Lab Manuals

CS603P - Software Architecture and Design (Practical)

Department of Computer Science, Virtual University of Pakistan

Lab Manual- 01

Topic: You pay now or you pay later

SIMCO pvt limited wants to develop an account system. They hire a Techno software development company for this development. After some time, SIMCO realize that Techno's staff do not giving proper attention towards requirement engineering process as well as design and architecture of the system. They are just rushing for direct development and final output. Some of the SIMCO's staff (Group A) supporting Techno way of working to get final product soon and digitalize the system while others (Group B) demands proper documentation of the system.

Give Your Thought:

You are required to give your point of view about group A and B. Which of the group is on right direction and consider following points while preparing your answer.

- Initial starting cost of the project according to group A and B point of view
- Extendable design and maintenance cost according to group A and B point of view
- Feature addition cost according to group A and B point of view
- Reduce complexity

Solution

In above scenario, group B is on right direction because Techno should give due time to the requirement engineering process and, design and architecture.

Initially, the cost of group A scenario is less then group B, because a software project required good amount of time as well as human effort to gather requirements, analyze these requirements then prepare design document. Software designer prepare design considering different angles like, requirements may change in the future so design should welcome the change in design and system, SIMCO may demand more features to add in future so design should be extendable. Software may require maintenance so software should be easy to understand to maintain.

While if a company do not give good amount of time towards documentation then it is true that company may save good time and money but later SIMCO's budget will increase a lot to maintain and adding more features.

If a project has zero or less documentation then it will be hard to understand the system so if a project is hard to understand then it is true that it will be hard to extend, maintain and hard to add more features. Because hard to understand means it is complex so a good documentation always reduce complexity, welcome changes and maintenance but required spending good amount of time and money upfront. So It is up to the SIMCO, either SIMCO pay now or pay later.

Lab Manual- 02

Topic: Identification of objects and actors from a case study

Case Study:

The hospital's manager inputs the patient's name, date of birth, address and emergency contact into the system along with the persons' health insurance numbers. A patient may have multiple health insurances (from different companies). All health insurance numbers are entered along with details of each company. The system then verifies the insurance credentials from the corresponding company. Upon successful validation, the patient is registered with the hospital.

Question

You are required to find the objects and the actors in the above case study.

Solution:

Actors:

Manager, Insurance Company

Objects:

Patient

Manager

Insurance company

Registration

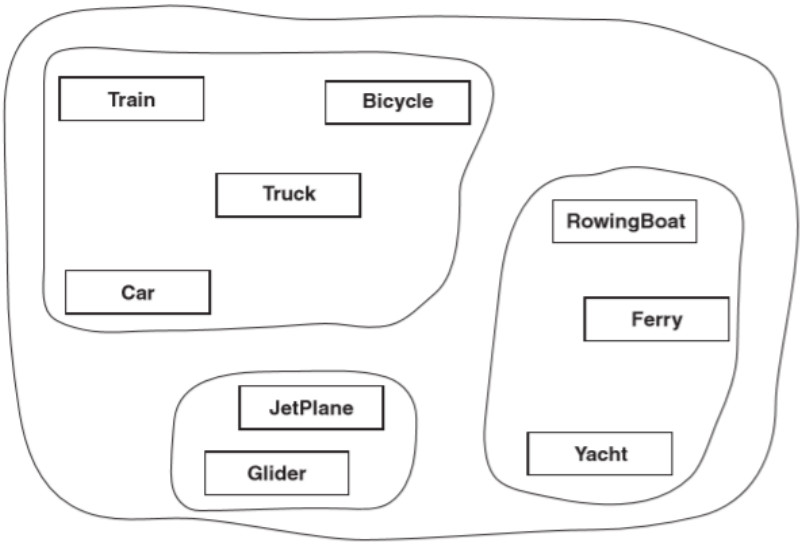
Lab Manual- 03

Topic: Class classification and relationships

Problem Statement

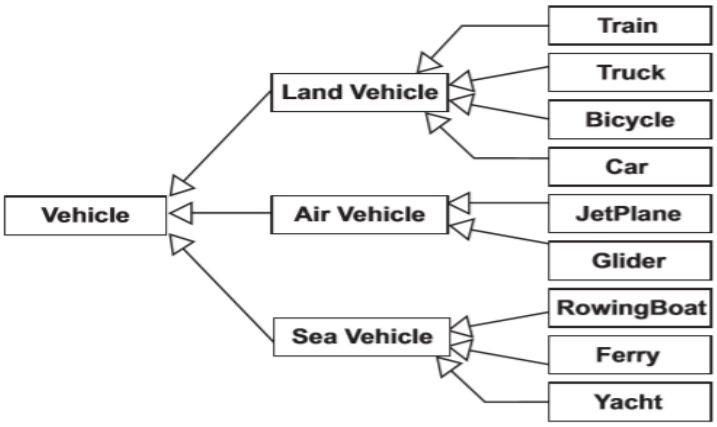
You are familiar with the Jet-Plane, Bicycle, Rowing-Boat, Train, Glider, Car, Ferry, Yacht and truck. You are required classify these classes into relevant groups and create class hierarchy.

Solution:



Classification

Relationship hierarchy:



Lab Manual- 04

Lab Title: Write a C++ program to implement single inheritance.

Objectives: Learn how to develop single inheritance in the C++ programming language.

Tool: C++

Single Inheritance Implementation in C++:

Steps:

1. Create 2 classes, one for the teacher and one for the student.
2. Using the get data () function, enter full name and id number values in a teacher class, then display the data using the put data () function.
3. Using the get data () and put data () functions, enter and show the marks mk1, mk2, mk3 and mk4 within the student class.
4. Within the student class, extend the teacher class by utilising data and functions.

Program:

```
#include<iostream.h>
class teacher
{
private:
char name[30];
long number;
public:
void getdata()
{
cout<<"\n \t Provide Full name:";
cin>>full name;
cout<<"\n \t Provide id number:";
cin>>id number;
}
void putdata()
{
```

```
cout<<"\n \t full name:"<<Full name;
cout<<"\n \t id number : "<<id number;
}
};
class student : public teacher
{
private:
int mk1,mk2,mk3,mk4;
public:
void getdata()
{
teacher :: getdata();
cout<<"\n \t Provide four subjects marks:";
cin>>mk1>>mk2>>mk3>>mk4;
}
void putdata()
{
        teacher :: putdata();
        cout<<"\n \t Four subjects marks:"<<mk1<<mk2<<mk3>>mk4;
        }

        };
        void main()
        {
        student sb1,sb2,sb3,sb4;
        cout<<"\n Provide student1 information :\n":
        sb1.getdata();
        cout<<"\n Provide student2 information :\n";l
        sb2.getdata();
        cout<<"\n Provide student3 information :\n";2
        sb3.getdata();
        cout<<"\n student 1 information:";
        sb1.putdata();
        cout<<"\n student 2 information:";
        sb2.putdata();
```

```
cout<<"\n student 3 information:";
sb3.putdata();

}
```

Output:

```
Provide student1 information
Provide full name: Haroon Ali
Provide id number:4
Provide 4 subject marks:80 75 95 88
Provide student 2 information
Provide full name: Ayesha Ali
Provide id number:6
Provide 4 subjects marks :75 95 65 77
Provide student 3 information
Provide full name: Ehsan Ali
Provide id number:8
Provide 4 subjects marks:55 98 84 90


Student 1 information
Full Name: Haroon Ali
Id Number:4
Four subject Marks: 80 75 95 88
Student 2 information
Full Name: Ayesha Ali
Id Number :6
Four subject Marks: 75 95 65 77
Student 3 information
Full Name: Ehsan Ali
Id Number :8
Four subject Marks: 55 98 84 90
```

Lab Manual- 05

Topic: SOLID Design Principles-Open-Closed Principle (OCP) - Example

You have to write a program which can calculate area of triangle. Your developed program was working fine but After few days a requirement arise and you want to calculate the area of rectangle also and for this purpose you have to modify the program. The modification of the program leads to the violation of the OCP principle. Now you have to write program in a way that follow the OCP principle. OCP mean open for extension and close for modification.

You have to write two types of programs:

a. First write pseudo code/program without using OCP principle

b. Second write program using OCP principle.

Solution:

Pseudo code/Program without using OCP principle

```
public class Triangle{
 public double width;
 public double height;
}


public class AreaCalculator{
  public double calculateTriangleArea(Triangle triangle){
    return (triangle.width* triangle.height)/2;
  }
}
```

If we want to calculate the area of Rectangle then the rectangle class will be like this:

```
public class Rectangle{
 public double length;
 public double width;
}
```

Then we modify **AreaCalculator** class to add rectangle calculations through a new method **calculateRectangleArea()** :

```
public class AreaCalculator{

 public double calculateTriangleArea(Triangle triangle){
   return (triangle.width* triangle.height)/2;
 }
 public double calculateRectangleArea(Rectangle rectangle){
   return rectangle.length *rectangle.width;
  }
}
```

If we want to add any other shape then again we have to modify AreaCalculator class which will make it more messy and difficult to maintain if we add more shapes.

Program using OCP principle

But If we use Open/Close principle then this problem can be solved. Following is the code that implements open close principle. In this program if area of new shape required we simply add class of that shape without modifying the existing class. Thus this is open for extension and closed for modification.

```
#include <iostream>

using namespace std;


class Shape

{

  protected:

    double width, height;

  public:

    void set_data (double a, double b)  //setter function

    {
```

```
        width = a;
        height = b;
    }
    virtual double area() = 0; //pure virtual function
};

class Rectangle: public Shape
{
public:
    double area ()
    {
        return (width * height);       //rectangle area calculation
    }
};

class Triangle: public Shape
{
public:
    double area ()
    {
        return (width * height)/2;  //triangle area calculation
    }
};
```

```
int main ()
{
   Shape *sPtr;

   Rectangle rectangle;
   sPtr = &rectangle;

   sPtr -> set_data (6,3);
   cout << "Rectangle' Area is " << sPtr -> area() << endl;   //output rectangle area

   Triangle triangle;
   sPtr = &triangle;

   sPtr -> set_data (5,6);
   cout << "Triangle's Area is " << sPtr -> area() << endl;       //output triangle area
   return 0;
}
```

Lab Manual- 06

Topic: Design Pattern

Tool/Language: (any) Java IDE/Java

Case Study:

Consider a scenario of a bank where we calculate monthly interest on different types of bank accounts. Initially, we only have two types of accounts:

- Current Account
- Savings Account

Current account with an interest rate of 2% per year and savings account with 4% per year. Our account types are defined by an enum.

```
enum AccountTypes {CURRENT, SAVINGS}
```

Here is our initial code for interest calculator

```
enum AccountTypes { CURRENT, SAVINGS}
public class InterestCalculator {

    public double calculateInterest(AccountTypes accountType, double Balance) {
        switch (accountType) {
            //Monthly interest rate is annual rate / 12 months.
            case CURRENT: return Balance * (0.02 / 12);
            case SAVINGS: return Balance * (0.04 / 12);
            default:
                return 0;
        }
    }
    public static void main(String[] args)
    {
        InterestCalculator i = new InterestCalculator();
        System.out.println(i.calculateInterest(AccountTypes.SAVINGS, 5000));

    }
}

```

Now, our next requirement is to add support for two other accounts.

- Freedom Account
- Munafa Account

The Freedom Account paying 6% per year, and Munafa Account that pays 7.5% per year but only if customer's minimum balance does not fall below PKR 100,000. We will modify our method accordingly

Here comes the complete code.

```
enum AccountTypes {CURRENT, SAVINGS, FREEDOM, MUNAFA}
public class InterestCalculator {

    public double calculateInterest(AccountTypes accountType, double Balance) {
        switch (accountType) {

            case CURRENT:
                //Monthly interest rate is annual rate / 12 months.
                return Balance * (0.02 / 12);

            case SAVINGS:
                return Balance * (0.04 / 12);

            case FREEDOM:
                return Balance * (0.06/12);

            case MUNAFA:
                if (Balance <100000) {
                    return 0;
                }
                else {
                    return Balance * (0.075/12);
                }

            default:
                return 0;
        }

    }
    public static void main(String[] args)
    {
        InterestCalculator i = new InterestCalculator();
        System.out.println("Inyterest Calculator!");
        System.out.println(i.calculateInterest(AccountTypes.MUNAFA, 100000));
    }
}
```

The finance team is planning to utilize our interest calculator for some other account like to calculate the interest on consumer loans. However, their interest rates are not fixed, they are linked to the state bank interest rate which we have to get through a web service. Not only are we dealing with more and more types of accounts, the calculation logic is also becoming more complex.

Now, the problem is, with every new requirement our code is going to deteriorate. And this will create difficult for us to understand the code. A single method is trying to handle multiple scenarios and code is getting complex, we need to address this issue.

Which pattern can help us to address this issue? Name the pattern and perform the necessary steps using java.

Solution:

Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Guidelines for the design pattern

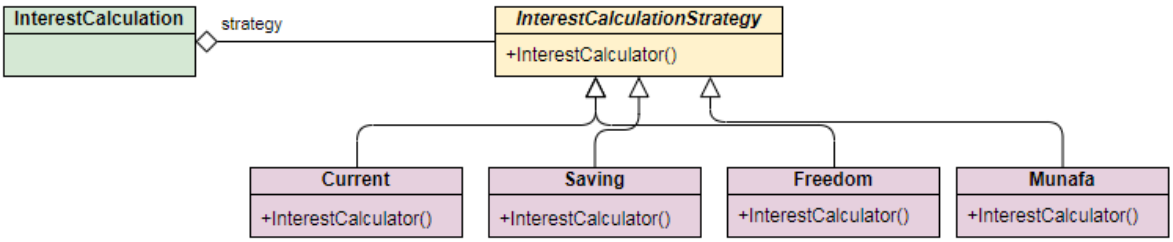
1. Figure out the areas in your code which are going to change and separate them.
2. Always program to an interface

In our scenario, according to the guidelines, we have successfully identified that we are changing the logic for calculating interest based on the account's types we are working with.

Now, considering the second guideline, we will define an interface to handle the input and output of our account balance and interest calculations.

```
1
2   /*Interface methods are implicitly public,
3   there is no need to write access modifier */
4
5   interface IntCalStrategy {
6
7       double calculateInterest(double accountBalance);
8
9   }
```

Note that our interface only deals with account balances - it doesn't care about account type, as each action will already be specific to a specific account type. Our next step is to develop a strategy to deal with each of our calculations.


InterestCalculation
strategy
InterestCalculationStrategy
+InterestCalculator()
Current
+InterestCalculator()
Saving
+InterestCalculator()
Freedom
+InterestCalculator()
Munafa
+InterestCalculator()

```
1
2       /*Interface methods are implicitly public,
3       there is no need to write access modifier */
4
5       interface IntCalStrategy {
6
7           double calculateInterest(double accountBalance);
8
9       }
10
11      //Current Account Interest Calculation Strategy
12      class CurrentAccIntCal implements IntCalStrategy {
13
14          @Override
15          public double calculateInterest(double accountBalance)
16          {
17              return accountBalance * (0.02 / 12);
18          }
19      }
20
21      //Savings Account Interest Calculation Strategy
22
23      class SavingsAccIntCal implements IntCalStrategy {
24
25          @Override
26          public double calculateInterest(double accountBalance) {
27              return accountBalance * (0.04 / 12);
28          }
29      }
30
31      //Freedom Account Interest Calculation Strategy
32
33      class FreedomAccIntCal implements IntCalStrategy {
34
35          @Override
36          public double calculateInterest(double accountBalance) {
37              return accountBalance * (0.06/12);
38          }
39      }
40
41      //Munafa Account Interest Calculation Strategy
42      class MunafaAccIntCal implements IntCalStrategy {
43
44          @Override
45          public double calculateInterest(double accountBalance) {
46              return accountBalance < 100000.00 ? 0 : accountBalance * (0.075/12);
47          }
48      }
49
```

Each calculation is now isolated to its own class, making the individual calculations much easier to understand - they are no longer surrounded with clutter. Next, we'll refactor our calculator.

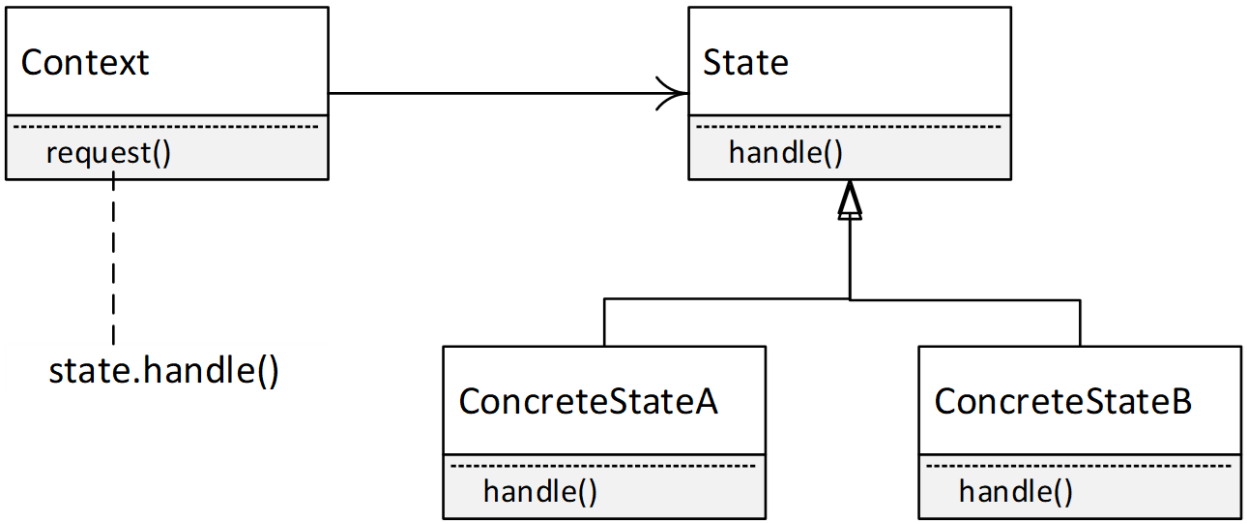
```java
public class InterestCalculator {

    //Strategies for calculating interest.
    private final IntCalStrategy CurrentStrategy = new CurrentAccIntCal();
    private final IntCalStrategy SavingsStrategy = new SavingsAccIntCal();
    private final IntCalStrategy FreedomStrategy = new FreedomAccIntCal();
    private final IntCalStrategy MunafaStrategy = new MunafaAccIntCal();


    public double calculateInterest(AccountTypes accountType, double Balance) {
        switch (accountType) {
            case CURRENT: return CurrentStrategy.calculateInterest(Balance);
            case SAVINGS: return SavingsStrategy.calculateInterest(Balance);
            case FREEDOM: return FreedomStrategy.calculateInterest(Balance);
            case MUNAFA: return MunafaStrategy.calculateInterest(Balance);
            default:
                return 0;
        }

    }
    public static void main(String[] args){

        InterestCalculator i = new InterestCalculator();
        System.out.println("Interest Calculator");
        System.out.println(i.calculateInterest(AccountTypes.MUNAFA, 1000));

    }
}

```

That’s it. We've moved the calculation logic out of the calculator itself.

Lab Manual- 07

Topic: State Design Pattern – An Example

As we studied in lectures, the State Pattern allows an object to change its behavior when it's internal state changes. The object will appear to change its class.



- **Context:** It shows an interface to the client for interaction. It maintains references to concrete state object which may be used to define current state of object.
- **State:** It defines interface for declaring what each concrete state should do.
- **ConcreteState:** It provides execution for functions defined in State.

Scenario:

Let's take an example of a mobile mode state; you are required to write a java code for a mobile mode. As we know that there are different alert modes in a mobile phone i.e., Sound, Vibration and Mute. Based on this alert mode, when an alert is to be done, behavior of the mobile changes.

The output should be as follows:

Mobile is in Sound Mode…

Mobile is in Sound Mode…

Mobile is in Vibration Mode…

Mobile is in Mute Mode…

Mobile is in Mute Mode…

Solution:

```
interface Mobile_Alert_Mode
{
	public void alert(Alert_Mode_Context amc);
}

class Alert_Mode_Context
{
	private Mobile_Alert_Mode presentMode;

	public Alert_Mode_Context()
	{
		presentMode = new Sound();
	}

	public void setMode(Mobile_Alert_Mode mode)
	{
		presentMode = mode;
	}

	public void alert()
	{
		presentMode.alert(this);
	}
}

class Sound implements Mobile_Alert_Mode
{
	@Override
	public void alert(Alert_Mode_Context amc)
```

```
	{
		System.out.println("Mobile is in Sound Mode...");
	}

}

class Vibration implements Mobile_Alert_Mode
{
	@Override
	public void alert(Alert_Mode_Context amc)
	{
		System.out.println("Mobile is in Vibration Mode...");
	}

}

class Mute implements Mobile_Alert_Mode
{
	@Override
	public void alert(Alert_Mode_Context amc)
	{
		System.out.println("Mobile is in Mute Mode...");
	}

}

public class Main
{
	public static void main(String[] args)
	{
```

```
        Alert_Mode_Context mobile_mode = new Alert_Mode_Context();
        mobile_mode.alert();
        mobile_mode.alert();
        mobile_mode.setMode(new Vibration());
        mobile_mode.alert();
        mobile_mode.setMode(new Mute());
        mobile_mode.alert();
        mobile_mode.alert();
    }
}
```

Output:

Mobile is in Sound Mode…

Mobile is in Sound Mode…

Mobile is in Vibration Mode…

Mobile is in Mute Mode…

Mobile is in Mute Mode…

Lab Manual- 08

Topic: Learn to overcome the issue of 'Bad Code smell' through Refactoring with Extract Method Technique

Identifying areas of bad design in a code and knowing when to apply code refactoring is a challenging task for the programmer/code designer. These areas of bad design are known as "Bad smells" or "Stinks" within a code. Code smell indicates that the written code is not well structured and planned due to which the software performance can be degraded. Refactoring is a process in which the overall code design should be re-examined/re-visited such that the internal structure of source code is improved while preserving its external behavior.

The Extract Method is one of the most commonly used technique for refactoring code. With this method, when we examine a piece of code, we can find that there is too much going on in one place in our code, and we further see that there are one or more "chunks" of the code that can be re-shaped in separate methods.

Give Your Thought:

You are provided with a sample programmed code compiled in Java language. The task is to apply Extract Method type of refactoring over the given code with the aim of improving Maintainability, Readability, Reusability etc. quality factors of the given code.

Sample Code Fragment:

```
import java.util.Scanner;
public class Main
{
   public static double BasicPay, NetSalary, IncomeTax,GPFund;
   public static int grade;
   public static void main(String[] args)
   {
      // Scanner used for taking input from user via keyboard
     Scanner kbd=new Scanner(System.in);
     System.out.println("Enter Basic pay");
     BasicPay=kbd.nextDouble();
     System.out.println("Enter Employee Grade");
     grade=kbd.nextInt();
     // Calculating tax on the inputted salary
     if(BasicPay<30000.0)
       System.out.println("Salary in not taxable");
      else if(BasicPay>=30000.0 && BasicPay<=50000.0)
        IncomeTax=(double)5/100*(BasicPay-30000);
      else
        IncomeTax=(double)7/100*(BasicPay-50000);
```

```
// Code fragment for calculating GPFund of the inputted salary
      if(grade<=16)
        GPFund=(double)7/100*BasicPay;
      else
        GPFund=(double)10/100*BasicPay;
// Code fragment for Displaying Output values
      System.out.println("Income Tax =" +IncomeTax);
      System.out.println("GP Fund Subscription =" +GPFund);
      NetSalary=BasicPay-(IncomeTax+GPFund);
      System.out.println("Net Salary of Employee = " +NetSalary);
   }
}
```

Solution:

Motivation for applying Extract Method Technique:

- The more lines found in a method, the harder it is to figure out what the method does.
- Less readability of the code.
- It is very difficult to be reused in other places in your program.
- Independent parts of the code are merged together so errors are more likely to occur in this code.

Steps for applying Extract Method Technique

Step 1: Identify the code to be extracted:

The first step is to identify independent parts of the code which can be regrouped into a single method performing its intended task. For example if we look at the following code it can easily be put into a single method named "**public void TaxCalculation()**" as the name suggests its intended functionality.

```
if(BasicPay<=30000.0)
   System.out.println("Salary in not taxable");
 else if(BasicPay>30000.0 && BasicPay<=50000.0)
   IncomeTax=(double)5/100*(BasicPay-30000);
 else
   IncomeTax=(double)7/100*(BasicPay-50000);
 System.out.println("Income Tax =" +IncomeTax);
```

Step 2: Create empty method and copy code:

After identification step the next step is to construct empty methods for each independent fragment of code. From the above program we have identified the following four methods/functions where the relevant code will be copied as shown in step 4.

1. ReadData()
2. TaxCalculation()
3. GPFundCalculation()
4. NetSalaryCalc()

Step 3: Call new method from original method:

In this step each of the above four methods will be called through an object in the main method as shown below.

```
public class Main
{
   public double BasicPay, NetSalary, IncomeTax=0,GPFund=0;
   public int grade;
   Scanner kbd=new Scanner(System.in);
   public static void main(String[] args) {

      Main obj=new Main();
      obj.ReadData();
      obj.TaxCalculation();
      obj.GPFundCalculation();
      obj.NetSalaryCalc();
   }
```

Step 4: Full Java code after applying Extract Method technique:

This java code is giving the same output as the given code (No change in its external behaviour) but with an improved software quality which can easily be maintained, read and reused by the developers.

```
import java.util.Scanner;
public class Main
{
   public double BasicPay, NetSalary, IncomeTax=0,GPFund=0;
   public int grade;
   Scanner kbd=new Scanner(System.in);
   public static void main(String[] args) {

      Main obj=new Main();
      obj.ReadData();
      obj.TaxCalculation();
```

```
        obj.GPFundCalculation();
        obj.NetSalaryCalc();
    }

    // 1. Extracted method for reading data
    public void ReadData()
    {
      System.out.println("Enter Basic pay");
      BasicPay=kbd.nextDouble();
      System.out.println("Enter Employee Grade");
      grade=kbd.nextInt();
    }

// 2. Extracted method for calculating tax on the inputted salary
    public void TaxCalculation()
    {
      if(BasicPay<=30000.0)
         System.out.println("Salary in not taxable");
       else if(BasicPay>30000.0 && BasicPay<=50000.0)
          IncomeTax=(double)5/100*(BasicPay-30000);
         // System.out.println("IncomeTax value = " + IncomeTax);
       else
          IncomeTax=(double)7/100*(BasicPay-50000);
       System.out.println("Income Tax =" +IncomeTax);
    }
// 3. Extracted method for calculating GPFund for the given grade
    public void GPFundCalculation()
    {
     if(grade<=16)
          GPFund=(double)7/100*BasicPay;
     else

          GPFund=(double)10/100*BasicPay;
    System.out.println("GP fund Subscription is " +GPFund);
    }
// 4. Extracted method for calculating net salary of the employee
    void NetSalaryCalc()
    {
      NetSalary=BasicPay-(IncomeTax+GPFund);
      System.out.println("Net Salary of Employee = " +NetSalary);
     }
}
```

Step 5: Output Returned from the Program:

This is the same output result obtained from the code provided at **1.** "**Sample Code Fragment**" portion and **2. "Step 4"** of the solution part.

```
Enter Basic pay
60000
Enter Employee Grade
17
Income Tax =700.0000000000001
GP Fund Subscription =6000.0
Net Salary of Employee = 53300.0
```

Lab Manual- 09

Topic: MVC (Model View Controller) Pattern Explained by implementing of a simple Java Program.

Model View Controller (MVC) Pattern in Java splits the given application into three interconnected parts which are (model, view and controller) in order to separate internal representations of information from the ways that information is presented to and accepted from the user.

In MVC Pattern the user is not allowed to directly make connection with the data sources, but he/she has to interact with the top-level layers to access the data. The user can interact with the view and with the controller, but not with the model.

MVC pattern has three layers, which are:

- Model – Shows the business layer of the application.
- View – Defines the presentation of the application.
- Controller – Manages the flow of the application.

Solution:

Procedure:

In order to implement MVC design pattern, we will create the following three classes.

- **StudentModel Class**, which contains only the pure application data, it contains no logic describing how to present the data to a user.
- **StudentView Class**, which defines the presentation of data. It knows how to access the model's data.
- **Controller Class**, which acts as an interface between the Model and the View parts and it intercepts all the incoming requests.

We will explore these classes one by one and finally merge them into a single program and run it to verify its outputted result.

Steps for implementing MVC (Model View Controller) Pattern

Step 1: Implementation of the "StudentModel" Class:

The **StudentModel** is the data layer which defines the business logic of the system and also represents the state of the application. Through this layer, we apply rules to data, which eventually represents the concepts our application manages. Now, let's create a model using Course Class.

```
class StudentModel {

  private int id;
  private String name;
  private String department;

  public int getId() {
     return id;
  }
  public String getName() {
     return name;
  }
  public String getDepartment() {
     return department;
  }
  public void setId(int id) {
     this.id = id;
  }
  public void setName(String name) {
     this.name = name;
  }
  public void setDepartment(String department) {
    this.department = department;
  }
}
```

Step 2: Implementation of the "StudentView" Class:

View represents the visualization of the data that model contains. Let's create a view using CourseView Class.

```
class StudentView {

    public void printStudentInformation(StudentModel std) {

        System.out.println("---------------------------------------------------------- ");

        System.out.println("Student's Record:");

      System.out.println("Student ID= "+std.getId()+" \nStudent Name= "+std.getName()+"

      \nDepartment = "+std.getDepartment());

      System.out.println("---------------------------------------------------------- ");

    }

}
```

Step 3: Implementation of the "Controller" Class:

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separated from each other. The code for the controller class is given as:

```
class Controller {
   private StudentModel studentModel;
   private StudentView studentView;

   public Controller(StudentModel studentModel, StudentView studentView) {
      super();
      this.studentModel = studentModel;
      this.studentView = studentView;
   }
   public void setStudentId (int id) {
      studentModel.setId(id);
   }
   public int getStudentId () {
      return studentModel.getId();
   }
   public void setStudentName (String name) {
      studentModel.setName(name);
   }
   public String getStudentName () {
      return studentModel.getName();
   }
   public void setStudentDepartment (String department) {
      studentModel.setDepartment(department);
   }
   public String getStudentDepartment() {
      return studentModel.getDepartment();
   }
   public void updateView(){
      studentView.printStudentInformation(studentModel);
   }
}
```

Step 4: Implementation of the "main method" in Main Class:

In this portion we will handle all the classes discussed above and to lead us to a proper output after fetching student's record from the user through keyboard. Then we will update/modify the same record with the help of the controller class as shown in Step 5.

Step 5: Full Program:

```
// Code obtained from Step 1
```

```
class StudentModel {

  private int id;
  private String name;
  private String department;

  public int getId() {
     return id;
  }
  public String getName() {
     return name;
  }
  public String getDepartment() {
     return department;
  }
  public void setId(int id) {
     this.id = id;
  }
  public void setName(String name) {
     this.name = name;
  }
  public void setDepartment(String department) {
    this.department = department;
  }
}

//2 Code obtained from Step 2
class StudentView {

    public void printStudentInformation(StudentModel std) {

        System.out.println("--------------------------------------------------- ");

        System.out.println("Student's Record:");

       System.out.println("Student ID= "+std.getId()+" \nStudent Name= "+std.getName()+"

       \nDepartment = "+std.getDepartment());

      System.out.println("--------------------------------------------------- "); }

}

//3. Code obtained from Step 3

class Controller {

   private StudentModel studentModel;
   private StudentView studentView;
```

```
   public Controller(StudentModel studentModel, StudentView studentView) {
      super();
      this.studentModel = studentModel;
      this.studentView = studentView;
   }
   public void setStudentId (int id) {
      studentModel.setId(id);
   }
   public int getStudentId () {
      return studentModel.getId();
   }
   public void setStudentName (String name) {
      studentModel.setName(name);
   }
   public String getStudentName () {
      return studentModel.getName();
   }
   public void setStudentDepartment (String department) {
      studentModel.setDepartment(department);
   }
   public String getStudentDepartment() {
      return studentModel.getDepartment();
   }
   public void updateView(){
      studentView.printStudentInformation(studentModel);
   }
}
//4. Main class's implementation
public class Main {

    public static void main(String[] args) {

        System.out.println("Sample program for implementing MVC Architecture");

        StudentModel studentModel = new StudentModel(); // Object declaration

        StudentView studentView = new StudentView();

        Controller controller = new Controller(studentModel, studentView);  // used for input

        Scanner scan= new Scanner(System.in); //System.in is a standard input stream.

        System.out.println("Taking input from user via keyboard.");

        System.out.print("Enter Student ID: ");

        int stId=scan.nextInt();
```

```
        System.out.println("Enter Student Name:");
        String sname= scan.next();   //reads string from keyboard
        System.out.println("Enter Department Name:");
        String dept= scan.next();
        studentModel.setId(stId);
        studentModel.setName(sname);
        studentModel.setDepartment(dept);
        studentView.printStudentInformation(studentModel);    //Showing the user entered Record
        System.out.println("After updation of Data through Controller Class");
        //Updating data through controller class's object
       controller.setStudentDepartment("Mathematics");
        controller.updateView();    //Viewing the updated Student's Record through Controller
    }}
```

Step 5: Running and Verifying the Output of the Program:

After running the program the user is required to give input about the student record, then it is displayed as shown. Now the modification in the same student record is performed through the Controller class as the client cannot directly access the other classes.

```
Sample program for implementing MVC Architecture
Taking input from user via keyboard.
Enter Student ID:
12
Enter Student Name:
M.Jamal
Enter Department Name:
CS
-----------------------------------------------------
Student's Record:
Student ID= 12
Student Name= M.Jamal
Department = CS
-----------------------------------------------------
After updation of Data through Controller Class
-----------------------------------------------------
Student's Record:
Student ID= 12
Student Name= M.Jamal
Department = Mathematics
-----------------------------------------------------
```

Lab Manual- 10

Lab Title: Learn to implement Process View of (4+1 View Model of Software Architecture) by using interaction diagrams.

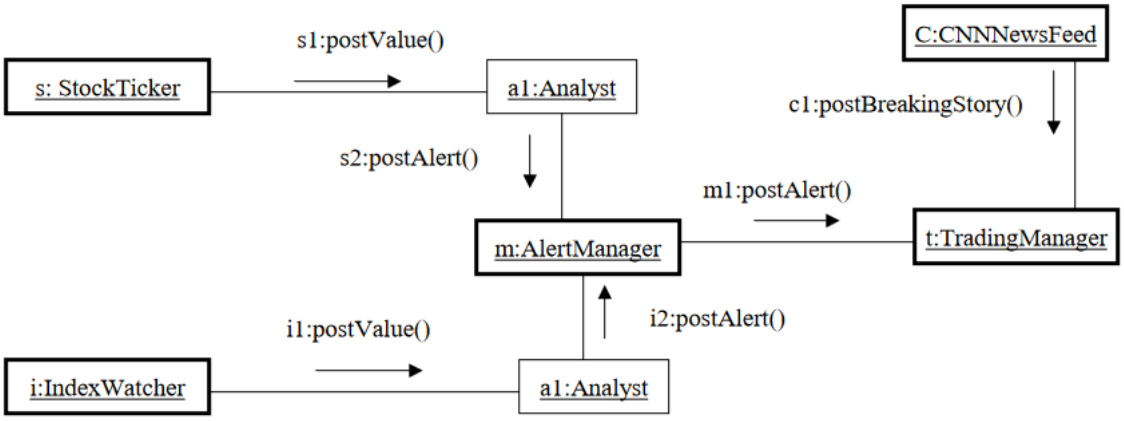
Objectives: Get the knowledge to capture "Process View" using interaction diagrams.

Tool: UML

Description: Interaction diagrams are useful for showing where two control flows could cross paths and, as a result, where coordination and synchronization issues need to be addressed.

Problem Statement: Suppose a trading system in which trading decision making are driven by data from 3 sources: a stock ticker, an index watcher, and a CNN News Feed. The data from the stock ticker and index watcher is first analysed before being forwarded to the trading manager via an alert manager. The CNN News Feed interacts with the Trading Manager directly. You are required to draw interaction diagram by using UML.

Solution:



Lab Manual- 11

Topic: Quality Attributes

Task:

Quality attribute (QA) is a measurable or testable property of a system that is used to indicate the extent to which the system meets the needs of its stakeholders. Considering the case study given below, identify the measurable properties of the given system, and mention the quality attribute against each identified property in the provided table.

CASE STUDY:

Software development process has a variety of artefacts like software specification document, design document, source files, test cases, test reports, etc. Each phase of software development produces it respective artefacts that are maintained separately. The management of these artefacts is a challenging task. We, CollabHub Inc., have been experiencing artefacts inconsistency issues for last 5 years.

After many meetings and brainstorming activities with stakeholders, we have decided to design a solution for our project managers to keep track of the software artefacts. We are already keeping track for out software artefacts using a good trace matrix. But it requires a lot of effort for maintenance.

The new system would be more like a hybrid project management tool that functions as a team collaboration platform as well as an artefact management system. The system will be specifically designed for software engineers and project managers, so it should have a conservative feel and appearance.

The system should allow the assignment of different roles and access permissions, but only the project manager should have the privilege of assigning roles to the project team. We need to make sure that artefacts links are accessible to the project team, and they are only accessing the links they are authorized to.

The system will also routinely handle the one-time, periodic, and ongoing project's artefacts with ease, including file indexing, inventory set up with retention schedules, and more. File indexing will be prevented by everyone except the project managers.

We need to make sure that it does not take more than 60 seconds for the system to update the file index. This can take a long time, but only if such processes are automated. In addition, the system should complete the request to display artefacts such as sequence diagrams, case diagrams or any other antiques within 5 seconds.

We need to make sure that the data does not leave the organization without encryption. As mentioned earlier, this system will be used by project teams, engineers and managers, so there should be proper FAQ and user guide in place. We have a strict deadline; we look forward to seeing this system by the end of this year.

Property	**Quality Attribute**
• • •	• • •

SOLUTION

Software development process has a variety of artefacts like software specification document, design document, source files, test cases, test reports, etc. Each phase of software development produces it respective artefacts that are maintained separately. The management of these artefacts is a challenging task. We, CollabHub Inc., have been experiencing artefacts inconsistency issues for last 5 years.

After many meetings and brainstorming activities with stakeholders, we have decided to design a solution for our project managers to keep track of the software artefacts. We are already keeping track for our software artefacts using a good trace matrix. But it requires a lot of effort for maintenance.

The new system would be more like a hybrid project management tool that functions as a team collaboration platform as well as an artefact management system. The system will be specifically designed for software engineers and project managers, so it should have a conservative feel and appearance. The system will be initially available for normal use from 06:00 to 23:59 hours Monday to Saturday and for system maintenance purposes from 00:00 to 02:00 hours once a week.

The system should allow the assignment of different roles and access permissions, but only the project manager should have the privilege of assigning roles to the project team. We need to make sure that artefacts links are accessible to the project team 99% of the time without failure, and they are only accessing the links they are authorized to.

The system will also routinely handle the one-time, periodic, and ongoing project's artefacts with ease, including file indexing, inventory set up with retention schedules, and more. File indexing will be prevented by everyone except the project managers.

We need to make sure that it does not take more than 60 seconds for the system to update the file indices. This can take a long time, but only if such processes are automated. In addition, the system should complete the request, to display artefacts such as sequence diagrams, use case diagrams or any other artefacts, within 5 seconds.

We need to make sure that the data does not leave the organization without encryption. As mentioned earlier, this system will be used by project teams, engineers, and managers, so there should be proper FAQ's and user guide in place. We have a strict deadline; we look forward to seeing this system by the end of this year.

Property	**Quality Attribute**
We are already keeping track for out software artefacts using a good trace matrix. But it requires a lot of effort for maintenance.	Maintainability
The system will be specifically designed for software engineers and project managers, so it should have a conservative feel and appearance.	Usability
The system will be initially available for normal use from 06:00 to 23:59 hours Monday to Saturday and for system maintenance purposes from 00:00 to 02:00 hours once a week.	Availability
The system should allow the assignment of different roles and access permissions, but only the project manager should have the privilege of assigning roles to the project team.	Security

We need to make sure that artefacts links are accessible to the project team 99% of the time without failure	Reliability
And they (team members) are only accessing the links they are authorized to.	Security
File indexing will be prevented by everyone except the project managers.	Security
We need to make sure that it does not take more than 60 seconds for the system to update the file indices.	Performance
In addition, the system should complete the request, to display artefacts such as sequence diagrams, use case diagrams or any other artefacts, within 5 seconds.	Performance
We need to make sure that the data does not leave the organization without encryption.	Security
As mentioned earlier, this system will be used by project teams, engineers, and managers, so there should be proper FAQ's and user guide in place.	Usability

Lab Manual- 12

Topic: Agility and Architecture Design

ABC Pvt. limited wants to develop an inventory system. They hired a Cyber Solution Pvt. Ltd. software development company for this development. First Cyber Solution Pvt. Ltd. have to develop system architecture based on initial requirements provided by the ABC Pvt. Ltd. company. After some time, ABC Pvt. Ltd. realize that Cyber Solution's staff do not giving proper attention towards design and architecture of the system. They just finalizing the architecture and just rushing for direct development and final output. The ABC Pvt. Ltd understands that requirements may change during development and the architecture may have to be updated. They want cyber solution Pvt. Ltd. to develop agile architecture which can later updated when any change in requirement occurs.

Do you think that it is a good approach to develop agile architecture which can be later updated?

How incremental and iterative approach to refining architecture is beneficial?

Solution:

a. Do you think that it is a good approach to develop agile architecture which can be later updated?

- Yes, Wherever possible, design your application so that it can change over time to address new requirements and challenges. Modern thinking on architecture assumes that your design will evolve over time and that you cannot know everything you need to know up front in order to fully architect your system. Your design will generally need to evolve during the implementation stages of the application as you learn more, and as you test the design against real-world requirements. Create your architecture with this evolution in mind so that it will be agile in terms of adapting to requirements that are not fully known at the start of the design process.

b. How incremental and iterative approach to refining architecture is beneficial?

- Consider using an incremental and iterative approach to refining your architecture. Do not try to get it all right the first time—design just as much as you can in order to start testing the design against requirements and assumptions. Iteratively add details to the design over multiple passes to make sure that you get the big decisions right first, and then focus on the details. A common pitfall is to dive into the details too quickly and get the big decisions wrong by making incorrect assumptions, or by failing to evaluate your

architecture effectively. This iterative and incremental approach allows you to get the big risks out of the way first, iteratively render your architecture, and use architectural tests to prove that each new baseline is an improvement over the last.

Lab Manual- 13

Topic: Architectural Patterns

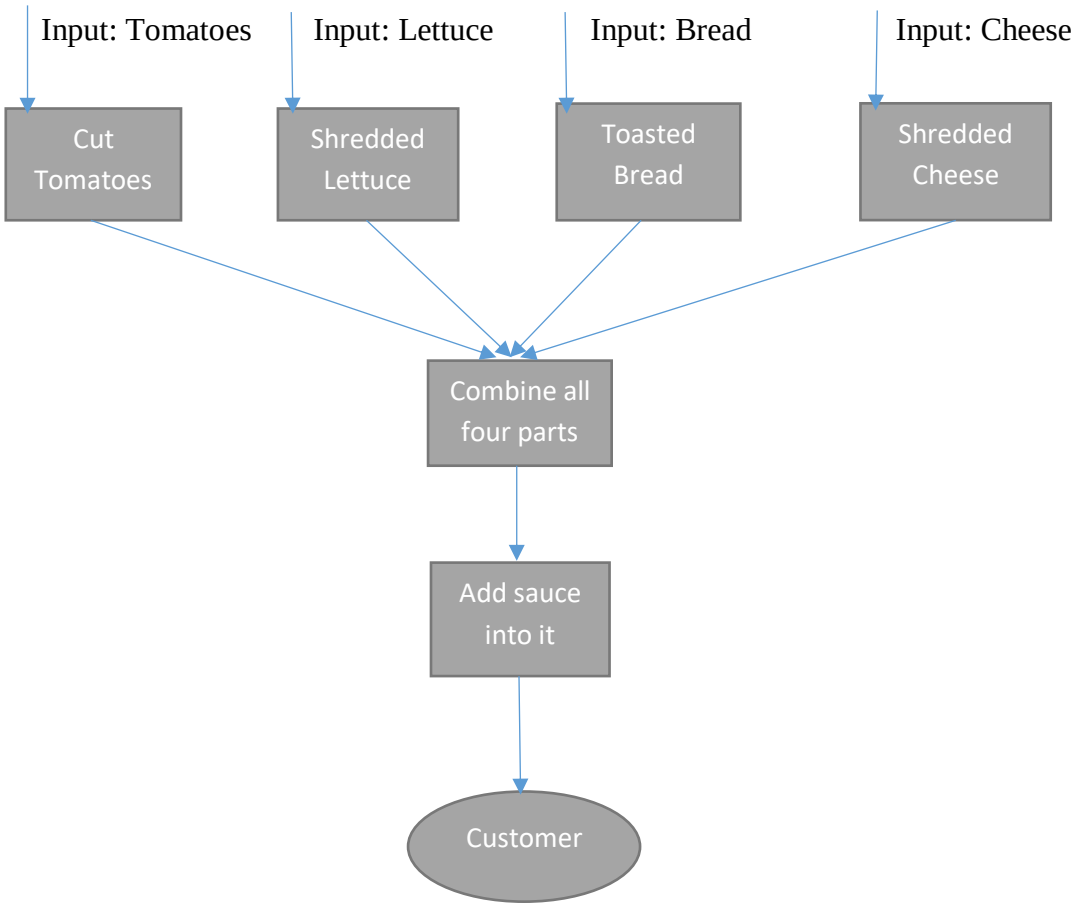


Figure 1

Give Your Thought:

You can see Figure 1 above. You are required to give the answers of the following question:

- Which architectural style is used in Figure 1?
- Explain the demonstration of the architectural style followed in Figure 1.
- Can we increase the size of architecture during implementation? If yes, then what will be its impact on the overall performance?
- What are the strengths and weaknesses of this architectural style?
- Give some examples related to this architectural style?

Solution:

1. Pipe and Filter
2. Figure 1 shows a simple and common method of making sandwiches. The first four filters can work concurrently for preparation of sandwiches in the beginning. Once they are done, the fifth filter can get the output and combine them together. Next, a following filter will add sauce in to it and pass it to customer through a pipe.
3. Yes. During implementation, we can easily increase the size of Pipe and Filter architecture by adding more pipes and filters in to it. However, its overall performance could be slow and buffer overflow could occur, when data and architecture size become very large.
4. Strengths and Weaknesses:
    - Typically, not a good choice for an interactive system, as it disallows cycles (which are important for user feedback).
    - having large numbers of independent filters can add substantial amounts of computational overhead, because each filter runs as its own thread or process.
    - May not be appropriate for long-running computations, without the addition of some form of checkpoint/restore functionality, as the failure of any filter (or pipe) can cause the entire pipeline to fail.
    - Pipes buffer data during communication. Because of this property, filters can execute asynchronously and concurrently.
    - A filter typically does not know the identity of its upstream or downstream filters. For this reason, pipeline pipe-and-filter systems have the property that the overall computation can be treated as the functional composition of the computations of the filters, making it easier for the architect to reason about end-to-end behavior.
    - Data transformation systems are typically structured as pipes and filters, with each filter responsible for one part of the overall transformation of the input data.
    - The independent processing at each step supports reuse, parallelization, and simplified reasoning about overall behavior.
5. Examples:
    - Systems built using UNIX pipes
    - The request processing architecture of the Apache web server
    - Yahoo! Pipes for processing RSS feeds
    - Many workflow engines
    - Many scientific computation systems that have to process and analyze large streams of captured data.

Lab Manual- 14

Topic: Monolithic Architecture – An Example

Scenario:

Suppose that you are developing an E-commerce Web application, in which orders are taken from customers, their available credit, and the required items are checked in the inventory and finally the orders are shipped to the customers. This application consists of various components including the Front-end, which implements the (GUI) user interface, along with some back-end services for checking the available credit, maintaining inventory and orders shipment.

Points to Consider:

The application :

- must be easy to understand and modify
- should be developed by a team of multiple developers
- should satisfy availability and scalability requirements
- must be developed using emerging technologies

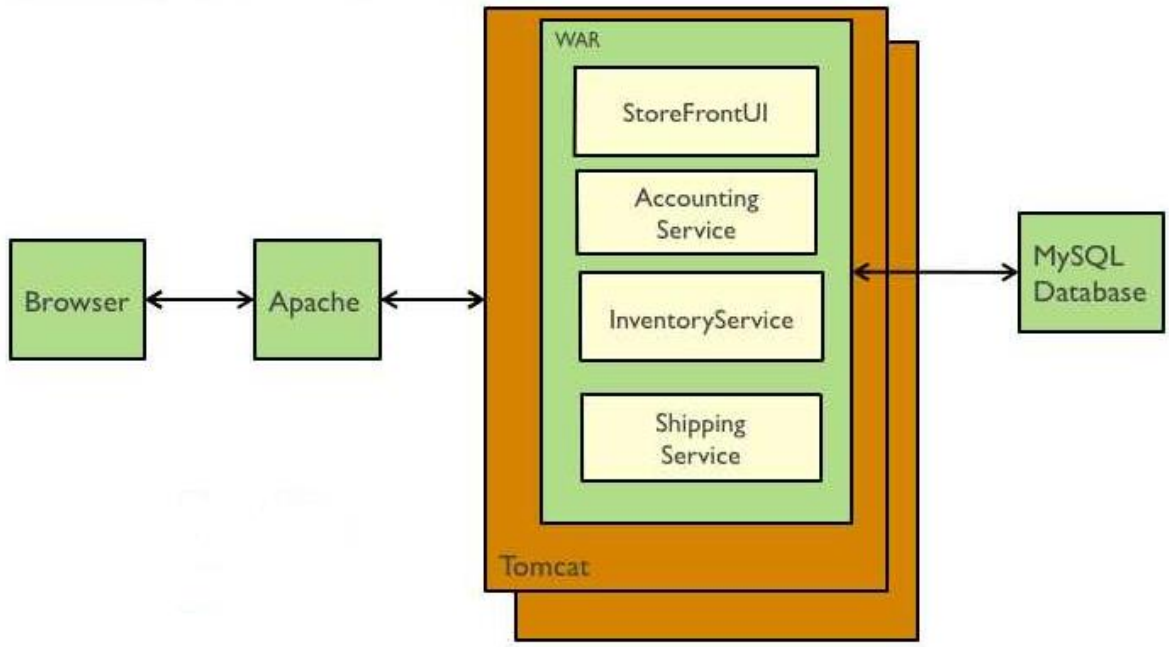
Problem:

What should be the application's deployment architecture?

Solution:

The application is deployed as a single monolithic application.

For example, a Java web application consists of a single WAR file that runs on a web container such as Tomcat. A Rails application consists of a single directory hierarchy deployed using either, for example, Phusion Passenger on Apache/Nginx or JRuby on Tomcat. You can run multiple instances of the application behind a load balancer in order to scale and improve availability.



Conclusion:

This solution has various benefits:

- Simple to develop
- Simple to deploy
- Simple to scale

Lab Manual- 15

Topic: Cloud Computing and Service-oriented Architecture – An Example

Scenario:

Suppose that you are developing an application that can be used for storage of user data/ files (e.g. Dropbox, Google Drive, and OneDrive etc.). It has the advantage of easy backup of user data. It will synchronize user files / data from selected drives, folders and the desktop automatically. This storage Application will allow users to access their files from anywhere (using the Internet) and a free storage of up to 10 GBs.

Your organization do not have their own physical infrastructure; they will hire these services from a third party service provider.

Problem:

What should be the application's deployment architecture?

Solution:

This application is deployed using Cloud Computing Architecture.

Some of the very popular cloud-computing service providers are:

- Microsoft Azure
- Oracle Cloud
- Google Cloud
- IBM Cloud
- Amazon Web Services
- Alibaba Cloud
- SAP

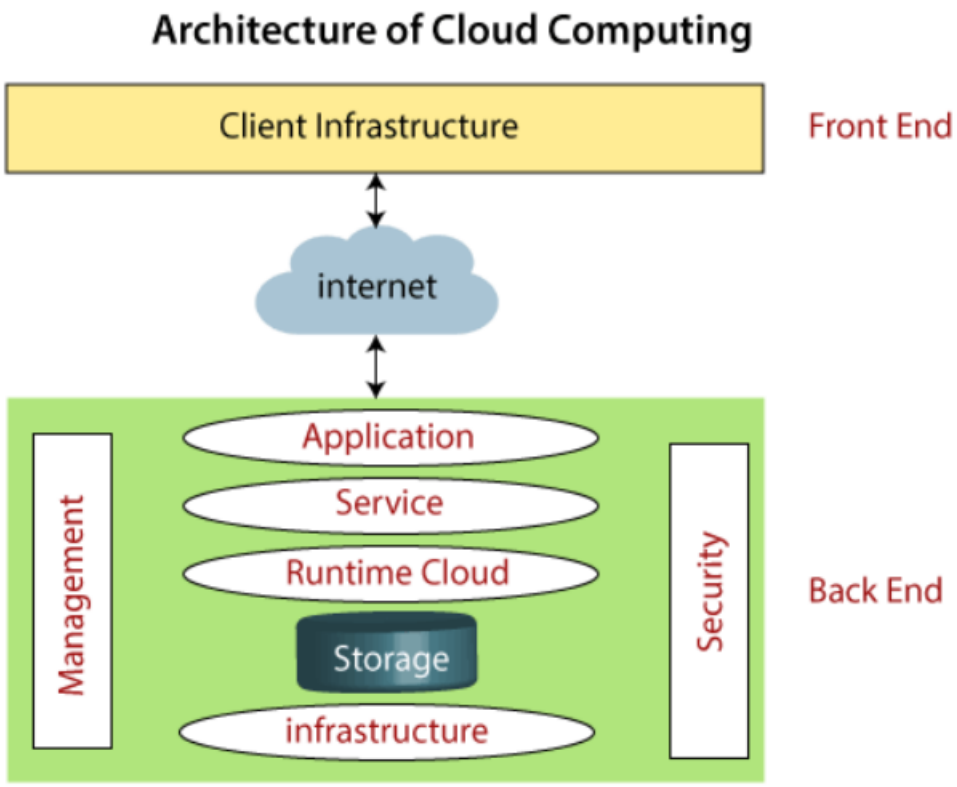
Cloud computing technology is used by both small as well as large organizations to store the information in cloud and access it from anywhere at any time using an Internet connection.

Cloud computing architecture is a combination of service-oriented architecture and event-driven architecture. Cloud computing architecture is divided into the following two parts -

- Front End

- Back End

The below diagram shows the architecture of cloud computing -



Conclusion:

This solution has a number of benefits including:

- Cost effective
- Flexible and Scalable
- Data Security
- Excellent accessibility
- Mobility

Lab Manual- 16

Topic: Micro-service Architecture – An Example

Scenario:

Suppose that you are developing an E-commerce Web application, in which orders are taken from customers, their available credit, and the required items are checked in the inventory and finally the orders are shipped to the customers. This application consists of various components including the Front-end, which implements the (GUI) user interface, along with some back-end services for checking the available credit, maintaining inventory and orders shipment.

Points to Consider:

The same application was designed (in our Lab manual-14) using the monolithic architecture. Despite its too many benefits, there are some drawbacks of monolithic architecture, because of the "all-in-one structure".

Difficulties with monolithic application, when it grows:

- Code base of large monolithic applications usually gets complicated to understand
- Scaling is always a challenging task
- Continuously integration / deployment become complex
- Increased development time due to large code base and slow / overloaded IDE
- Very difficult to change the underlying technology because everything is tightly coupled and dependant on each other.

Alternate Solution:

In order to address the issues faced during adopting monolithic architecture, we need to use / identify an architecture that structures this application as a set of loosely coupled, collaborating services which are:

- Highly maintainable and testable
- Loosely coupled with other services
- Independently deployable
- Capable of being developed by small teams

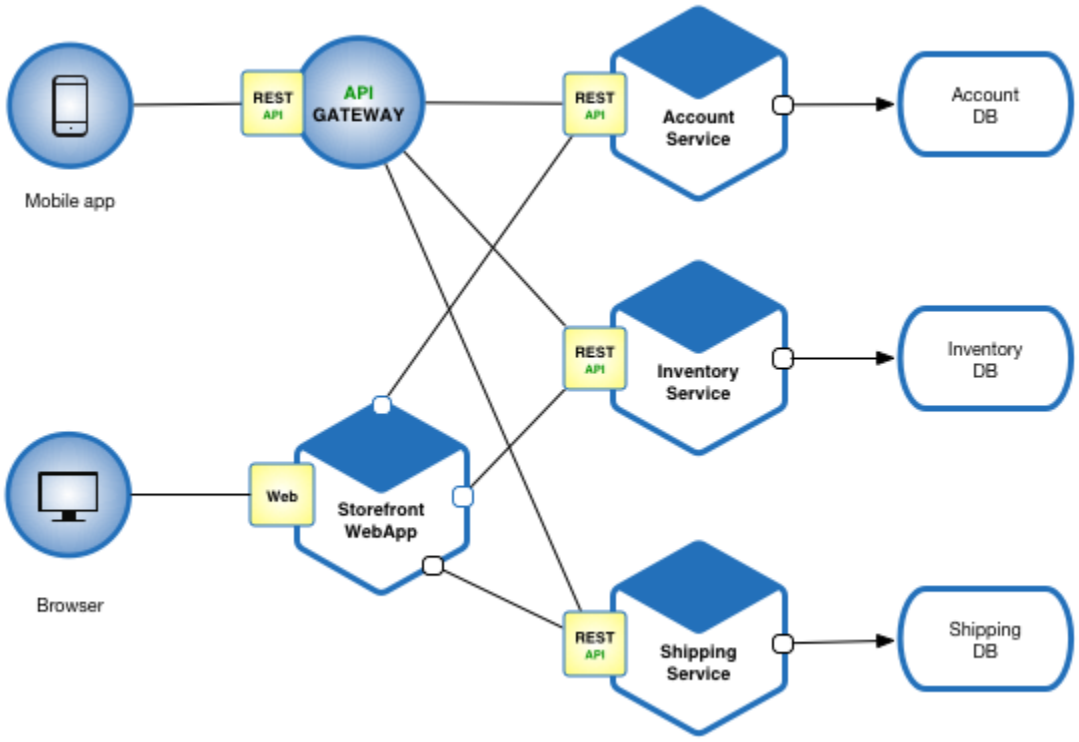
Problem:

What should be the application's deployment architecture?

Solution:

This application is deployed using Micro-service architecture.

For example, services communicate using either synchronous protocols such as HTTP/REST or asynchronous protocols such as AMQP. Services can be developed and deployed independently of one another. Each service has its own database in order to be decoupled from other services.



Conclusion:

This solution has a number of benefits:

- Improved maintainability - each service is relatively small and so is easier to understand and change
- Better testability - services are smaller and faster to test
- Better deployability - services can be deployed independently
- It enables you to organize the development effort around multiple, autonomous teams. Each team can develop, test, deploy and scale their services independently of all of the other teams.
- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests.
- Eliminates any long-term commitment to a technology stack.