## ****Exercise 1: Singleton Pattern****

## Logger.java

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);

}

}

### TestSingleton.java

public class TestSingleton {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Starting...");

Logger logger2 = Logger.getInstance();

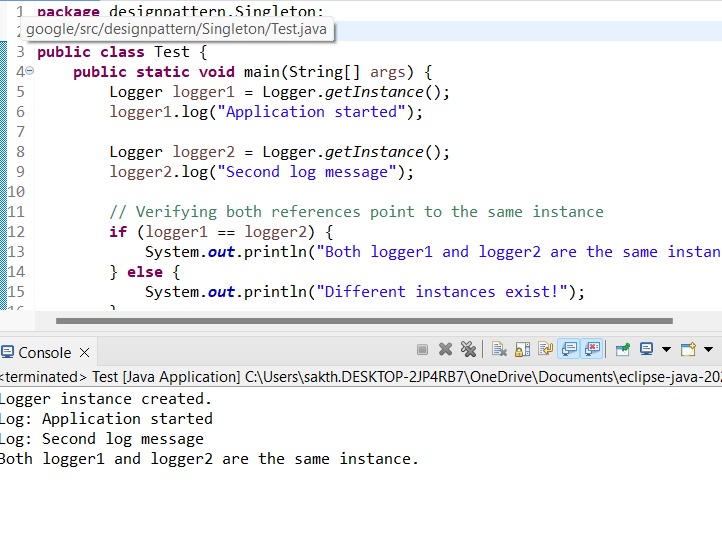
logger2.log("Running...");

System.out.println(logger1 == logger2 ? "Same instance." : "Different instances.");

}

}

### Output



## ****Exercise 2: Factory Method Pattern****

### Document.java

public interface Document {

void open();

}

WordDocument.java

public class WordDocument implements Document {

public void open() {

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

}

}

### PdfDocument.java

public class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF document.");

}

}

### ExcelDocument.java

public class ExcelDocument implements Document {

public void open() {

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

}

}

### DocumentFactory.java

public abstract class DocumentFactory {

public abstract Document createDocument();

}

### WordFactory.java / PdfFactory.java / ExcelFactory.java

public class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

public class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

public class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

### TestFactory.java

public class TestFactory {

public static void main(String[] args) {

DocumentFactory factory = new PdfFactory();

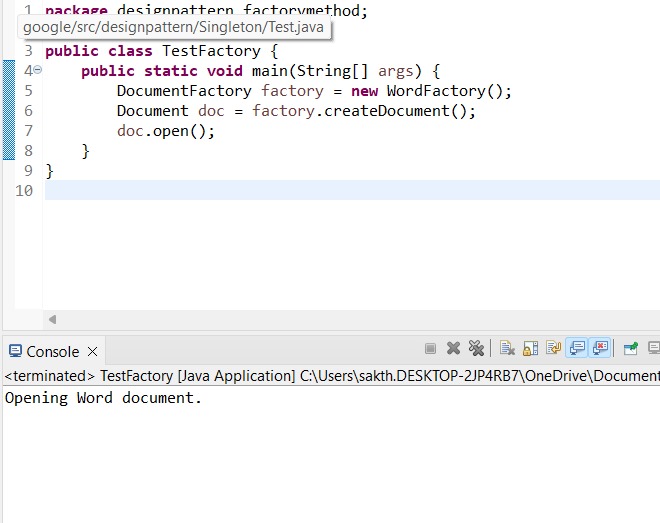
Document doc = factory.createDocument();

doc.open();

}

}

**Output**



## ****Exercise 3: Builder Pattern****

### Computer.java

public class Computer {

private String CPU;

private String RAM;

private String storage;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

}

public static class Builder {

private String CPU;

private String RAM;

private String storage;

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 Computer build() {

return new Computer(this);

}

}

public void displaySpecs() {

System.out.println("CPU: " + CPU + ", RAM: " + RAM + ", Storage: " + storage);

}

}

### TestBuilder.java

public class TestBuilder {

public static void main(String[] args) {

Computer pc = new Computer.Builder()

.setCPU("i7")

.setRAM("16GB")

.setStorage("1TB SSD")

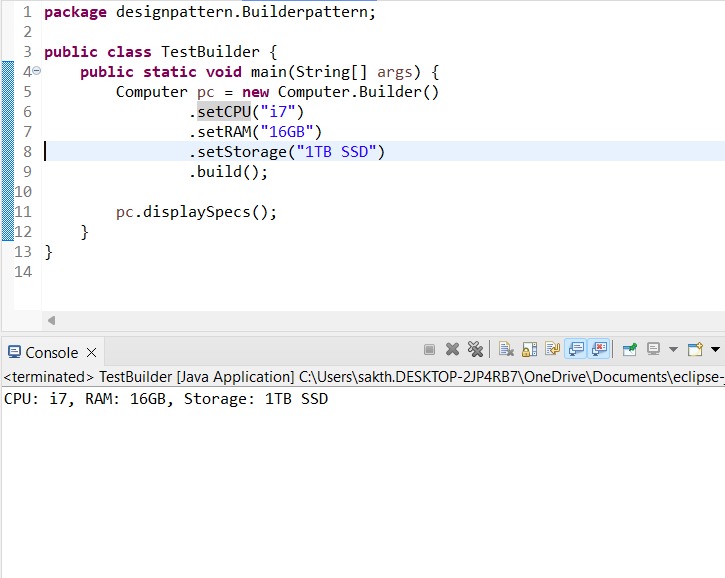
.build();

pc.displaySpecs();

}

}

**Output**



## ****Exercise 4: Adapter Pattern****

### PaymentProcessor.java

java

CopyEdit

public interface PaymentProcessor {

void processPayment(double amount);

}

### PayPalGateway.java

public class PayPalGateway {

public void makePayment(double amount) {

System.out.println("Paid via PayPal: " + amount);

}

}

### PayPalAdapter.java

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway gateway = new PayPalGateway();

public void processPayment(double amount) {

gateway.makePayment(amount);

}

}

### TestAdapter.java

public class TestAdapter {

public static void main(String[] args) {

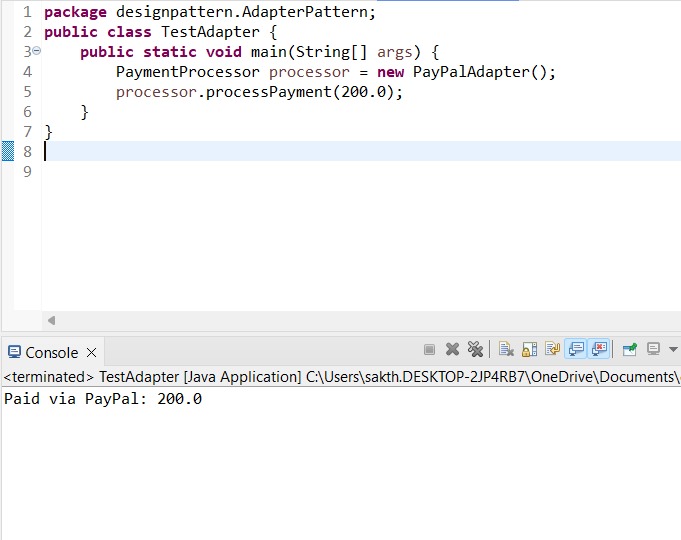
PaymentProcessor processor = new PayPalAdapter();

processor.processPayment(200.0);

}

}

**Output**



## ****Exercise 5: Decorator Pattern****

### Notifier.java

public interface Notifier {

void send(String message);

}

### EmailNotifier.java

java

CopyEdit

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

### NotifierDecorator.java

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappee;

public NotifierDecorator(Notifier notifier) {

this.wrappee = notifier;

}

public void send(String message) {

wrappee.send(message);

}

}

### SMSNotifier.java / SlackNotifier.java

public class SMSNotifier extends NotifierDecorator {

public SMSNotifier(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

public class SlackNotifier extends NotifierDecorator {

public SlackNotifier(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

### TestDecorator.java

public class TestDecorator {

public static void main(String[] args) {

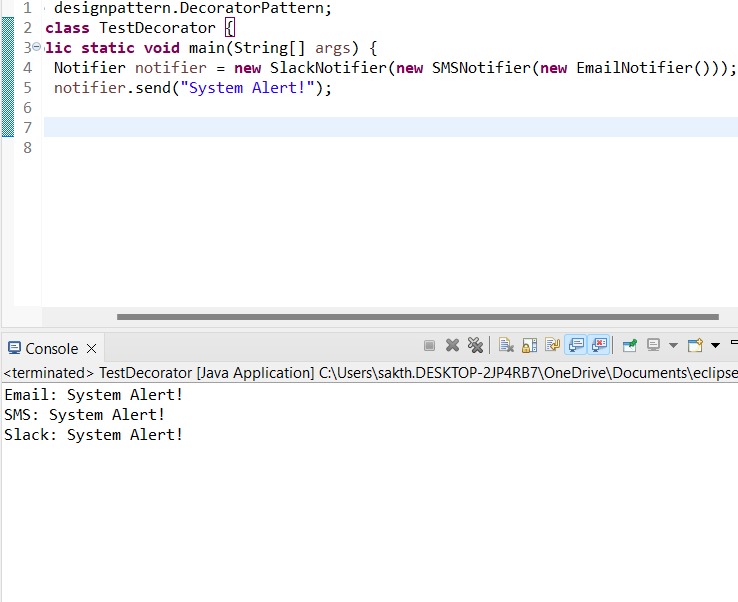
Notifier notifier = new SlackNotifier(new SMSNotifier(new EmailNotifier()));

notifier.send("System Alert!");

}

}

**Output**



## ****Exercise 6: Proxy Pattern****

### Image.java

public interface Image {

void display();

}

### RealImage.java

public class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromDisk();

}

private void loadFromDisk() {

System.out.println("Loading " + filename);

}

public void display() {

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

}

}

### ProxyImage.java

public class ProxyImage implements Image {

private RealImage realImage;

private String filename;

public ProxyImage(String filename) {

this.filename = filename;

}

public void display() {

if (realImage == null) {

realImage = new RealImage(filename);

}

realImage.display();

}

}

### TestProxy.java

public class TestProxy {

public static void main(String[] args) {

Image image = new ProxyImage("cat.jpg");

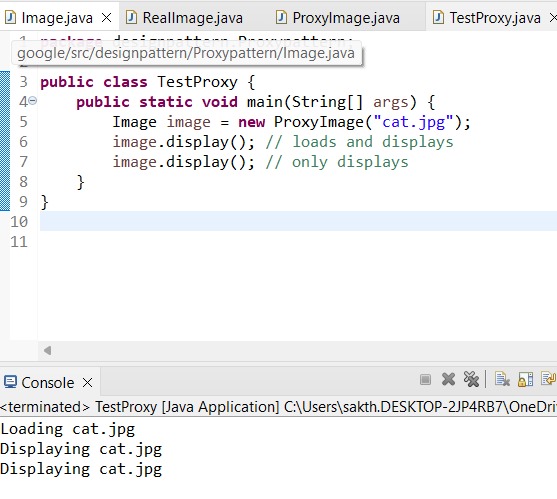
image.display(); // loads and displays

image.display(); // only displays

}

}

**Output**



## ****Exercise 7: Observer Pattern****

### Observer.java

public interface Observer {

void update(String stock);

}

### Stock.java

public interface Stock {

void register(Observer o);

void deregister(Observer o);

void notifyObservers();

}

### StockMarket.java

import java.util.\*;

public class StockMarket implements Stock {

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

private String stockUpdate;

public void setStockUpdate(String update) {

this.stockUpdate = update;

notifyObservers();

}

public void register(Observer o) {

observers.add(o);

}

public void deregister(Observer o) {

observers.remove(o);

}

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockUpdate);

}

}

}

### MobileApp.java / WebApp.java

public class MobileApp implements Observer {

public void update(String stock) {

System.out.println("MobileApp received: " + stock);

}

}

public class WebApp implements Observer {

public void update(String stock) {

System.out.println("WebApp received: " + stock);

}

}

### TestObserver.java

public class TestObserver {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile = new MobileApp();

Observer web = new WebApp();

market.register(mobile);

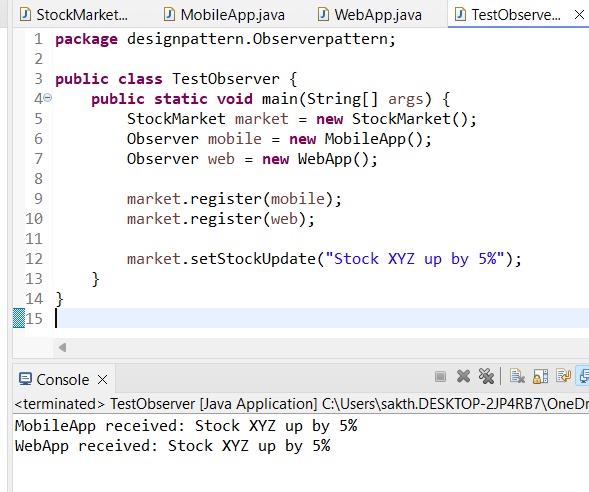
market.register(web);

market.setStockUpdate("Stock XYZ up by 5%");

}

}

**Output**



## ****Exercise 8: Strategy Pattern****

### PaymentStrategy.java

public interface PaymentStrategy {

void pay(double amount);

}

### CreditCardPayment.java / PayPalPayment.java

public class CreditCardPayment implements PaymentStrategy {

public void pay(double amount) {

System.out.println("Paid with Credit Card: " + amount);

}

}

public class PayPalPayment implements PaymentStrategy {

public void pay(double amount) {

System.out.println("Paid with PayPal: " + amount);

}

}

### PaymentContext.java

public class PaymentContext {

private PaymentStrategy strategy;

public PaymentContext(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void executePayment(double amount) {

strategy.pay(amount);

}

}

### TestStrategy.java

public class TestStrategy {

public static void main(String[] args) {

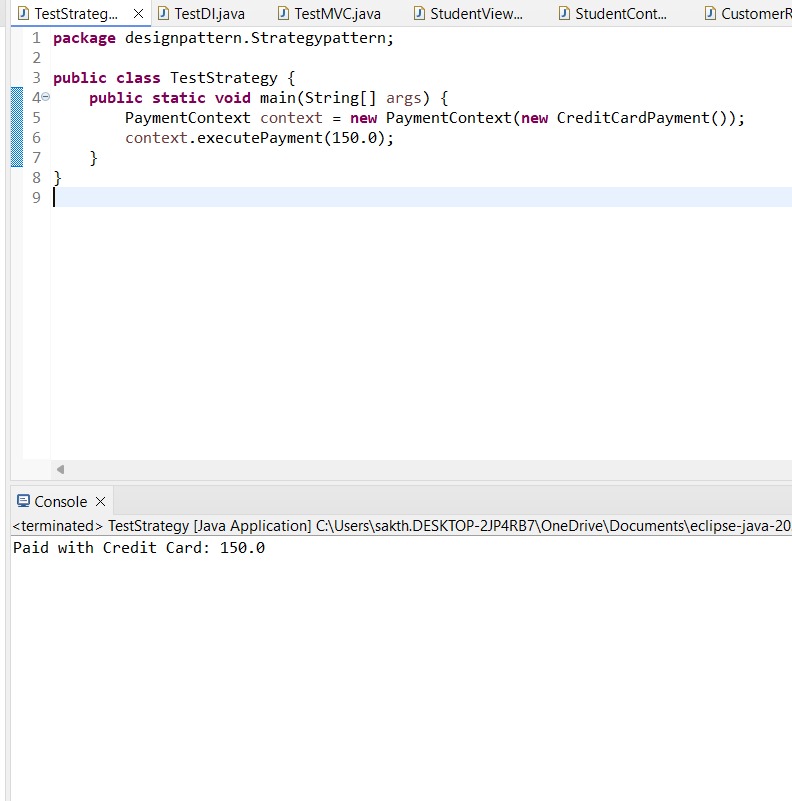
PaymentContext context = new PaymentContext(new CreditCardPayment());

context.executePayment(150.0);

}

}

**Output**



## ****Exercise 9: Command Pattern****

### Command.java

public interface Command {

void execute();

}

### Light.java

public class Light {

public void turnOn() {

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

}

public void turnOff() {

System.out.println("Light OFF");

}

}

### LightOnCommand.java / LightOffCommand.java

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

public class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

### RemoteControl.java

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

### TestCommand.java

public class TestCommand {

public static void main(String[] args) {

Light light = new Light();

Command on = new LightOnCommand(light);

Command off = new LightOffCommand(light);

RemoteControl remote = new RemoteControl();

remote.setCommand(on);

remote.pressButton();

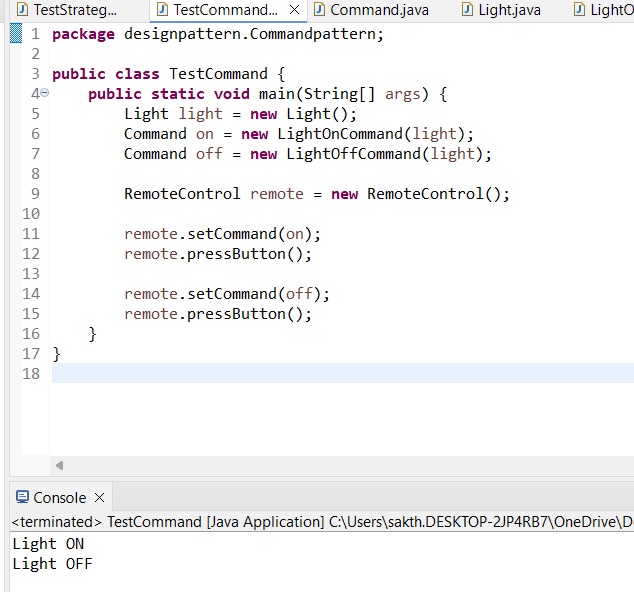
remote.setCommand(off);

remote.pressButton();

}

}

**Output**



## ****Exercise 10: MVC Pattern****

### Student.java (Model)

public class Student {

private String name;

private int id;

private String grade;

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

this.name = name;

this.id = id;

this.grade = grade;

}

// Getters and Setters

public String getName() { return name; }

public int getId() { return id; }

public String getGrade() { return grade; }

public void setGrade(String grade) { this.grade = grade; }

}

### StudentView.java (View)

public class StudentView {

public void displayStudentDetails(Student student) {

System.out.println("ID: " + student.getId() + ", Name: " + student.getName() + ", Grade: " + student.getGrade());

}

}

### StudentController.java (Controller)

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void updateGrade(String grade) {

model.setGrade(grade);

}

public void display() {

view.displayStudentDetails(model);

}

}

### TestMVC.java

public class TestMVC {

public static void main(String[] args) {

Student student = new Student("John", 101, "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.display();

controller.updateGrade("A+");

controller.display();

}

}

**Output**

## WhatsApp Image 2025-06-22 at 20.56.19_0a109147 ****Exercise 11: Dependency Injection****

### CustomerRepository.java

public interface CustomerRepository {

String findCustomerById(int id);

}

### CustomerRepositoryImpl.java

public class CustomerRepositoryImpl implements CustomerRepository {

public String findCustomerById(int id) {

return "Customer#" + id;

}

}

### CustomerService.java

public class CustomerService {

private CustomerRepository repository;

public CustomerService(CustomerRepository repository) {

this.repository = repository;

}

public void displayCustomer(int id) {

System.out.println("Found: " + repository.findCustomerById(id));

}

}

### TestDI.java

public class TestDI {

public static void main(String[] args) {

CustomerRepository repo = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repo);

service.displayCustomer(123);

}

}

**Output**

