**Lab Assignment 5, 6**

**Adapter pattern**

@author fa16-bse-102

package AdapterPattern;

public class AdapterMain {

public static void main(String[] args) {

Moveable honda = new honda();

MoveableAdapter speedadapter = new MoveableAdapterImpl(honda);

System.out.println(speedadapter.speed());

}

}

**Honda class:**

package AdapterPattern;

public class Honda implements Moveable{

@Override

public double speed() {

return 285;

}

**}**

**Moveable:**

package AdapterPattern;

public interface Moveable{

public double speed();

}

**Moveable adopter:**

package AdapterPattern;

public interface MoveableAdapter{

public double speed();

}

**Moveable Adopter Imp:**

package AdapterPattern;

public class MoveableAdapterImpl implements MoveableAdapter{

public Moveable luxuryCars;

MoveableAdapterImpl(Moveable obj){

this.luxuryCars = obj;

}

@Override

public double speed() {

return convertmphtokmph(luxuryCars.speed());

}

private double convertMPHtoKMPH(double mph) {

return mph \* 1.60934;

}

}

**Builder pattern**

package BuilderPattern;

public abstract class CD implements Packing{

@Override

public abstract String pack(); }

**CD Builder:**

ackage BuilderPattern;

public class CDBuilder {

public CDType buildSonyCD(){

CDType cd=new CDType();

cd.addItem(new Sony());

return cd;

}

public CDType buildSamsungCD(){

CDType cd=new CDType();

cd.addItem(new Samsung());

return cd;

}

}

**CD Type:**

package BuilderPattern;

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());

}

}

}

**Company:**

package BuilderPattern;

public abstract class Company extends CD{

@Override

public abstract int price();

}

**New Pattern:**

package BuilderPattern;

import BuilderPattern.CDBuilder;

import BuilderPattern.CDType;

public class NewPatterns {

public static void main(String[] args) {

CDBuilder cdBuilder=new CDBuilder();

CDType cdType1=cdBuilder.buildSonyCD();

cdType1.showItems();

CDType cdType2=cdBuilder.buildSamsungCD();

cdType2.showItems();

}

}

**Packing:**

package BuilderPattern;

public interface Packing {

public String pack();

public int price();

}

**Sumsung:**

package BuilderPattern;

public class Samsung extends Company {

@Override

public int price(){

return 50;

}

@Override

public String pack(){

return "Samsung CD";

}

}

**Sony**

package BuilderPattern;

public class Sony extends Company{

@Override

public int price(){

return 200;

}

@Override

public String pack(){

return "Sony CD";

}

}

**-:FacadePattren:-**

**BlackBerry**

package FacadePattern;

public class Blackberry implements MobileShop{

@Override

public void ModelNo() {

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

}

@Override

public void Price() {

System.out.println("Rs. 10,000");

}

}

**Mobileshop**

package FacadePattern;

public interface MobileShop {

public void ModelNo();

public void Price();

}

**Sumsung**

package FacadePattern;

public class Samsung implements MobileShop{

@Override

public void ModelNo() {

System.out.println("Samsung S5");

}

@Override

public void Price() {

System.out.println("Rs. 20,000");

}

}

**Shop**

package FacadePattern;

public class Shop {

MobileShop samsung;

MobileShop blackberry;

Shop(){

blackberry = new Blackberry();

samsung = new Samsung();

}

public void BlackberrySale(){

blackberry.ModelNo();

blackberry.Price();

}

public void SamsungSale(){

samsung.ModelNo();

samsung.Price();

}

}

**Main class**

package FacadePattern;

public class FascadeMain {

public static void main(String[] args) {

Shop s = new Shop();

s.BlackberrySale();

s.SamsungSale();

}

}

**-:DecorePattern:-**

**Circle**

package DecoratorPattern;

public class Circle implements Shape{

@Override

public void draw(){

System.out.println("draw a Circle ");

}

}

**Rectangle**

package DecoratorPattern;

public class Rectangle implements Shape{

@Override

public void draw(){

System.out.println("draw a Rectangle ");

}

}

**Shape**

package DecoratorPattern;

public interface Shape {

public void draw();

}

**RedShapeDecorator**

package DecoratorPattern;

public class RedShapeDecorator extends ShapeDecorator{

RedShapeDecorator(Shape shape){

super(shape);

}

public void setRedBorder(){

System.out.println("Red Border is set");

}

public void draw(){

shapeobj.draw();

}

}

**ShapeDecortor**

package DecoratorPattern;

public abstract class ShapeDecorator implements Shape{

Shape shapeobj;

ShapeDecorator(Shape shape){

this.shapeobj = shape;

}

}

**Main class**

package DecoratorPattern;

public class Main {

public static void main(String args[]){

Shape obj = new Circle();

ShapeDecorator redobj = new RedShapeDecorator(obj);

redobj.draw();

}

}

**Lab assignment 4**

**Factory pattern**

**AbstractFactory**

package patterns;

import Colors.Color;

import Shapes.Shape;

public abstract class AbstractFactory {

public abstract Shape getShape(String ShapeName);

public abstract Color getColor(String ColorName);

}

**FactoryDemo**

package patterns;

import Shapes.Shape;

import Shapes.ShapeFactory;

import java.io.\*;

public class FactoryDemo {

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

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

String choice;

Shape shapeselected;

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

choice = br.readLine();

ShapeFactory factory = new ShapeFactory();

shapeselected = factory.getShape(choice);

shapeselected.draw();

System.out.println("Enter Color: ");

choice = br.readLine();

FactoryProducer fobj = new FactoryProducer();

AbstractFactory ffobj = fobj.getFactory("color");

ffobj.getColor(choice).fill();

//======================================

}

}

**FactoryProducer**

package patterns;

import Colors.ColorFactory;

import Shapes.ShapeFactory;

public class FactoryProducer {

public AbstractFactory getFactory(String choice){

if(choice.equalsIgnoreCase("shape")){

return new ShapeFactory();

}else if(choice.equalsIgnoreCase("color")){

return new ColorFactory();

}

return null;

}

}

**Shape**

**Circle**

package Shapes;

import Shapes.Shape;

public class Circle implements Shape {

@Override

public void draw() {

System.out.println("Circle Draw");

}

}

**Rectangle**

package Shapes;

import Shapes.Shape;

public class Rectangle implements Shape{

@Override

public void draw() {

System.out.println("Rectangle Draw");

}

}

**Shape**

package Shapes;

public interface Shape {

public void draw();

}

**ShapeFactory**

package Shapes;

import Colors.Color;

import Shapes.Shape;

import patterns.AbstractFactory;

public class ShapeFactory extends AbstractFactory{

@Override

public Shape getShape(String ShapeName){

if(ShapeName.equals("Circle")){

return this.getCircle();

}else if(ShapeName.equals("Square")){

return this.getSquare();

}else if(ShapeName.equals("Rectangle")){

return this.getRectangle();

}

return null;

}

public Shape getCircle(){

return new Circle();

}

public Shape getSquare(){

return new Square();

}

public Shape getRectangle(){

return new Rectangle();

}@Override

public Color getColor(String ColorName) {

return null;

}

}

**Square**

package Shapes;

import Shapes.Shape;

public class Square implements Shape {

@Override

public void draw() {

System.out.println("Square Draw");

}

}

* **Colors**

**ColorFactory**

package Colors;

import Shapes.Shape;

import patterns.AbstractFactory;

public class ColorFactory extends AbstractFactory{

@Override

public Color getColor(String ColorName){

if(ColorName.equals("Blue")){

return this.getBlue();

}else if(ColorName.equals("Red")){

return this.getRed();

}else if(ColorName.equals("Green")){

return this.getGreen();

}

return null;

}

public Color getRed(){

return new Red();

}

public Color getGreen(){

return new Green();

}

public Color getBlue(){

return new Blue();

} @Override

public Shape getShape(String ShapeName) {

return null;

}

}

**Color**

package Colors;

public interface Color {

public void fill();

}

**Red**

package Colors;

public class Red implements Color{

@Override

public void fill() {

System.out.println("Red Filled");

}

}

**Blue**

package Colors;

public class Blue implements Color{

@Override

public void fill() {

System.out.println("Red Filled");

}

}

**Green**

package Colors;

public class Green implements Color{

@Override

public void fill() {

System.out.println("Red Filled");

}

}