**LAB 5, 6**

**Task1: Builder pattern**

Create Packing interface

**Packing.java**

public interface Packing {

            public String pack();

            public int price();

}

Create an abstract class CD which will implement Packing interface.

**CD.java**

public abstract class CD implements Packing{

public abstract String pack();

}

**Company.java**

public abstract class Company extends CD{

   public abstract int price();

}

Create 2 implementation classes of Company: Sony and Samsung

**File: Sony.java**

public class Sony extends Company{

    @Override

        public int price(){

                        return 20;

      }

    @Override

    public String pack(){

             return "Sony CD";

        }

}//End of the Sony class.

**Samsung.java**

public class Samsung extends Company {

    @Override

        public int price(){

                        return 15;

    }

    @Override

    public String pack(){

             return "Samsung CD";

        }

}//End of the Samsung class.

4) Create the CDType class

**CDType.java**

import java.util.ArrayList;

import java.util.List;

public class CDType {

             private List<Packing> items=new ArrayList<Packing>();

             public void addItem(Packing packs) {

                    items.add(packs);

             }

             public void getCost(){

              for (Packing packs : items) {

                        packs.price();

              }

             }

             public void showItems(){

              for (Packing packing : items){

             System.out.print("CD name : "+packing.pack());

             System.out.println(", Price : "+packing.price());

          }

            }

}//End of the CDType class.

5) Create the CDBuilder class

**CDBuilder.java**

public class CDBuilder {

                  public CDType buildSonyCD(){

                     CDType cds=new CDType();

                     cds.addItem(new Sony());

                     return cds;

              }

              public CDType buildSamsungCD(){

             CDType cds=new CDType();

             cds.addItem(new Samsung());

             return cds;

              }

}// End of the CDBuilder class.

6) Create the BuilderDemo class

**BuilderDemo.java**

public class BuilderDemo{

 public static void main(String args[]){

   CDBuilder cdBuilder=new CDBuilder();

   CDType cdType1=cdBuilder.buildSonyCD();

   cdType1.showItems();

   CDType cdType2=cdBuilder.buildSamsungCD();

   cdType2.showItems();

 }

}

------------------------------------------------------------------------------

**Task2:Adaptor Pattern**

|  |
| --- |
| public interface Movable {  // returns speed in MPH  double getSpeed();  } |

create one concrete implementation of interface

|  |  |
| --- | --- |
|  | public class BugattiVeyron implements Movable {  @Override  public double getSpeed() {  return 268;  }  } |

|  |  |
| --- | --- |
|  | public interface MovableAdapter {  // returns speed in KM/H  double getSpeed();  } |

|  |  |
| --- | --- |
|  | public class MovableAdapterImpl implements MovableAdapter {  private Movable luxuryCars;  // standard constructors  @Override  public double getSpeed() {  return convertMPHtoKMPH(luxuryCars.getSpeed());  }  private double convertMPHtoKMPH(double mph) {  return mph \* 1.60934;  }  } |

|  |  |
| --- | --- |
|  | @Test  public void whenConvertingMPHToKMPH\_thenSuccessfullyConverted() {  Movable bugattiVeyron = new BugattiVeyron();  MovableAdapter bugattiVeyronAdapter = new MovableAdapterImpl(bugattiVeyron);  assertEquals(bugattiVeyronAdapter.getSpeed(), 431.30312, 0.00001);  } |

------------------------------------------------------------------------------

**Task3: Façade Design Pattern**

Create a MobileShop interface.

MobileShop.java

public interface MobileShop {

    public void modelNo();

    public void price();

}

Create a Iphone implementation class that will implement Mobileshop interface.

**Iphone.java**

public class Iphone implements MobileShop {

    @Override

    public void modelNo() {

        System.out.println(" Iphone 6 ");

    }

    @Override

    public void price() {

    System.out.println(" Rs 65000.00 ");

    }

}

Create a Samsung implementation class that will implement Mobileshop interface.

**Samsung.java**

public class Samsung implements MobileShop {

    @Override

    public void modelNo() {

    System.out.println(" Samsung galaxy tab 3 ");

    }

    @Override

    public void price() {

        System.out.println(" Rs 45000.00 ");

    }

}

Create a Blackberry implementation class that will implement Mobileshop interface .

**Blackberry.java**

public class Blackberry implements MobileShop {

    @Override

    public void modelNo() {

    System.out.println(" Blackberry Z10 ");

    }

    @Override

    public void price() {

        System.out.println(" Rs 55000.00 ");

    }

}

Create a ShopKeeper concrete class that will use MobileShop interface.

**ShopKeeper.java**

public class ShopKeeper {

    private MobileShop iphone;

    private MobileShop samsung;

    private MobileShop blackberry;

    public ShopKeeper(){

        iphone= new Iphone();

        samsung=new Samsung();

        blackberry=new Blackberry();

    }

    public void iphoneSale(){

        iphone.modelNo();

        iphone.price();

    }

        public void samsungSale(){

        samsung.modelNo();

        samsung.price();

    }

   public void blackberrySale(){

    blackberry.modelNo();

    blackberry.price();

        }

}

Creating a client that can purchase the mobiles from MobileShop through ShopKeeper.

**FacadePatternClient.java**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class FacadePatternClient {

    private static int  choice;

    public static void main(String args[]) throws NumberFormatException, IOException{

        do{

            System.out.print("========= Mobile Shop ============ \n");

            System.out.print("            1. IPHONE.              \n");

            System.out.print("            2. SAMSUNG.              \n");

            System.out.print("            3. BLACKBERRY.            \n");

            System.out.print("            4. Exit.                     \n");

            System.out.print("Enter your choice: ");

            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

            choice=Integer.parseInt(br.readLine());

            ShopKeeper sk=new ShopKeeper();

            switch (choice) {

            case 1:

                {

                  sk.iphoneSale();

                    }

                break;

       case 2:

                {

                  sk.samsungSale();

                    }

                break;

       case 3:

                           {

                           sk.blackberrySale();

                           }

                   break;

            default:

            {

                System.out.println("Nothing You purchased");

            }

                return;

            }

      }while(choice!=4);

   }

}

**-------------------------------------------------------------------**

**Task4:Decorator Pattern**

Create an interface.

**Shape.java**

public interface Shape {

void draw();

}

Create concrete classes implementing the same interface.

Rectangle.java

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Shape: Rectangle");

}

}

**Circle.java**

public class Circle implements Shape {

@Override

public void draw() {

System.out.println("Shape: Circle");

}

}

Create abstract decorator class implementing the Shape interface.

**ShapeDecorator.java**

public abstract class ShapeDecorator implements Shape {

protected Shape decoratedShape;

public ShapeDecorator(Shape decoratedShape){

this.decoratedShape = decoratedShape;

}

public void draw(){

decoratedShape.draw();

}

}

Create concrete decorator class extending the ShapeDecorator class.

**RedShapeDecorator.java**

public class RedShapeDecorator extends ShapeDecorator {

public RedShapeDecorator(Shape decoratedShape) {

super(decoratedShape);

}

@Override

public void draw() {

decoratedShape.draw();

setRedBorder(decoratedShape);

}

private void setRedBorder(Shape decoratedShape){

System.out.println("Border Color: Red");

}

}

Use the RedShapeDecorator to decorate Shape objects.

**DecoratorPatternDemo.java**

public class DecoratorPatternDemo {

public static void main(String[] args) {

Shape circle = new Circle();

Shape redCircle = new RedShapeDecorator(new Circle());

Shape redRectangle = new RedShapeDecorator(new Rectangle());

System.out.println("Circle with normal border");

circle.draw();

System.out.println("\nCircle of red border");

redCircle.draw();

System.out.println("\nRectangle of red border");

redRectangle.draw();

}

}

----------------------------------------------------------------------