# Week10\_DesignPatterns\_HandsOn

## 1.Singleton Pattern

Logger.java

package com.singleton.example;

public class Logger {

private static Logger instance;

private Logger() {

System.out.println("LOgger instance created");

}

public static Logger getInstance() {

if(instance==null) {

instance=new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("Log:" + message);

}

}

LoggerTest.java

package com.singleton.example;

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First log message");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message");

if(logger1==logger2) {

System.out.println("Both logger instance are the same");

}else {

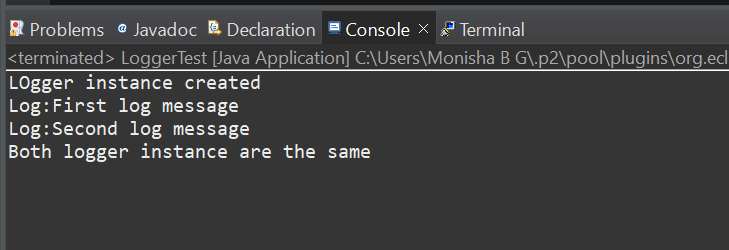
System.out.println("different logger instances");

}

}

}

Output Screenshot:



## 2.Factory Method Pattern

Document.java

package com.factory.documents;

public interface Document {

void open();

}

ExcelDocument.java

package com.factory.documents;

public class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("Opening Excel Document..");

}

}

PdfDocument.java

package com.factory.documents;

public class PdfDocument implements Document {

@Override

public void open() {

System.out.println("Opening Pdf Document..");

}

}

Worddocument.java

package com.factory.documents;

public class WordDocument implements Document {

@Override

public void open() {

System.out.println("opening Word document..");

}

}

DocumentFactory.java

package com.factory.factories;

import com.factory.documents.Document;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

ExcelDocumentFactory.java

package com.factory.factories;

import com.factory.documents.Document;

import com.factory.documents.ExcelDocument;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

PdfDocumentFactory.java

package com.factory.factories;

import com.factory.documents.Document;

import com.factory.documents.PdfDocument;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

WordDocumentFactory.java

package com.factory.factories;

import com.factory.documents.Document;

import com.factory.documents.WordDocument;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

FactoryMEthodTest.java

package com.factory.test;

import com.factory.documents.Document;

import com.factory.factories.\*;

public class FactoryMethodTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

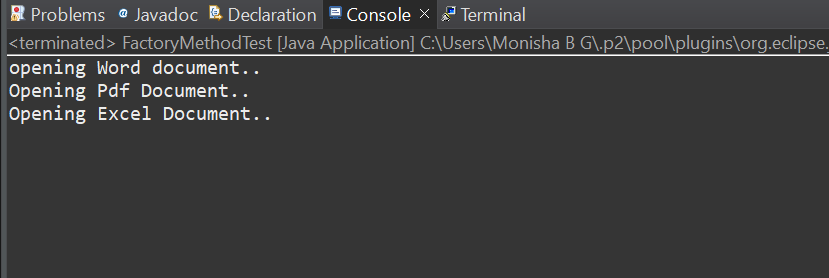
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

Output Screenshot:



## 3.Builder Pattern

Computer.java

package com.builder.build;

public class Computer {

private final String CPU;

private final String RAM;

private final String Storage;

private final String graphicsCard;

private final String operatingSystem;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.Storage = builder.Storage;

this.graphicsCard = builder.graphicsCard;

this.operatingSystem = builder.operatingSystem;

}

public String getCPU() {

return CPU;

}

public String getRAM() {

return RAM;

}

public String getStorage() {

return Storage;

}

public String getGraphicsCard() {

return graphicsCard;

}

public String getOperatingSystem() {

return operatingSystem;

}

@Override

public String toString() {

return "Computer Configuration:\n" +

"CPU: " + CPU + "\n" +

"RAM: " + RAM + "\n" +

"Storage: " + Storage + "\n" +

"Graphics Card: " + graphicsCard + "\n" +

"Operating System: " + operatingSystem;

}

public static class Builder {

private String CPU;

private String RAM;

private String Storage;

private String graphicsCard;

private String operatingSystem;

public Builder setCPU(String CPU) {

this.CPU = CPU;

return this;

}

public Builder setRAM(String RAM) {

this.RAM = RAM;

return this;

}

public Builder setStorage(String storage) {

this.Storage = Storage;

return this;

}

public Builder setGraphicsCard(String graphicsCard) {

this.graphicsCard = graphicsCard;

return this;

}

public Builder setOperatingSystem(String operatingSystem) {

this.operatingSystem = operatingSystem;

return this;

}

public Computer build() {

return new Computer(this);

}

}

}

BuilderPatterntest.java

package com.builder.build;

public class BuilderPatterntest {

public static void main(String[] args) {

Computer gamingPC = new Computer.Builder()

.setCPU("Intel Core i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.setOperatingSystem("Windows 11")

.build();

System.out.println(gamingPC);

System.out.println();

Computer officePC = new Computer.Builder()

.setCPU("Intel Core i5")

.setRAM("8GB")

.setStorage("512GB SSD")

.setOperatingSystem("Windows 10")

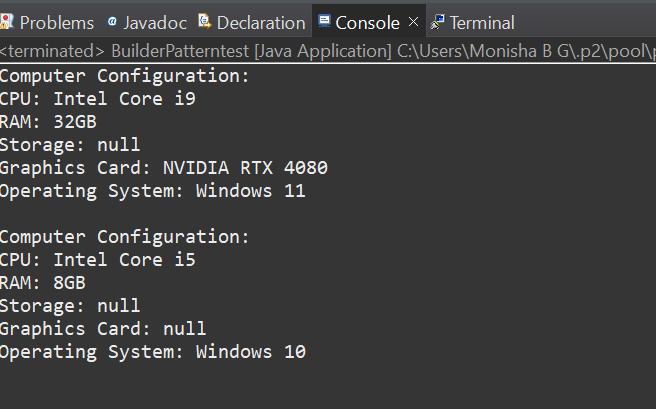
.build();

System.out.println(officePC);

}

}

Output Screenshot:



## 4.Adapter Pattern

PayPalAdapter.java

package com.adapter.example;

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway payPalGateway;

public PayPalAdapter(PayPalGateway gateway) {

this.payPalGateway=gateway;

}

@Override

public void processPayment(double amount) {

payPalGateway.sendMoney(amount);

}

}

StripeAdapter.java

package com.adapter.example;

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripeGateway;

public StripeAdapter(StripeGateway gateway) {

this.stripeGateway = gateway;

}

@Override

public void processPayment(double amount) {

stripeGateway.makePayment(amount);

}

}

PayPalGateway.java

package com.adapter.example;

public class PayPalGateway {

public void sendMoney(double amount) {

System.out.println("Paid $"+ amount + "using PayPal");

}

}

StripeGateway.java

package com.adapter.example;

public class StripeGateway {

public void makePayment(double amount) {

System.out.println("Paid $" +amount + "using Stripe");

}

}

PaymentProcessor.java

package com.adapter.example;

public class StripeGateway {

public void makePayment(double amount) {

System.out.println("Paid $" +amount + "using Stripe");

}

}

AdapterPatternTest.java

package com.adapter.example;

public class AdapterPatternTest {

public static void main(String[] args) {

PaymentProcessor paypal = new PayPalAdapter(new PayPalGateway());

paypal.processPayment(1500.00);

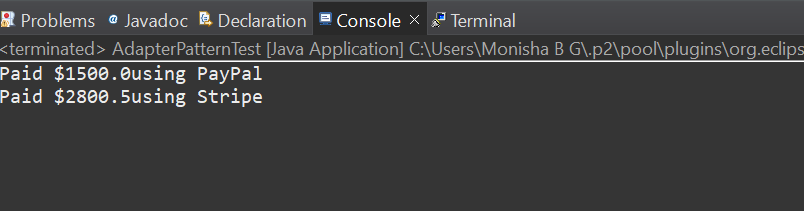
PaymentProcessor stripe = new StripeAdapter(new StripeGateway());

stripe.processPayment(2800.50);

}

}

Output Screenshot:



## 5.Decorator Pattern

Notifier.java

package com.decorator.example;

public interface Notifier {

void send(String message);

}

EmailNotifier.java

package com.decorator.example;

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

// TODO Auto-generated method stub

System.out.println("Sending Email:" + message);

}

}

NotifierDecorator.java

package com.decorator.example;

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

@Override

public void send(String message) {

// TODO Auto-generated method stub

wrappedNotifier.send(message);

}

}

SMSNotifierDecorator.java

package com.decorator.example;

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSMS(message);

}

private void sendSMS(String message) {

System.out.println("Sending SMS: " + message);

}

}

SlackNotifierDecorator.java

package com.decorator.example;

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSlack(message);

}

private void sendSlack(String message) {

System.out.println("Sending Slack Message: " + message);

}

}

DecoratorPatternTest.java

package com.decorator.example;

public class DecoratorPatternTest {

public static void main(String args[]) {

Notifier notifier =new EmailNotifier();

notifier=new SMSNotifierDecorator(notifier);

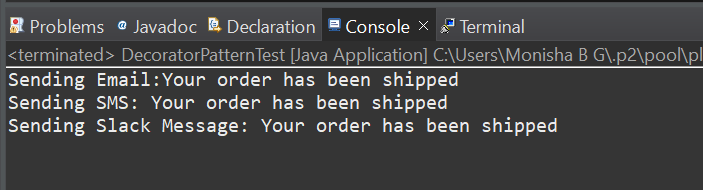
notifier=new SlackNotifierDecorator(notifier);

notifier.send("Your order has been shipped");

}

}

Output Screenshot:



## 6.Command Pattern

Light.java

package com.command.example;

public class Light {

public void turnOn() {

System.out.println("Light is ON");

}

public void turnOff() {

System.out.println("Light is Off");

}

}

Command.java

package com.command.example;

public interface Command {

void execute();

}

LightOffCommand.java

package com.command.example;

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

// TODO Auto-generated method stub

light.turnOff();

}

}

LightOnCommand.java

package com.command.example;

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

@Override

public void execute() {

// TODO Auto-generated method stub

light.turnOn();

}

}

RemoteControl.java

package com.command.example;

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

CommandPatternTest.java

package com.command.example;

public class CommandPatternTest {

public static void main(String args[]) {

Light livingRoomLight = new Light();

Command lightOn = new LightOnCommand(livingRoomLight);

Command lightOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

System.out.println("Turning ON the light:");

remote.setCommand(lightOn);

remote.pressButton();

System.out.println("\nTurning OFF the light:");

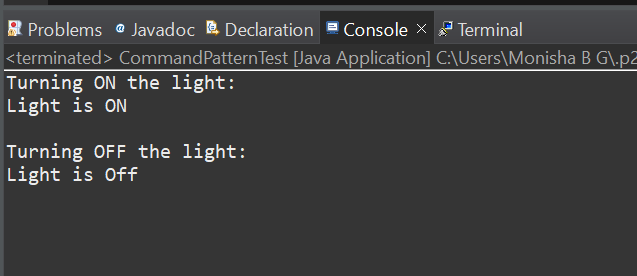
remote.setCommand(lightOff);

remote.pressButton();

}

}

Output Screenshot:



## 7.MVC Pattern

Student.java

package com.mvc.example;

public class Student {

private String name;

private String id;

private String grade;

public Student(String name, String id, String grade) {

this.name = name;

this.id = id;

this.grade = grade;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getGrade() {

return grade;

}

public void setGrade(String grade) {

this.grade = grade;

}

}

StudentController.java

package com.mvc.example;

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void setStudentName(String name) {

model.setName(name);

}

public void setStudentId(String id) {

model.setId(id);

}

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public String getStudentName() {

return model.getName();

}

public String getStudentId() {

return model.getId();

}

public String getStudentGrade() {

return model.getGrade();

}

public void updateView() {

view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

}

}

StudentView.java

package com.mvc.example;

public class StudentView {

public void displayStudentDetails(String name, String id, String grade) {

System.out.println("Student Details:");

System.out.println("Name : " + name);

System.out.println("ID : " + id);

System.out.println("Grade: " + grade);

}

}

MVCPatternTest.java

package com.mvc.example;

public class MVCPatternTest {

public static void main(String[] args) {

Student student = new Student("Alice", "S101", "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

System.out.println("\nUpdating student grade to B...");

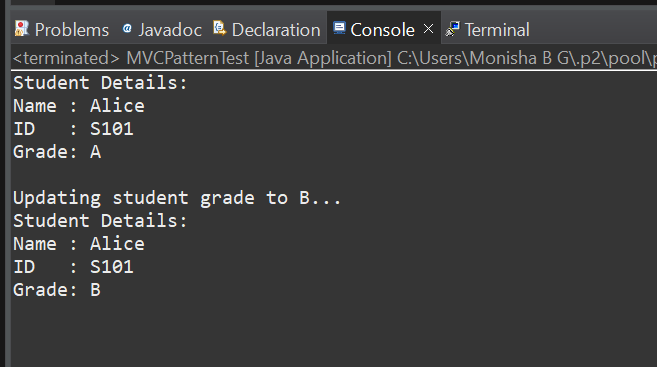
controller.setStudentGrade("B");

controller.updateView();

}

}

Output Screenshot:



## 8.Observer Pattern

MobileApp.java

package com.observer.example;

public class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name=name;

}

@Override

public void update(double price) {

// TODO Auto-generated method stub

System.out.println(name+ " -mobile app recieved stock update: $" + price );

}

}

Observer.java

package com.observer.example;

public interface Observer {

void update(double price);

}

Stock.java

package com.observer.example;

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

StockMArket.java

package com.observer.example;

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers = new ArrayList<>();

private double stockPrice;

public void setStockPrice(double price) {

this.stockPrice=price;

notifyObservers();

}

public double getStockPrice() {

return stockPrice;

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers() {

// TODO Auto-generated method stub

for(Observer o: observers) {

o.update(stockPrice);

}

}

}

WebApp.java

package com.observer.example;

public class WebApp implements Observer {

private String name;

public WebApp(String name){

this.name=name;

}

@Override

public void update(double price) {

// TODO Auto-generated method stub

System.out.println(name + "-WebApp recieved stock update: $" + price);

}

}

ObserverPatternTEst.java

package com.observer.example;

public class ObserverPatternTest {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile1 = new MobileApp("Monisha");

Observer web1 = new WebApp("Philo");

market.registerObserver(mobile1);

market.registerObserver(web1);

System.out.println("Setting price to $120.50");

market.setStockPrice(120.50);

System.out.println("\nSetting price to $121.00");

market.setStockPrice(121.00);

market.removeObserver(mobile1);

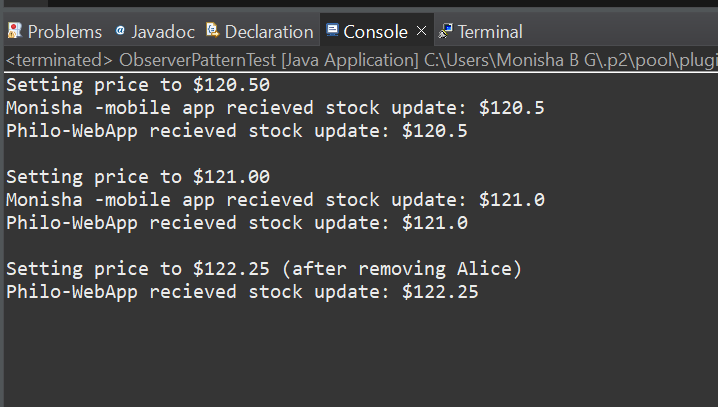
System.out.println("\nSetting price to $122.25 (after removing Alice)");

market.setStockPrice(122.25);

}

}

Output Screenshot:



## 10.Proxy Pattern

Image.java

package com.proxy.example;

public interface Image {

void display();

}

proxyImage.java

package com.proxy.example;

public class ProxyImage implements Image {

private String filename;

private RealImage realImage;

public ProxyImage(String filename) {

this.filename=filename;

}

@Override

public void display() {

// TODO Auto-generated method stub

if(realImage==null) {

realImage=new RealImage(filename);

}else {

System.out.println("Using chched image: " +filename);

}

realImage.display();

}

}

RealImage.java

package com.proxy.example;

public class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename=filename;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

System.out.println("Loading image from remote server: " + filename);

try {

Thread.sleep(1000);

}

catch(InterruptedException e) {

e.printStackTrace();

}

}

@Override

public void display() {

// TODO Auto-generated method stub

System.out.println("Displaying: " + filename);

}

}

ProxyPatternTest.java

package com.proxy.example;

public class ProxyPatternTest {

public static void main(String[] args) {

Image image1 = new ProxyImage("photo1.jpg");

Image image2 = new ProxyImage("photo2.jpg");

image1.display();

image1.display();

image2.display();

}

}

Output Screenshot:

