**Design Patterns and Design Principles Solution**

**Exercise 1: Implementing the Singleton Pattern**

**Project Structure :**

SingletonPatternExample/

├── Logger.java

└── SingletonTest.java

**Code :**

**Logger.java - (Singleton Class)**

**package** singleton;

**public** **class** Logger {

**private** **static** **volatile** Logger *instance* = **null**;

// Private constructor

**private** Logger() {

System.***out***.println("Logger Instance Created");

}

// Public method to provide access to the instance

**public** **static** Logger getInstance() {

**if** (*instance* == **null**) {

**synchronized** (Logger.**class**) {

**if** (*instance* == **null**) {

*instance* = **new** Logger();

}

}

}

**return** *instance*;

}

**public** **void** log(String message) {

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

}

}

**SingletonTest.java - (Test Class)**

**package** singleton;

**public** **class** SingletonTest {

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

Logger logger1 = Logger.*getInstance*();

logger1.log("This is the first log message.");

Logger logger2 = Logger.*getInstance*();

logger2.log("This is the second log message.");

**if** (logger1 == logger2) {

System.***out***.println("Only one instance exists. Singleton verified.");

} **else** {

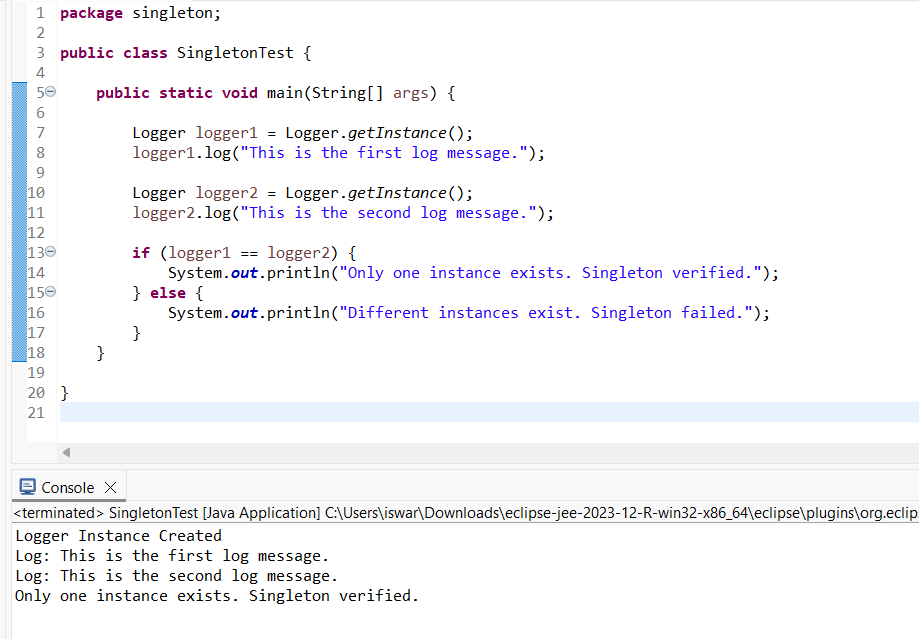
System.***out***.println("Different instances exist. Singleton failed.");

}

}

}

**Output :**

****

**Exercise 2: Implementing the Factory Method Pattern**

**Project Structure :**

FactoryMethodPatternExample/

├── Document.java (Interface)

├── Word.java (Concrete)

├── Pdf.java (Concrete)

├── Excel.java (Concrete)

├── DocumentFactory.java (Abstract Factory)

├── WordFactory.java

├── PdfFactory.java

├── ExcelFactory.java

└── FactoryTest.java

**Code :**

**Document.java**

**package** factory;

**public** **interface** Document {

**void** open();

}

**Excel.java**

**package** factory;

**public** **class** Excel **implements** Document {

@Override

**public** **void** open() {

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

}

}

**Pdf.java**

**package** factory;

**public** **class** Pdf **implements** Document {

@Override

**public** **void** open() {

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

}

}

**Word.java**

**package** factory;

**public** **class** Wordf **implements** Document {

@Override

**public** **void** open() {

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

}

}

**DocumentFactory.java**

**package** factory;

**public** **abstract** **class** DocumentFactory {

**public** **abstract** Document createDocument();

}

**ExcelFactory.java**

**package** factory;

**public** **class** ExcelFactory **extends** DocumentFactory {

@Override

**public** Document createDocument() {

**return** **new** Excel();

}

}

**PdfFactory.java**

**package** factory;

**public** **class** PdfFactory **extends** DocumentFactory {

@Override

**public** Document createDocument() {

**return** **new** Pdf();

}

}

**WordFactory.java**

**package** factory;

**public** **class** WordFactory **extends** DocumentFactory {

@Override

**public** Document createDocument() {

**return** **new** Word();

}

}

**FactoryTest.java**

**package** factory;

**public** **class** FactoryTest {

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

DocumentFactory wordFactory = **new** WordFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = **new** PdfFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = **new** ExcelFactory();

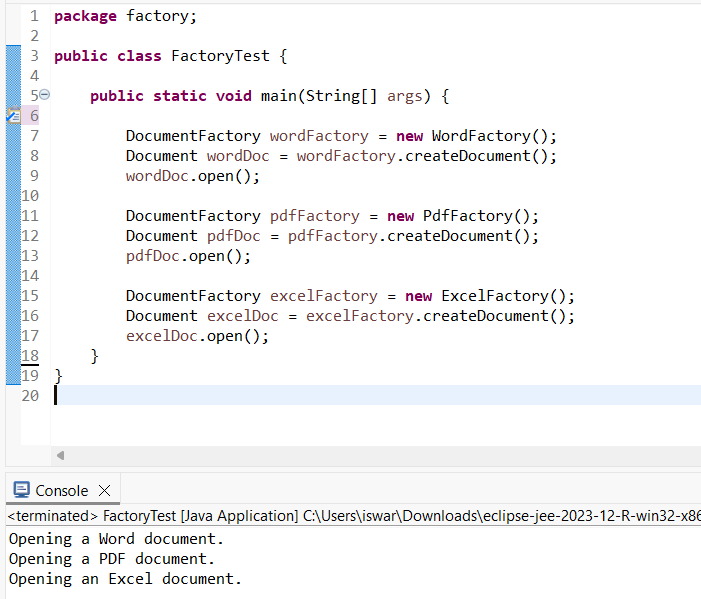
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output :**

****

**Exercise 3: Implementing the Builder Pattern**

**Project Structure :**

BuilderPatternExample/

├── Computer.java

└── BuilderTest.java

**Code :**

**Computer.java**

**package** builder;

**public** **class** Computer {

**private** String cpu;

**private** String ram;

**private** String storage;

**private** String graphicsCard;

**private** **boolean** wifiEnabled;

**private** **boolean** bluetoothEnabled;

**private** Computer(Builder builder) {

**this**.cpu = builder.cpu;

**this**.ram = builder.ram;

**this**.storage = builder.storage;

**this**.graphicsCard = builder.graphicsCard;

**this**.wifiEnabled = builder.wifiEnabled;

**this**.bluetoothEnabled = builder.bluetoothEnabled;

}

**public** **static** **class** Builder {

**private** String cpu;

**private** String ram;

**private** String storage;

**private** String graphicsCard;

**private** **boolean** wifiEnabled;

**private** **boolean** bluetoothEnabled;

**public** Builder(String cpu, String ram) {

**this**.cpu = cpu;

**this**.ram = ram;

}

**public** Builder setStorage(String storage) {

**this**.storage = storage;

**return** **this**;

}

**public** Builder setGraphicsCard(String graphicsCard) {

**this**.graphicsCard = graphicsCard;

**return** **this**;

}

**public** Builder enableWiFi(**boolean** value) {

**this**.wifiEnabled = value;

**return** **this**;

}

**public** Builder enableBluetooth(**boolean** value) {

**this**.bluetoothEnabled = value;

**return** **this**;

}

**public** Computer build() {

**return** **new** Computer(**this**);

}

}

**public** **void** showConfig() {

System.***out***.println("Computer Configuration:");

System.***out***.println("CPU: " + cpu);

System.***out***.println("RAM: " + ram);

System.***out***.println("Storage: " + storage);

System.***out***.println("Graphics Card: " + graphicsCard);

System.***out***.println("WiFi Enabled: " + wifiEnabled);

System.***out***.println("Bluetooth Enabled: " + bluetoothEnabled);

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

}

}

**BuilderTest.java**

**package builder;**

**public class BuilderTest {**

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

**Computer basicComputer = new Computer.Builder("Intel i5", "8GB")**

**.setStorage("512GB SSD")**

**.build();**

**basicComputer.showConfig();**

**Computer gamingComputer = new Computer.Builder("AMD Ryzen 7", "32GB")**

**.setStorage("1TB SSD")**

**.setGraphicsCard("NVIDIA RTX 4070")**

**.enableWiFi(true)**

**.enableBluetooth(true)**

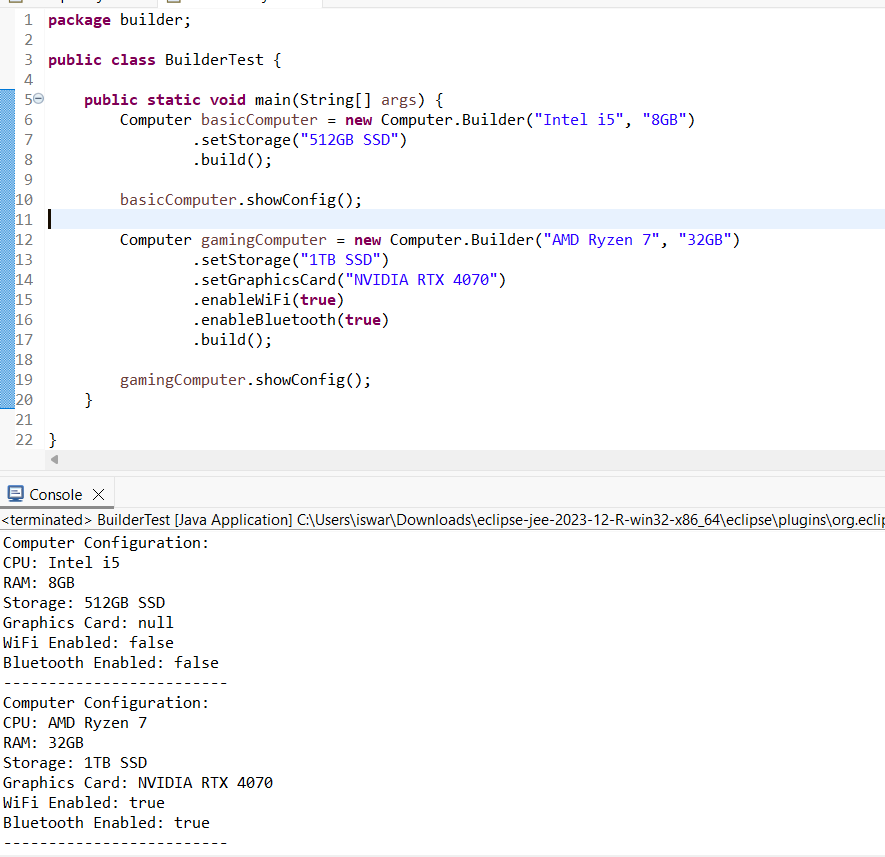
**.build();**

**gamingComputer.showConfig();**

**}**

**}**

**Output :**

****

**Exercise 4: Implementing the Adapter Pattern**

**Project Structure :**

AdapterPatternExample/

├── PaymentProcessor.java (Target Interface)

├── StripeGateWay.java (Adaptee)

├── PayPalGateWay.java (Adaptee)

├── StripeAdapter.java (Adapter)

├── PayPalAdapter.java (Adapter)

└── AdapterTest.java

**Code :**

**PaymentProcessor.java**

**package adapter;**

**public interface PaymentProcessor {**

**void processPayment(double amount);**

**}**

**StripeGateWay.java**

**package adapter;**

**public class StripeGateWay {**

**public void doPayment(double amount) {**

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

**}**

**}**

**PayPalGateWay.java**

**package** adapter;

**public** **class** PayPalGateway {

**public** **void** makePayment(**double** amount) {

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

}

}

**StripeAdapter.java**

**package adapter;**

**public class StripeAdapter implements PaymentProcessor {**

**private StripeGateWay stripeGateway;**

**public StripeAdapter() {**

**this.stripeGateway = new StripeGateWay();**

**}**

**@Override**

**public void processPayment(double amount) {**

**stripeGateway.doPayment(amount);**

**}**

**}**

**PayPalAdapter.java**

**package** adapter;

**public** **class** PayPalAdapter **implements** PaymentProcessor {

**private** PayPalGateway payPalGateway;

**public** PayPalAdapter() {

**this**.payPalGateway = **new** PayPalGateway();

}

@Override

**public** **void** processPayment(**double** amount) {

payPalGateway.makePayment(amount);

}

}

**AdapterTest.java**

**package** adapter;

**public** **class** AdapterTest {

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

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

paypal.processPayment(10000.0);

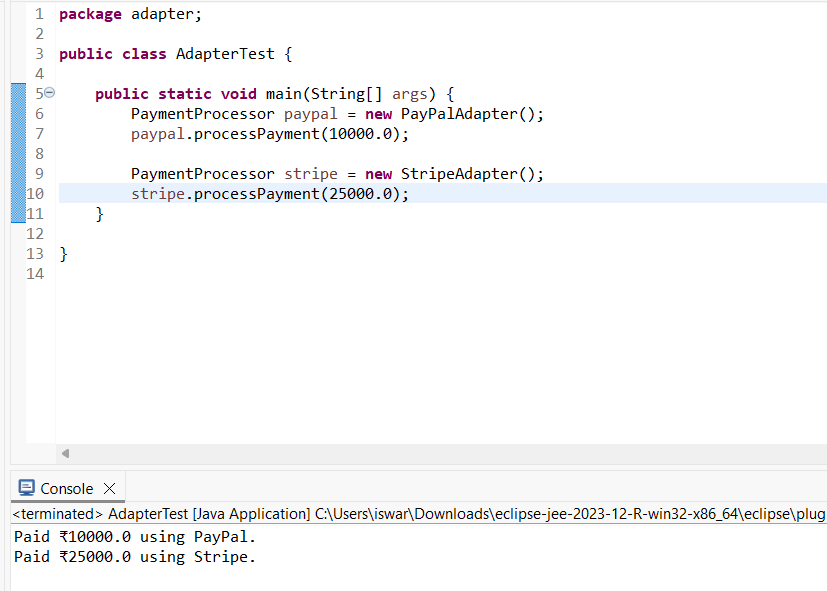
PaymentProcessor stripe = **new** StripeAdapter();

stripe.processPayment(25000.0);

}

}

**Output :**

****

**Exercise 5: Implementing the Decorator Pattern**

**Project Structure :**

DecoratorPatternExample/

├── Notifier.java (Component Interface)

├── EmailNotifier.java (Concrete Component)

├── NotifierDecorator.java (Abstract Decorator)

├── SMSNotifierDecorator.java

├── SlackNotifierDecorator.java

└── DecoratorTest.java

**Code :**

**Notifier.java**

**package decorator;**

**public interface Notifier {**

**void send(String message);**

**}**

**EmailNotifier.java**

**package decorator;**

**public class EmailNotifier implements Notifier {**

**@Override**

**public void send(String message) {**

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

**}**

**}**

**NotifierDecorator.java**

**package** decorator;

**public** **abstract** **class** NotifierDecorator **implements** Notifier {

**protected** Notifier wrappedNotifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.wrappedNotifier = notifier;

}

@Override

**public** **void** send(String message) {

wrappedNotifier.send(message);

}

}

**SMSNotifierDecorator.java**

**package decorator;**

**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** decorator;

**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);

}

}

**DecoratorTest.java**

**package decorator;**

**public class DecoratorTest {**

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

**Notifier baseNotifier = new EmailNotifier();**

**Notifier smsNotifier = new SMSNotifierDecorator(baseNotifier);**

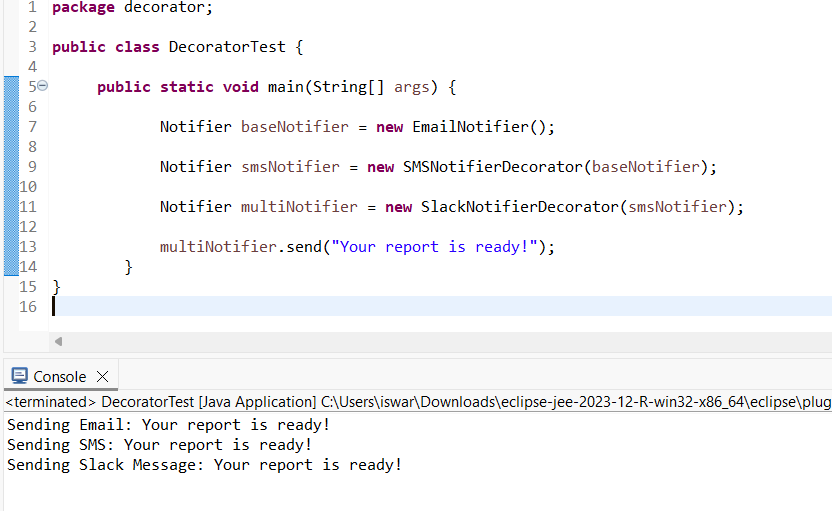
**Notifier multiNotifier = new SlackNotifierDecorator(smsNotifier);**

**multiNotifier.send("Your report is ready!");**

**}**

**}**

**Output :**

****

**Exercise 6: Implementing the Proxy Pattern**

**Project Structure :**

ProxyPatternExample/

├── Image.java (Interface)

├── RealImage.java (Real Subject)

├── ProxyImage.java (Proxy)

└── ProxyTest.java

**Code :**

**Image.java**

**package proxy;**

**public interface Image {**

**void display();**

**}**

**RealImage.java**

**package proxy;**

**public class RealImage implements Image {**

**private String filename;**

**public RealImage(String filename) {**

**this.filename = filename;**

**loadFromServer();**

**}**

**private void loadFromServer() {**

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

**try {**

**Thread.*sleep*(2000);**

**} catch (InterruptedException e) {**

**Thread.*currentThread*().interrupt();**

**}**

**}**

**@Override**

**public void display() {**

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

**}**

**}**

**ProxyImage.java**

**package** proxy;

**public** **class** ProxyImage **implements** Image {

**private** String filename;

**private** RealImage realImage;

**public** ProxyImage(String filename) {

**this**.filename = filename;

}

@Override

**public** **void** display() {

**if** (realImage == **null**) {

System.***out***.println("Creating RealImage instance lazily...");

realImage = **new** RealImage(filename);

} **else** {

System.***out***.println("Using cached RealImage instance...");

}

realImage.display();

}

}

**ProxyTest.java**

**package proxy;**

**public class ProxyTest {**

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

**Image image1 = new ProxyImage("design-pattern.png");**

**image1.display();**

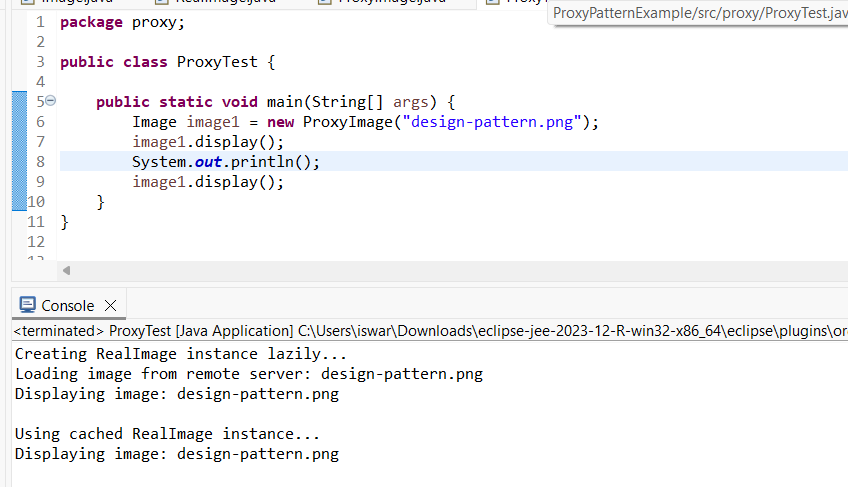
**System.*out*.println();**

**image1.display();**

**}**

**}**

**Output :**

****

**Exercise 7: Implementing the Observer Pattern**

**Project Structure :**

ObserverPatternExample/

├── Stock.java (Subject Interface)

├── Observer.java (Observer Interface)

├── StockMarket.java (Concrete Subject)

├── MobileApp.java (Observer)

├── WebApp.java (Observer)

└── ObserverTest.java

**Code :**

**Stock.java**

**package observer;**

**public interface Stock {**

**void registerObserver(Observer observer);**

**void removeObserver(Observer observer);**

**void notifyObservers();**

**}**

**Observer.java**

**package observer;**

**public interface Observer {**

**void update(String stockName, double stockPrice);**

**}**

**StockMarket.java**

**package** observer;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StockMarket **implements** Stock {

**private** List<Observer> observers;

**private** String stockName;

**private** **double** stockPrice;

**public** StockMarket() {

**this**.observers = **new** ArrayList<>();

}

@Override

**public** **void** registerObserver(Observer observer) {

observers.add(observer);

System.***out***.println(observer + " registered.");

}

@Override

**public** **void** removeObserver(Observer observer) {

observers.remove(observer);

System.***out***.println(observer + " removed.");

}

@Override

**public** **void** notifyObservers() {

**for** (Observer observer : observers) {

observer.update(stockName, stockPrice);

}

}

**public** **void** setStock(String name, **double** price) {

**this**.stockName = name;

**this**.stockPrice = price;

System.***out***.println("\nStock updated: " + name + " = ₹" + price);

notifyObservers();

}

}

**MobileApp.java**

**package observer;**

**public class MobileApp implements Observer {**

**private String user;**

**public MobileApp(String user) {**

**this.user = user;**

**}**

**@Override**

**public void update(String stockName, double stockPrice) {**

**System.*out*.println("MobileApp [" + user + "] - Stock Update: " + stockName + " = ₹" + stockPrice);**

**}**

**@Override**

**public String toString() {**

**return "MobileApp[" + user + "]";**

**}**

**}**

**WebApp.java**

**package observer;**

**public class WebApp implements Observer {**

**private String sessionId;**

**public WebApp(String sessionId) {**

**this.sessionId = sessionId;**

**}**

**@Override**

**public void update(String stockName, double stockPrice) {**

**System.*out*.println("WebApp [Session: " + sessionId + "] - Stock Update: " + stockName + " = ₹" + stockPrice);**

**}**

**@Override**

**public String toString() {**

**return "WebApp[Session:" + sessionId + "]";**

**}**

**}**

**ObserverTest.java**

**package observer;**

**public class ObserverTest {**

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

**StockMarket market = new StockMarket();**

**Observer mobile1 = new MobileApp("UserA");**

**Observer mobile2 = new MobileApp("UserB");**

**Observer web1 = new WebApp("Session101");**

**market.registerObserver(mobile1);**

**market.registerObserver(mobile2);**

**market.registerObserver(web1);**

**market.setStock("TCS", 3870.25);**

**market.setStock("Infosys", 1480.60);**

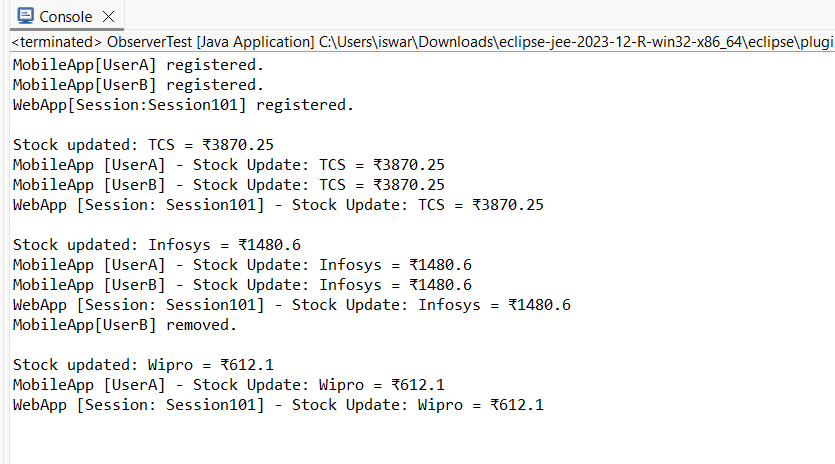
**market.removeObserver(mobile2);**

**market.setStock("Wipro", 612.10);**

**}**

**}**

**Output :**

****

**Exercise 8: Implementing the Strategy Pattern**

**Project Structure :**

StrategyPatternExample/

├── PaymentStrategy.java (Strategy Interface)

├── CreditCardPayment.java (Concrete Strategy)

├── PayPalPayment.java (Concrete Strategy)

├── PaymentContext.java (Context)

└── StrategyTest.java

**Code :**

**PaymentStrategy.java**

**package strategy;**

**public interface PaymentStrategy {**

**void pay(double amount);**

**}**

**CreditCardPayment.java**

**package strategy;**

**public class CreditCardPayment implements PaymentStrategy {**

**private String cardNumber;**

**public CreditCardPayment(String cardNumber) {**

**this.cardNumber = cardNumber;**

**}**

**@Override**

**public void pay(double amount) {**

**System.*out*.println("Paid ₹" + amount + " using Credit Card (\*\*\*\* " + cardNumber.substring(cardNumber.length() - 4) + ").");**

**}**

**}**

**PayPalPayment.java**

**package strategy;**

**public class PayPalPayment implements PaymentStrategy {**

**private String email;**

**public PayPalPayment(String email) {**

**this.email = email;**

**}**

**@Override**

**public void pay(double amount) {**

**System.*out*.println("Paid ₹" + amount + " using PayPal (account: " + email + ").");**

**}**

**}**

**package strategy;**

**public class PaymentContext {**

**private PaymentStrategy strategy;**

**public void setPaymentStrategy(PaymentStrategy strategy) {**

**this.strategy = strategy;**

**}**

**public void executePayment(double amount) {**

**if (strategy == null) {**

**System.*out*.println("Payment strategy not set!");**

**return;**

**}**

**strategy.pay(amount);**

**}**

**}**

**StrategyTest.java**

**package strategy;**

**public class StrategyTest {**

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

**PaymentContext context = new PaymentContext();**

**context.setPaymentStrategy(new CreditCardPayment("1234567812345678"));**

**context.executePayment(1500.00);**

**System.*out*.println();**

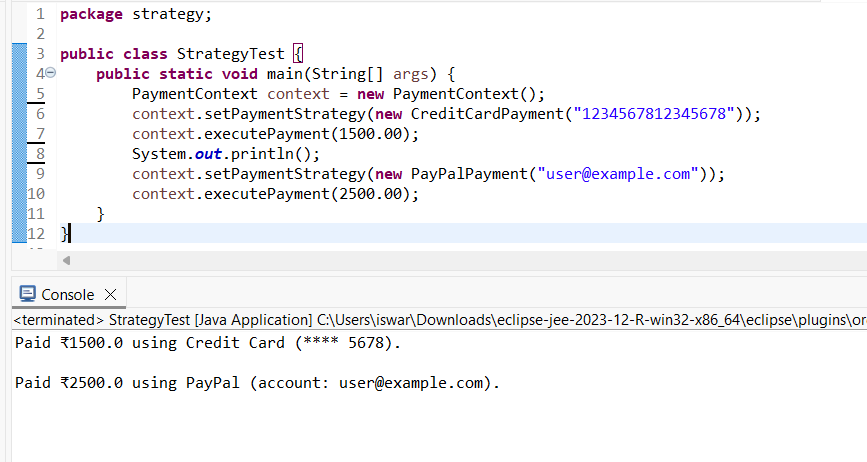
**context.setPaymentStrategy(new PayPalPayment("user@example.com"));**

**context.executePayment(2500.00);**

**}**

**}**

**Output :**

****

**Exercise 9: Implementing the Command Pattern**

**Project Structure :**

CommandPatternExample/

├── Command.java (Interface)

├── LightOnCommand.java (Concrete Command)

├── LightOffCommand.java (Concrete Command)

├── Light.java (Receiver)

├── RemoteControl.java (Invoker)

└── CommandTest.java

**Code :**

**Command.java**

**package command;**

**public interface Command {**

**void execute();**

**}**

**LightOnCommand.java**

**package command;**

**public class LightOnCommand implements Command {**

**private Light light;**

**public LightOnCommand(Light light) {**

**this.light = light;**

**}**

**@Override**

**public void execute() {**

**light.turnOn();**

**}**

**}**

**LightOfCommand.java**

**package command;**

**public class LightOfCommand implements Command {**

**private Light light;**

**public LightOfCommand(Light light) {**

**this.light = light;**

**}**

**@Override**

**public void execute() {**

**light.turnOff();**

**}**

**}**

**Light.java**

**package command;**

**public class Light {**

**public void turnOn() {**

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

**}**

**public void turnOff() {**

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

**}**

**}**

**RemoteControl.java**

**package command;**

**public class RemoteControl {**

**private Command command;**

**public void setCommand(Command command) {**

**this.command = command;**

**}**

**public void pressButton() {**

**if (command != null) {**

**command.execute();**

**} else {**

**System.*out*.println("No command assigned.");**

**}**

**}**

**}**

**CommandTest.java**

**package command;**

**public class RemoteControl {**

**private Command command;**

**public void setCommand(Command command) {**

**this.command = command;**

**}**

**public void pressButton() {**

**if (command != null) {**

**command.execute();**

**} else {**

**System.*out*.println("No command assigned.");**

**}**

**}**

**}**

**Output :**

****

**Exercise 10: Implementing the MVC Pattern**

**Project Structure :**

MVCPatternExample/

├── Student.java (Model)

├── StudentView.java (View)

├── StudentController.java (Controller)

└── MVCExampleTest.java

**Code :**

**Student.java**

**package mvc;**

**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; }**

**}**

**StudentView.java**

**package mvc;**

**public class StudentView {**

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

**System.*out*.println("\n--- Student Details ---");**

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

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

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

**}**

**}**

**StudentController.java**

**package** mvc;

**public** **class** StudentController {

**private** Student student;

**private** StudentView view;

**public** StudentController(Student student, StudentView view) {

**this**.student = student;

**this**.view = view;

}

// Getters

**public** String getStudentName() { **return** student.getName(); }

**public** String getStudentId() { **return** student.getId(); }

**public** String getStudentGrade() { **return** student.getGrade(); }

// Setters

**public** **void** setStudentName(String name) { student.setName(name); }

**public** **void** setStudentGrade(String grade) { student.setGrade(grade); }

// Update View

**public** **void** updateView() {

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

}

}

**MVCExampleTest.java**

**package mvc;**

**public class MVCExampleTest {**

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

**Student student = new Student("Iswarya", "S12345", "A");**

**StudentView view = new StudentView();**

**StudentController controller = new StudentController(student, view);**

**controller.updateView();**

**controller.setStudentName("Iswarya Lakshmi");**

**controller.setStudentGrade("A+");**

**controller.updateView();**

**}**

**}**

**Output :**

****

**Exercise 11: Implementing the Dependency Injection**

**Project Structure :**

DependencyInjectionExample/

├── CustomerRepository.java (Interface)

├── CustomerRepositoryImpl.java (Concrete Repository)

├── CustomerService.java (Service using Constructor DI)

└── DIExampleTest.java (Main/Test)

**Code :**

**CustomerRepository.java**

**package di;**

**public interface CustomerRepository {**

**String findCustomerById(String customerId);**

**}**

**CustomerRepositoryImpl.java**

**package di;**

**public class CustomerRepositoryImpl implements CustomerRepository {**

**@Override**

**public String findCustomerById(String customerId) {**

**return "Customer[ID=" + customerId + ", Name=Iswarya Lakshmi]";**

**}**

**}**

**CustomerService.java**

**package di;**

**public class CustomerService {**

**private final CustomerRepository customerRepository;**

**public CustomerService(CustomerRepository customerRepository) {**

**this.customerRepository = customerRepository;**

**}**

**public void getCustomerDetails(String customerId) {**

**String details = customerRepository.findCustomerById(customerId);**

**System.*out*.println("Fetched from service: " + details);**

**}**

**}**

**DIExampleTest.java**

**package di;**

**public class DIExampleTest {**

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

**CustomerRepository repository = new CustomerRepositoryImpl();**

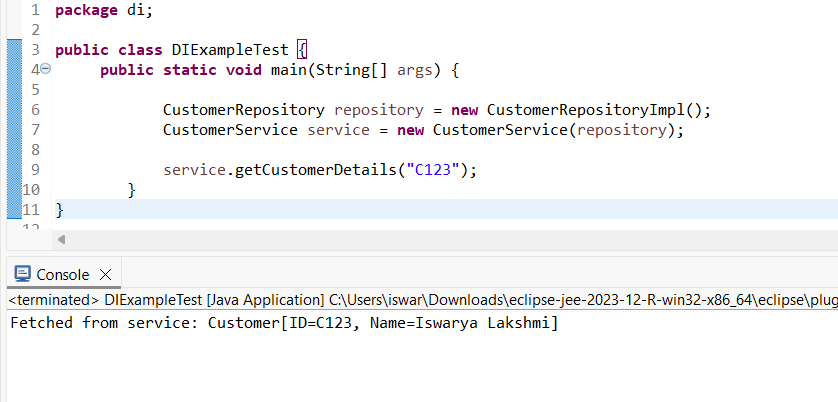
**CustomerService service = new CustomerService(repository);**

**service.getCustomerDetails("C123");**

**}**

**}**

**Output :**

****