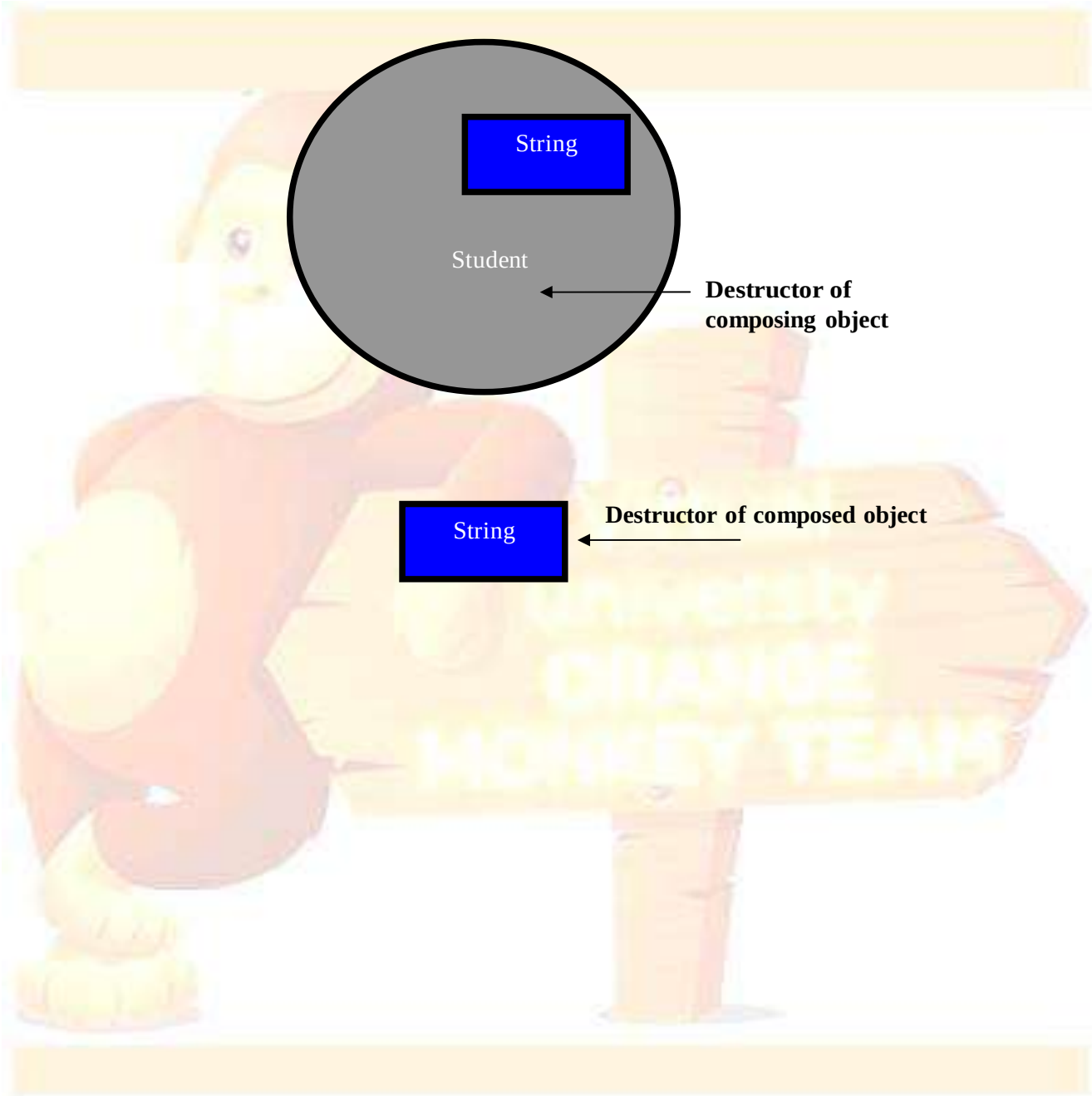
Constructor calling:
Constructors are called from composed objects to composing objects.



ORANGE MONKEY TEAM

Destructor calling:

Destructors are called from composing objects to composed objects.

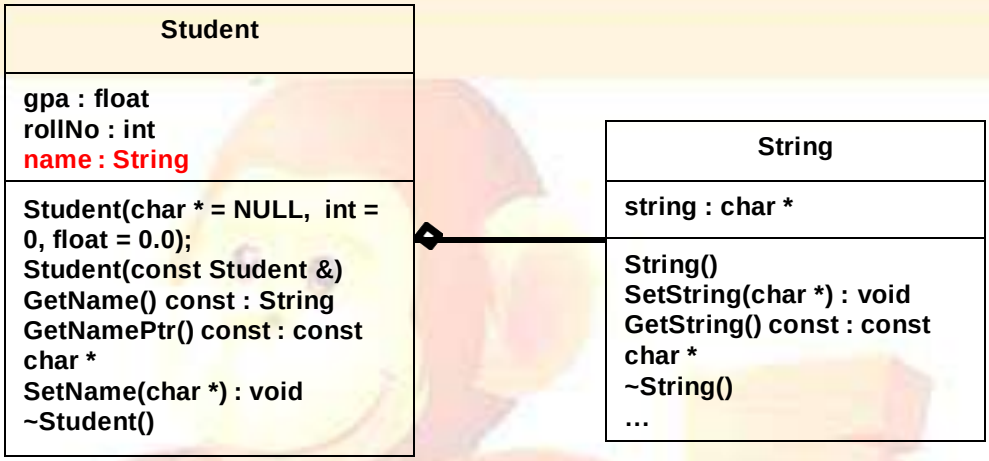


ORANGE MONKEY TEAM

Lecture No.15

Composition:

We saw composition in last lecture, its Conceptual notation is given below,



We created student object in main as by passing name, rollno and gpa and then displayed the name of student using GetNamePtr member function of student class,

```
int main(){
  Student aStudent("Fakhir", 899,3.1);
  cout << endl;
  cout << "Name:" << aStudent.GetNamePtr()<< endl;
  return 0;
  }
```

Output:

The output of our code is given below,

```
Constructor::String..
Constructor::Student..
Name: Fakhir
Destructor::Student..
Destructor::String..
```

Constructor Code:

Let us see the constructor code again,

```
Student::Student(char * n, int roll, float g){
      cout <<"Constructor::Student..\n";
      name.SetString(n);
      rollNumber = roll;
      gpa = g;
    }
```

ORANGE MONKEY TEAM

In this code we are setting string data member **name** of Student class using **SetString** but the problem in this approach is that we have to call SetString method explicitly to set value in string class, the reason for it is that our String class doesn't support setting its value while creating its object this is the reason we have to use the function SetString in the constructor.

This is an overhead and also not very good way to set any object value, we want to initialize our string sub-object **name** in the **student** class as we initialize other objects using constructor. For achieving this functionality we add an overloaded constructor in the **String** class that takes char string as parameter and initialize the String class object with this value using the Student constructor's "*Member initialization list*" as shown below in bold text:

```
class String{
        char *ptr;
public:
        String();
        String(char *);  // constructor with char * as parameter
        String(const String &);
        void SetName(char *);
        ~String();
        …
};


 String::String(char * str){
  if(str != NULL){
                ptr = new char[strlen(str)+1];
                strcpy(ptr, str);
 }

else ptr = NULL;
    cout << "Overloaded Constructor::String..\n";

}
```

Now Student class constructor code is modified as follows:

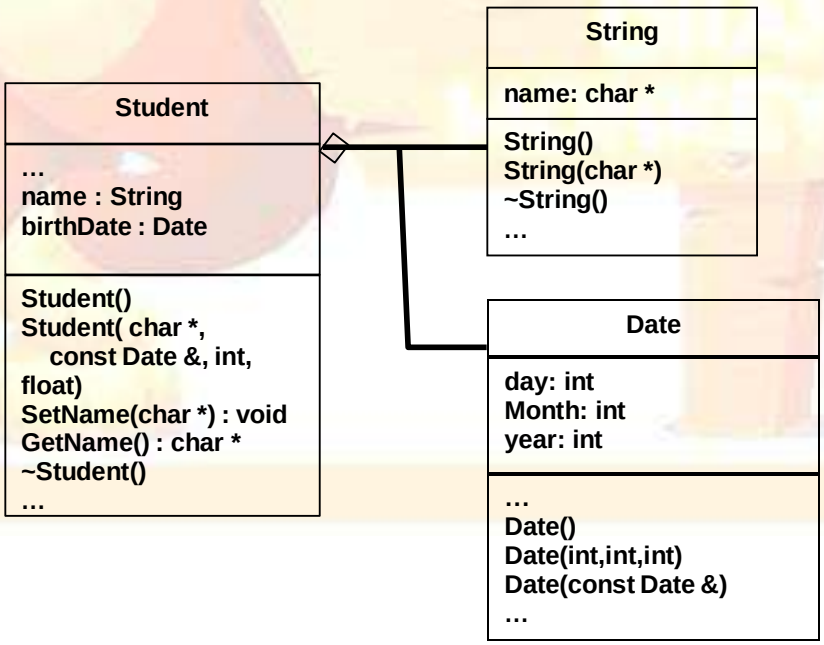
```
class Student{
private:
        float gpa;
        int rollNumber;
        String name;
public:
        …
        Student(char *=NULL, int=0, float=0.0);
};
```

```
Student::Student(char * n,int roll, float g): name(n)  {

        cout << "Constructor::Student..\n";
        rollNumber = roll;
        gpa = g;

 }

int main(){
        Student aStudent("Fakhir", 899, 3.1);
        cout << endl;
        cout  << "Name:"  << aStudent.GetNamePtr()          << endl;
        return 0;
}
```

Output:

```
Overloaded Constructor::String..
Constructor::Student..

Name: Fakhir
Destructor::Student..
Destructor::String..
```

Now suppose we want to add date object in student class to store student **Birth Date,** the conceptual diagram will be as given below,



ORANGE MONKEY TEAM

Student class is modified as follows:

```
class Student{
private:
        ...
        Date birthDate;
        String name;
public:
        Student(char *, const Date &, int, float);
        ~Student();
        ...
};
```

Composition

```
 Student::Student(char * n, const Date & d, int roll, flaot g): name(n),birthDate(d) {
        cout << "Constructor::Student..\n";
        rollNumber = roll;
        gpa = g;
 }

 Student::~Student(){
        cout << "Destructor::Student..\n";
 }
int main(){
        Date _date(31, 12, 1982);
        Student aStudent("Fakhir", _date,899,3.5);
        return 0;
}
```

Output:

```
        Overloaded Constructor::Date..
        Copy Constructor::Date..
        Overloaded Constructor::String..
        Constructor::Student..
        Destructor::Student..
        Destructor::String..
        Destructor::Date..
        Destructor::Date..
```

Aggregation

In composition we made separate object of those concepts that we think were worthy to be implemented as an object within other object to make our code simpler and to make functionality modular (divided in parts) and understandable like we made String class in Student class, but in real life most situations are such that two distinct (different) objects and one object is using services of the other one like student and teacher, student and librarian, room and chair, passenger and bus, book and

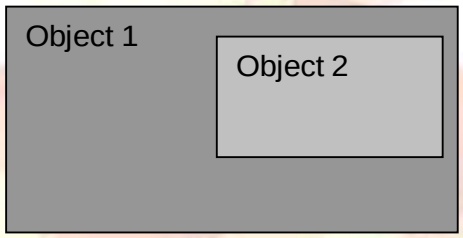
bookshelf, person and computer so on. In this case we cannot make one object as part of other object because they exist independently and only provide services to each other like in case of,

Student and Teacher: Student or Teacher object cannot be composed of other one yet they are taking services of each other.

Passenger and Bus: Passenger and Bus are taking services of each other but exist standalone also, bus includes passengers but passenger are not Part of Bus they can exist independently as well.

Composition vs. Aggregation

Aggregation is a *weak relationship* than composition because in this relationship two classes get services of each other but can exist independently as well, main difference is memory organization of two objects as shown below,

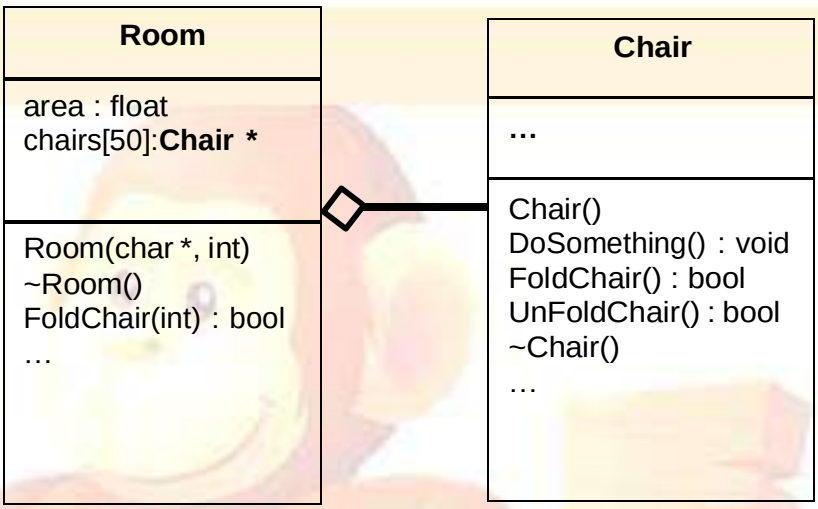


Composition

Aggregation

Example:

Take the example of Room and Chair as given below,



Aggregation C++ implementation:

In aggregation, a pointer or reference to an object is created inside a class. The sub-object has a life that is **NOT** dependant on the life of its master class.
e.g

- Chairs can be moved inside or outside at anytime
- When Room is destroyed, the chairs may or **may not** be destroyed

Aggregation:

```
class Room{
private:
        float area;
        Chair * chairs[50];
Public:
        Room();
        void AddChair(Chair *, int chairNo);
        Chair * GetChair(int chairNo);
        bool FoldChair(int chairNo);
        …
};

Room::Room(){
for(int i = 0; i < 50; i++)
chairs[i] = NULL;
```

ORANGE MONKEY TEAM

```
}
void Room::AddChair(Chair * chair1, int chairNo){
        if(chairNo >= 0 && chairNo < 50)
                chairs[chairNo] = chair1;
}

Chair * Room::GetChair(int chairNo){
        if(chairNo >= 0 && chairNo < 50)
                return chairs[chairNo];
        else
                return NULL;
}

bool Room::FoldChair(int chairNo){
 if(chairNo >= 0 && chairNo < 50)
        return chairs[chairNo]->FoldChair();
 else
        return false;
}
int main(){
  Chair ch1;
  {
    Room r1;
        r1.AddChair(&ch1, 1);
        r1.FoldChair(1);
  }
   ch1.UnFoldChair(1);
  return 0;
}
```

Friend Functions

The functions which are not member functions of the class yet they can access all private members of the class are called friend functions.

Why they are needed?

They are needed in situations where we have written code for some function in one class and it need to be used by other classes as well for example,
Suppose we wrote the code to compute a complex mathematical formulae in one class but later it was required by other classes as well, in that case we will make that function friend of all other classes.

Are friend functions against the concept of Object Oriented Programming?

It can be said that friend functions are against the principle of object oriented programming because they violate the principle of encapsulation which clearly says that each object methods and functions should be encapsulated in it. But there we are making our private member accessible to other outside functions.

ORANGE MONKEY TEAM

Consider the following class:

```
class X{
private:
        int a, b;
public:
        void MemberFunction();
        …
};
```

Suppose we have a global function **DoSomething** that need to access the private members of class X, when we will try to access them compiler will generate error as outside world can not access private members of a class except its member functions.

```
void DoSomething(X obj){

        obj.a = 3; //Error
        obj.b = 4; //Error
}
```

Friend Functions

In order to access the member variables of the class, we must make function friend of that class,

```
class X{
private:
  int a, b;
public:
        …
   friend void DoSomething(X obj);
};
```

Now the function **DoSomething** can access data members of class X

```
void DoSomething(X obj){

        obj.a = 3;
        obj.b = 4;
}
```

Friend Functions

Prototypes of friend functions appear in the class definition.
But friend functions are NOT member functions.

Friend Functions

Friend functions can be placed anywhere in the class without any effect
Access specifiers don't affect friend functions or classes

ORANGE MONKEY TEAM

```
class X{
        ...
private:
        friend void DoSomething(X);
public:
        friend void DoAnything(X);
         ...
};
```

Friend Functions

While the definition of the friend function is:

```
void DoSomething(X obj){

        obj.a = 3;              // No Error
        obj.b = 4;              // No Error
        ...
}
```

friend keyword is not given in definition.

Friend Functions

If keyword **friend** is used in the function definition, it's a syntax error

```
//Error...

friend void DoSomething(X obj){
        ...
}
```

Friend Classes

Similarly, one class can also be made friend of another class:

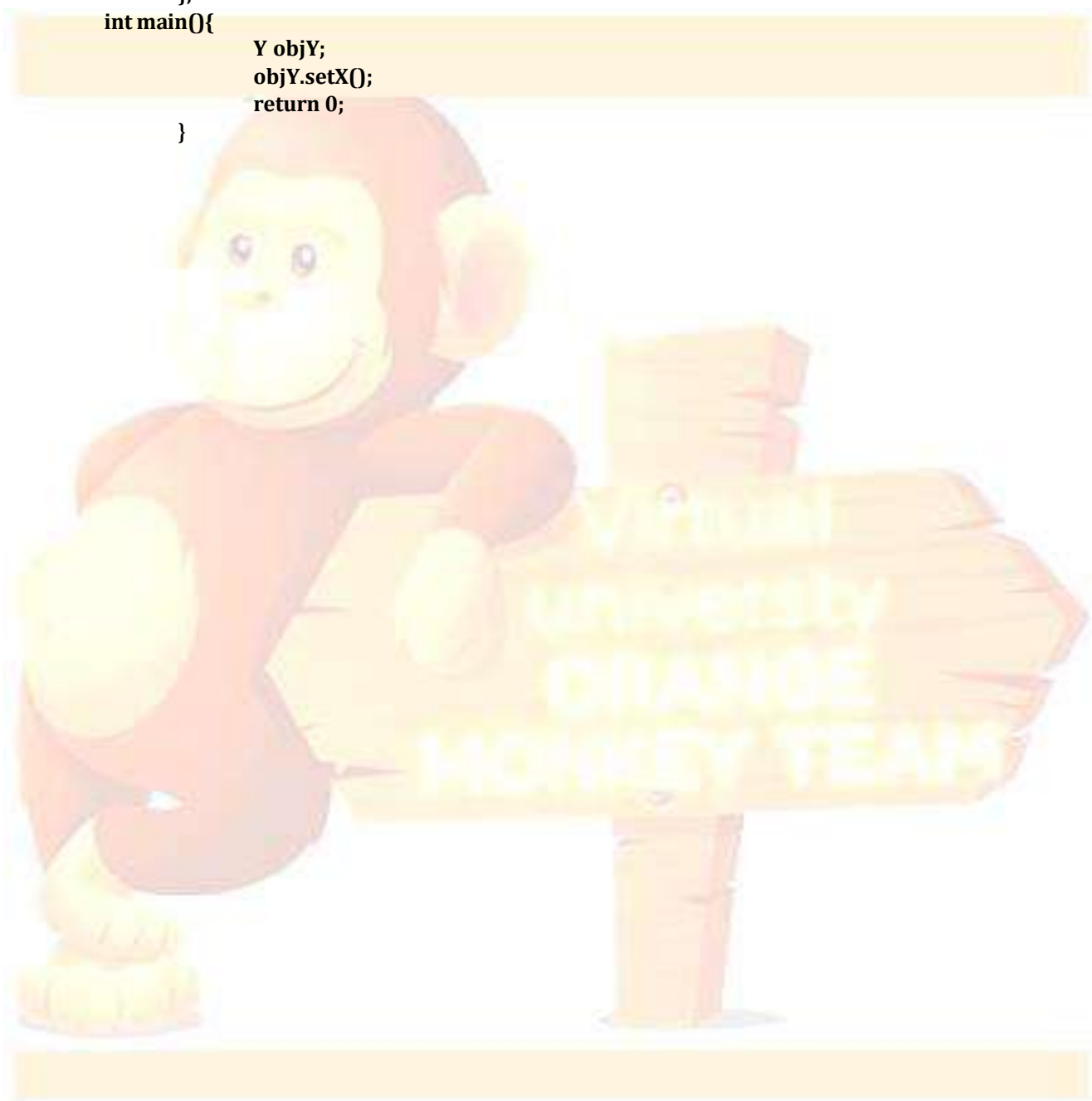
```
class X{
        friend class Y;
        ...
};
```

Member functions of class Y can access private data members of class X

```
class X{
        friend class Y;
        private:
                int x_var1, x_var2;
          ...
        };
class Y{
        private:
                int y_var1, y_var2;
```

ORANGE MONKEY TEAM

```
        X objX;
    public:
        void setX(){
            objX.x_var1 = 1;
        }
    };
int main(){
        Y objY;
        objY.setX();
        return 0;
    }
```



Lecture No.16

16.1.Operator overloading

Consider the following class,

```
class Complex{
private:
        double real, img;
public:
        Complex Add(const Complex &);
        Complex Subtract(const Complex &);
        Complex Multiply(const Complex &);
        …
};
```

We want to write function to add two complex no. objects, the Add function implementation to add two complex numbers is shown below, this function is taking one complex no object and its adding the current (with reference to which it will be called) and is returning result in new object,

```
Complex Complex::Add(const Complex & c1){
        Complex t;
        t.real = real + c1.real;
        t.img  = img  + c1.img;
        return t;
};

// adds the contents of c2 to c1
// creating new object c3 and assigning it result of c1+c2
```

Now we can add two complex no. objects using the following statement,

```
Complex c3 = c1.Add(c2);
```

In this statement two operations are taking place,

1. One is addition of two objects using the function call Add and returning the result in a new object.
   Complex c3 = **c1.Add(c2);**

2. Second is copy of that temporary object to newly created object c3 using copy constructor.
   Complex c3 = c1.Add(c2);

But there are two issues with this implementation,

1. We can't add two complex no. objects by simple writing '+' as we can add basic data types like int or float as shown below,

int a = 3;

ORANGE MONKEY TEAM

int b = 5;
c = a + b; // correct

Complex c1(2,3), c2(4,5);
Complex c3 = c1 + c2; // error

Instead we have to explicitly write,

Complex c3 = c1.Add(c2)

If we give our complex no. class code some user in compiled form for use, user will need to know how we have written **Add** function (no. of parameters , return type) to add two complex no. objects so that he can call Add function correctly.

2. If we want to perform add operation on more than two objects in a single mathematical statement like:

c1+c2+c3+c4

We are unable to do it.

We have to explicitly write,

c1.Add(c2.Add(c3.Add(c4)))

Alternative way is:

t1 = c3.Add(c4);
t2 = c2.Add(t1);
t3 = c1.Add(t2);

This is also overhead, especially if the mathematical expression is large,

Converting it to C++ code will involve complicated mixture of function calls
Code will become less readable
Chances of human mistakes will become very high
Code produced will be very hard to maintain

The solution to this problem is simple that we can write normal operators like +,-,*, and so on for our user defined classes as well,

It is "*Operator overloading*"

Using operator overloading we can perform basic operations (like addition, subtraction, multiplication, division and so on…) on our own defined classes objects in the similar way as we perform them on basic built-in types(like int, float, long, double etc.).
C++ allows us to overload common operators like **+**, **-** or * etc…
With operator overloading Mathematical statements don't have to be explicitly converted into function calls as we had to do to add two complex no objects using function call **Add**.

ORANGE MONKEY TEAM

Operator overloading

Assume that operator **+** has been overloaded then actual C++ code becomes:

c1+c2+c3+c4

The resultant code is very easy to read, write and maintain

Operator overloading

C++ automatically overloads operators for pre-defined types as these have also been implemented as classes by c++.

Example of predefined types:

int
float
double
char
long

Operator overloading

```
float x;
int y;
x = 102.02 + 0.09; // overloaded operator '+' for float type will be called by c++
Y = 50 + 47; // overloaded operator '+' for int type will be called by c++
```

The compiler probably calls the correct overloaded low level function for addition i.e:

```
// for integer addition:
Add(int a, int b)

// for float addition:
Add(float a, float b)
```

Operator overloading

Operator functions are not usually called directly, they are automatically invoked to evaluate the operations they implement by compiler.

List of operators that can be overloaded in C++:

```
new    delete     new[]      delete[]
+      -     *     /     %     ^     &     |     ~
!      =     <     >     +=    -=    *=    /=    %=
^=     &=    |=    <<    >>    >>=   <<=   ==    !=
<=     >=    &&    ||    ++    --    ,     ->*   ->
()     []
```

List of operators that can't be overloaded:

.	.*	::	?:	#	##

Reason: They take actual current object name, rather than value in their argument as you have seen previously in the use of dot (**'.'**) operator,

Student std;
int roll = std.getRollNo() // dot operator is performing on actual function (getRollNo) of class Student that will vary from program to program.

?: is the only ternary operator in C++ and can't be overloaded.

The precedence of an operator:

The precedence of an operator is order of evaluation which operator will be evaluated first in expression.
The precedence of an operator is **NOT** affected due to overloading.
Example:

c1*c2+c3
c3+c2*c1
In both lines multiplication * will be done first and then addition.

Associativity:

Associativity is **NOT** changed due to overloading
Following arithmetic expression always is evaluated from left to right:
c1 + c2 + c3 + c4

Unary operators and assignment operator are right associative, e.g:
a=b=c is same as **a=(b=c)**
All other operators are left associative:
c1+c2+c3 is same as
(c1+c2)+c3

Important things to consider:
Always write code representing the operator for example adding subtraction code inside the + operator will create chaos.
Creating a new operator is a syntax error (whether unary, binary or ternary), you cannot create $.

Arity of Operators

Arity (no of operands it works on) of an operator is NOT affected by overloading
Example:
Division operator will take exactly two operands in any case:
b = c / d

ORANGE MONKEY TEAM

General syntax of Operators Overloading:

In case of member functions of a class:

```
return_type class_name::operator operator_symbol( parameters ){
        /*code*/
}
```

In case of non member functions of a class (in this case we will make overloaded operator function as friend function):

```
return_type operator operator_symbol( parameters ){
        /*code*/
}
```

For example:

```
Complex& Complex::operator + (const Complex & c){
/*code*/
}

Complex& operator + (const Complex & c){
/*code*/
}
```

Binary Operators Overloading:

Binary operators act on two quantities.
Examples of binary operators:

+	-	*	/	%	^	&	|	~
!	=	<	>	+=	-=	*=	/=	%=
^=	&=	|=	<<	>>	>>=	<<=	==	!=
<=	>=	&&	||	,	->*	->		

General syntax of Binary Operators Overloading:

In case of member function of a class:

```
TYPE class_name::operator operator_symbol( TYPE rhs ){
        /*code*/
}
```

In case of non-member function of a class:

```
TYPE class_name::operator operator_symbol( TYPE rhs1, TYPE rhs2 ){
        /*code*/
}
```

ORANGE MONKEY TEAM

The "**operator OP**" must have at least one formal parameter of type class (user defined type)

Following is an error:
int operator + (int, int);

Examples:

Overloading + operator:

```
class Complex{
private:
        double real, img;
public:
        …
        Complex operator +(const Complex & rhs);
};

Complex Complex::operator +( const Complex & rhs){
        Complex t;
        t.real = real + rhs.real;
        t.img = img + rhs.img;
        return t;
}
```

The return type is Complex so as to facilitate complex statements like:
Complex t = c1 + c2 + c3;

The above statement is automatically converted by the compiler into appropriate function calls:
(c1.operator +(c2)).operator +(c3);

If the return type was **void**,

```
class Complex{
        …
 public:
        void operator+(
                const Complex & rhs);
};

void Complex::operator+(const Complex & rhs){
        real = real + rhs.real;
        img = img + rhs.img;
};
```

We have to do the same operation **c1+c2+c3** as:
c1+c2
c1+c3
// final result is stored in c1

Drawbacks of void return type:

Assignments and cascaded expressions are not possible
One of the existing objects is used to store result
Code is less readable
Debugging is tough
Code is very hard to maintain

Lecture No.17

Binary operators (cont.)

The binary operator is always called with reference to the left hand argument.

Example:
In **c1+c2**,
c1.operator+(c2)
// c1 is calling overloaded + operator and c2 is being passed as
// reference in that function.
In **c2+c1**,
c2.operator+(c1)
// c2 is calling overloaded + operator and c1 is being passed as
// reference in that function.

Adding basic data type to complex number class:

The overloading code we discussed before for complex no. class can add two complex number objects but it can not handle the following situation:

Complex c1;
c1 + 2.325 (as we are adding basic data type double to complex no.)

To do this, we have to modify the **Complex** class.

Modifying the complex class:

```
class Complex{
...
Complex operator+(const Complex & rhs);
Complex operator+(const double& rhs);
};

Complex operator + (const double& rhs){
        Complex t;
        t.real = real + rhs;
        t.img = img;
        return t;
}
```

Now we can write both forms of statements,
Complex c2, c3;
Complex c1 = c2 + c3;
Complex c4 = c2 + 235.01;
But problem arises if we do the following:

Complex c5 = 450.120 + c1;
The + operator is called with reference to **450.120**

ORANGE MONKEY TEAM

No predefined overloaded + operator is there that takes **Complex** as an argument

Now if we write the following two functions to the class, we can add a **Complex** to a **real** or vice versa :

```
Class Complex{
...
friend Complex operator + (const Complex & lhs, const double & rhs);
friend Complex operator + (const double & lhs, const Complex & rhs);
};
```

We have made them as friend so that we can write them as non member functions and they are not called with respect to complex no. class object instead we pass both arguments (complex no. object and double value) to this function compiler invoke them according to arguments passed. Their implementation is similar,

```
Complex operator +(const Complex & lhs, const double& rhs){

        Complex t;
        t.real = lhs.real + rhs;
        t.img = lhs.img;
        return t;
}

Complex operator + (const double & lhs, const Complex & rhs){

        Complex t;
        t.real = lhs + rhs.real;
        t.img = rhs.img;
        return t;
}
```

Binary operators

So adding three overloaded versions for + operator as shown below allow us to write code to,
Add two complex objects
Add complex object and a double value.

```
Class Complex{
...
Complex operator + (const Complex &);
friend Complex operator + (const    Complex &, const double &);
friend Complex operator + (const    double &, const Complex &);
};
```

Non members which are not friend call also achieve this functionality but in that case we need extra four functions two getters and two setters each for real and imag part. Compiler searches overloaded operator code in member function first and then in non member functions.

ORANGE MONKEY TEAM

Binary operators

Other binary operators are overloaded in similar to the + operator as demonstrated in the above examples
Example:

```
Complex operator * (const Complex & c1, const Complex & c2);
Complex operator / (const Complex & c1, const Complex & c2);
Complex operator - (const Complex & c1, const Complex & c2);
```

Overloading Assignment operator

As we know compiler can generate the following three functions for a class if required on its own,

- Default Constructor (in case we have not written any other constructor for a class)
- Copy Constructor
- Assignment Operator

However as we discussed previously if our class has any data member using dynamic memory then we had to write our own code for default constructor, copy constructor and similarly assignment operator as compiler generated version of these functions performs shallow copy that creates dangling pointer, and memory leakage issues in case of dynamic memory allocation.
We have already seen code of default constructor and copy constructor and the code for overloaded assignment operator is similar,

Consider the string class:

```
class String{
        int size;
        char * bufferPtr;
public:
        String(); // default constructor
        String(char *); // overloaded constructor
        String(const String &); // copy constructor
        …
};
String::String(){

                bufferPtr = NULL;
               size = 0;
}

String::String(char * ptr){
        if(ptr != NULL){
                size = strlen(ptr);
                bufferPtr = new char[size+1];
                strcpy(bufferPtr, ptr);
        }
```

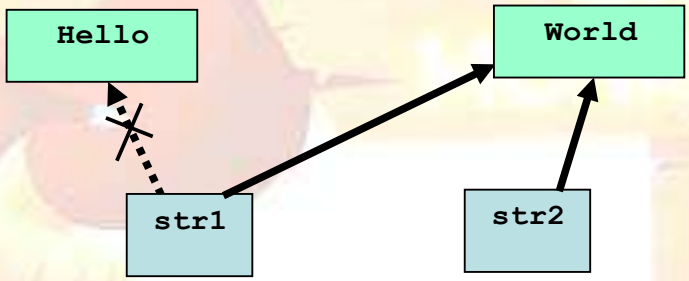
ORANGE MONKEY TEAM

```
        else{
                bufferPtr = NULL;
size = 0;
}
}

String::String(const String & rhs){
        size = rhs.size;

        if(rhs.size != 0){
                bufferPtr = new char[size+1];
                strcpy(bufferPtr, ptr);
        }
        else
                bufferPtr = NULL;

}

int main(){
        String str1("Hello");
        String str2("World");
        str1 = str2;[7][8]
        return 0;
}
```

Member wise copy assignment

Assignment operator (Shallow Copy)

Result of **str1 = str2** (memory leak)



Second issue is dangling pointer issue as was in the case of copy constructor.

Modified Assignment Operator Code:

So we add overloaded assignment operator to perform deep copy as given below,

[7] Here by term member wise copy we mean copying values of members of class one by one blindly also called bitwise copy.
[8] In ANSI c++ standard term member wise copy has also been used to indicate logical copy (the deep copy)

ORANGE MONKEY TEAM

```
class String{
        ...
public:
        ...
        void operator =(const String &);
};

void String::operator = (const String & rhs){
        size = rhs.size;
        if(rhs.size != 0){
          delete [] bufferPtr; // resolving inaccessible memory issue
          bufferPtr = new char[rhs.size+1]; // creating new dynamic memory
          strcpy(bufferPtr,rhs.bufferPtr); // deep copy
        }
        else
                bufferPtr = NULL;

}

int main(){
        String str1("ABC");
        String str2("DE"), str3("FG");
        str1 = str2;          // Valid...
        str1 = str2 = str3; // Error...
        return 0;
}
```

The problem in statement
str1 = str2 = str3 is,

str1=str2=str3 is resolved as:
str1.operator=(str2.operator=(str3))
Assignment operator is beiing called two times one for part str2 = str3 and then for str1 = (str2 = str3) as assignment operator is right associate so first **str2=str3** will be executed, and str2 will become equal to str3, then first overloaded assignment operator execution result will be assigned to s1,
str1.operator=(**str2.operator=(str3)**)

Problem is when compiler will try to invoke second assignment operator to assign value to str1 error will be returned becuase of void return type of oveloaded assignment operator the reasn is explained below,
As we have studied before values are passed as parameters in operator overloadnig,
str2 = str3 means str2.operator = (str3)
// str3 is being called as parameter
and
str1 = str2 = str3 means str3.operator(str2.operator = (str3))
// str2.operator = (str3) is being passed as parameter

This issue can be resolved by introducing minor change in our code of copy assignment operator to make it return String object instead of void as shown below,

```
class String{
...
public:
        ...
        String & operator = (const String &);
};

String & String :: operator = (const String & rhs){
        size = rhs.size;
        delete [] bufferPtr;
        if(rhs.size != 0){
          bufferPtr = new char[rhs.size+1];
          strcpy(bufferPtr,rhs.bufferPtr);
        }
        else bufferPtr = NULL;
        return *this;
}
```

Now we are returning the value by reference of the object with respect to which this overloaded assignment operator will be called. It will be str2 in the case of str2 = str3, now when part **str1 = (**str2 = str3**)** will be executed, str2 will be passed as argument, that will be assigned to str1.

ORANGE MONKEY TEAM

Lecture No.18

Self assignment problem:

In we assign same string to itself as done in main function below our program will produce unexpected results as source and destination operands for copying are same,

```
int main(){
        String str1("Fakhir");
        str1 = str1; // Self Assignment problem...
        return 0;
}
```

Result of str1 = str1

We can resolve this issue by adding a simple if condition to ensure that both strings are not same

```
String & String :: operator = (const String & rhs){
 if(this != &rhs){
        size = rhs.size;
        delete [] bufferPtr; // deleting memory of left hand side operand
        if(rhs.bufferPtr != NULL){
          bufferPtr = new char[rhs.size+1];
          strcpy(bufferPtr,rhs.bufferPtr);
      // memory access violation or incorrect data copy
          }
        else bufferPtr = NULL;
 }
 return *this;
}
```

Now self-assignment is properly handled:

```
int main(){
        String str1("Fakhir");
        str1 = str1;
        return 0;
}
```

We can make return type String & as constant to avoid assignment to sub expressions, like (str1 = str2) = str3

ORANGE MONKEY TEAM

```
class String{
…
public:
        …
        const String & operator=[9]
                        (const String &);
};

int main(){
        String s1("ABC"),
               s2("DEF"),
               s3("GHI");
        // Error…
        (s1 = s2) = s3;
        return 0;
}
```

But as we can do that with primitive types so we can allow assignment to sub expressions by making return type as String & only as we have done before.

```
int main(){
        int a, b, c;
        (a = b) = c;
        return 0;
}
```

Other Binary operators

Overloading += operator:

```
class Complex{
double real, img;
public:
Complex & operator+=(const Complex & rhs);
Complex & operator+=(const double & rhs);
…
};

Complex & Complex::operator += (const Complex & rhs){
        real = real + rhs.real;
        img = img + rhs.img;
        return * this;
}

Complex & Complex::operator += (const double & rhs){
        real = real + rhs;
        return * this;
```

[9] We have seen previously that we should not return handle to any private data member

ORANGE MONKEY TEAM

```
}

int main(){
        Complex c1, c2, c3;
        c1 += c2;
        c3 += 0.087;
        return 0;
}
```

Friend Functions and Operator overloading

Friend functions minimize encapsulation as we can access private data of any class using friend functions,
This can result in:

- Data vulnerability
- Programming bugs
- Tough debugging

Hence, use of friend functions must be limited we can overload operators without declaring them friend functions of a class, for example the + operator can be defined as a non-member, non-friend function as shown below,
(Three versions of overloaded + operator for Complex no class to handle three kinds of statements)

- obj1 + obj2
    - + obj1
- obj1 + 3.78

```
Complex operator + (const Complex & a, const Complex & b){
        Complex t = a; // creating temporary object t to store a+b
        return t += b; // returning t by reference
}

Complex operator + (const double & a, const Complex & b){
        Complex t = b;
        return t += a;
}

Complex operator + (const Complex & a, const double & b){
        Complex t = a;
        return t += b;
}
```

Other Binary operators

The operators
-=, /=, *=, |=, %=, &=, ^=, <<=, >>=, !=
can be overloaded in a very similar fashion.

ORANGE MONKEY TEAM

Lecture No.19

Overloading stream insertion extraction operators

Stream Insertion operator

Often we need to display the data on the screen c++ provides us insertion operator ('<<')to put data on output stream default is console but it can be any file or network socket as well to send data on network from our program.

Example:

```
int i=1, j=2;
cout << "i= "<< i << "\n";
cout << "j= "<< j << "\n";
```

Stream Extraction operator

We also need to get data from the console or from file or network this is achieved through c++ provided stream extraction operator ('>>') that is used to get data from input stream, again default input stream is from console.

Example:

```
int i,j;
cin >> i >> j; // getting value of i and j from user
```

Explanation:

cin and **cout** are objects of istream and ostream classes used for input and output respectively, the insertion and extractions operators have been overloaded in istream and ostream classes to do these tasks.

When we write lines like,

```
int i;
cin>> i;
cout << i;
```

Actually we are using istream and ostream class objects and using these objects we are calling these classes overloaded (>> and <<) operators that have been overloaded for all basic types like integer, float , long , double and char *.
We have seen previously that actual call of overloaded operators for a class takes place by passing the objects to overloaded function as parameter like shown below,

```
cin>>i;
istream & operator >> (istream & in, int & i)
```

Here cin will be passed as istream object along with int i to be displayed and code of this functions is returning istream object by reference & to accommodate multiple input statement in a single line like,

```
int i , j;
cin>>i >> j;
```

as we did in our string class before.

ORANGE MONKEY TEAM

Same goes for insertion operator <<
cout << i;
ostream & operator >> (ostream & os, const int & i)

Stream insertion and extraction operator have been overloaded for basic data types but if we try to use them for user defined data types like our Complex no. class compiler will generate error as it will not find any overloaded operator code for our complex no class

```
Complex c1;
cout << c1; // Error
cout << c1 << 2; // Error cascaded statement
```

// Compiler error: binary '<<' : no operator // defined which takes a right-hand operand of type 'class Complex'

Same error will be for stream extraction operator so will need to overload these two operators (<< and >>) for our Complex no. class.

Overloading Stream Insertion Operator

First we try to overload insertion << operator as member function as we did before,

```
class Complex{
        …
public:
        …
        void operator << (const
                                Complex & rhs);
};
```

But when we will call this overloaded function is main compiler will generate errors as shown below,

```
int main(){
        Complex c1;
        cout << c1;             // Error
        c1 << cout;
        c1 << cout << 2; // Error
        return 0;
};
```

```
class Complex{
        …
public:
        …
        void operator << (ostream &);
};

void Complex::operator << (ostream & os){
```

```
        os      << '(' << real
                << ',' << img << ')';
}
```

Now the statement c1 << cout will work but it has two limitations,
Difficult to understand and remember statement syntax (c1 << cout ;)
Cascaded statements not possible (cout << c1 << 2 ;)

Better syntax is given below to resolve these two issues,

```
class Complex{
  ...
friend ostream & operator << (ostream & os, const Complex & c);
};
```

Stream Insertion operator

```
// we want the output as: (real, img)
ostream & operator << (ostream & os, const Complex & c){
        os << '(' << c.real
          << ','
          << c.img << ')';
        return os;
}
```

ostream reference can not be const as it store the data in its buffer to insert on output stream, however Complex reference will be constant as we are only getting data from Complex object and inserting it to output stream.

```
Complex c1(1.01, 20.1), c2(0.01, 12.0);
cout << c1 << endl << c2;
```

Stream Insertion operator

Output:

```
( 1.01 , 20.1 )
( 0.01 , 12.0 )
```

Now cascading statements are also possible as given below,

```
cout << c1 << c2;
```

is equivalent to

operator<<(**operator<<(cout,c1)**,c2);

Because insertion operator is Left to right associative so first left part **cout << c1** << c2 will be executed and then the next part as opposed to copy assignment operator that will right associative.

Same thing can be done with stream extraction operator,

Overloading Stream Extraction Operator:

ORANGE MONKEY TEAM

```
class Complex{
  ...
friend istream & operator >> (istream & i, Complex & c);
};
```

istream can not be cont and istream buffer will change as we will get data from it and assign it to complex reference similarly Complex object can not be const for stream extraction operator as well because we will add data to it and hence its state will change.

Stream Extraction Operator Code:

```
istream & operator << (istream & in, Complex & c){
        in >> c.real;
        in >> c.img;
        return in;
}
```

Main Program:

```
Complex c1(1.01, 20.1);
cin >> c1;
// suppose we entered    // 1.0025 for c1.real and     // 0.0241 for c1.img
cout << c1;
```

Output:

```
( 1.0025 , 0.0241 )
```

Other Binary operators:

Overloading comparison operators (Equality and Inequality operators)

```
class Complex{
public:
        bool operator == (const Complex & c);
//friend bool operator == (const //Complex & c1, const Complex & c2);
        bool operator != (const Complex & c);
//friend bool operator != (const //Complex & c1, const Complex & c2);
        ...
};
```

Equality operator:

```
bool Complex::operator ==(const Complex & c){
        if((real == c.real) &&
          (img == c.img)){
```

ORANGE MONKEY TEAM

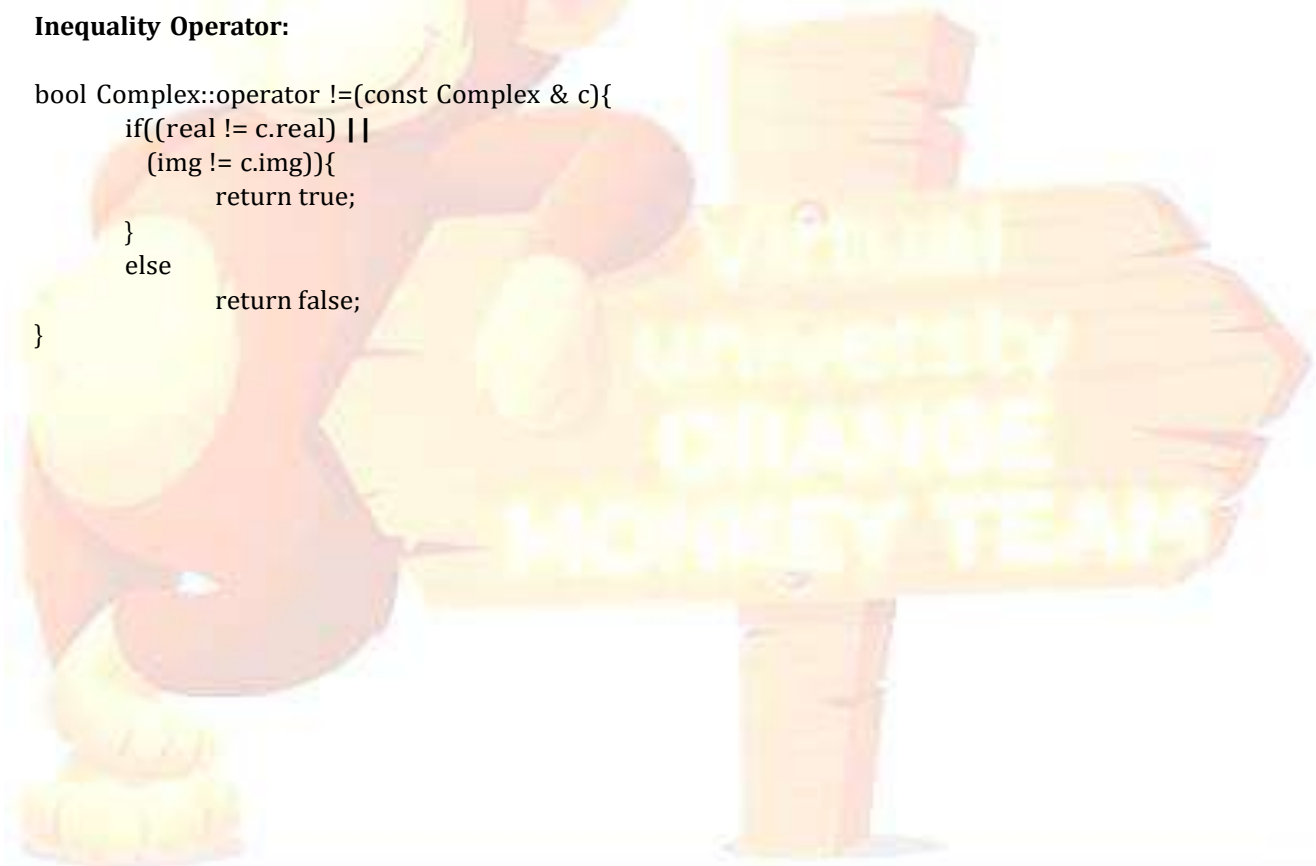
```
                return true;
        }
        else
                return false;
}
```

As non member friend function:

```
bool operator ==(const Complex& lhs, const Complex& rhs){
        if((lhs.real == rhs.real) &&
          (lhs.img == rhs.img)){
                return true;
        }
        else
                return false;
}
```

Inequality Operator:

```
bool Complex::operator !=(const Complex & c){
        if((real != c.real) ||
          (img != c.img)){
                return true;
        }
        else
                return false;
}
```



Lecture No.20

Modified String Class:

We have seen the following string class till now,

```
class String{
private:
        char * bufferPtr;
    int size;
public:
        String();
        String(char * ptr);
        void SetString(char * ptr);
        const char * GetString();
        ...
};

int main(){
        String str1("Test");
        String str2;
        str2.SetString("Ping");
        return 0;
}
```

What if we want to change the string from "*Ping*" to "*Pong*"?? *{ONLY 1 character to be changed…}*

Possible solution:

- Call: **str2.SetString("Pong");**
- This will delete the current buffer and allocate a new one
- Too much overhead if string is too big

Or, we can add a function which changes a character at nth location

```
class String{
        ...
public:
        void SetChar(char c, int pos);
        ...
};

 void SetChar(char c, int pos){
        if(bufferPtr != NULL){
                if(pos>0 && pos<=size)
                        bufferPtr[pos] = c;
        }
}
```

Other Binary Operators

Now we can efficiently change a single character:

```
String str1("Ping");
```

ORANGE MONKEY TEAM

str1.SetChar('o', 2);
// str1 is now changed to "Pong"

Subscript [] Operator

There is another elegant solution present for this problem. It is subscript operator that is used on basic char [] data type as shown below,

```
char array[5] = "Ping";
array[1] = 'o';
```

We want to get same functionality for our own defined String class for this we overload the subscript **"[]"** operator.

We want function like given below for subscript operator in our String class,

```
int main(){
        String str2;
        str2.SetString("Ping");
        str[2] = 'o';
// acting as l-value (left value so that we can assign it some value)
        cout << str[2];
// acting as r-value (we are reading value using subscript operator)
        return 0;
}
```

Overloading Subscript [] Operator

Subscript operator must be overloaded as member function of the class with one parameter of integer type,

```
class String{
        …
public:
        char & operator[](int);
        …
};

char & String::operator[]( int pos){
        assert(pos>0 && pos<=size);
        return stringPtr[pos-1];
}
int main() {
        String s1("Ping");
        cout <<str.GetString()<< endl;
        s1[2] = 'o';
        cout << str.GetString();
        return 0;
}
```

Output:

ORANGE MONKEY TEAM

```
Ping
Pong
```

Overloading Function () operator

Must be a member function
Any number of parameters can be specified
Any return type can be specified
Operator() can perform any generic operation

Function Operator

```
class String{
        …
public:
        char & operator()(int);
        …
};

char & String::operator()
                                (int pos){
        assert(pos>0 && pos<=size);
        return bufferPtr[pos-1];
}
int main(){
String s1("Ping");
char g = s1(2);  // g = 'i'
s1(2) = 'o';
cout << g  <<  "\n";
cout << str.GetString();
return 0;
}
```

Output:

```
i
Pong
```

Function Operator performing Sub String operation,

```
class String{
        …
public:
        String operator()(int, int);
        …
};

String String::operator()(int index,     int subLength){
        assert(index>0 && index+subLength-1<=size);
        char * ptr = new char[subLength+1];
```

```
        for (int i=0; i < subLength; ++i)
    ptr[i] = bufferPtr[i+index-1];
        ptr[subLength] = '\0';
        String str(ptr);
        delete [] ptr;
        return str;
}

int main(){
        String s("Hello World");
        // "<<" is overloaded
        cout << s(1, 5);
        return 0;
}
```

Function Operator
Output:

```
Hello
```

Unary Operators

Unary operators take one operand, they act on the object with reference to which they have been called as shown below,
& * + - ++ -- ! ~

Examples:

- **--x**
- **-(x++)**
- **!(*ptr ++)**

Unary Operators

Unary operators are usually prefix, except for **++** and **--**
++ and **--** both act as prefix and postfix

Example:
h++;
g-- + ++h - --i;

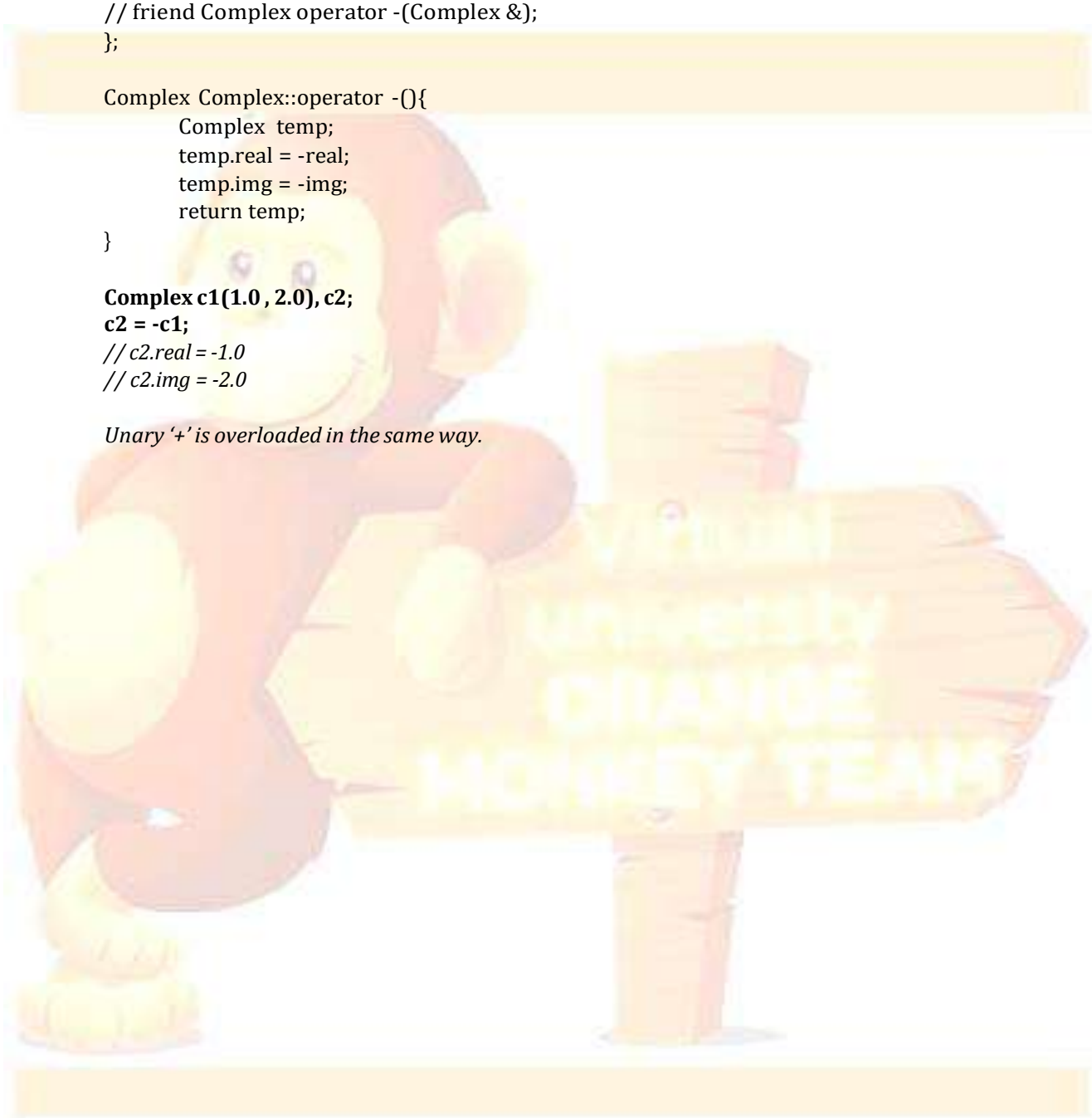
General syntax for unary operators
As Member Functions:
TYPE & operator OP (); // no argument the object with respect to which it is called is taken as one operand
As Non-member Functions:
Friend TYPE & operator OP (TYPE & t);
// one argument object with respect to which it is called.

Overloading unary '-':

```
class Complex{
...
Complex operator - ( );
// friend Complex operator -(Complex &);
};

Complex Complex::operator -(){
        Complex  temp;
        temp.real = -real;
        temp.img = -img;
        return temp;
}
```

Complex c1(1.0 , 2.0), c2;
c2 = -c1;
// c2.real = -1.0
// c2.img = -2.0

Unary '+' is overloaded in the same way.



ORANGE MONKEY TEAM

Lecture No.21

Unary Operators

Behavior of ++ and -- for pre-defined types:

- Post-increment **++:**

Post-increment operator ++ increments the current value and then returns the previous value

- Post-decrement --: Works exactly like post ++

Example:

```
int x = 1, y = 2;
cout << y++ << endl;
cout << y;
```

Output:

2
3

Example:

```
int y = 2;
y++++;// Error
y++ = x;          // Error
```

Behavior of ++ and -- for pre-defined types:

- Pre-increment ++:

Pre-increment operator ++ increments the current value and then returns it's reference

- Pre-decrement --:

Works exactly like Pre-increment ++

Example:

```
int y = 2;
cout << ++y << endl;
cout << y << endl;
```

Output:

3
3

Example:

```
int x = 2, y = 2;
++++y;
cout << y;
++y = x;
cout << y;
```

Output:

4
2

ORANGE MONKEY TEAM

Example (Pre-increment):

```
class Complex{
        double real, img;
public:
...
        Complex & operator ++ ();
    // friend Complex & operator ++(Complex &);
};

Complex & Complex::operator++(){       // member function
        real = real + 1;
        return * this;
}

Complex & operator ++ (Complex & h){ // non member function
        h.real += 1;
        return h;
}
```

Example:

```
Complex h1, h2, h3;
++h1;
```

Function **operator++()** returns a reference so that the object can be used as an *lvalue*

```
++h1 = h2 + ++h3;
```

How does a compiler know whether it is a pre-increment or a post-increment?

A post-fix unary operator is implemented using:
Member function with 1 dummy int argument
OR
Non-member function with two arguments

In post increment, current value of the object is stored in a temporary variable
Current object is incremented
Value of the temporary variable is returned

Post-increment operator:class

```
Complex{
...
        Complex operator ++ (int);
// friend Complex operator ++(const Complex &, int);
};

Complex Complex::operator ++ (int){
        complex t = *this;
```

ORANGE MONKEY TEAM

```
        real += 1;
        return t;
}

Complex operator ++ (const
                    Complex & h, int){
        complex t = h;
        h.real += 1;
        return t;
}
```

How does a compiler know whether it is a pre-increment or a post-increment?

The dummy parameter in the operator function tells compiler that it is post-increment
Example:

```
Complex h1, h2, h3;
h1++;
h3++ = h2 + h3++; // Error…
```

The *pre* and *post* decrement operator **--** is implemented in exactly the same way

Type Conversion

The compiler automatically performs a type coercion of compatible types
e.g:

```
int f = 0.021;
double g = 34;
```

// type **float** *is automatically converted into* **int***. Compiler only issues a warning…*

Type Conversion

The user can also explicitly convert between types:

```
int g = (int)0.0210;
double h = double(35);
```

// type **float** *is explicitly converted (casted) into* **int***. Not even a warning*
// is issued now…

Type Conversion

For user defined classes, there are two types of conversions
From any other type to current type
From current type to any other type

We can do the following type conversion by our self,
Conversion from any other type to current type:
Requires a constructor with a single parameter
For example,

```
String str = 135;
```

Conversion from current type to any other type:
Requires an overloaded operator

ORANGE MONKEY TEAM

For example,
String str;
char * ptr = str;

Conversion from other type to current type (int to String):

```
class String{
...
public:
        String(int a);
        char * GetStringPtr()const;
};
String::String(int a){
        cout << "String(int) called..." << endl;
        char array[15];
        itoa(a, array, 10);
        size = strlen(array);
        bufferPtr = new char [size + 1];
        strcpy(bufferPtr, array);
}
char * String::GetStringPtr() const{
        return bufferPtr;
}
int main(){
        String s = 345;
        cout << s.GetStringPtr() << endl;
        return 0;
}
```

Output:

```
String(int) called...
345
```

Automatic conversion like shown above using constructor has drawbacks conversion takes place transparently even if the user didn't wanted the conversion for example see the code below, in it user can write the following code to initialize the string with a single character:

```
int main(){

        String s = 'A';
// ASCII value of A that is 65 is being taken automatically it may be what the user //
wants, perhaps user intent was String s = "A" to store A in the string object
// but instead 65 is being stored

        cout    << s.GetStringPtr()<< endl << s.GetSize() << endl;
        return 0;
```

ORANGE MONKEY TEAM

}

Output:

```
String(int) called...
65
2
```

Keyword **explicit**

There is a mechanism in C++ to restrict automatic conversions like this using constructor code it is to use keyword **explicit** with such constructors if we have used this keyword then casting must be explicitly performed by the user.
Keyword **explicit** only works with constructors.

Example:

```
class String{
        …
public:
        …
        explicit String(int);
};

int main(){
        String s;
        s = 'A';            // Error…
        return 0;
}

int main(){
        String s1, s2;
        s1 = String(101);       // valid, explicit casting…
        // OR
        s2 = (String)204;
        return 0;
}
```

Type Conversion

There is another method for type conversion:
"Operator overloading"
It is used for converting from current type (user defined) to any other basic type or user defined type.

General Syntax:

$TYPE_1$::operator $TYPE_2$();
Like,

```
String::operator char * ();  // TYPE1 = String , TYPE2 = char *
```

ORANGE MONKEY TEAM

It will convert our string object to char *.

We write such functions as member function of the class.
NO return type and arguments are specified
Return type is implicitly taken to be TYPE2 by compiler

Type Conversion

Overloading pre-defined types:

```
class String{
        …
public:
        …
        operator int();
        operator char *();
};
String::operator int(){
        if(size > 0)
                return atoi(bufferPtr);
        else
                return -1;
}

String::operator  char  *(){
        return bufferPtr;
}


int main(){
        String s("2324");
        cout << (int)s << endl << (char *)s;
   //    int a = (int)s;
   //    int a = int (s);
        return 0;
}
```

Output:

```
2324
2324
```

User Defined types:

User-defined types can be overloaded in exactly the same way
Only prototype is shown below:

```
class String{
        …
        operator  Complex();
        operator   HugeInt();
        operator IntVector();
```

```
};
```

Drawbacks of Type Conversion Operator:

```
class String{
        …
public:
        …
        String(char *);
        operator int();
};

int main(){
        String s("Fakhir");
        // << is NOT overloaded
        cout << s; // compiler is automatically converting s to int
        return 0;
}
```

Output:

Junk Returned…

To avoid this problem DO NOT use *type conversion operators* instead use separate member function for such type conversion as shown below,

Modifying String class:

```
class String{
        …
public:
        …
        String(char *);
        int AsInt();
};
int String::AsInt(){
        if(size > 0)
                return atoi(bufferPtr);
        else
                return -1;
}
int main(){
        String s("434");
        // << is NOT overloaded
        cout << s; //error
        cout << s.AsInt();
        return 0;
}
```

ORANGE MONKEY TEAM

Lecture No.22

Practical implementation of Inheritance in c++

Topics to be discussed in this lecture are,

- Inheritance in Classes
- UML Notation of Inheritance
- Types of Inheritance in c++
- IS A relationship
- Accessing members
- Allocation in memory
- Constructors
- Base class initializers
- Initializing members
- Destructors
- Order of execution constructors and destructors
- Examples of C++ implementation of inheritance

Inheritance in Classes

If a class B inherits from class A, then B contains all the characteristics (information structure and behavior) of class A.
The class whose behavior is being inherited is called base class and the class who inherits the behavior of base class is called derived class. Base class is also called parent class and child class is called also derived class
Besides inherited characteristics, derived class may have its own unique characteristics

UML Notation

We use arrow from derived class to the parent class to show inheritance as shown below,

Inheritance in C++

In c++ we can inherit a class from another class in three ways,

- Public
- Private
- Protected

ORANGE MONKEY TEAM

"IS A" Relationship

Inheritance represents "IS A" relationship for example "a student IS A person".
In general words we can say that inheritance represents,

"Derived class IS A kind of Parent class"

C++ Syntax of Inheritance

```
class ChildClass
        : public BaseClass{
...
};
```

Example

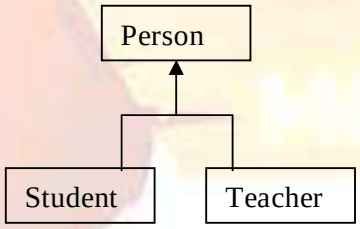
```
class Person{
...
};
class Student: public Person{
...
};
```

Accessing Members

Public members of base class become public member of derived class.
Private members of base class are not accessible from outside of base class, even in the derived class (Information Hiding)

Example

In the code given below Student and Teacher classes has been derived from single Person class,




```
class Person{
   char *name;
   int age;
   ...
public:
   const char *GetName() const;
   int GetAge() const;
   ...
};
```

ORANGE MONKEY TEAM

```
class Teacher: public
   Person{
   char * dept;
   int course;
   ...
public:
   char * GetDept() const;
   int GetCourse() const;
   void Print() const;
   ...
};
```

```
class Student: public
   Person{
   int semester;
   int rollNo;
   ...
public:
   int GetSemester() const;
   int GetRollNo() const;
   void Print() const;
   ...
};
```

Example

```
void Student::Print()
{
        cout << name << " is in" << " semester " << semester;
}
```

Error

corrected Code:

```
void Student::Print()
{
        cout << GetName() << " is in semester " << semester;
}
int main(){
        Student stdt;

        stdt.semester = 0;//error
        stdt.name = NULL; //error
        cout << stdt.GetSemester();
        cout << stdt.GetName();

        return 0;
}
```

Explanation of above code (char * data type)

In C++ char arrays (char []) are handled in two ways one way is statically using statements like,

char name[30]; // static array of length 30 characters

or dynamically as shown below,

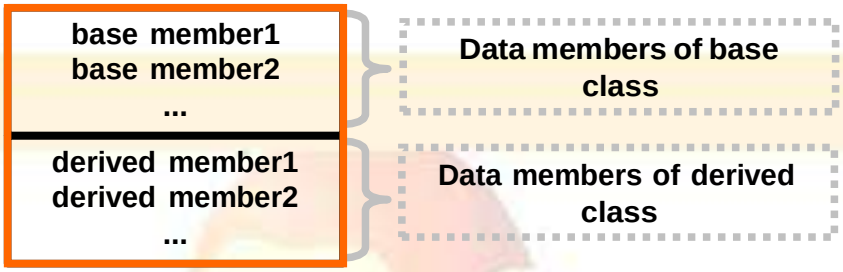
char * name;

name = new char[30];

In dynamic creation of arrays we simply store char * in class and assign it a dynamic memory according to our need using new operator.

Allocation in Memory

The object of derived class is represented in memory as follows

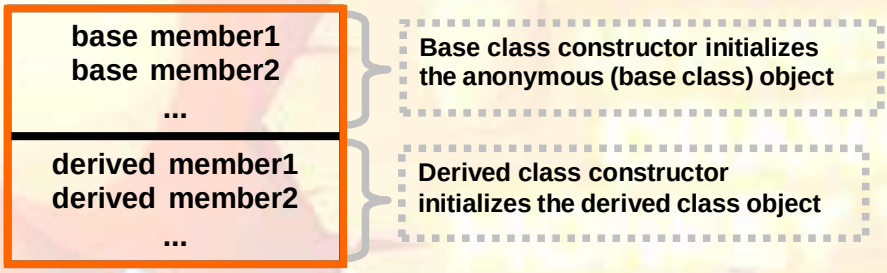


Derived Class Object

Every object of derived class has an anonymous object of base class

Constructors

- The anonymous object of base class must be initialized using constructor of base class
- When a derived class object is created the constructor of base class is executed before the constructor of derived class



Derived Class Object

Example

```
class Parent{
public:
        Parent(){ cout <<
                "Parent Constructor...";}
};
class Child : public Parent{
public:
        Child(){        cout <<
                "Child Constructor...";}
};
```

ORANGE MONKEY TEAM

```
int main(){
        Child cobj;
        return 0;
}
```

Output:

```
Parent Constructor...
Child Constructor...
```

Constructor

- If default constructor of base class does not exist then the compiler will try to generate a default constructor for base class and execute it before executing constructor of derived class
- If the user has given only an overloaded constructor for base class, the compiler will not generate default constructor for base class

Example

```
class Parent{
public:
        Parent(int i){}
};
class Child : public Parent{
public:
        Child(){}
} Child_Object;  //ERROR
```

Definition of Some Terms:

Default constructor: Default constructor is such constructor which either has no parameter or if it has some parameters these have default values. The benefit of default constructor is that it can be used to create class object without passing any argument.

Implicit Default constructor:

Compiler generates implicit default constructor for any class in case we have not given any constructor for the class.

Explicit Default constructor:

If user has given constructor for any class without any arguments or with all arguments with default values then it is also default constructor according to definition but it is explicit (user defined) default constructor.

Now if a base class has only **non-default constructor** (constructor with parameters without default values), then when we will create object of any class derived from this base class compiler will not be able to call base class constructor as base class has no default constructor (constructor that can be called without giving any parameters) so compiler will generate error.

We can avoid this error by calling base class non-default constructor in derived class constructor initializer list by ourself.

ORANGE MONKEY TEAM

Base Class Initializer

- C++ has provided a mechanism to explicitly call a constructor of base class from derived class

- The syntax is similar to member initializer and is referred as base-class initialization

Example

```
class Parent{
public:
        Parent(int i){...};
};
class Child : public Parent{
public:
        Child(int i): Parent(i)
        {...}
};
```

Example

```
class Parent{
public:
        Parent(){cout <<
                "Parent Constructor...";}
        ...
};
class Child : public Parent{
public:
        Child():Parent()
        {cout << "Child Constructor...";}
        ...
};
```

Base Class Initializer

- User can provide base class initializer and member initializer simultaneously

Example

```
class Parent{
public:
        Parent(){...}
};
class Child : public Parent{
        int member;
public:
        Child():member(0), Parent()
        {...}
};
```

Base Class Initializer

- The base class initializer can be written after member initializer for derived class

ORANGE MONKEY TEAM

- The base class constructor is executed before the initialization of data members of derived class.

Initializing Members

- Derived class can only initialize members of base class using overloaded constructors
  - Derived class can not initialize the public data member of base class using member initialization list

Example

```
class Person{
public:
        int  age;
        char *name;
        ...
public:
        Person();
};
```

Example

```
class Student: public Person{
private:
        int semester;
...
public:
        Student(int a):age(a)
        {                                   //error
        }
};
```

Reason

- It will be an assignment not an initialization

Destructors

- Destructors are called in reverse order of constructor called
- Derived class destructor is called before the base class destructor is called

Example

```
class Parent{
public:
        Parent(){cout <<"Parent Constructor";}
        ~Parent(){cout<<"Parent Destructor";}
};

class Child : public Parent{
public:
        Child(){cout << "Child Constructor";}
        ~Child(){cout << "Child Destructo";}
};
```

Example

Output: **Parent Constructor**

ORANGE MONKEY TEAM

Child Constructor
Child Destructor
Parent Destructor

ORANGE MONKEY TEAM

© Virtual University of Pakistan

AL-JUNAID TECH INSTITUTE

CS304-OOP

MIDTERM MCQS

Prepared by: JUNAID MALIK

AL-JUNAID TECH INSTITUTE

www.vulmshelp.com

Language Courses Training Available

I'm providing paid courses in different languages within 3 Months,Certificate will be awarded after completion.

- HTML
- CSS
- JAVASCRIPT
- BOOTSTRAPS
- JQUERY
- PHP MYSQL
- NODES.JS
- REACT JS

LMS Handling Services

LMS Activities Paid Task

Assignments 95% Results

Quizes 95% Results

GDB 95% Results

For CS619 Project Feel Free To Contact With Me

Ph# 0304-1659294

Email: junaidfazal08@gmail.com

AL-JUNAID TECH INSTITUTE

SR NO	MCQS QUESTION	ANSWER
1	In expression c1*c2+c3-c4 which of the following will be executed in first order	C1*c2
2	Inheritance is a way to	Add features to existing classes without rewriting them
3	---------- represent “IS A ”relationship	Inheritance
4	Constructor without any parameter is called:	Default constructor
5	Which of the following stores the address of a variable?	Pointer
6	A Post-fix unary operator is implemented in C ++ using Member function with	1 dummy int argument
7	minimum classes required in a program for implementing multiple inheritance	3
8	Which of the following statement(s) describe constant member functions?	All of the given option
9	composition is a strong relation relationship because	Composed object cant exist independency
10	How many interactions are there in the following scenario Imran has the car. he drive the car and car has four wheels and doors	3
11	if we extend our model ,and the rest of the model is not affected then it is called-----	Flexibility
12	choose correct declaration of overload*=1 operator for class complex as Member function	Complex and operator*=(const complex & c)
13	Consider the following class A{int a,b,c; public: A():B(10),C(0)a(7) };	A,b,c
14	Class is a blue print of	Objects

AL-JUNAID TECH INSTITUTE

15	Suppose we have a class student which of the following statement is correctly declaring the pointer to the object of student class?	Student*object
16	In c++ composition is relationship between -------and------objects	Whole, part
17	Which of the following Depicts the proper definition of class?	a class is description of a kind of object
18	in class attributes are represented by ---- and------ respectively	data member, Member function
19	which of the following represents the two Association	Employee works for the company
20	Entities in OO model are represented by	Rectangle
21	Which of the following features of OOP is used to derive a class from another?	Inheritance
22	In C ++ which of the following is defined as a stream insertion operator?	<<
23	choose the correct declaration of overloaded equality (==)operator for class string as a Member function	Bool operator==(const string and s)
24	---- is the relationship between the part object and the whole object	Composition
25	object hides its information from others this concept is known as	information hiding
26	the concept of derived classes is involved in	Inheritance
27	abstraction provides information according to	user perspective
28	which of the following is not an object Association	Inheritance
29	which of the following classes are used by amphibious vehicle to inherit characteristics	both land and water vehicle
29	object data includes	both states and attributes of an object

AL-JUNAID TECH INSTITUTE

30	choose correct declaration of overloaded inequality operator for class string as non member friend function	friend pool operator !=(const string &s1,const string &2)
31	How many objects of a given class may be constructed in an application?	as many as the application ask for
32	how many objects of a given class may be constructed in an application	as many as the application ask for
33	Which of the following statements describe constant member functions?	all of the given option
34	student *ptr=new student(" Ali"); is an example of	static memory allocation
35	in------- base class can be replaced by the derived class	extension
36	Which of the following is a correct way of declaring constant variable ROLLNO?	cosnst int Roll no;
37	which of the following operator doesn't take argument hit overloaded	none of the given choices
38	which of the following is part of the Class diagram	all of the given option
39	a child inherits characteristic from its	parent
40	which of the following issues can be caused by the use of friend function	programming Bugs
41	which of the following is not true about constant number function	all of the given option
42	Which one is not the main features of object oriented programming?	exception handling
43		
44	in object oriented programming, objects communicate with each other through	messages
45	static data member is declared	inside the class
46	the concept of derived classes is involved in	inheritance
47	Which of the following can we attribute of an object "Ali"?	address

AL-JUNAID TECH INSTITUTE

48	Which of the following keyword tell the compiler to substitute the code within the function definition for every instance of a function call?	inline
49	inheritance is the relationship between	classes
50	how many objects are involved in the binary Association	2
51	all objects of same class has -------- attributes	same
52	which of the following concept is more close to encapsulation	information hiding
53	member initialization list is used to	initialize data member
54	aggregation between objects show that	objects have week relationship
55	there is /are ways to create user defined types for objects in C ++	two
56	composition is a strong relationship because:	composed object can not exist independently
57	in polymorphism messenger can be interpreted in different ways depending upon the----- class	receiver
58	which of the following is most suitable advantage of information hiding	simplifies the model
59	the process of extracting common characteristics from two or More classes and combining them into a generalized superclass is called	Generalization
60	------- operator is used to create objects at run time	new
61	choose the correct option to declare an overloaded Stream insertion operator for userdefind class "string" as a non-member friend function	friend out stream and operator<<(ostream and os,const string & s)
62	static data member is declared	inside the class
63	an instance of user defined type is called	object

AL-JUNAID TECH INSTITUTE

64	which of the following is true about constructor	all of the given option
65	Which of the following correct way of declaring constant variable ROLLNO?	const int ROLLNO;
66	inheritance is shown by	filled arrow
67	which of the following operators operate on one operand	unary operator
68	which of the following will happen when a constant function tried to change the value of data members of the class	compile time error will occur
69	in object oriented programming, objects communicate with each other through---	messages
70	How the information hidden within an object can be accessed?	through its interface
71	object orientation approach can be achieved using object and their-------	interactions
72	constructors have ----- return type	no
73	which of the following is an example of unary operator	increment operator(++)
74	memory is allocated to non-static members only when	object is created
75	in class B inherit from class A which of the following is child class	class B
76	the------ tells the compiler what task the function will be performing	function definition
77	Consider the statement" room has chair" which of the following type of Association exists between whom and chair?	aggregation
78	how many objects are involved in the N-ary Association	more than 3
79	deconstruct is a function which has the same name as that of class but starts with a----- sign	~
80	which of the following is a tangible entity	car

AL-JUNAID TECH INSTITUTE

81	member initializer list is used to	initialize data members
82	we can get the address of variable stored in pointer using-----	& symbol
83	two sentence object oriented programming book in bookshelf is an example of	aggregation
84	which of the following best defines the obstruction	hide the implementation and show current perspective
85	which of the following define default constructor 5 class student	student(){//..}
86	object data includes	both state and attributes of an object
87	minimum classes required in a program for implementing multiple inheritance	3
88	information hiding can be achieved through-------	encapsulation, abstraction
89	compiler generated defaulter structure is called----- hand user written default constructor is called----	implicit, explicit
90	which of the following is true about constructor	all of the given option
91	In which of the following different forms office single entity exist?	polymorphism
92	data members of a class student can be:	all of the given option
93	the----- turn off the compiler what does the function will be performing	function definition
94	Suppose a class does not have any constructor what will happen when an object of this class is created?	compiler will call implicit default constructor
95	how can we differentiate between constructor and destructor	destructor are preceded with a Tilda(~) symbol, and constructor are not preceded with any symbol

AL-JUNAID TECH INSTITUTE

96	---- is used for hiding the details of an object from the other parts of a program	encapsulation
97	Class diagram include	class name, attributes, operation
98	through interface was access objects----------	behavior
99	composition is a strong relationship because	compose object cannot exist independently
100	copy constructor is called when:	an object is created in term of Preexisting object
101	choose the correct option to declare an overloaded stream insertion operator for a user defined class string as a non-Member friend function	friend ostream & operator(ostream &os.const string &s)
102	a post-fix unary operator implemented in C plus plus using non Member function with---- arguments	two
103	in class attributes and behaviors are represented by---- and--- respectively	data members, member functions
104	which of the following best describes the relationship between book and book chapter classes	composition
105	which of the following is true about destructor	all of the given option
106	in composition----- are called from composing objects to composed subjects	destructors
107	which of the following can be used an alternative to Static members	Global variables
108	which of the following statement is true about static function of a class	it is used to access static data members
109	which of the following operator does not take any arguments if overloaded	none of the given choices
110	what is the output of the following code int main(){int cost x=10;cout<<++x;return0;}	error

AL-JUNAID TECH INSTITUTE

111	abstraction includes-------	relevant information
112	Which of the following can be the behavior of an object "Usman"?	eat
113	a class defined within another class is	nested class
114	when an object is initialized, is automatically called by the compiler	constructor
115	the compiler treats the Member function of a class as	inline function
116	an object has attributes operation ,and----	unique identity
117	Which of the following is not an access specifier in C ++?	hidden
118	which of the following parts of an object exhibits its state	data
119	---- is the relationship between the part object and the whole object	composition
120	how can we identify classes from a given problem statement	by extracting nouns from problem statement
121	what "friend" should be placed in	function declaration
122	generalization is-------- approach	bottom-up
123	static data members can be accessed through--------	dot operator and scope resolution operator
124	composition age----- and aggregation is---- relationship	strong, weak
125	suppose there is an object of type person, which of the following can be considered as one of its attributes	name
126	Which of the following depicts the proper definition of class?	a class is a description of a kind of object
127	a child inherits characteristics from its------	parent
128	the car is composed of wheels what is the relationship between car and wheels?	composition

AL-JUNAID TECH INSTITUTE

129	------ is creating objects of one class inside another class	composition
130	the other name of specialization is-------	restriction
131	suppose a is an int type static data member class test then which of the following is correct way of initialization the static variable outside the class test	int test::a=0
132	Which of the following stores the address of variable?	Pointer
133		
134		
135	static data member is declared	inside the class
136	polymorphism makes the system	all of the given option
137	which of the following statement best describes the constructor	constructor is used to initialize the data members of a class
138	data member(s) of a class student can be	all of the given option
139	which of the following is a strong relationship	composition
140	all objects of same class has------- attributes	same
141	in C ++ composition is a relationship between----and---- objects	whole, part
142	suppose 12 class string assignment operator is overloaded with following declaration void operator=(const string &); what will happen when we will write following statement in main() str=str2=str3;where,str1.str2,str3 are object of class string	compiler will generate compile time error
143	in class B inherits from class A then it contains all characteristics of----	class A

AL-JUNAID TECH INSTITUTE

144	which of the following is not an example of multiple inheritance	car
145	which of the following can be created without any object of a class?	static data members
146	consider the following code segment class test{int a;int b;int c;}	junk value,5, junk value
147	object orientation approach can be achieved using objects and their-----	interactions
148	member functions defined inside a class declaration are------- by default	private
149	how can we identify classes from a given problem statement	by extracting nouns from problem statement
150	which of the following statement is not true about static variable of a class?	static variable belongs to particular instance of a class
151	there are------ types of constructor in C++	3
152	statement yasir is a friend of Ali and Ali is a friend of yasir is an example of	two way Association
153	this pointer cannot be passed implicitly to-------- functions	static member
154	which of the following issues can be caused by the use of friend function	all of the given
155	in------ send a message does not need to know exact class of the receiver	none of the given option
156	minimum classes required in a program for implementing multiple inheritance	3
157	-------Associates the object of the exact two classes	binary Association
158	main characteristics of information hiding is	keep the model simple
159	static data member is declared	inside the class
160	in expression b=c/d, the arity of division(/) operator is	2

AL-JUNAID TECH INSTITUTE

161	we can allocate dynamic memory in C ++ through operator	new
162	Which of the following is true about accessor functions?	all of the given option
163	in---- base class cannot always be replaced by the derived class	specialization
164	abstraction help to	reduce the complexity
165	relationship in which child object gets destroyed if parent object is destroyed	composition
166	object oriented programming is better than procedural programming because:	it is more close to reality
167	The derived class inherits behavior from the base class and also exhibits its own behavior. this concept is known as---	Overriding
168	the other name of subtyping is	Extension
169	A good model is------ related to life problem	closely
170	in generalization general class represents the -------- behavior of the class	common
171	how can we identify from a given problem statement?	by extracting noun from problem statement
172	which of the following is not a access specifier in C ++	hidden
173	in composition------ are called from composing objects to composed objects	destructors
174	a class have only---- destructor	one
175	suppose you have following C + + statement int old value=10; int new value=++old value; what will be the value of old value and newValue after executing above statements ?	oldVlue=11,newVlue=11
176	in constant Member function the type of this pointer is	constant pointer to constant data
177	----- is creating objects of one class inside another class	composition

178	-------- Associates the objects of the exact three classes	ternary Association
179	generalization is implemented through	inheritance
180	which of the following is strong relationship	composition
181	composition is a strong relationship because	composed object cannot exist independently
182	----- is set of function of an object exposed of other objects	interface
183	suppose student is a class, which of the following defines constructor with one parameter for class student	student(char*aName);
184	which of the following is not true about constant Member function	all of the given option
185	----- define the order of evaluation of an operator is an expression	operator precedence
186	choose correct declaration of overloaded inequality(!=) operator for class string as non member friend function	friend bool operator!=(const string &s1,cont string &s2)
187	which of the following is most suitable advantage of information hiding	simplifies the model
188	a fan has wings, which type of relation exists between fan and Wings in this sentence	composition
189	multiple inheritance is implemented between class through------ symbol	comma(,)
190	postfix unary operator is implemented in C ++ using non Member function with arguments	two
191	Supposed derived class is inherited from base class what happens when a derived class object is created	the constructor of base class exact is executed before the constructor of derived class
192	assignment operator are	right associative

193	a postfix unary operator is implemented in C ++ using number function with	1 dummy int argument
194	which of the following is true about exception functions	all of the given option

Question No: 1

Which part of an object exhibits its state?

- ► **Data (Page 23)**
- ► Operations
- ► Any public part
- ► Any private part

Question No: 2

Inheritance is a way to

- ► organize data.
- ► pass arguments to objects of classes.
- ► **add features to existing classes without rewriting them. (Page 27)**
- ► improve data-hiding and encapsulation.

Question No: 3

Suppose you have been given the following design,

"A person has a name, age, address and sex. You are designing a class to represent a type of person called a patient. This kind of person may be given a diagnosis, have a spouse and may be alive".

Given that the person class has already been created, what of the following would be appropriate to include when you design the patient class?

AL-JUNAID TECH INSTITUTE
www.vulmshelp.com

- **registration date and diagnosis**
- age and sex
- sex and diagnosis
- diagnosis and age

Question No: 4

What problem(s) may occur when we copy objects without using deep copy constructor?

- Dangling pointer
- Memory Leakage
- **All of the given (Page 147)**
- System crash

Question No: 5

this pointers are not accessible for static member functions.

- True (Page 114)
- False

Question No: 6

A static member function cannot be declared.

- Static
- Implicit
- Explicit
- **Virtual**

Question No: 7

___ remain in memory even when all objects of a class have been destroyed.

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

- **Static variables (Page 111)**
- Instance variable
- Primitive variables
- None of given

Question No: 8

Friend functions are _____ functions of a class.

- None of given
- object member
- **non-member (Page 136)**
- data member

Question No: 9

_____, which means if A declares B as its friend it does NOT mean that A can access private data of B. It only means that B can access all data of A.

- **Friendship is one way only**
- Friendship is two way only
- NO Friendship between classes
- Any kind of friendship

Question No: 10

The statement objA=objB; will cause a compiler error if the objects are of different classes.

- **True**
- False

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 11

Identify which of the following overloaded operator function's declaration is appropriate for the given call?

Rational_number_1 + 2.325

Where Rational_number_1 is an object of user defined class Rational_number.

- Rational_number operator+(Rational_number & obj);
- **Rational_number operator+(double& obj); (Page 145)**
- Rational_number operator+(Rational_number &obj, double& num);
- operator+(double& obj);

Question No: 12

Which operator can not be overloaded?

- The relation operator (>=)
- Assignment operator (=)
- Script operator ([])
- **Conditional operator (? :) (Page 141)**

Question No: 13

To convert from a user-defined class to a basic type, you would most likely use

- a built-in conversion operator.
- a one-argument constructor.
- an overloaded = operator.
- **a conversion operator that's a member of the class.**

0304-1659294
www.vulmshelp.com

Question No: 14

The technique in which we visualize our programming problems according to real life's problems is called

---

- ► structured programming
- ► **object oriented Programming (Page 9)**
- ► procedural programming
- ► non of the given

Question No: 15

In object orientated programming, a class of objects cans _____ properties from another class of objects

- ► Utilize
- ► Borrow
- ► **Inherit**
- ► Adopt

Question No: 16

A C++ class is similar to ----------

- ► **Structure**
- ► Header File
- ► Library File
- ► None of the given

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 17

What problem(s) may occur when we copy objects without using deep copy constructor?

- ► Dangling pointer
- ► Memory Leakage
- ► **All of the given (Page 147)**
- ► System crash

Question No: 18

Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

- ► The automatic assignment operator is used
- ► **The copy constructor is used**
- ► Compiler error
- ► Run-time error

Question No: 19

a'A static member function can be called, even when a class is not ____.

- ► Declared
- ► Define
- ► **Instantiated C++ How to Program, Fifth Edition(Page 871)**
- ► Called

Question No: 20

Identify which of the following overloaded operator function's declaration is appropriate for the given call?

Rational_number_1 + 2.325

Where Rational_number_1 is an object of user defined class Rational_number.

0304-1659294 www.vulmshelp.com

- Rational_number operator+(Rational_number & obj);
- **Rational_number operator+(double& obj); (Page 145)**
- Rational_number operator+(Rational_number &obj, double& num);
- operator+(double& obj);

Question No:21

____ provide the facility to access the data member.

- **accesser function (Page 68)**
- private function
- inline function
- None of the given

Question No: 22

Constant objects cannot change their state,

- **True (Page 105)**
- False

Question No: 23

The ____ relationship indicates that an object contains other objects.

- **None of given (Page 53)**

0304-1659294
www.vulmshelp.com

- ‘has-a’
- ‘is-a’
- ‘be-

Question No: 24

Which one of the following features of OOP is used to derive a class from another?

- Encapsulation
- Polymorphism
- Data hiding
- **Inheritance (Page 25)**

Question No: 25

___ is a relationship

- **Inheritance (Page 25)**
- Polymarphism
- abstraction
- encapsulation

Question No: 26

____ satisfy the condition of polymorphism

- Carbon
- Diamond
- Coal
- **all of the given (Page 56)**

Question No: 27

A generalization-specialization relation between classes are implemented using

0304-1659294
www.vulmshelp.com

- ► data hiding
- ► friend classes
- ► encapsulation
- ► **inheritance (Page 49)**

Question No: 28

The >= operator can be overloaded.

- ► **True (Page 140)**
- ► False

Question No: 29

In order to free the memory occupied by the object, we use ----------

- ► Constructor
- ► **Destructor (Page 92)**
- ► Shallow Copy
- ► Deep Copy

Question No: 30

Which of the following is not an example of multiple inheritances?----------

- ► Mermaid
- ► **Woman (Lecture No.5)**
- ► None of the given
- ► Amphibious Vehicle

AL JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 31

Static variable can be initialized more than once.

- ► **True**
- ► False

Question No: 32

A generic class showing all the common attributes and a behavior of other classes represents a very important feature in oop called --------

- ► **Inheritance (Page 26)**
- ► Encapsulation
- ► Polymarphism
- ► Abstraction

Question No: 33

We can get only one unique value which can be used by all the objects of that class by the use of,

- ►**static variables**
- ►dynamic variables
- ►instance variables
- ►data members

Question No: 34

A member function having the same name as that of a class and a ~ sign with it is called,

- ►Constructor

0304-1659294
www.vulmshelp.com

►Getter

►Setter

►**Destructor (Page 92)**

Question No: 35

Using encapsulation we can achieve

►**Information hiding (Page 16)**

►Least interdependencies among modules

►Implementation independence

►All of given options

Question No: 36

Inheritance is a way to

►**make general classes into more specific classes. (Page 27)**

►pass arguments to objects of classes.

►improve data hiding and encapsulation.

►providing class growth through natural selection.

Question No: 37

Static variable can be initialized more than once.

- True
- **False (Page 107)**

0304-1659294
www.vulmshelp.com

Question No: 38

For classes with common behavior, you can save effort by placing the common behavior in a ____.

►Derived Class

►**Base class (Page 29)**

►Deprived Class

►Named class

Question No: 39

Which of the following are an advantage of OOP?

►OOP makes it easy to re-use the code

►It provides an ability to create one user defined data type by extending the other

►It provides the facility of defining Abstract data types through which real world entities can be defined better.

►**All of the given options** Click here for detail

Question No: 40

The >= operator can be overloaded.

►**True (Page 140)**

►False

Question No: 41

A static member function cannot be declared.

►Static

►Implicit

►Explicit

►**Virtual**

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 42

Static variables act like a global variable in the context or scope of the class.

►**True**

►False

Question No: 43

The compiler won't object if you overload the * operator to perform division.

►**True**

►False

Question No: 44

We can use "this" pointer in the constructor in the body and even in the initialization list of any class if we are careful,

►TRUE

►**False**

Question No: 45

A C++ class is similar to ----------

►**Structure**

►Header File

►Library File

►None of the given

Question No: 46

Which operator can not be overloaded?

►The relation operator (>=)

►Assignment operator (=)

0304-1659294
www.vulmshelp.com

►Script operator ([])

►**Conditional operator (? :) (Page 141)**

Question No: 47

An overloaded operator always requires one less argument than its number of operands.

►**True**

►False

Question No: 48

A generalization-specialization relation between classes are implemented using

►data hiding

►friend classes

►encapsulation

►**inheritance (Page 49)**

Question No: 49

In OOP a class is an example of ___

►Data Type

►Abstract Type

►**User Defined Type (Page 66)**

►None of the given

Question No: 50

A class can be identified from a statement by ----------

►**Noun (Page 58)**

►Pronoun

►Verb

►Adverb

AL JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Question No: 51

The members of a class that can be accessed without creating the object of the class is called

►Private member

►Data Member

►**Public Member (How to Program page 983)**

►Static

Question No: 52

Suppose there is an object of type Person, which of the following can be considered as one of its attributes

► Name

► Age

► Work()

► **Both Name and Age (page 13)**

Question No: 53

What a derived class can add?

► New data members

► New member functions and New friend functions

► New constructors and destructor

► **All of given**

Question No: 54

____ is/are used to access information hidden within an object?

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

► Interface

► Private data members

► **Private member functions (Page 69)**

► Both public and private members

Question No: 55

this pointers are not accessible for static member functions.

► **True (Page 14)**

► False

Question No: 56

A static member function cannot be declared.

► Static

► Implicit

► Explicit

► **Virtual**

Question No: 57

C++ compiler does not allow to dynamically allocate memory for objects

► **False**

► True

Question No: 58

Given the following class

class Base{

int Age=33;

}

How you can improve above class with respect to accessing the field Age?

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

► Define the variable Age as private

► Define the variable Age as protected

► **Define the variable Age as private and create a get method that returns it and a set method that updates it**

► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 59

Friend class and friend function can be used as an alternate to each other

► **True**

► False

Question No: 60

Which of the following operators always takes no argument if overloaded?

► /

► -

► +

► **++**

Question No: 61

Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

► The automatic assignment operator is used

► **The copy constructor is used**

► Compiler error

► Run-time error

0304-1659294
www.vulmshelp.com

Question No:62

Assume a class C with objects obj1, obj2, and obj3. For the statement obj3 = obj1 - obj2 to work correctly, if the overloaded - operator must

► take two arguments.

► **return a value**

► create a named temporary object.

► take four arguments

Question No: 63

Which operator can not be overloaded?

► The relation operator (>=)

► Assignment operator (=)

► Script operator ([])

► **Conditional operator (? :) (Page 141)**

Question No: 64

We achieve independence of internal implementation from its external interface through----------

► Encapsulation

► Information Hiding

► **Abstraction**

► both encapsulation and information hiding

Question No: 65

0304-1659294
www.vulmshelp.com

Which one of the following is not an object association?

► Simple Assocation

► **Inheritance (Page 49)**

► Aggregation

► Composition

Question No: 66

Keeping in view the principle of abstraction, which of the above information the company needs to save as employee's record?

► 2, 4

► 1, 3, 5

► **1, 2, 3 (Lecture No.3)**

► 1, 2, 3, 4

Question No: 67

What a derived class can add?

► New data members

► New member functions and New friend functions

► New constructors and destructor

► **All of given**

Question No: 68

The concept of derived classes is involved in,

AL JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

► **inheritance (Page 25)**

► encapsulation

► structure

► array

Question No: 69

Your chief Software designer has shown you a sketch of the new Computer parts system she is about to create. At the top of the hierarchy is a Class called Computer and under this are two child classes. One is called LinuxPC and one is called WindowsPC. The main difference between the two is that one runs the Linux operating System and the other runs the Windows System (of course another difference is that one needs constant re-booting and the other runs reliably). Under the WindowsPC are two Sub classes one called Server and one Called Workstation. How might you appraise your designers work?

► Give the goahead for further design using the current scheme

► **Ask for a re-design of the hierarchy with changing the Operating System to a field rather than Class**

► Ask for the option of WindowsPC to be removed as it will soon be obsolete

► Change the hierarchy to remove the need for the superfluous Computer Class.

Question No: 70

Consider the code below,

```
class Fred {
public:
Fred();

...
};
```

0304-1659294
www.vulmshelp.com

```
int main()
{
Fred a[10];
Fred* p = new Fred[10];
...
}
```

Select the best option,

- ► Fred a[10]; calls the default constructor 09 times

Fred* p = new Fred[10]; calls the default constructor 10 times

- ► Produce an error
- ► **Fred a[10]; calls the default constructor 11 times**

Fred* p = new Fred[10]; calls the default constructor 11 times

- ► Fred a[10]; calls the default constructor 10 times

Fred* p = new Fred[10]; calls the default constructor 10 times

Question No:71

Which construct is the source for the creation of an object?

- ► Destructor of the class
- ► **New operator**
- ► Delete operator
- ► Constructor of the class

Question No: 72

this pointers are not accessible for static member functions.

- ► **True (Page 14)**
- ► False

0304-1659294
www.vulmshelp.com

Question No: 73

When a variable is define as static in a class then all object of this class,

- ► Have different copies of this variable
- ► **Have same copy of this variable**
- ► Can not access this variable
- ► None of given

Question No: 74

___ remain in memory even when all objects of a class have been destroyed.

- ► **Static variables (Page 111)**
- ► Instance variable
- ► Primitive variables
- ► None of given

Question No: 75

Given the following class

```
class Base{
    int Age=33;
}
```

How you can improve above class with respect to accessing the field Age?

- ► Define the variable Age as private
- ► Define the variable Age as protected
- ► **Define the variable Age as private and create a get method that returns it and a set method that updates it**
- ► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 76

The life of sub object is not dependant on the life of master class in _____.

0304-1659294 www.vulmshelp.com

- Composition
- **Aggregation (Page 134)**
- Separation
- None of the given

Question No: 77

Which one is not keyword in C++?

- operator
- **B_op**
- const
- None of given

Question No: 78

The >= operator can be overloaded.

- **True (Page 140)**
- False

0304-1659294
www.vulmshelp.com

Question No: 79

A template provides a convenient way to make a family of

- variables and data members
- **functions and classes**
- classes and exceptions
- programs and algorithms

Question No: 80

Which one of the following terms must relate to polymorphism?

- ► Static allocation
- ► Static typing
- ► **Dynamic binding (How to program , page 1103)**
- ► Dynamic allocation

Question No: 81

What is true about function templates?

- ► The compiler generates only one copy of the function template
- ► **The compiler generates a copy of function respective to each type of data (Page 257)**
- ► The compiler can only generate copy for the int type data
- ► None of the given.

0304-1659294
www.vulmshelp.com

Question No: 82

Which of the following is the best approach if it is required to have more than one functions having exactly same functionality and implemented on different data types?

- ► **Templates**
- ► Overloading
- ► Data hiding
- ► Encapsulation

Question No: 83

template <>

class Vector<char*> { }

This is an example of partial specialization.

► **True (Page 281)**

► False

Question No: 84

Classes like TwoDimensionalShape and ThreeDimensionalShape would normally be concrete, while classes like Sphere and Cube would normally be abstract.

► True

► **False Click here for Detail**

Question No: 85

A non-virtual member function is defined in a base class and overridden in a derived class; if that function is called through a base-class pointer to a derived class object, the derived-class version is used.

► True

► **False**

Question No: 86

Assume a class Derv that is privately derived from class Base. An object of class Derv located in main() can access

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

► **public members of Derv.**

► protected members of Derv.

► private members of Derv.

► protected members of Base.

Question No: 87

In order to define a class template, the first line of definition must be:

► **template <typename T> (Page 281)**

► typename <template T>

► Template Class <ClassName>

► Class <Template T>

Question No: 88

If there is a pointer p to objects of a base class, and it contains the address of an object of a derived class, and both classes contain a nonvirtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the ___ class to be executed.

► Base

► Derived

► Abstract

► **virtual (not sure)**

Question No: 89

When the base class and the derived class have a member function with the same name, you must be more specific which function you want to call (using _____).

0304-1659294
www.vulmshelp.com

- ► scope resolution operator
- ► dot operator
- ► null operator
- ► **Operator overloading (Page 204)**

Question No: 90

Non Template Friend functions of a class are friends of ____instance/s of that class.

- ► **All**
- ► One specific
- ► All instances of one date type
- ► None of the given options

Question No: 91

The find() algorithm

- ► finds matching sequences of elements in two containers.
- ► finds a container that matches a specified container.
- ► **takes iterators as its first two arguments.**
- ► takes container elements as its first two arguments.

Question No: 92

If you define a vector v with the default constructor, and define another vector w with a one-argument constructor to a size of 11, and insert 3 elements into each of these vectors with push_back(), then the size() member function will return __ for v and ___ for w.

- ► 11 for v and 3 for w.
- ► 0 for v and 0 for w.
- ► 0 for v and 3 for w.
- ► **3 for v and 11 for w.**

Question No: 93

0304-1659294 www.vulmshelp.com

Which of the following may not be an integral part of an object?

- ► State
- ► Behavior
- ► Protected data members
- ► **All of given (not sure)**

Question No: 94

Which is not the Advantage of inheritance?

- ► providing class growth through natural selection.
- ► facilitating class libraries.
- ► **avoiding the rewriting of code.**
- ► providing a useful conceptual framework

How can we differentiate between constructor and destructors?

Destructor are preceded by a tilde(~) symbol and constructor are not preceded with any symbol.

If you do not initialize static variable of int type then it is automatically initialized with _____

0

Choose the correct statement

All of the given options

___associates the objects of the exact two classes.

Binary association

Which of the following is the advantage of inheritance?

Reusability

Member function defined inside a class declaration are___by default

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Public

Class is a single class provides functionally to operate on different type of data.

Template

Which of the following may inherit from an ordinary class?

All of the given

Which of the following represents complete specialization? Template <int.char>

Are called Non-type parameters

Which of the following is correct way to define a template class X?

typename<class T > class X {};

Which of the following is a sequence Container?

Vector

In resolution order of function template compiler searches for in the end.

Generic Template

May inherit from a complete specialization.

All of the given option

Which of the following may inherit from an ordinary class?

All of the given option

Which of the following is NOT an Associate Container?

Group

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

When was specialize a function template .it is called

Function template Overriding

Which of the following types of Inheritance is used to model "Implemented in terms of" relationship?

Private

In case of protected inheritance, public members of base class will be in derived class?

Protected

In C++ by default access of classes is .

Private

If there is a pointer p to objects of a base class , and it contains the address of an object of a derived Class, and both classes contain a virtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the class to be executed.

Derived

In case of private inheritance, private members of base class will be in derived class.

Hidden

Defines the order of evaluation of an operator in an expression.

Operator Precedence

Which of the following is the correct syntax of declaring static variable "count" of type int?

Static int count;

Defining a function in class body, the compiler treats that function as:

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Inline function

What is the output of the following code?Int main()

{intconst x = 10; cout<< ++x; return 0;}

11

Which of the following statements best describes the Constructor?

Constructor is used to initialize the data members of a class.

What will the value of variables a, b, and c after instantiating an object of above class?

Junk Value, 5, Junk value

Which of the following is a strong relationship?

Inheritance binding means that target function for a call is selected at compile time. Static

In case of public inheritance, protected members of base class will be in derived class.

Protected

A template provides a convenient way to make a family of

functions and classes

0304-1659294
www.vulmshelp.com

Function overriding is done in context of,

Derived and base class

In Protected Inheritance the public members of base class become in derived class.

Protected

In resolution order compiler search firstly .

Ordinary function

AL-JUNAID TECH INSTITUTE

Compiler generated copy constructor performs .

Both Shallow aperformscopy

A pure virtual function is a virtual function that

causes its class to be abstract

Methodologies to the development of reusable software relate to .

Generic programming

In C++ generic programming is done using .

Packages

Consider the code below, class class1{

Private:

Function func1 of class1 is in class2.

Private

When derived class has user defined assignment operator.

Compiler itself calls base Class default constructor.

A Child class can call constructor of its parent class through,

Both from its constructor initialization list or body

How many objects of a given class may be constructed in an application?

As many as the Application asks for

Which of the following is True about Destructor?

All of the given

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

How many objects are involved in the N-ary association?

More than 3

Which of the following can be the behavior of an object "Usman"?

Eat

In C++, Composition is a relationship between and objects.

Parent, Child

Generalization is approach.

Top-bottom

The tells the compiler what task the function will be performing.

Function declaration

Suppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of overloaded Assignment operator for the following statement to work correctly.

String& operator =(const String &);

In operator overloading, which of the following operator takes one or no argument.

<<

Class is blueprint of .

Objects

Polymorphism makes the system:

All of the given

If class B inherits from class A then it contains all characteristics of .

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Class A

What is meant by multiple inheritance?

Deriving a derived class from more than one base class

_____Class is a single class provides functionally to operate on different type of Data.

Template

Which of the following may inherit from an ordinary class?

All of the given

Which of the following represents complete specialization?

Template <int.char>

The parameters given in template definition other than those used for mentoring templates types are called

Non-type parameters

Which of the following is correct way to define a template class X?

Typename<class T > class X {};

Which of the following is a sequence Container?

Vector

In resolution order of function template compiler searches for_____in the end.

Generic template

_____may inherit from a complete specialization.

All of the given option

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following may inherit from an ordinary class?

All of the given option

Which of the following is NOT an Associate Container?

Group

Which of the following is NOT casting operator in C++ standard?

Var_cast

Which of the following types of Inheritance is used to model "Implemented in terms of" relationship?

Private

In case of protected inheritance, public members of base class will be_____in derived Class?

Protected

In C++ by default access of classes is_____.

Private

If there is a pointer p to objects of a base class , and it contains the address of an object of a Derived class, and both classes contain a virtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the_____class to be executed.

Derived

Suppose we have defined derived class copy constructor but have not defined base class Copy constructor then compiler will,

Use base class default constructor

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In case of private inheritance, private members of base class will be_____in derived Class.

Hidden

_____defines the order of evaluation of an operator in an expression.

Operator Precedence

Which of the following is the correct syntax of declaring static variable "count" of type int?

Static int count;

When was specialize a function template .it is called_____

Function template overriding

Which of the following is/are key component(s) of standard Template Library?

All of the given option

Each_____of a template class by default becomes function template.

Member function

In statement "template<class U,int |=5>"the non type parameter is

Int |

Solution order highest priority is given to_____in template

Complete template

In case of template specialization ,if compiler cannot find required complete

Partial specialization

AL-JUNAID TECH INSTITUTE
0304 1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

_____is a pointer which is declared outside the _____

Cursor, Container

Suppose a template class "Test" has a static data member. How many copies of this static daya member following line of C++ code?

Three

In statement template <class T,class U ,int |=5> the non-parameter is_____.

Class U

A template argument is preceded by the keyword_____.

Class.

We can have_____type of member functions in a class.

All of the given

Which of the following is a strong relationship?

Inheritance

_____Binding means that target function for a call is selected at compile time.

Static

In case of public inheritance, protected members of base class will be_____in derived Class.

Protected

A template provides a convenient way to make a family of

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Functions and classes

Function overriding is done in context of,

Derived and base class

Which of the following is TRUE.

Derived class pointer can be used as Base class pointer

The default inheritance mode is,

Private inheritance

In Protected Inheritance the public members of base class become_____in derived class.

Protected

In resolution order compiler search firstly_____.

Ordinary function

Compiler generated copy constructor performs_____.

Both Shallow and Deep copy

A pure virtual function is a virtual function that

Causes its class to be abstract

Methodologies to the development of reusable software relate to_____.

Generic programming

In C++ generic programming is done using_____.

Packages

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Consider the code below,

Class class1{

Private:

Void func1();

};

Class class2 : private class1 {

};

Function func1 of class1 is_____in class2.

Private

When derived class has user defined assignment operator.

Compiler itself calls base class default constructor.

A Child class can call constructor of its parent class through,

Both from its constructor initialization list or body

How many objects of a given class may be constructed in an application?

As many as the application asks for

The other name of specialization is_____.

Restriction

How can we differentiate between constructors and destructors?

Destructors are preceded with a tilde (~) symbol, and constructors are not preceded with any symbol.

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

How many entities are there in the following scenario?Ali lives in the house that is near the Tree and he also drives his car.

2

Which of the following is True about Destructor?

All of the given

How many objects are involved in the N-ary association?

More than 3

Which of the following can be the behavior of an object "Usman"?

Eat

In C++, Composition is a relationship between_____and_____objects.

Parent, child

If you have three classes in a C++ program A, B, and C where class A inherits from class B, then class_____contains all the characteristics of class_____.

A, B

In case when we define the function outside the class then we must use the keyword _____to make the function inline.

Inline

Mermaid is an example of:

Multiple Inheritance

AL JUNAID TECH INSTITUTE
0304 1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Assignment operators are:

Right associative

Generalization is_____approach.

Top-bottom

The_____tells the compiler what task the function will be performing.

Function declaration

Uppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of Overloaded assignment operator for the following statement to work correctly.

String& operator =(const String &);

In operator overloading, which of the following operator takes one or no argument.

<<

Class is blueprint of_____.

Objects

Polymorphism makes the system:

All of the given

If class B inherits from class A then it contains all characteristics of_____.

Class A

What is meant by multiple inheritance?

Deriving a derived class from more than one base class

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In C++, which of the following operator can only be overloaded as a member function of the class?

Stream Extraction Operator: >>

Insertion operator is_____associative.

Left to Right

In expression c1*c2+c3-c4 which of the following will be executed in first order?

C1*c2

Suppose for a class String, assignment operator is overloaded with following declaration.

Void operator =(const String &);

What will happen when we will write following statement in main()?

Str1 = str2 = str3

Where, str1, str2 and str3 are objects of class String.

Compiler will generate compile time error.

Identify the abstract class from the given statement:

"Vehicle class is base class. Bus, Car, and Truck are derived classes"

Vehicle

Which of the following operators operate on one operand?

Unary Operators

Subtyping means that derived class is behaviorally_____with the base class.

Compatible

AL JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In_____, Base class can't always be replaced by the derived class.

Specialization

The overloaded '-' operator for Complex class will be called with reference to_____in the following statement

Complex C3 = C1 – C2;

Complex

What is a class?

A class is a description of a kind of object

In polymorphism, messages can be interpreted in different ways depending upon the _____class.

Receiver

Suppose there is an object of type Person, which of the following can be considered as one of its attributes?

Name

Which of the following is True about class?

All of the given

In C++, which of the following keywords works only with constructors?

Explicit

Choose correct declaration of overloaded stream insertion operator for class String as non-member friend function.

Friend ostream& operator << ();

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following classes are used by Amphibious vehicle to inherit characteristics?

Both Land & Water Vehicle

Suppose derived class is inherited from base class. What happens when a derived class object is created?

The constructor of base class is executed before the constructor of derived class.

Suppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of Overloaded assignment operator for the following statement to work correctly.

Str1 = str2 = str3;

String& operator =(const String &);

In C++, which of the following is defined as stream insertion operator?

<<

Choose correct declaration of overloaded inequality (!=) operator for class String as non-Member friend function.

None of the given options

Relationship in which child object gets destroyed if parent object is destroyed.

Composition

How the information hidden within an object can be accessed?

Through its interface

Which of the following defines default constructor for class "Student"?

Student() {//…

}

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

The sub-object's life is not dependent on the life of master class in_____.

Aggregation

Hiding the implementation details makes program:

Easy to understand

The functions defined inside the class are by default_____.

Inline

Which of the following statement is TRUE about static function of a class?

It is used to access static data members.

Deconstruct is a function which has the same name as that of class but starts with a—– sign

~

'this' pointer cannot be passed implicitly to_____functions.

Static Member

Which of the following can be the behavior of an object "Usman"?

Eat

In Object Oriented programming, objects communicate with each other through _____

Messages

Constructor is used to_____the objects of a class.

Initialize

AL JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following is NOT an access specifier in C++?

Hidden

The other name of subtyping is_____.

Extension

Suppose a class does not have any constructor. What will happen when an object of this class is created?

Compiler will call implicit default constructor.

In case when we define the function outside the class then we must use the keyword _____To make the function inline.

Inline

In constant member function the type of this pointer is:

Constant pointer to constant data

In_____, base class can be replaced by the derived class.

Extension

We can get the address of a variable stored in a pointer using_____:

& symbol

A child inherits characteristics from its_____.

Parent

AL JUNAID TECH INSTITUTE
0304 1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following function declaration is correct to overload the + operator as member function in complex class?

Complex operator +(const Complex &rhs);

Which of the following is not true about constant member function?

All of the given options

What is the general syntax of overloading Unary Operator as member function of the class?

TYPE & operator OP ();

_____is creating objects of one class inside another class.

Composition

Which of the following is true about constructor?

All of the given option

Memory is allocated to non static members only, when

Object is created

A child inherit characteristics from its ?

Parents

A post-fix unary operator is implemented in C++ member function with

1 dummy int arguments

Which of the following is the example of objects in school?

Books , pages

AL JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Suppose student is a class, which of the following constructor with one parameter for class student?

Student()

If you have three classes in a C++ program A, B, and C where class A inherits from class B, then class_____contains all the characteristics of class_____.

A,C

In_____base class can be replaced by the derived class.

Extension

The concept of derived class is involved in _____

Inheritance

Suppose person is a class which of the following statement defines an object of class person?

Create person object

Which of the following is true about Accessor Functions?

All of the given options.

The other name of subtyping is

Extension

Which of the following statement best describes the constructor

Constructor is used to initialize the data member of a class.

Which of the following is true about inline function?

It is used by compilers to improve efficiency of the program

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

Which of the following is defined as stream extraction operator in c++?

>>

An object has

_____interfac

e(s)One or more than one

Which of the following is an advantage of

encapsulation?Better understanding

Which of the following represents the two-way

association?Employee works for the company

___is the weakest link between

object.Simple association

In composition_____are called from composing objects to compound

objects.Constructor

In OOP we can achieve reusability though _____ Inheritance

Generalization is implemented through

Inheritance

Suppose str1, str2, and str3 are objects of class string. Choose appropriate declaration of overloaded

Assignment operator for the following statement to work

correctly.String& operator=(const String &)

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In C++ which of the following is defined as stream insertion operator.

<<

Abstraction hides the

_____deta

ils.Relevant

Minimum classes required in a program for implementing multiple

inheritance.3

Composition is_____and aggregation is_____relationship.

Strong, weak

Minimum classes required for implementing multiple

inheritance.At least 3

We can access a private static variable through.

Static data member

The process of extracting common characteristics from two or more classes and combiningthem into generalized superclass is called ___

Generalization

Which of the following is true about

polymorphism?All of the given

Which of the following can be the behavior of an object "Usman"?

0304-1659294
www.vulmshelp.com

Eat

Composition is

_____relations

hipPart whole

Which of the following is a strong

relationship?Composition

Which of the following defines default constructor for class student?

Student(){/…

}

This car is composed of wheels. What is the relationship between car and wheels?

Composition

In_____base class can be replaced by the derived

class.Extension

___defines the order of evaluation of an operator is an

expression.Operator precedence

In composition___are called from composing objects to composed

objects.Destructor

0304-1659294
www.vulmshelp.com

How many objects of a given class may be constructed in an application?As many as the application asks for

Relationship in which child object gets destroyed, If parent object is destroyed.Composition

Which of the following is not true about constructor?

Constructor is a special function which can have different name than class name.

Suppose test is a class, void show() is its public member function. Which of the following is correct call of Show() function using Test pointer ptr?Ptr->show()

Which of the following features of OOP is used to deal with only relevant details?Abstraction

In C++ the subscript operator must be overloaded as a____of the class, with one parameter of ___type.

Member function, int

Identify the abstract class from the given statement

"vehicle class is base class. Bus, Car and truck are derived classes"

Vehicle

In Object Oriented programming objects communicate with each other through _____

Messages

 0304-1659294

_____associates the objects of the exact three classes.Ternary Association

An objects has attributes, operations and _____

Unique identity

In C++ which of the following keywords works only with constructors?

Explicit

Which of the following allows to reuse characteristics of more than one parent class?

Multiple Inheritance

How many objects are involved in the Binary association?2

Which of the following features of OOP is used to deal with only relevant details?

Abstraction

The overloaded '-' operator for complex class will be called with reference to____in the following

Statement

Complex C3 = C1 –

C2

Complex

Static data member is declared

Inside the class

Class is blue print of ___ Object

Which of the following is true about inline functions?All of the given options

How many objects can be created once a class is declared?As many as needed

Suppose you have following C++ statementsIntoldValue=10

intviewValue * ++ old/Value

What will be the value of old/Value and new/Value after executing above statement?

Old/Value=11, new/Value=11

In____base class can't always be replaced by the derived class.

Specialization

Which of the following will happen when a constant function tries to change the value of datamembers of the class?

Complex time error will occur

_____is set of functions of an objects exposed to other objects.Interface

How many objects are involved in the Ternary association?3

0304-1659294
www.vulmshelp.com

Mermaid is an example of Multiple inheritance

Which of the following is the correct syntax of declaring static variable 'count' of type int?

Static int count

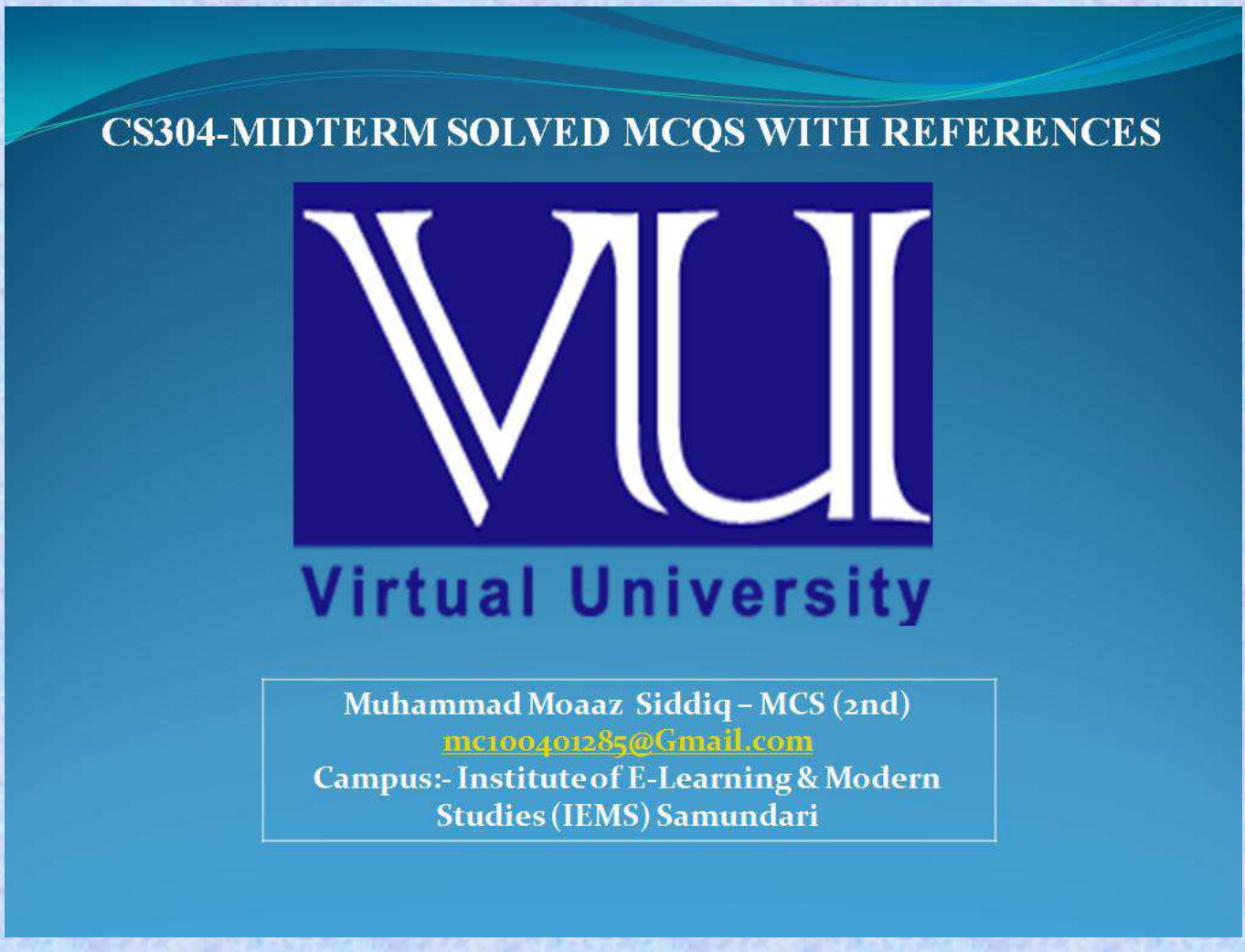
Which of the following operators operate on one operand? Unary Operators

Object Data includes ____

Both static and attributes of an objects

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

Proude To Be Virtulian MOAAZ SIDDIQ	**CS304- Object Oriented Programming** LATEST SOLVED MCQS FROM MIDTERM PAPERS	**Nov 07,2011**
MC100401285	Moaaz.pk@gmail.com MC100401285@gmail.com	**Latest Mcqs**



Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
Which part of an object exhibits its state?

- **Data (Page 23)**
- Operations
- Any public part
- Any private part

Question No: 2 (Marks: 1) - Please choose one
Inheritance is a way to

- organize data.
- pass arguments to objects of classes.
- **add features to existing classes without rewriting them. (Page 27)**
- improve data-hiding and encapsulation.

Question No: 3 (Marks: 1) - Please choose one
Suppose you have been given the following design,
"A person has a name, age, address and sex. You are designing a class to represent a type of person called a patient. This kind of person may be given a diagnosis, have a spouse and may be alive". Given that the person class has already been created, what of the following would be appropriate to include when you design the patient class?

- **registration date and diagnosis** Click here for Detail
- age and sex
- sex and diagnosis
- diagnosis and age

Question No: 4 (Marks: 1) - Please choose one
What problem(s) may occur when we copy objects without using deep copy constructor?

- Dangling pointer
- Memory Leakage
- **All of the given (Page 147)**
- System crash

Question No: 5 (Marks: 1) - Please choose one
this pointers are not accessible for static member functions.

- **True (Page 114)**
- False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 6 (Marks: 1) - Please choose one
A static member function cannot be declared.

- ► Static
- ► Implicit
- ► Explicit
- ► **Virtual** **Click here for detail**

Question No: 7 (Marks: 1) - Please choose one
_____ remain in memory even when all objects of a class have been destroyed.

- ► **Static variables (Page 111)**
- ► Instance variable
- ► Primitive variables
- ► None of given

Question No: 8 (Marks: 1) - Please choose one
Friend functions are _____ functions of a class.

- ► None of given
- ► object member
- ► **non-member (Page 136)**
- ► data member

Question No: 9 (Marks: 1) - Please choose one
_____, which means if A declares B as its friend it does NOT mean that A can access private data of B. It only means that B can access all data of A.

- ► **Friendship is one way only**
- ► Friendship is two way only
- ► NO Friendship between classes
- ► Any kind of friendship

Question No: 10 (Marks: 1) - Please choose one
The statement objA=objB; will cause a compiler error if the objects are of different classes.

- ► **True**
- ► False

Question No: 11 (Marks: 1) - Please choose one
Identify which of the following overloaded operator function's declaration is appropriate for the given call?

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Rational_number_1 + 2.325
Where Rational_number_1 is an object of user defined class Rational_number.

- Rational_number operator+(Rational_number & obj);
- **Rational_number operator+(double& obj);** **(Page 145)**
- Rational_number operator+(Rational_number &obj, double& num);
- operator+(double& obj);

Question No: 12 (Marks: 1) - Please choose one
Which operator can not be overloaded?

- The relation operator (>=)
- Assignment operator (=)
- Script operator ([])
- **Conditional operator (? :) (Page 141)**

Question No: 13 (Marks: 1) - Please choose one
To convert from a user-defined class to a basic type, you would most likely use

- a built-in conversion operator.
- a one-argument constructor.
- an overloaded = operator.
- **a conversion operator that's a member of the class.**

Question No: 14 (Marks: 1) - Please choose one
The technique in which we visualize our programming problems according to real life's problems is called
---

- structured programming
- **object oriented Programming** **(Page 9)**
- procedural programming
- non of the given

Question No: 15 (Marks: 1) - Please choose one
In object orientated programming, a class of objects cans _____ properties from another class of objects

- Utilize
- Borrow
- **Inherit** **Click here for detail**
- Adopt

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 16 (Marks: 1) - Please choose one
A C++ class is similar to ----------

- ► **Structure** **Click here for detial**
- ► Header File
- ► Library File
- ► None of the given

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
What problem(s) may occur when we copy objects without using deep copy constructor?

- ► Dangling pointer
- ► Memory Leakage
- ► **All of the given** **(Page 147)**
- ► System crash

Question No: 2 (Marks: 1) - Please choose one
Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

- ► The automatic assignment operator is used
- ► **The copy constructor is used** **Click here for detial**
- ► Compiler error
- ► Run-time error

Question No: 3 (Marks: 1) - Please choose one
a'A static member function can be called, even when a class is not _____.

- ► Declared
- ► Define
- ► **Instantiated** **C++ How to Program, Fifth Edition(Page 871)**
- ► Called

Question No: 4 (Marks: 1) - Please choose one
Identify which of the following overloaded operator function's declaration is appropriate for the given call?
Rational_number_1 + 2.325
Where Rational_number_1 is an object of user defined class Rational_number.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► Rational_number operator+(Rational_number & obj);
► **Rational_number operator+(double& obj); (Page 145)**
► Rational_number operator+(Rational_number &obj, double& num);
► operator+(double& obj);

Question No: 5 (Marks: 1) - Please choose one
_____ provide the facility to access the data member.

► **accesser function (Page 68)**
► private function
► inline function
► None of the given

Question No: 6 (Marks: 1) - Please choose one
Constant objects cannot change their state,

► **True (Page 105)**
► False

Question No: 7 (Marks: 1) - Please choose one
The _____ relationship indicates that an object contains other objects.

► **None of given (Page 53)**
► 'has-a'
► 'is-a'
► 'be-

Question No: 8 (Marks: 1) - Please choose one
Which one of the following features of OOP is used to derive a class from another?

► Encapsulation
► Polymorphism
► Data hiding
► **Inheritance (Page 25)**

Question No: 9 (Marks: 1) - Please choose one
_____ is a relationship

► **Inheritance (Page 25)**
► Polymarphism
► abstraction
► encapsulation

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 10 (Marks: 1) - Please choose one
_____satisfy the condition of polymorphism

- ► Carbon
- ► Diamond
- ► Coal
- ► **all of the given (Page 56)**

Question No: 11 (Marks: 1) - Please choose one
A generalization-specialization relation between classes are implemented using

- ► data hiding
- ► friend classes
- ► encapsulation
- ► **inheritance (Page 49)**

Question No: 12 (Marks: 1) - Please choose one
The >= operator can be overloaded.

- ► **True (Page 140)**
- ► False

Question No: 13 (Marks: 1) - Please choose one
In order to free the memory occupied by the object, we use ----------

- ► Constructor
- ► **Destructor (Page 92)**
- ► Shallow Copy
- ► Deep Copy

Question No: 14 (Marks: 1) - Please choose one
Which of the following is not an example of multiple inheritances?----------

- ► Mermaid
- ► **Woman (Lecture No.5)**
- ► None of the given
- ► Amphibious Vehicle

Question No: 15 (Marks: 1) - Please choose one
Static variable can be initialized more than once.

- ► **True** Click here for detail
- ► False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 16 (Marks: 1) - Please choose one
A generic class showing all the common attributes and a behavior of other classes represents a very important feature in oop called --------

- **Inheritance (Page 26)**
- Encapsulation
- Polymarphism
- Abstraction

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
We can get only one unique value which can be used by all the objects of that class by the use of,

- **static variables** Click here for detail
- dynamic variables
- instance variables
- data members

Question No: 2 (Marks: 1) - Please choose one
A member function having the same name as that of a class and a ~ sign with it is called,

- Constructor
- Getter
- Setter
- **Destructor (Page 92)**

Question No: 3 (Marks: 1) - Please choose one
Using encapsulation we can achieve

- **Information hiding (Page 16)**
- Least interdependencies among modules
- Implementation independence
- All of given options

Question No: 4 (Marks: 1) - Please choose one
Inheritance is a way to

- **make general classes into more specific classes. (Page 27)**
- pass arguments to objects of classes.
- improve data hiding and encapsulation.
- providing class growth through natural selection.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 5 (Marks: 1) - Please choose one
Static variable can be initialized more than once.
True
False (Page 107)

Question No: 6 (Marks: 1) - Please choose one
For classes with common behavior, you can save effort by placing the common behavior in a _____.

►Derived Class
►**Base class (Page 29)**
►Deprived Class
►Named class

Question No: 7 (Marks: 1) - Please choose one
Which of the following are an advantage of OOP?

►OOP makes it easy to re-use the code
►It provides an ability to create one user defined data type by extending the other
►It provides the facility of defining Abstract data types through which real world entities can be defined better.
►**All of the given options** **Click here for detail**

Question No: 8 (Marks: 1) - Please choose one
The >= operator can be overloaded.

►**True (Page 140)**
►False

Question No: 9 (Marks: 1) - Please choose one
A static member function cannot be declared.

►Static
►Implicit
►Explicit
►**Virtual** **Click here for detail**

Question No: 10 (Marks: 1) - Please choose one
Static variables act like a global variable in the context or scope of the class.

►**True** **click here for detail**
►False

Question No: 11 (Marks: 1) - Please choose one
The compiler won't object if you overload the * operator to perform division.

►**True** **Click here for detail**
►False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 12 (Marks: 1) - Please choose one
We can use "this" pointer in the constructor in the body and even in the initialization list of any class if we are careful,

- ►TRUE
- ►**False**

Question No: 13 (Marks: 1) - Please choose one
A C++ class is similar to ----------

- ►**Structure** **Click here for detail**
- ►Header File
- ►Library File
- ►None of the given

Question No: 14 (Marks: 1) - Please choose one
Which operator can not be overloaded?

- ►The relation operator (>=)
- ►Assignment operator (=)
- ►Script operator ([])
- ►**Conditional operator (? :) (Page 141)**

Question No: 15 (Marks: 1) - Please choose one
An overloaded operator always requires one less argument than its number of operands.

- ►**True (Page 896)**
- ►False

Question No: 16 (Marks: 1) - Please choose one
A generalization-specialization relation between classes are implemented using

- ►data hiding
- ►friend classes
- ►encapsulation
- ►**inheritance (Page 49)**

Question No: 17 (Marks: 1) - Please choose one
In OOP a class is an example of _____

- ►Data Type
- ►Abstract Type
- ►**User Defined Type (Page 66)**
- ►None of the given

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 18 (Marks: 1) - Please choose one
A class can be identified from a statement by ----------

►**Noun (Page 58)**
►Pronoun
►Verb
►Adverb

Question No: 19 (Marks: 1) - Please choose one
The members of a class that can be accessed without creating the object of the class is called

►Private member
►Data Member
►**Public Member (How to Program page 983)**
►Static

MIDTERM EXAMINATION
Spring 2010

Question No: 1 (Marks: 1) - Please choose one
Suppose there is an object of type Person, which of the following can be considered as one of its attributes

► Name
► Age
► Work()
► **Both Name and Age (page 13)**

Question No: 2 (Marks: 1) - Please choose one
What a derived class can add?

► New data members
► New member functions and New friend functions
► New constructors and destructor
► **All of given** **Click here for Detail**

Question No: 3 (Marks: 1) - Please choose one
_____ is/are used to access information hidden within an object?

► Interface
► Private data members
► **Private member functions (Page 69)**
► Both public and private members

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 4 (Marks: 1) - Please choose one
this pointers are not accessible for static member functions.

► **True (Page 14)**
► False

Question No: 5 (Marks: 1) - Please choose one
A static member function cannot be declared.

► Static
► Implicit
► Explicit
► **Virtual** **Click here for detail**

Question No: 6 (Marks: 1) - Please choose one
C++ compiler does not allow to dynamically allocate memory for objects

► **False** **Click here for detail**
► True

Question No: 7 (Marks: 1) - Please choose one
Given the following class
class Base{
int Age=33;
}
How you can improve above class with respect to accessing the field Age?

► Define the variable Age as private
► Define the variable Age as protected
► **Define the variable Age as private and create a get method that returns it and a set method that updates it**
Click here for detail
► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 8 (Marks: 1) - Please choose one
Friend class and friend function can be used as an alternate to each other

► **True** **Click here for detail**
► False

Question No: 9 (Marks: 1) - Please choose one
Which of the following operators always takes no argument if overloaded?

► /
► -

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

► +
► ++

Question No: 10 (Marks: 1) - Please choose one
Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

► The automatic assignment operator is used
► **The copy constructor is used** **Click here for detail**
► Compiler error
► Run-time error

Question No: 11 (Marks: 1) - Please choose one
Assume a class C with objects obj1, obj2, and obj3. For the statement obj3 = obj1 - obj2 to work correctly, if the overloaded - operator must

► take two arguments.
► **return a value** **Click here for detail**
► create a named temporary object.
► take four arguments

Question No: 12 (Marks: 1) - Please choose one
Which operator can not be overloaded?

► The relation operator (>=)
► Assignment operator (=)
► Script operator ([])
► **Conditional operator (? :)** **(Page 141)**

Question No: 13 (Marks: 1) - Please choose one
We achieve independence of internal implementation from its external interface through----------

► Encapsulation
► Information Hiding
► **Abstraction** **Click here for detial**
► both encapsulation and information hiding

Question No: 14 (Marks: 1) - Please choose one
Which one of the following is not an object association?

► Simple Assocation
► **Inheritance** **(Page 49)**
► Aggregation
► Composition

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Keeping in view the principle of abstraction, which of the above information the company needs to save as employee's record?

- ► 2, 4
- ► 1, 3, 5
- ► **1, 2, 3 (Lecture No.3)**
- ► 1, 2, 3, 4

Question No: 4 (Marks: 1) - Please choose one
What a derived class can add?

- ► New data members
- ► New member functions and New friend functions
- ► New constructors and destructor
- ► **All of given** **Click here for Detail**

Question No: 5 (Marks: 1) - Please choose one
The concept of derived classes is involved in,

- ► **inheritance (Page 25)**
- ► encapsulation
- ► structure
- ► array

Question No: 6 (Marks: 1) - Please choose one
Your chief Software designer has shown you a sketch of the new Computer parts system she is about to create. At the top of the hierarchy is a Class called Computer and under this are two child classes. One is called LinuxPC and one is called WindowsPC. The main difference between the two is that one runs the Linux operating System and the other runs the Windows System (of course another difference is that one needs constant re-booting and the other runs reliably). Under the WindowsPC are two Sub classes one called Server and one Called Workstation. How might you appraise your designers work?

- ► Give the goahead for further design using the current scheme
- ► **Ask for a re-design of the hierarchy with changing the Operating System to a field rather than Class type** **Click here for detail**
- ► Ask for the option of WindowsPC to be removed as it will soon be obsolete
- ► Change the hierarchy to remove the need for the superfluous Computer Class.

Question No: 7 (Marks: 1) - Please choose one
Consider the code below,
class Fred {
public:
Fred();

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

```
...
};
int main()
{
Fred a[10];
Fred* p = new Fred[10];
...
}
```

Select the best option,

► Fred a[10]; calls the default constructor 09 times
Fred* p = new Fred[10]; calls the default constructor 10 times
► Produce an error
► **Fred a[10]; calls the default constructor 11 times**
Fred* p = new Fred[10]; calls the default constructor 11 times
► Fred a[10]; calls the default constructor 10 times
Fred* p = new Fred[10]; calls the default constructor 10 times

Question No: 8 (Marks: 1) - Please choose one
Which construct is the source for the creation of an object?

► Destructor of the class
► **New operator** **Click here for detial**
► Delete operator
► Constructor of the class

Question No: 9 (Marks: 1) - Please choose one
this pointers are not accessible for static member functions.
► **True (Page 14)**
► False

Question No: 10 (Marks: 1) - Please choose one
When a variable is define as static in a class then all object of this class,

► Have different copies of this variable
► **Have same copy of this variable** **Click here for detail**
► Can not access this variable
► None of given

Question No: 11 (Marks: 1) - Please choose one
_____ remain in memory even when all objects of a class have been destroyed.
► **Static variables (Page 111)**
► Instance variable
► Primitive variables
► None of given

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 12 (Marks: 1) - Please choose one
Given the following class
class Base{
int Age=33;
}
How you can improve above class with respect to accessing the field Age?

- ► Define the variable Age as private
- ► Define the variable Age as protected
- ► **Define the variable Age as private and create a get method that returns it and a set method that updates it** **Click here for detail**
- ► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 13 (Marks: 1) - Please choose one
The life of sub object is not dependant on the life of master class in _____.

- ► Composition
- ► **Aggregation (Page 134)**
- ► Separation
- ► None of the given

Question No: 14 (Marks: 1) - Please choose one
Which one is not keyword in C++?

- ► operator
- ► **B_op** **Click here for detail**
- ► const
- ► None of given

Question No: 15 (Marks: 1) - Please choose one
The >= operator can be overloaded.

- ► **True (Page 140)**
- ► False

Question No: 16 (Marks: 1) - Please choose one
Identify which of the following overloaded operator function's declaration is appropriate for the given call?
Rational_number_1 + 2.325
Where Rational_number_1 is an object of user defined class Rational_number.

- ► Rational_number operator+(Rational_number & obj);
- ► **Rational_number operator+(double& obj);**
- ► Rational_number operator+(Rational_number &obj, double& num);
- ► operator+(double& obj);

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

MIDTERM EXAMINATION
Spring 2009

Question No: 1 (Marks: 1) - Please choose one
A template provides a convenient way to make a family of

- ► variables and data members
- ► **functions and classes** **Click here for detail**
- ► classes and exceptions
- ► programs and algorithms

Question No: 2 (Marks: 1) - Please choose one
Which one of the following terms must relate to polymorphism?

- ► Static allocation
- ► Static typing
- ► **Dynamic binding (How to program , page 1103)** **Click here for more detail**
- ► Dynamic allocation

Question No: 3 (Marks: 1) - Please choose one
What is true about function templates?

- ► The compiler generates only one copy of the function template
- ► **The compiler generates a copy of function respective to each type of data (Page 257)**
- ► The compiler can only generate copy for the int type data
- ► None of the given.

Question No: 4 (Marks: 1) - Please choose one
Which of the following is the best approach if it is required to have more than one functions having exactly same functionality and implemented on different data types?

- ► **Templates** **Click here for detial**
- ► Overloading
- ► Data hiding
- ► Encapsulation

Question No: 5 (Marks: 1) - Please choose one
template <>
class Vector<char*> { }

This is an example of partial specialization.

- ► **True (Page 281)**
- ► False

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 6 (Marks: 1) - Please choose one
Classes like TwoDimensionalShape and ThreeDimensionalShape would normally be concrete, while classes like Sphere and Cube would normally be abstract.

- ► True
- ► **False** **Click here for Detail**

Question No: 7 (Marks: 1) - Please choose one
A non-virtual member function is defined in a base class and overridden in a derived class; if that function is called through a base-class pointer to a derived class object, the derived-class version is used.

- ► True
- ► **False** **Click here for Detial**

Question No: 8 (Marks: 1) - Please choose one
Assume a class Derv that is privately derived from class Base. An object of class Derv located in main() can access

- ► **public members of Derv.** **Click here for Detail**
- ► **protected members of Derv.**
- ► private members of Derv.
- ► protected members of Base.

Question No: 9 (Marks: 1) - Please choose one
In order to define a class template, the first line of definition must be:

- ► **template <typename T> (Page 281)**
- ► typename <template T>
- ► Template Class <ClassName>
- ► Class <Template T>

Question No: 10 (Marks: 1) - Please choose one
If there is a pointer p to objects of a base class, and it contains the address of an object of a derived class, and both classes contain a nonvirtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the _____ class to be executed.

- ► Base
- ► Derived
- ► Abstract
- ► **virtual** **(not sure)**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 11 (Marks: 1) - Please choose one
When the base class and the derived class have a member function with the same name, you must be more specific which function you want to call (using _____).

- ► scope resolution operator
- ► dot operator
- ► null operator
- ► **Operator overloading (Page 204) Click here for Detail**

Question No: 12 (Marks: 1) - Please choose one
Non Template Friend functions of a class are friends of _____instance/s of that class.

- ► **All Click here for detail**
- ► One specific
- ► All instances of one date type
- ► None of the given options

Question No: 13 (Marks: 1) - Please choose one
The find() algorithm

- ► finds matching sequences of elements in two containers.
- ► finds a container that matches a specified container.
- ► **takes iterators as its first two arguments. Click here for Detail**
- ► takes container elements as its first two arguments.

Question No: 14 (Marks: 1) - Please choose one
If you define a vector v with the default constructor, and define another vector w with a one-argument constructor to a size of 11, and insert 3 elements into each of these vectors with push_back(), then the size() member function will return _____ for v and _____ for w.

- ► 11 for v and 3 for w.
- ► 0 for v and 0 for w.
- ► 0 for v and 3 for w.
- ► **3 for v and 11 for w.**

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Question No: 15 (Marks: 1) - Please choose one
Which of the following may not be an integral part of an object?

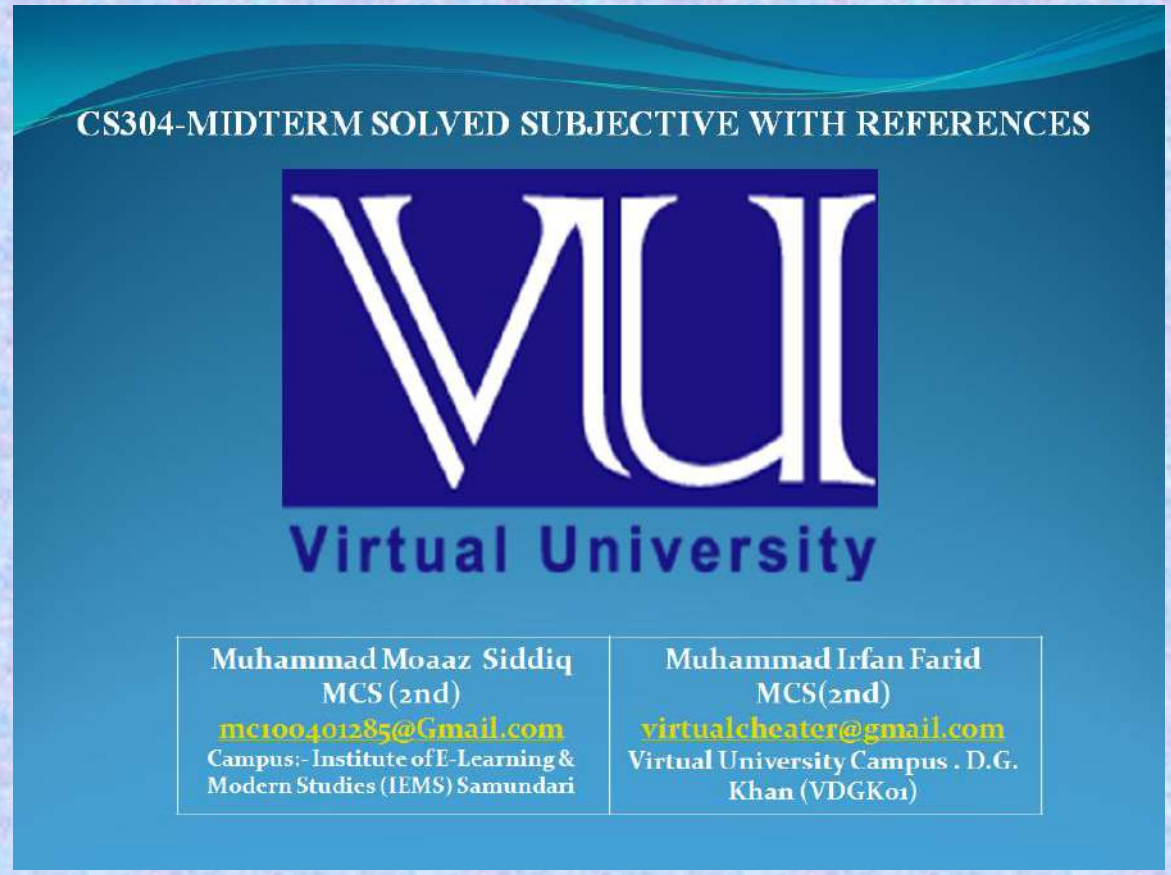
- ► State
- ► Behavior
- ► Protected data members
- ► **All of given (not sure)**

Question No: 16 (Marks: 1) - Please choose one
Which is not the Advantage of inheritance?

- ► providing class growth through natural selection.
- ► facilitating class libraries.
- ► **avoiding the rewriting of code.** **Click here for Detail**
- ► providing a useful conceptual framework.

Muhammad Moaaz Siddiq – MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Proude To Be Virtulian MOAAZ SIDDIQ	**CS304- Object Oriented Programming** LATEST SOLVED SUBJECTIVES FROM MIDTERM PAPERS	**NOV 29,2011**
MC100401285	**Moaaz.pk@gmail.com** **Mc100401285@gmail.com**	**PSMD01**
MC110200885	**Virtualcheater@gmail.com**	**VDGK01**



MIDTERM EXAMINATION 2011

Q.1 can constant object access the none constant member function of each class.

Answer:- (Page 106)

"Const objects can access only const member functions so chances of change of state of const objects once they are created are eliminated."

Q.2. Give at least two problems that we should check when we overloading assignments operator ("=") in string class (unsolved)

Q3. Give c++ code to overloaded unary "--" oprators to comples member class.

Answer:- (Page 165)

```
class Complex{
...
Complex operator -- (int);
// friend Complex operator --(const Complex &, int);
}
Complex Complex::operator -- (int){
complex t = *this;
real - = 1;
return t;
}
Complex operator -- (const
Complex & h, int){
complex t = h;
h.real - = 1;
return t;
}
```

Q4.What is simple association? Explain it with the help of example.

Answer:- (Page 49)

The two interacting objects have no intrinsic relationship with other object. It is the weakest link between objects. It is a reference by which one object can interact with some other object.

- Customer gets cash from cashier
- Employee works for a company

Q5.explain the deference between the static variable of a class with none static variable with the help of example

Answer:- click here for detail

The difference between static and non static members is only that a non static member is tied to an instance of a class although a static member is tied to the class, and not to a particular instance. That is, a static member is shared by all instances of a class although a non static member exists for each instance of class.

MIDTERM EXAMINATION 2011

write c++ code for operator() .....5 marks

Answer:- (Page 161)

Function Operator

```
class String{
...
public:
char & operator()(int);
...
};
char & String::operator()
(int pos){
assert(pos>0 && pos<=size);
return bufferPtr[pos-1];
}
int main(){
String s1("Ping");
char g = s1(2); // g = 'i'
s1(2) = 'o';
cout << g << "\n";
cout << str.GetString();
return 0;
}
```

Write c++ code for operator [] using string class.....5 marks.....

Answer:- (Page 160)

Subscript operator must be overloaded as member function of the class with one parameter of integer type,

```
class String{
...
public:
char & operator[](int);
...
};
char & String::operator[]( int pos){
assert(pos>0 && pos<=size);
return stringPtr[pos-1];
}
int main() {
String s1("Ping");
cout <<str.GetString()<< endl;
s1[2] = 'o';
cout << str.GetString();
return 0;
}
```

static data members.....2.mark

Answer:- (Page 108)

"A variable that is part of a class, yet is not part of any object of that class, is called static data member"

MIDTERM EXAMINATION 2011

What is interface give two real life examples?

Answer:- (Page 18)

Interface is a set of functions of an object that he wants to expose to other objects.

Interface of a Car is set of these functions

• Accelerate • Change Gear • Apply Brakes• Turn Lights On/Off

Interface of a Phone is set of these functions

• Input Number• Place Call • Disconnect Call • Add number to address book • Remove number • Update number

What is aggregation give example?

Answer:- (Page 53)

An object may contain a collection (aggregate) of other objects, the relationship between the container and the contained object is called aggregation, For Example: A room and its components like table, chair and bed etc are example of aggregation.

What is an abstract class?

Answer:- (Page 21)

Taking part of the objects and their functions that is related to the problem and ignore all other irrelative characteristic called abstraction

MIDTERM EXAMINATION 2010

Question No: 17 (Marks: 2)

Can we create an array of objects for a class having user defined constructor? Justify your answer.

Answer:- (Page 114)

There must always be a default constructor if we want to create array of objects

Question No: 18 (Marks: 2)

Friend functions increase 'Programming bugs'. What is your opinion?

Answer:-

Yes friend functions can increase programming bugs some time.

Question No: 19 (Marks: 2)

Explain two benefits of **setter functions.**

Answer:- click here for detail

- They help to define the variable in an interface
- To make a variable read-only
- notify other methods / classes when the value is changed

Question No: 20 (Marks: 3)

What are binary operators? Give an example of binary operators overloading using any class.

Answer:- (Page 142)

Binary Operators Overloading:

Binary operators act on two quantities.

Examples of binary operators:

Examples:

Overloading + operator:

```
class Complex{
private:
double real, img;
public:
…
Complex operator +(const Complex & rhs);
};
Complex Complex::operator +( const Complex & rhs){
Complex t;
t.real = real + rhs.real;
t.img = img + rhs.img;
return t;
}
```

The binary operator is always called with reference to the left hand argument. (Page 145)

Question No: 21 (Marks: 3)

Give c++ code to overload assignment operator for string class.

Answer:- (Page 148)

```
class String{
int size;
char * bufferPtr;
public:
String(); // default constructor
String(char *); // overloaded constructor
String(const String &); // copy constructor
…
};
String::String(){
bufferPtr = NULL;
size = 0;
}
String::String(char * ptr){
if(ptr != NULL){
size = strlen(ptr);
bufferPtr = new char[size+1];
strcpy(bufferPtr, ptr);
}
```

```
else{
bufferPtr = NULL;
size = 0;
}
}
String::String(const String & rhs){
size = rhs.size;
if(rhs.size != 0){
bufferPtr = new char[size+1];
strcpy(bufferPtr, ptr);
}
else
bufferPtr = NULL;
}
int main(){
String str1(“Hello");
String str2(“World”);
str1 = str2;78
return 0;
}
```

Question No: 22 (Marks: 5)
Writ c++ code to overload **subscript []** operator for **String class.**
Answer:- (Page 160)

Question No: 23 (Marks: 5)
Detect and correct compile time error(s) in the following code.

```
class  Exam
{
     char *ExamName;
     int No_of_paper;

public:
      Exam()
      {
           ExamName = "Final Term";
           No_of_paper = 5;
           }

void setname( char* name)  const
{
    ExamName = name;
    }
void setpaper(int paper)   const
{
    No_of_paper = paper;
```

```
    }
char* getname()
{
     return ExamName;
     }
int getpaper()
{
   return No_of_paper;
}
 };

int main()
{
const Exam exam1;

cout << " Exam           = "<<exam1.getname()<<endl;
cout << " Numbe of paper  = " << exam1.getpaper();

getch();
return 0;
}
```

Answer:-

Corrected Code

```
class  Exam
{
     char *ExamName;
     int No_of_paper;

public:
      Exam()
      {
           ExamName = "Final Term";
           No_of_paper = 5;
           }

void setname( char* name)
{
    ExamName=name;
    }
void setpaper(int paper)
{
    No_of_paper = paper;
    }
char* getname()
```

```
{
     return ExamName;
     }
int getpaper()
{
   return No_of_paper;
}
 };

int main()
{
Exam exam1;

cout << " Exam          = "<<exam1.getname()<<endl;
cout << " Numbe of paper  = " << exam1.getpaper();

getch();
return 0;
}
```

MIDTERM EXAMINATION 2010

Question No: 17 (Marks: 2)
If, within a class, da is a member variable, will the statement **this.da=25;** assign 25 to da? False
Answer:-
"this" is pointer, so using pointer to object we use the - > operator to access object member function and data member,

Question No: 18 (Marks: 2)
Explain the scope of 'private' member of a class with one example.

Answer:-
The scope of private data member of a class is that, only an access is allowed within the class. Only member function of class can access and modify the value of data member value. Derived class or class instance/object can't directly access. Data Member will created for each separate object in the memory and will eliminated when object will be destroy. Object could be destroy by calling its destructor. Example

```
Class arthimetic
{
   Private:
         int a,b;
   Public:
         Add()
         {
```

```
            A+b;  // direct access by member function Correct Access
        }
};

Int main()
{
        Arthimetic c1;
        C1.a=4;// is illegal access to class private data member error will generate by compiler
}
```

Question No: 19 (Marks: 2)
Friend functions increase 'Programming bugs'. What is your opinion?
Answer:- repeat

Question No: 20 (Marks: 3)

Assuming that class X does not use any overloaded operators and has a member function to subtract two objects of X and placing the result in third object using which it has been called, write a statement that we will use to subtract two objects of class X placing the result in third object for class X. You can take any names for three objects of X. **(unsolved)**

Question No: 21 (Marks: 3)
Give c++ code to overload unary '-- 'operator for **complex** numbers class.
Answer:- repeat

Question No: 22 (Marks: 5)
Write c++ of overloading ^ operator in **complex numbers class.**

If we have two objects of complex number class as follows,
Complex obj1,obj2;
and we write statement,
Complex obj3 = obj1 ^ obj2;
obj3 real and imaginary parts will be,

obj3.real = (obj1.real) $^{\textbf{obj2.real}}$ and **obj3.img = (obj1.img)** $^{\textbf{obj2.img}}$
Hint: You can use c++ built in function **power(x,y)** that returns the result of x^y.

Answer:-

```
Class complex
{
        Private:
                Double  val;
        Complex operator ^(complex c1)
        {
                Return Power(val,c1.val);
        }
};
```

Question No: 22 (Marks: 5) (Another related question come in Nov-2011 mid paper)

Write c++ of overloading - operator in complex numbers class which can work in these give situation.
If we have two objects of complex number class as follows,

```
Complex obj1;
Complex obj3 = 47.4- obj2;
Complex obj4 = obj2-47.4;
```

Answer:-

Handout Page No.146

We have made them as friend so that we can write them as non member functions and they ar not called with respect to complex no. Class object instead we pass both arguments (complex no. object and double value) to this function compiler invoke them according to arguments passed. Their implementation is similar as give below.

```
Class complex
{
        private:
                double  real, img;
        public:
        friend complex operator -(double c1, complex obj);
        Complex operator -(double c1)
        {
                Complex temp;
                temp.real=real-c1;
                temp.img=img-c1;
                return temp;
        }
};
//Friend function Definition
        Complex operator -( double c1, complex obj)
        {
                Complex temp;
                temp.real=c1-real;
                temp.img=c1-img;
                return temp;
        }
```

Question No: 23 (Marks: 5)

What are Accessor Functions, Explain with an example?

Answer:- (Page93)

Accessor functions are used to access private data of the object, we provide accessor functions to get and set private data members of the class.

Example

```
class Student{
…
int rollNo;
public:
```

```
void setRollNo(int aRollNo){
rollNo = aRollNo;
}
};
```

Avoiding Error

```
void Student::setRollNo(int aRollNo){
if(aRollNo < 0){
rollNo = 0;
}
else
{
rollNo = aRollNo;
}
}
```

MIDTERM EXAMINATION
2010

Q.1

What is difference between simple association and composition?

Answer:- (PAGE 51-52)

In Association, interacting objects have no intrinsic relationship with other object. It is the weakest link between objects. While in Composition An object may be composed of other smaller objects, the relationship between the "part" objects and the "whole".

Q.2

Friend functions minimize "Encapsulation", What is your opinion?

Answer:- (PAGE 153)

Friend functions minimize encapsulation as we can access private data of any class using friend functions,
This can result in:

- Data vulnerability
- Programming bugs
- Tough debugging

Q.3

Write three important properties of constructors?

Tell the reason why we can not overload the following four operators in c++

. , .* , :: , ?:

Considering the complex number class can we subtract two complex numbers by overloading plus "+ "Operator. Justify your answer as well.

Answer:-

Constructor Properties (Page 74)

- Constructor is a special function having same name as the class name
- Constructor does not have return type
- Constructors are commonly public members

The reason why we can not overload the following four operators in c++ **(Click here for detail)**

Because they are strictly compile time. There is a very strong argument against sizeof and unary &, since they

Muhammad Moaaz Siddiq
MCS (2nd)
mc100401285@Gmail.com
Campus:- Institute of E-Learning & Modern Studies (IEMS) Samundari

Muhammad Irfan Farid
MCS(2nd)
virtualcheater@gmail.com
Virtual University Campus . D.G. Khan (VDGK01)

are defined for all types, and have results which can be used in constant expressions (evaluated at "compile" time)

Q.4
How we resolve the following problems in overloading of assignment operator in string class, (explain with the help of c++ code)

lf referencing
signing a string value to more than one strings in a single line like, stringobject1 = string object2 = ingobject3 = stringobject4

Answer:-
Handout Page:150

```
class String{
…
public:
        …
      String & operator = (const String &);
};
String & String :: operator = (const String & rhs){
  size = rhs.size;
  delete [] bufferPtr;
  if(rhs.size != 0){
  bufferPtr = new char[rhs.size+1];
  strcpy(bufferPtr,rhs.bufferPtr);
}
else bufferPtr = NULL;
return *this;
}
```

Q.5
Consider the class given below explain the order in which variables e,f and g will be initialized after creating object of this class,

```
class XYZ{
        int e;
        int f;
        int g;
public:
        XYZ ();
};

XYZ:: XYZ ():g(30),e(10),f(20)
{
        …
}
```

Answer:-
The value will give in the way which they are defined in the class.
E=10,f=30 and g=20

MIDTERM EXAMINATION 2010

Question No: 17 (Marks: 2) **(unsolved)**

Can we create an array of objects for a class having default constructor?. Justify your answer.

Question No: 18 (Marks: 2)

Friend functions increase 'Data vulnerability', what is your opinion?

Answer:- (Page 153)

Friend functions minimize encapsulation as we can access private data of any class using friend functions, This can result in:

- Data vulnerability
- Programming bugs
- Tough debugging

Question No: 19 (Marks: 2)

Explain two benefits of setter functions.

Answer:- (Page 67)

Setters and getters functions are provided by class to access the its members it also minimizes the changes to move the objects in inconsistent state as we can write checks in our setter functions for example we can check that whether the user has entered correct age value and has not entered negative value for age.

Question No: 20 (Marks: 3)

Consider the class given below what will be the values in variables x,y and z after creating object of this class,

```
class ABC{
int x;
int y;
int z;
public:
ABC();
};
ABC::ABC():x(10),z(x),y(x)
{
…
}
```

Answer:-

X=10,z=10 and y=10

Question No: 21 (Marks: 3)

Explain what type of copy the default assignment operator "=" does when applied to objects. (shallow copy or deep copy)

Answer:- (Page 147)

if our class has any data member using dynamic memory then we had to write our own code for default constructor, copy constructor and similarly assignment operator as compiler generated version of these

functions performs shallow copy that creates dangling pointer, and memory leakage issues in case of dynamic memory allocation.

Question No: 22 (Marks: 5)
What is composition? Explain it with the help of an example.
Answer:- (Page 52)
An object may be composed of other smaller objects, the relationship between the "part" objects and the "whole" object is known as Composition, and Composition is represented by a line with a filled-diamond head towards the composer object. For example a man and his body parts have the relation of "part" objects and the "whole" object is known as Composition of man.

Question No: 23 (Marks: 5)
How we can overload Stream Extraction and Insertion Operators in c++? Give example code for Complex Number Class.
Answer:- (Page 155-157)
Overloading Stream Insertion Operator

```
class Complex{
...
friend ostream & operator << (ostream & os, const Complex & c);
};
```

Stream Insertion operator

```
// we want the output as: (real, img)
ostream & operator << (ostream & os, const Complex & c){
        os << '(' << c.real
         << ','
         << c.img << ')';
         return os;
}
```

Overloading Stream Extraction Operator

```
class Complex{
...
friend istream & operator >> (istream & i, Complex & c);
};
```

AL-JUNAID TECH INSTITUTE

CS304-OOP

MIDTERM MCQS

Prepared by: JUNAID MALIK

AL-JUNAID TECH INSTITUTE

www.vulmshelp.com

Language Courses Training Available

I'm providing paid courses in different languages within 3 Months,Certificate will be awarded after completion.

- HTML
- CSS
- JAVASCRIPT
- BOOTSTRAPS
- JQUERY
- PHP MYSQL
- NODES.JS
- REACT JS

LMS Handling Services

LMS Activities Paid Task

Assignments 95% Results

Quizes 95% Results

GDB 95% Results

For CS619 Project Feel Free To Contact With Me

Ph# 0304-1659294

Email: junaidfazal08@gmail.com

AL-JUNAID TECH INSTITUTE

SR NO	MCQS QUESTION	ANSWER
1	In expression c1*c2+c3-c4 which of the following will be executed in first order	C1*c2
2	Inheritance is a way to	Add features to existing classes without rewriting them
3	---------- represent “IS A ”relationship	Inheritance
4	Constructor without any parameter is called:	Default constructor
5	Which of the following stores the address of a variable?	Pointer
6	A Post-fix unary operator is implemented in C ++ using Member function with	1 dummy int argument
7	minimum classes required in a program for implementing multiple inheritance	3
8	Which of the following statement(s) describe constant member functions?	All of the given option
9	composition is a strong relation relationship because	Composed object cant exist independency
10	How many interactions are there in the following scenario Imran has the car. he drive the car and car has four wheels and doors	3
11	if we extend our model ,and the rest of the model is not affected then it is called-----	Flexibility
12	choose correct declaration of overload*=1 operator for class complex as Member function	Complex and operator*=(const complex & c)
13	Consider the following class A{int a,b,c; public: A():B(10),C(0)a(7) };	A,b,c
14	Class is a blue print of	Objects

AL-JUNAID TECH INSTITUTE

15	Suppose we have a class student which of the following statement is correctly declaring the pointer to the object of student class?	Student*object
16	In c++ composition is relationship between -------and------objects	Whole, part
17	Which of the following Depicts the proper definition of class?	a class is description of a kind of object
18	in class attributes are represented by ---- and------ respectively	data member, Member function
19	which of the following represents the two Association	Employee works for the company
20	Entities in OO model are represented by	Rectangle
21	Which of the following features of OOP is used to derive a class from another?	Inheritance
22	In C ++ which of the following is defined as a stream insertion operator?	<<
23	choose the correct declaration of overloaded equality (==)operator for class string as a Member function	Bool operator==(const string and s)
24	---- is the relationship between the part object and the whole object	Composition
25	object hides its information from others this concept is known as	information hiding
26	the concept of derived classes is involved in	Inheritance
27	abstraction provides information according to	user perspective
28	which of the following is not an object Association	Inheritance
29	which of the following classes are used by amphibious vehicle to inherit characteristics	both land and water vehicle
29	object data includes	both states and attributes of an object

AL-JUNAID TECH INSTITUTE

30	choose correct declaration of overloaded inequality operator for class string as non member friend function	friend pool operator !=(const string &s1,const string &2)
31	How many objects of a given class may be constructed in an application?	as many as the application ask for
32	how many objects of a given class may be constructed in an application	as many as the application ask for
33	Which of the following statements describe constant member functions?	all of the given option
34	student *ptr=new student(" Ali"); is an example of	static memory allocation
35	in------- base class can be replaced by the derived class	extension
36	Which of the following is a correct way of declaring constant variable ROLLNO?	cosnst int Roll no;
37	which of the following operator doesn't take argument hit overloaded	none of the given choices
38	which of the following is part of the Class diagram	all of the given option
39	a child inherits characteristic from its	parent
40	which of the following issues can be caused by the use of friend function	programming Bugs
41	which of the following is not true about constant number function	all of the given option
42	Which one is not the main features of object oriented programming?	exception handling
43		
44	in object oriented programming, objects communicate with each other through	messages
45	static data member is declared	inside the class
46	the concept of derived classes is involved in	inheritance
47	Which of the following can we attribute of an object "Ali"?	address

AL-JUNAID TECH INSTITUTE

48	Which of the following keyword tell the compiler to substitute the code within the function definition for every instance of a function call?	inline
49	inheritance is the relationship between	classes
50	how many objects are involved in the binary Association	2
51	all objects of same class has -------- attributes	same
52	which of the following concept is more close to encapsulation	information hiding
53	member initialization list is used to	initialize data member
54	aggregation between objects show that	objects have week relationship
55	there is /are ways to create user defined types for objects in C ++	two
56	composition is a strong relationship because:	composed object can not exist independently
57	in polymorphism messenger can be interpreted in different ways depending upon the----- class	receiver
58	which of the following is most suitable advantage of information hiding	simplifies the model
59	the process of extracting common characteristics from two or More classes and combining them into a generalized superclass is called	Generalization
60	------- operator is used to create objects at run time	new
61	choose the correct option to declare an overloaded Stream insertion operator for userdefind class "string" as a non-member friend function	friend out stream and operator<<(ostream and os,const string & s)
62	static data member is declared	inside the class
63	an instance of user defined type is called	object

AL-JUNAID TECH INSTITUTE

64	which of the following is true about constructor	all of the given option
65	Which of the following correct way of declaring constant variable ROLLNO?	const int ROLLNO;
66	inheritance is shown by	filled arrow
67	which of the following operators operate on one operand	unary operator
68	which of the following will happen when a constant function tried to change the value of data members of the class	compile time error will occur
69	in object oriented programming, objects communicate with each other through---	messages
70	How the information hidden within an object can be accessed?	through its interface
71	object orientation approach can be achieved using object and their-------	interactions
72	constructors have ----- return type	no
73	which of the following is an example of unary operator	increment operator(++)
74	memory is allocated to non-static members only when	object is created
75	in class B inherit from class A which of the following is child class	class B
76	the------ tells the compiler what task the function will be performing	function definition
77	Consider the statement" room has chair" which of the following type of Association exists between whom and chair?	aggregation
78	how many objects are involved in the N-ary Association	more than 3
79	deconstruct is a function which has the same name as that of class but starts with a----- sign	~
80	which of the following is a tangible entity	car

AL-JUNAID TECH INSTITUTE

81	member initializer list is used to	initialize data members
82	we can get the address of variable stored in pointer using-----	& symbol
83	two sentence object oriented programming book in bookshelf is an example of	aggregation
84	which of the following best defines the obstruction	hide the implementation and show current perspective
85	which of the following define default constructor 5 class student	student(){//..}
86	object data includes	both state and attributes of an object
87	minimum classes required in a program for implementing multiple inheritance	3
88	information hiding can be achieved through-------	encapsulation, abstraction
89	compiler generated defaulter structure is called----- hand user written default constructor is called----	implicit, explicit
90	which of the following is true about constructor	all of the given option
91	In which of the following different forms office single entity exist?	polymorphism
92	data members of a class student can be:	all of the given option
93	the----- turn off the compiler what does the function will be performing	function definition
94	Suppose a class does not have any constructor what will happen when an object of this class is created?	compiler will call implicit default constructor
95	how can we differentiate between constructor and destructor	destructor are preceded with a Tilda(~) symbol, and constructor are not preceded with any symbol

AL-JUNAID TECH INSTITUTE

96	---- is used for hiding the details of an object from the other parts of a program	encapsulation
97	Class diagram include	class name, attributes, operation
98	through interface was access objects----------	behavior
99	composition is a strong relationship because	compose object cannot exist independently
100	copy constructor is called when:	an object is created in term of Preexisting object
101	choose the correct option to declare an overloaded stream insertion operator for a user defined class string as a non-Member friend function	friend ostream & operator(ostream &os.const string &s)
102	a post-fix unary operator implemented in C plus plus using non Member function with---- arguments	two
103	in class attributes and behaviors are represented by---- and--- respectively	data members, member functions
104	which of the following best describes the relationship between book and book chapter classes	composition
105	which of the following is true about destructor	all of the given option
106	in composition----- are called from composing objects to composed subjects	destructors
107	which of the following can be used an alternative to Static members	Global variables
108	which of the following statement is true about static function of a class	it is used to access static data members
109	which of the following operator does not take any arguments if overloaded	none of the given choices
110	what is the output of the following code int main(){int cost x=10;cout<<++x;return0;}	error

AL-JUNAID TECH INSTITUTE

111	abstraction includes-------	relevant information
112	Which of the following can be the behavior of an object "Usman"?	eat
113	a class defined within another class is	nested class
114	when an object is initialized, is automatically called by the compiler	constructor
115	the compiler treats the Member function of a class as	inline function
116	an object has attributes operation ,and----	unique identity
117	Which of the following is not an access specifier in C ++?	hidden
118	which of the following parts of an object exhibits its state	data
119	---- is the relationship between the part object and the whole object	composition
120	how can we identify classes from a given problem statement	by extracting nouns from problem statement
121	what "friend" should be placed in	function declaration
122	generalization is-------- approach	bottom-up
123	static data members can be accessed through--------	dot operator and scope resolution operator
124	composition age----- and aggregation is---- relationship	strong, weak
125	suppose there is an object of type person, which of the following can be considered as one of its attributes	name
126	Which of the following depicts the proper definition of class?	a class is a description of a kind of object
127	a child inherits characteristics from its------	parent
128	the car is composed of wheels what is the relationship between car and wheels?	composition

AL-JUNAID TECH INSTITUTE

129	------ is creating objects of one class inside another class	composition
130	the other name of specialization is-------	restriction
131	suppose a is an int type static data member class test then which of the following is correct way of initialization the static variable outside the class test	int test::a=0
132	Which of the following stores the address of variable?	Pointer
133		
134		
135	static data member is declared	inside the class
136	polymorphism makes the system	all of the given option
137	which of the following statement best describes the constructor	constructor is used to initialize the data members of a class
138	data member(s) of a class student can be	all of the given option
139	which of the following is a strong relationship	composition
140	all objects of same class has------- attributes	same
141	in C ++ composition is a relationship between----and---- objects	whole, part
142	suppose 12 class string assignment operator is overloaded with following declaration void operator=(const string &); what will happen when we will write following statement in main() str=str2=str3;where,str1.str2,str3 are object of class string	compiler will generate compile time error
143	in class B inherits from class A then it contains all characteristics of----	class A

AL-JUNAID TECH INSTITUTE

144	which of the following is not an example of multiple inheritance	car
145	which of the following can be created without any object of a class?	static data members
146	consider the following code segment class test{int a;int b;int c;}	junk value,5, junk value
147	object orientation approach can be achieved using objects and their-----	interactions
148	member functions defined inside a class declaration are------- by default	private
149	how can we identify classes from a given problem statement	by extracting nouns from problem statement
150	which of the following statement is not true about static variable of a class?	static variable belongs to particular instance of a class
151	there are------ types of constructor in C++	3
152	statement yasir is a friend of Ali and Ali is a friend of yasir is an example of	two way Association
153	this pointer cannot be passed implicitly to-------- functions	static member
154	which of the following issues can be caused by the use of friend function	all of the given
155	in------ send a message does not need to know exact class of the receiver	none of the given option
156	minimum classes required in a program for implementing multiple inheritance	3
157	-------Associates the object of the exact two classes	binary Association
158	main characteristics of information hiding is	keep the model simple
159	static data member is declared	inside the class
160	in expression b=c/d, the arity of division(/) operator is	2

AL-JUNAID TECH INSTITUTE

161	we can allocate dynamic memory in C ++ through operator	new
162	Which of the following is true about accessor functions?	all of the given option
163	in---- base class cannot always be replaced by the derived class	specialization
164	abstraction help to	reduce the complexity
165	relationship in which child object gets destroyed if parent object is destroyed	composition
166	object oriented programming is better than procedural programming because:	it is more close to reality
167	The derived class inherits behavior from the base class and also exhibits its own behavior. this concept is known as---	Overriding
168	the other name of subtyping is	Extension
169	A good model is------ related to life problem	closely
170	in generalization general class represents the -------- behavior of the class	common
171	how can we identify from a given problem statement?	by extracting noun from problem statement
172	which of the following is not a access specifier in C ++	hidden
173	in composition------ are called from composing objects to composed objects	destructors
174	a class have only---- destructor	one
175	suppose you have following C + + statement int old value=10; int new value=++old value; what will be the value of old value and newValue after executing above statements ?	oldVlue=11,newVlue=11
176	in constant Member function the type of this pointer is	constant pointer to constant data
177	----- is creating objects of one class inside another class	composition

178	-------- Associates the objects of the exact three classes	ternary Association
179	generalization is implemented through	inheritance
180	which of the following is strong relationship	composition
181	composition is a strong relationship because	composed object cannot exist independently
182	----- is set of function of an object exposed of other objects	interface
183	suppose student is a class, which of the following defines constructor with one parameter for class student	student(char*aName);
184	which of the following is not true about constant Member function	all of the given option
185	----- define the order of evaluation of an operator is an expression	operator precedence
186	choose correct declaration of overloaded inequality(!=) operator for class string as non member friend function	friend bool operator!=(const string &s1,cont string &s2)
187	which of the following is most suitable advantage of information hiding	simplifies the model
188	a fan has wings, which type of relation exists between fan and Wings in this sentence	composition
189	multiple inheritance is implemented between class through------ symbol	comma(,)
190	postfix unary operator is implemented in C ++ using non Member function with arguments	two
191	Supposed derived class is inherited from base class what happens when a derived class object is created	the constructor of base class exact is executed before the constructor of derived class
192	assignment operator are	right associative

193	a postfix unary operator is implemented in C ++ using number function with	1 dummy int argument
194	which of the following is true about exception functions	all of the given option

Question No: 1

Which part of an object exhibits its state?

- ► **Data (Page 23)**
- ► Operations
- ► Any public part
- ► Any private part

Question No: 2

Inheritance is a way to

- ► organize data.
- ► pass arguments to objects of classes.
- ► **add features to existing classes without rewriting them. (Page 27)**
- ► improve data-hiding and encapsulation.

Question No: 3

Suppose you have been given the following design,

"A person has a name, age, address and sex. You are designing a class to represent a type of person called a patient. This kind of person may be given a diagnosis, have a spouse and may be alive".

Given that the person class has already been created, what of the following would be appropriate to include when you design the patient class?

AL-JUNAID TECH INSTITUTE
www.vulmshelp.com

- **registration date and diagnosis**
- age and sex
- sex and diagnosis
- diagnosis and age

Question No: 4

What problem(s) may occur when we copy objects without using deep copy constructor?

- Dangling pointer
- Memory Leakage
- **All of the given (Page 147)**
- System crash

Question No: 5

this pointers are not accessible for static member functions.

- True (Page 114)
- False

Question No: 6

A static member function cannot be declared.

- Static
- Implicit
- Explicit
- **Virtual**

Question No: 7

___ remain in memory even when all objects of a class have been destroyed.

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

- **Static variables (Page 111)**
- Instance variable
- Primitive variables
- None of given

Question No: 8

Friend functions are _____ functions of a class.

- None of given
- object member
- **non-member (Page 136)**
- data member

Question No: 9

_____, which means if A declares B as its friend it does NOT mean that A can access private data of B. It only means that B can access all data of A.

- **Friendship is one way only**
- Friendship is two way only
- NO Friendship between classes
- Any kind of friendship

Question No: 10

The statement objA=objB; will cause a compiler error if the objects are of different classes.

- **True**
- False

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 11

Identify which of the following overloaded operator function's declaration is appropriate for the given call?

Rational_number_1 + 2.325

Where Rational_number_1 is an object of user defined class Rational_number.

- Rational_number operator+(Rational_number & obj);
- **Rational_number operator+(double& obj); (Page 145)**
- Rational_number operator+(Rational_number &obj, double& num);
- operator+(double& obj);

Question No: 12

Which operator can not be overloaded?

- The relation operator (>=)
- Assignment operator (=)
- Script operator ([])
- **Conditional operator (? :) (Page 141)**

Question No: 13

To convert from a user-defined class to a basic type, you would most likely use

- a built-in conversion operator.
- a one-argument constructor.
- an overloaded = operator.
- **a conversion operator that's a member of the class.**

0304-1659294
www.vulmshelp.com

Question No: 14

The technique in which we visualize our programming problems according to real life's problems is called

---

- ► structured programming
- ► **object oriented Programming (Page 9)**
- ► procedural programming
- ► non of the given

Question No: 15

In object orientated programming, a class of objects cans _____ properties from another class of objects

- ► Utilize
- ► Borrow
- ► **Inherit**
- ► Adopt

Question No: 16

A C++ class is similar to ----------

- ► **Structure**
- ► Header File
- ► Library File
- ► None of the given

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 17

What problem(s) may occur when we copy objects without using deep copy constructor?

- ► Dangling pointer
- ► Memory Leakage
- ► **All of the given (Page 147)**
- ► System crash

Question No: 18

Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

- ► The automatic assignment operator is used
- ► **The copy constructor is used**
- ► Compiler error
- ► Run-time error

Question No: 19

a'A static member function can be called, even when a class is not ____.

- ► Declared
- ► Define
- ► **Instantiated C++ How to Program, Fifth Edition(Page 871)**
- ► Called

Question No: 20

Identify which of the following overloaded operator function's declaration is appropriate for the given call?

Rational_number_1 + 2.325

Where Rational_number_1 is an object of user defined class Rational_number.

0304-1659294 www.vulmshelp.com

- Rational_number operator+(Rational_number & obj);
- **Rational_number operator+(double& obj); (Page 145)**
- Rational_number operator+(Rational_number &obj, double& num);
- operator+(double& obj);

Question No:21

____ provide the facility to access the data member.

- **accesser function (Page 68)**
- private function
- inline function
- None of the given

Question No: 22

Constant objects cannot change their state,

- **True (Page 105)**
- False

Question No: 23

The ____ relationship indicates that an object contains other objects.

- **None of given (Page 53)**

0304-1659294
www.vulmshelp.com

- ‘has-a’
- ‘is-a’
- ‘be-

Question No: 24

Which one of the following features of OOP is used to derive a class from another?

- Encapsulation
- Polymorphism
- Data hiding
- **Inheritance (Page 25)**

Question No: 25

___ is a relationship

- **Inheritance (Page 25)**
- Polymarphism
- abstraction
- encapsulation

Question No: 26

____ satisfy the condition of polymorphism

- Carbon
- Diamond
- Coal
- **all of the given (Page 56)**

Question No: 27

A generalization-specialization relation between classes are implemented using

0304-1659294
www.vulmshelp.com

- ► data hiding
- ► friend classes
- ► encapsulation
- ► **inheritance (Page 49)**

Question No: 28

The >= operator can be overloaded.

- ► **True (Page 140)**
- ► False

Question No: 29

In order to free the memory occupied by the object, we use ----------

- ► Constructor
- ► **Destructor (Page 92)**
- ► Shallow Copy
- ► Deep Copy

Question No: 30

Which of the following is not an example of multiple inheritances?----------

- ► Mermaid
- ► **Woman (Lecture No.5)**
- ► None of the given
- ► Amphibious Vehicle

AL JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 31

Static variable can be initialized more than once.

- ► **True**
- ► False

Question No: 32

A generic class showing all the common attributes and a behavior of other classes represents a very important feature in oop called --------

- ► **Inheritance (Page 26)**
- ► Encapsulation
- ► Polymarphism
- ► Abstraction

Question No: 33

We can get only one unique value which can be used by all the objects of that class by the use of,

- ►**static variables**
- ►dynamic variables
- ►instance variables
- ►data members

Question No: 34

A member function having the same name as that of a class and a ~ sign with it is called,

- ►Constructor

0304-1659294
www.vulmshelp.com

►Getter

►Setter

►**Destructor (Page 92)**

Question No: 35

Using encapsulation we can achieve

►**Information hiding (Page 16)**

►Least interdependencies among modules

►Implementation independence

►All of given options

Question No: 36

Inheritance is a way to

►**make general classes into more specific classes. (Page 27)**

►pass arguments to objects of classes.

►improve data hiding and encapsulation.

►providing class growth through natural selection.

Question No: 37

Static variable can be initialized more than once.

- True
- **False (Page 107)**

0304-1659294
www.vulmshelp.com

Question No: 38

For classes with common behavior, you can save effort by placing the common behavior in a ____.

►Derived Class

►**Base class (Page 29)**

►Deprived Class

►Named class

Question No: 39

Which of the following are an advantage of OOP?

►OOP makes it easy to re-use the code

►It provides an ability to create one user defined data type by extending the other

►It provides the facility of defining Abstract data types through which real world entities can be defined better.

►**All of the given options** Click here for detail

Question No: 40

The >= operator can be overloaded.

►**True (Page 140)**

►False

Question No: 41

A static member function cannot be declared.

►Static

►Implicit

►Explicit

►**Virtual**

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

Question No: 42

Static variables act like a global variable in the context or scope of the class.

►**True**

►False

Question No: 43

The compiler won't object if you overload the * operator to perform division.

►**True**

►False

Question No: 44

We can use "this" pointer in the constructor in the body and even in the initialization list of any class if we are careful,

►TRUE

►**False**

Question No: 45

A C++ class is similar to ----------

►**Structure**

►Header File

►Library File

►None of the given

Question No: 46

Which operator can not be overloaded?

►The relation operator (>=)

►Assignment operator (=)

0304-1659294
www.vulmshelp.com

►Script operator ([])

►**Conditional operator (? :) (Page 141)**

Question No: 47

An overloaded operator always requires one less argument than its number of operands.

►**True**

►False

Question No: 48

A generalization-specialization relation between classes are implemented using

►data hiding

►friend classes

►encapsulation

►**inheritance (Page 49)**

Question No: 49

In OOP a class is an example of ___

►Data Type

►Abstract Type

►**User Defined Type (Page 66)**

►None of the given

Question No: 50

A class can be identified from a statement by ----------

►**Noun (Page 58)**

►Pronoun

►Verb

►Adverb

AL JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Question No: 51

The members of a class that can be accessed without creating the object of the class is called

►Private member

►Data Member

►**Public Member (How to Program page 983)**

►Static

Question No: 52

Suppose there is an object of type Person, which of the following can be considered as one of its attributes

► Name

► Age

► Work()

► **Both Name and Age (page 13)**

Question No: 53

What a derived class can add?

► New data members

► New member functions and New friend functions

► New constructors and destructor

► **All of given**

Question No: 54

____ is/are used to access information hidden within an object?

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

► Interface

► Private data members

► **Private member functions (Page 69)**

► Both public and private members

Question No: 55

this pointers are not accessible for static member functions.

► **True (Page 14)**

► False

Question No: 56

A static member function cannot be declared.

► Static

► Implicit

► Explicit

► **Virtual**

Question No: 57

C++ compiler does not allow to dynamically allocate memory for objects

► **False**

► True

Question No: 58

Given the following class

class Base{

int Age=33;

}

How you can improve above class with respect to accessing the field Age?

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

► Define the variable Age as private

► Define the variable Age as protected

► **Define the variable Age as private and create a get method that returns it and a set method that updates it**

► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 59

Friend class and friend function can be used as an alternate to each other

► **True**

► False

Question No: 60

Which of the following operators always takes no argument if overloaded?

► /

► -

► +

► **++**

Question No: 61

Suppose that the Test class does not have an overloaded assignment operator. What happens when an assignment a=b; is given for two Test objects a and b?

► The automatic assignment operator is used

► **The copy constructor is used**

► Compiler error

► Run-time error

0304-1659294
www.vulmshelp.com

Question No:62

Assume a class C with objects obj1, obj2, and obj3. For the statement obj3 = obj1 - obj2 to work correctly, if the overloaded - operator must

► take two arguments.

► **return a value**

► create a named temporary object.

► take four arguments

Question No: 63

Which operator can not be overloaded?

► The relation operator (>=)

► Assignment operator (=)

► Script operator ([])

► **Conditional operator (? :) (Page 141)**

Question No: 64

We achieve independence of internal implementation from its external interface through----------

► Encapsulation

► Information Hiding

► **Abstraction**

► both encapsulation and information hiding

Question No: 65

0304-1659294
www.vulmshelp.com

Which one of the following is not an object association?

► Simple Assocation

► **Inheritance (Page 49)**

► Aggregation

► Composition

Question No: 66

Keeping in view the principle of abstraction, which of the above information the company needs to save as employee's record?

► 2, 4

► 1, 3, 5

► **1, 2, 3 (Lecture No.3)**

► 1, 2, 3, 4

Question No: 67

What a derived class can add?

► New data members

► New member functions and New friend functions

► New constructors and destructor

► **All of given**

Question No: 68

The concept of derived classes is involved in,

AL JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

► **inheritance (Page 25)**

► encapsulation

► structure

► array

Question No: 69

Your chief Software designer has shown you a sketch of the new Computer parts system she is about to create. At the top of the hierarchy is a Class called Computer and under this are two child classes. One is called LinuxPC and one is called WindowsPC. The main difference between the two is that one runs the Linux operating System and the other runs the Windows System (of course another difference is that one needs constant re-booting and the other runs reliably). Under the WindowsPC are two Sub classes one called Server and one Called Workstation. How might you appraise your designers work?

► Give the goahead for further design using the current scheme

► **Ask for a re-design of the hierarchy with changing the Operating System to a field rather than Class**

► Ask for the option of WindowsPC to be removed as it will soon be obsolete

► Change the hierarchy to remove the need for the superfluous Computer Class.

Question No: 70

Consider the code below,

```
class Fred {
public:
Fred();

...
};
```

0304-1659294
www.vulmshelp.com

```
int main()
{
Fred a[10];
Fred* p = new Fred[10];
...
}
```

Select the best option,

- ► Fred a[10]; calls the default constructor 09 times

Fred* p = new Fred[10]; calls the default constructor 10 times

- ► Produce an error
- ► **Fred a[10]; calls the default constructor 11 times**

Fred* p = new Fred[10]; calls the default constructor 11 times

- ► Fred a[10]; calls the default constructor 10 times

Fred* p = new Fred[10]; calls the default constructor 10 times

Question No:71

Which construct is the source for the creation of an object?

- ► Destructor of the class
- ► **New operator**
- ► Delete operator
- ► Constructor of the class

Question No: 72

this pointers are not accessible for static member functions.

- ► **True (Page 14)**
- ► False

0304-1659294
www.vulmshelp.com

Question No: 73

When a variable is define as static in a class then all object of this class,

- ► Have different copies of this variable
- ► **Have same copy of this variable**
- ► Can not access this variable
- ► None of given

Question No: 74

___ remain in memory even when all objects of a class have been destroyed.

- ► **Static variables (Page 111)**
- ► Instance variable
- ► Primitive variables
- ► None of given

Question No: 75

Given the following class

```
class Base{
    int Age=33;
}
```

How you can improve above class with respect to accessing the field Age?

- ► Define the variable Age as private
- ► Define the variable Age as protected
- ► **Define the variable Age as private and create a get method that returns it and a set method that updates it**
- ► Define the variable Age as protected and create a set method that returns it and a get method that updates it

Question No: 76

The life of sub object is not dependant on the life of master class in _____.

0304-1659294 www.vulmshelp.com

- Composition
- **Aggregation (Page 134)**
- Separation
- None of the given

Question No: 77

Which one is not keyword in C++?

- operator
- **B_op**
- const
- None of given

Question No: 78

The >= operator can be overloaded.

- **True (Page 140)**
- False

0304-1659294
www.vulmshelp.com

Question No: 79

A template provides a convenient way to make a family of

- variables and data members
- **functions and classes**
- classes and exceptions
- programs and algorithms

Question No: 80

Which one of the following terms must relate to polymorphism?

- ► Static allocation
- ► Static typing
- ► **Dynamic binding (How to program , page 1103)**
- ► Dynamic allocation

Question No: 81

What is true about function templates?

- ► The compiler generates only one copy of the function template
- ► **The compiler generates a copy of function respective to each type of data (Page 257)**
- ► The compiler can only generate copy for the int type data
- ► None of the given.

0304-1659294
www.vulmshelp.com

Question No: 82

Which of the following is the best approach if it is required to have more than one functions having exactly same functionality and implemented on different data types?

- ► **Templates**
- ► Overloading
- ► Data hiding
- ► Encapsulation

Question No: 83

template <>

class Vector<char*> { }

This is an example of partial specialization.

► **True (Page 281)**

► False

Question No: 84

Classes like TwoDimensionalShape and ThreeDimensionalShape would normally be concrete, while classes like Sphere and Cube would normally be abstract.

► True

► **False Click here for Detail**

Question No: 85

A non-virtual member function is defined in a base class and overridden in a derived class; if that function is called through a base-class pointer to a derived class object, the derived-class version is used.

► True

► **False**

Question No: 86

Assume a class Derv that is privately derived from class Base. An object of class Derv located in main() can access

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

► **public members of Derv.**

► protected members of Derv.

► private members of Derv.

► protected members of Base.

Question No: 87

In order to define a class template, the first line of definition must be:

► **template <typename T> (Page 281)**

► typename <template T>

► Template Class <ClassName>

► Class <Template T>

Question No: 88

If there is a pointer p to objects of a base class, and it contains the address of an object of a derived class, and both classes contain a nonvirtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the ___ class to be executed.

► Base

► Derived

► Abstract

► **virtual (not sure)**

Question No: 89

When the base class and the derived class have a member function with the same name, you must be more specific which function you want to call (using _____).

0304-1659294
www.vulmshelp.com

- ► scope resolution operator
- ► dot operator
- ► null operator
- ► **Operator overloading (Page 204)**

Question No: 90

Non Template Friend functions of a class are friends of ____instance/s of that class.

- ► **All**
- ► One specific
- ► All instances of one date type
- ► None of the given options

Question No: 91

The find() algorithm

- ► finds matching sequences of elements in two containers.
- ► finds a container that matches a specified container.
- ► **takes iterators as its first two arguments.**
- ► takes container elements as its first two arguments.

Question No: 92

If you define a vector v with the default constructor, and define another vector w with a one-argument constructor to a size of 11, and insert 3 elements into each of these vectors with push_back(), then the size() member function will return __ for v and ___ for w.

- ► 11 for v and 3 for w.
- ► 0 for v and 0 for w.
- ► 0 for v and 3 for w.
- ► **3 for v and 11 for w.**

Question No: 93

0304-1659294 www.vulmshelp.com

Which of the following may not be an integral part of an object?

- ► State
- ► Behavior
- ► Protected data members
- ► **All of given (not sure)**

Question No: 94

Which is not the Advantage of inheritance?

- ► providing class growth through natural selection.
- ► facilitating class libraries.
- ► **avoiding the rewriting of code.**
- ► providing a useful conceptual framework

How can we differentiate between constructor and destructors?

Destructor are preceded by a tilde(~) symbol and constructor are not preceded with any symbol.

If you do not initialize static variable of int type then it is automatically initialized with _____

0

Choose the correct statement

All of the given options

___associates the objects of the exact two classes.

Binary association

Which of the following is the advantage of inheritance?

Reusability

Member function defined inside a class declaration are___by default

AL-JUNAID TECH INSTITUTE 0304-1659294 www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Public

Class is a single class provides functionally to operate on different type of data.

Template

Which of the following may inherit from an ordinary class?

All of the given

Which of the following represents complete specialization? Template <int.char>

Are called Non-type parameters

Which of the following is correct way to define a template class X?

typename<class T > class X {};

Which of the following is a sequence Container?

Vector

In resolution order of function template compiler searches for in the end.

Generic Template

May inherit from a complete specialization.

All of the given option

Which of the following may inherit from an ordinary class?

All of the given option

Which of the following is NOT an Associate Container?

Group

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

When was specialize a function template .it is called

Function template Overriding

Which of the following types of Inheritance is used to model "Implemented in terms of" relationship?

Private

In case of protected inheritance, public members of base class will be in derived class?

Protected

In C++ by default access of classes is .

Private

If there is a pointer p to objects of a base class , and it contains the address of an object of a derived Class, and both classes contain a virtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the class to be executed.

Derived

In case of private inheritance, private members of base class will be in derived class.

Hidden

Defines the order of evaluation of an operator in an expression.

Operator Precedence

Which of the following is the correct syntax of declaring static variable "count" of type int?

Static int count;

Defining a function in class body, the compiler treats that function as:

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Inline function

What is the output of the following code?Int main()

{intconst x = 10; cout<< ++x; return 0;}

11

Which of the following statements best describes the Constructor?

Constructor is used to initialize the data members of a class.

What will the value of variables a, b, and c after instantiating an object of above class?

Junk Value, 5, Junk value

Which of the following is a strong relationship?

Inheritance binding means that target function for a call is selected at compile time. Static

In case of public inheritance, protected members of base class will be in derived class.

Protected

A template provides a convenient way to make a family of

functions and classes

0304-1659294
www.vulmshelp.com

Function overriding is done in context of,

Derived and base class

In Protected Inheritance the public members of base class become in derived class.

Protected

In resolution order compiler search firstly .

Ordinary function

AL-JUNAID TECH INSTITUTE

Compiler generated copy constructor performs .

Both Shallow aperformscopy

A pure virtual function is a virtual function that

causes its class to be abstract

Methodologies to the development of reusable software relate to .

Generic programming

In C++ generic programming is done using .

Packages

Consider the code below, class class1{

Private:

Function func1 of class1 is in class2.

Private

When derived class has user defined assignment operator.

Compiler itself calls base Class default constructor.

A Child class can call constructor of its parent class through,

Both from its constructor initialization list or body

How many objects of a given class may be constructed in an application?

As many as the Application asks for

Which of the following is True about Destructor?

All of the given

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

How many objects are involved in the N-ary association?

More than 3

Which of the following can be the behavior of an object "Usman"?

Eat

In C++, Composition is a relationship between and objects.

Parent, Child

Generalization is approach.

Top-bottom

The tells the compiler what task the function will be performing.

Function declaration

Suppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of overloaded Assignment operator for the following statement to work correctly.

String& operator =(const String &);

In operator overloading, which of the following operator takes one or no argument.

<<

Class is blueprint of .

Objects

Polymorphism makes the system:

All of the given

If class B inherits from class A then it contains all characteristics of .

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Class A

What is meant by multiple inheritance?

Deriving a derived class from more than one base class

_____Class is a single class provides functionally to operate on different type of Data.

Template

Which of the following may inherit from an ordinary class?

All of the given

Which of the following represents complete specialization?

Template <int.char>

The parameters given in template definition other than those used for mentoring templates types are called

Non-type parameters

Which of the following is correct way to define a template class X?

Typename<class T > class X {};

Which of the following is a sequence Container?

Vector

In resolution order of function template compiler searches for_____in the end.

Generic template

_____may inherit from a complete specialization.

All of the given option

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following may inherit from an ordinary class?

All of the given option

Which of the following is NOT an Associate Container?

Group

Which of the following is NOT casting operator in C++ standard?

Var_cast

Which of the following types of Inheritance is used to model "Implemented in terms of" relationship?

Private

In case of protected inheritance, public members of base class will be_____in derived Class?

Protected

In C++ by default access of classes is_____.

Private

If there is a pointer p to objects of a base class , and it contains the address of an object of a Derived class, and both classes contain a virtual member function, ding(), then the statement p->ding(); will cause the version of ding() in the_____class to be executed.

Derived

Suppose we have defined derived class copy constructor but have not defined base class Copy constructor then compiler will,

Use base class default constructor

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In case of private inheritance, private members of base class will be_____in derived Class.

Hidden

_____defines the order of evaluation of an operator in an expression.

Operator Precedence

Which of the following is the correct syntax of declaring static variable "count" of type int?

Static int count;

When was specialize a function template .it is called_____

Function template overriding

Which of the following is/are key component(s) of standard Template Library?

All of the given option

Each_____of a template class by default becomes function template.

Member function

In statement "template<class U,int |=5>"the non type parameter is

Int |

Solution order highest priority is given to_____in template

Complete template

In case of template specialization ,if compiler cannot find required complete

Partial specialization

AL-JUNAID TECH INSTITUTE
0304 1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

_____is a pointer which is declared outside the _____

Cursor, Container

Suppose a template class "Test" has a static data member. How many copies of this static daya member following line of C++ code?

Three

In statement template <class T,class U ,int |=5> the non-parameter is_____.

Class U

A template argument is preceded by the keyword_____.

Class.

We can have_____type of member functions in a class.

All of the given

Which of the following is a strong relationship?

Inheritance

_____Binding means that target function for a call is selected at compile time.

Static

In case of public inheritance, protected members of base class will be_____in derived Class.

Protected

A template provides a convenient way to make a family of

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Functions and classes

Function overriding is done in context of,

Derived and base class

Which of the following is TRUE.

Derived class pointer can be used as Base class pointer

The default inheritance mode is,

Private inheritance

In Protected Inheritance the public members of base class become_____in derived class.

Protected

In resolution order compiler search firstly_____.

Ordinary function

Compiler generated copy constructor performs_____.

Both Shallow and Deep copy

A pure virtual function is a virtual function that

Causes its class to be abstract

Methodologies to the development of reusable software relate to_____.

Generic programming

In C++ generic programming is done using_____.

Packages

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Consider the code below,

Class class1{

Private:

Void func1();

};

Class class2 : private class1 {

};

Function func1 of class1 is_____in class2.

Private

When derived class has user defined assignment operator.

Compiler itself calls base class default constructor.

A Child class can call constructor of its parent class through,

Both from its constructor initialization list or body

How many objects of a given class may be constructed in an application?

As many as the application asks for

The other name of specialization is_____.

Restriction

How can we differentiate between constructors and destructors?

Destructors are preceded with a tilde (~) symbol, and constructors are not preceded with any symbol.

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

How many entities are there in the following scenario?Ali lives in the house that is near the Tree and he also drives his car.

2

Which of the following is True about Destructor?

All of the given

How many objects are involved in the N-ary association?

More than 3

Which of the following can be the behavior of an object "Usman"?

Eat

In C++, Composition is a relationship between_____and_____objects.

Parent, child

If you have three classes in a C++ program A, B, and C where class A inherits from class B, then class_____contains all the characteristics of class_____.

A, B

In case when we define the function outside the class then we must use the keyword _____to make the function inline.

Inline

Mermaid is an example of:

Multiple Inheritance

AL JUNAID TECH INSTITUTE
0304 1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Assignment operators are:

Right associative

Generalization is_____approach.

Top-bottom

The_____tells the compiler what task the function will be performing.

Function declaration

Uppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of Overloaded assignment operator for the following statement to work correctly.

String& operator =(const String &);

In operator overloading, which of the following operator takes one or no argument.

<<

Class is blueprint of_____.

Objects

Polymorphism makes the system:

All of the given

If class B inherits from class A then it contains all characteristics of_____.

Class A

What is meant by multiple inheritance?

Deriving a derived class from more than one base class

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In C++, which of the following operator can only be overloaded as a member function of the class?

Stream Extraction Operator: >>

Insertion operator is_____associative.

Left to Right

In expression c1*c2+c3-c4 which of the following will be executed in first order?

C1*c2

Suppose for a class String, assignment operator is overloaded with following declaration.

Void operator =(const String &);

What will happen when we will write following statement in main()?

Str1 = str2 = str3

Where, str1, str2 and str3 are objects of class String.

Compiler will generate compile time error.

Identify the abstract class from the given statement:

"Vehicle class is base class. Bus, Car, and Truck are derived classes"

Vehicle

Which of the following operators operate on one operand?

Unary Operators

Subtyping means that derived class is behaviorally_____with the base class.

Compatible

AL JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In_____, Base class can't always be replaced by the derived class.

Specialization

The overloaded '-' operator for Complex class will be called with reference to_____in the following statement

Complex C3 = C1 – C2;

Complex

What is a class?

A class is a description of a kind of object

In polymorphism, messages can be interpreted in different ways depending upon the _____class.

Receiver

Suppose there is an object of type Person, which of the following can be considered as one of its attributes?

Name

Which of the following is True about class?

All of the given

In C++, which of the following keywords works only with constructors?

Explicit

Choose correct declaration of overloaded stream insertion operator for class String as non-member friend function.

Friend ostream& operator << ();

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following classes are used by Amphibious vehicle to inherit characteristics?

Both Land & Water Vehicle

Suppose derived class is inherited from base class. What happens when a derived class object is created?

The constructor of base class is executed before the constructor of derived class.

Suppose str1, str2 and str3 are objects of class String. Choose appropriate declaration of Overloaded assignment operator for the following statement to work correctly.

Str1 = str2 = str3;

String& operator =(const String &);

In C++, which of the following is defined as stream insertion operator?

<<

Choose correct declaration of overloaded inequality (!=) operator for class String as non-Member friend function.

None of the given options

Relationship in which child object gets destroyed if parent object is destroyed.

Composition

How the information hidden within an object can be accessed?

Through its interface

Which of the following defines default constructor for class "Student"?

Student() {//…

}

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

The sub-object's life is not dependent on the life of master class in_____.

Aggregation

Hiding the implementation details makes program:

Easy to understand

The functions defined inside the class are by default_____.

Inline

Which of the following statement is TRUE about static function of a class?

It is used to access static data members.

Deconstruct is a function which has the same name as that of class but starts with a—– sign

~

'this' pointer cannot be passed implicitly to_____functions.

Static Member

Which of the following can be the behavior of an object "Usman"?

Eat

In Object Oriented programming, objects communicate with each other through _____

Messages

Constructor is used to_____the objects of a class.

Initialize

AL JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following is NOT an access specifier in C++?

Hidden

The other name of subtyping is_____.

Extension

Suppose a class does not have any constructor. What will happen when an object of this class is created?

Compiler will call implicit default constructor.

In case when we define the function outside the class then we must use the keyword _____To make the function inline.

Inline

In constant member function the type of this pointer is:

Constant pointer to constant data

In_____, base class can be replaced by the derived class.

Extension

We can get the address of a variable stored in a pointer using_____:

& symbol

A child inherits characteristics from its_____.

Parent

AL JUNAID TECH INSTITUTE
0304 1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Which of the following function declaration is correct to overload the + operator as member function in complex class?

Complex operator +(const Complex &rhs);

Which of the following is not true about constant member function?

All of the given options

What is the general syntax of overloading Unary Operator as member function of the class?

TYPE & operator OP ();

_____is creating objects of one class inside another class.

Composition

Which of the following is true about constructor?

All of the given option

Memory is allocated to non static members only, when

Object is created

A child inherit characteristics from its ?

Parents

A post-fix unary operator is implemented in C++ member function with

1 dummy int arguments

Which of the following is the example of objects in school?

Books , pages

AL JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

Suppose student is a class, which of the following constructor with one parameter for class student?

Student()

If you have three classes in a C++ program A, B, and C where class A inherits from class B, then class_____contains all the characteristics of class_____.

A,C

In_____base class can be replaced by the derived class.

Extension

The concept of derived class is involved in _____

Inheritance

Suppose person is a class which of the following statement defines an object of class person?

Create person object

Which of the following is true about Accessor Functions?

All of the given options.

The other name of subtyping is

Extension

Which of the following statement best describes the constructor

Constructor is used to initialize the data member of a class.

Which of the following is true about inline function?

It is used by compilers to improve efficiency of the program

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

Which of the following is defined as stream extraction operator in c++?

>>

An object has

_____interfac

e(s)One or more than one

Which of the following is an advantage of

encapsulation?Better understanding

Which of the following represents the two-way

association?Employee works for the company

___is the weakest link between

object.Simple association

In composition_____are called from composing objects to compound

objects.Constructor

In OOP we can achieve reusability though _____ Inheritance

Generalization is implemented through

Inheritance

Suppose str1, str2, and str3 are objects of class string. Choose appropriate declaration of overloaded

Assignment operator for the following statement to work

correctly.String& operator=(const String &)

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com

AL-JUNAID TECH INSTITUTE

In C++ which of the following is defined as stream insertion operator.

<<

Abstraction hides the

_____deta

ils.Relevant

Minimum classes required in a program for implementing multiple

inheritance.3

Composition is_____and aggregation is_____relationship.

Strong, weak

Minimum classes required for implementing multiple

inheritance.At least 3

We can access a private static variable through.

Static data member

The process of extracting common characteristics from two or more classes and combiningthem into generalized superclass is called ___

Generalization

Which of the following is true about

polymorphism?All of the given

Which of the following can be the behavior of an object "Usman"?

0304-1659294
www.vulmshelp.com

Eat

Composition is

_____relations

hipPart whole

Which of the following is a strong

relationship?Composition

Which of the following defines default constructor for class student?

Student(){/…

}

This car is composed of wheels. What is the relationship between car and wheels?

Composition

In_____base class can be replaced by the derived

class.Extension

___defines the order of evaluation of an operator is an

expression.Operator precedence

In composition___are called from composing objects to composed

objects.Destructor

0304-1659294
www.vulmshelp.com

How many objects of a given class may be constructed in an application?As many as the application asks for

Relationship in which child object gets destroyed, If parent object is destroyed.Composition

Which of the following is not true about constructor?

Constructor is a special function which can have different name than class name.

Suppose test is a class, void show() is its public member function. Which of the following is correct call of Show() function using Test pointer ptr?Ptr->show()

Which of the following features of OOP is used to deal with only relevant details?Abstraction

In C++ the subscript operator must be overloaded as a____of the class, with one parameter of ___type.

Member function, int

Identify the abstract class from the given statement

"vehicle class is base class. Bus, Car and truck are derived classes"

Vehicle

In Object Oriented programming objects communicate with each other through _____

Messages

 0304-1659294

_____associates the objects of the exact three classes.Ternary Association

An objects has attributes, operations and _____

Unique identity

In C++ which of the following keywords works only with constructors?

Explicit

Which of the following allows to reuse characteristics of more than one parent class?

Multiple Inheritance

How many objects are involved in the Binary association?2

Which of the following features of OOP is used to deal with only relevant details?

Abstraction

The overloaded '-' operator for complex class will be called with reference to____in the following

Statement

Complex C3 = C1 –

C2

Complex

Static data member is declared

Inside the class

Class is blue print of ___ Object

Which of the following is true about inline functions?All of the given options

How many objects can be created once a class is declared?As many as needed

Suppose you have following C++ statementsIntoldValue=10

intviewValue * ++ old/Value

What will be the value of old/Value and new/Value after executing above statement?

Old/Value=11, new/Value=11

In____base class can't always be replaced by the derived class.

Specialization

Which of the following will happen when a constant function tries to change the value of datamembers of the class?

Complex time error will occur

_____is set of functions of an objects exposed to other objects.Interface

How many objects are involved in the Ternary association?3

0304-1659294
www.vulmshelp.com

Mermaid is an example of Multiple inheritance

Which of the following is the correct syntax of declaring static variable 'count' of type int?

Static int count

Which of the following operators operate on one operand? Unary Operators

Object Data includes ____

Both static and attributes of an objects

AL-JUNAID TECH INSTITUTE
0304-1659294
www.vulmshelp.com