**Exercise 1: Implementing the Singleton Pattern**

**Code:  
=>** Logger.java

**package** ex\_no1;

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

// Step 1: Create a private static instance of the same class

**private** **static** Logger *instance*;

// Step 2: Private constructor to prevent external instantiation

**private** Logger() {

System.***out***.println("Logger initialized.");

}

// Step 3: Public static method to provide access to the instance

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

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

*instance* = **new** Logger(); // lazy initialization

}

**return** *instance*;

}

// Example logging method

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

System.***out***.println("[LOG] " + message);

}

}

=>Main.java

**package** ex\_no1;

**public** **class** Main {

**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.");

// Check if both references point to the same object

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

System.***out***.println("Both logger instances are the same (Singleton confirmed).");

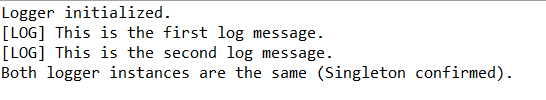
} **else** {

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

}

}

}

**Output:  
**

****

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

**Code:**

=> Document.java (interface)

**package** ex\_no2;

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

**void** open();

}

=> WordDocument.java

**package** ex\_no2;

**public** **class** WordDocument **implements** Document {

@Override

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

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

}

}

=> ExcelDocument.java

**package** ex\_no2;

**public** **class** ExcelDocument **implements** Document {

@Override

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

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

}

}

=> PdfDocument.java

**package** ex\_no2;

**public** **class** PdfDocument **implements** Document {

@Override

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

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

}

}

=> DocumentFactory.java

**package** ex\_no2;

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

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

}

=> WordFactory.java

**package** ex\_no2;

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

@Override

**public** Document createDocument() {

**return** **new** WordDocument();

}

}

=> PdfFactory.java

**package** ex\_no2;

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

@Override

**public** Document createDocument() {

**return** **new** PdfDocument();

}

}

=> ExcelFactory.java

**package** ex\_no2;

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

@Override

**public** Document createDocument() {

**return** **new** ExcelDocument();

}

}

=> Main.java

**package** ex\_no2;

**public** **class** Main {

**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();

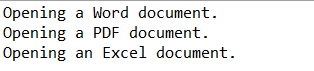
DocumentFactory excelFactory = **new** ExcelFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

} }

**Output:**



**Exercise 3: Implementing the Builder Pattern**

**Code:**

**=>** Computer.java

**package** ex\_no3;

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

// Required attributes

**private** String cpu;

**private** String ram;

// Optional attributes

**private** String storage;

**private** String graphicsCard;

**private** **boolean** isBluetoothEnabled;

// Private constructor

**private** Computer(Builder builder) {

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

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

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

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

**this**.isBluetoothEnabled = builder.isBluetoothEnabled;

}

// Nested static Builder class

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

**private** String cpu;

**private** String ram;

**private** String storage;

**private** String graphicsCard;

**private** **boolean** isBluetoothEnabled;

**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 setBluetoothEnabled(**boolean** isBluetoothEnabled) {

**this**.isBluetoothEnabled = isBluetoothEnabled;

**return** **this**;

}

**public** Computer build() {

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

}

}

@Override

**public** String toString() {

**return** "Computer [CPU=" + cpu + ", RAM=" + ram + ", Storage=" + storage +

", GraphicsCard=" + graphicsCard + ", Bluetooth=" + isBluetoothEnabled + "]";

}

}

=> Main.java

**package** ex\_no3;

**public** **class** Main {

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

// Basic configuration

Computer basicPC = **new** Computer.Builder("Intel i3", "8GB").build();

// Gaming configuration

Computer gamingPC = **new** Computer.Builder("Intel i9", "32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.setBluetoothEnabled(**true**)

.build();

System.***out***.println("Basic PC: " + basicPC);

System.***out***.println("Gaming PC: " + gamingPC);

}

}

**Output:**



**Exercise 4: Implementing the Adapter Pattern**

**Code:**

**=>** PaymentProcessor.java

**package ex\_no4;**

**public interface PaymentProcessor {**

**void processPayment(double amount);**

**}**

**=>** StripeGateway.java

**package** ex\_no4;

**public** **class** StripeGateway {

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

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

}

}

**=>** PayPalGateway.java

**package** ex\_no4;

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

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

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

}

}

**=>** StripeAdapter.java

**package** ex\_no4;

**public** **class** StripeAdapter **implements** PaymentProcessor {

**private** StripeGateway stripe;

**public** StripeAdapter(StripeGateway stripe) {

**this**.stripe = stripe;

}

@Override

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

stripe.makeStripePayment(amount);

}

}

=> PayPalAdapter.java

**package** ex\_no4;

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

**private** PayPalGateway paypal;

**public** PayPalAdapter(PayPalGateway paypal) {

**this**.paypal = paypal;

}

@Override

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

paypal.sendPayPalPayment(amount);

}

}

=> Main.java

**package** ex\_no4;

**public** **class** Main {

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

// Using Stripe via Adapter

StripeGateway stripe = **new** StripeGateway();

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

stripeProcessor.processPayment(250.00);

// Using PayPal via Adapter

PayPalGateway paypal = **new** PayPalGateway();

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

paypalProcessor.processPayment(500.00);

}

}

**Output:**

****

**Exercise 5: Implementing the Decorator Pattern**

**Code:**

**=>** Notifier.java

**package** ex\_no5;

**public** **interface** Notifier {

**void** send(String message);

}

=> EmailNotifier.java  
**package** ex\_no5;

**public** **class** EmailNotifier **implements** Notifier {

@Override

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

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

}

}

=> Notifier Decorator.java

**package** ex\_no5;

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

**protected** Notifier notifier;

**public** NotifierDecorator(Notifier notifier) {

**this**.notifier = notifier;

}

@Override

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

notifier.send(message);

}

}

=> SlackNotifierDecorator.java

**package** ex\_no5;

**public** **class** SlackNotifierDecorator **extends** NotifierDecorator {

**public** SlackNotifierDecorator(Notifier notifier) {

**super**(notifier);

}

@Override

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

**super**.send(message);

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

}

}

=> Main.java

**package** ex\_no5;

**public** **class** Main {

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

// Basic Email notification

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

// Add SMS notification on top of Email

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

// Add Slack on top of Email + SMS

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

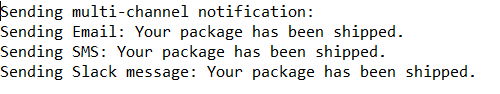
System.***out***.println("Sending multi-channel notification:");

multiChannelNotifier.send("Your package has been shipped.");

}

}

**Output:**

****

**Exercise 6: Implementing the Proxy Pattern**

**Code:**

**=>** Image.java

**package ex\_no6;**

**public interface Image {**

**void display();**

**}**

**=>** RealImage.java

**package** ex\_no6;

**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);

}

@Override

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

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

}

}

=> ProxyImage.java

**package** ex\_no6;

**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**) {

realImage = **new** RealImage(filename); // Lazy initialization

}

realImage.display();

}

}

=> Main.java

**package** ex\_no6;

**public** **class** Main {

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

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

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

// Image is not loaded yet

System.***out***.println("First call to image1:");

image1.display(); // Loads from server

System.***out***.println("\nSecond call to image1:");

image1.display(); // Uses cached image

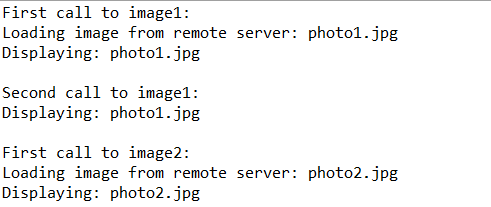
System.***out***.println("\nFirst call to image2:");

image2.display(); // Loads from server

}

}

**Output:**

****

**Exercise 7: Implementing the Observer Pattern**

**Code:**

**=>**Stock.java

**package** ex\_no7;

**public** **interface** Stock {

**void** registerObserver(Observer o);

**void** removeObserver(Observer o);

**void** notifyObservers();

}

**=>**StockMarket.java

**package** ex\_no7;

**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** newPrice) {

System.***out***.println("Stock price updated to $" + newPrice);

**this**.stockPrice = newPrice;

notifyObservers();

}

@Override

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

observers.add(o);

}

@Override

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

observers.remove(o);

}

@Override

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

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

observer.update(stockPrice);

}

}

}

**=>**StockMarket.java

**package** ex\_no7;

**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** newPrice) {

System.***out***.println("Stock price updated to $" + newPrice);

**this**.stockPrice = newPrice;

notifyObservers();

}

@Override

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

observers.add(o);

}

@Override

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

observers.remove(o);

}

@Override

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

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

observer.update(stockPrice);

}

}

}

=> Observer.java

**package** ex\_no7;

**public** **interface** Observer {

**void** update(**double** stockPrice);

}

=> MobileApp.java

**package** ex\_no7;

**public** **class** MobileApp **implements** Observer {

**private** String appId;

**public** MobileApp(String appId) {

**this**.appId = appId;

}

@Override

**public** **void** update(**double** stockPrice) {

System.***out***.println("MobileApp " + appId + " received stock update: $" + stockPrice);

}

}

=> WebApp.java

**package** ex\_no7;

**public** **class** WebApp **implements** Observer {

**private** String browser;

**public** WebApp(String browser) {

**this**.browser = browser;

}

@Override

**public** **void** update(**double** stockPrice) {

System.***out***.println("WebApp (" + browser + ") received stock update: $" + stockPrice);

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

}

}

=>Main.java

**package** ex\_no7;

**public** **class** Main {

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

StockMarket stockMarket = **new** StockMarket();

Observer mobileApp = **new** MobileApp("App001");

Observer webApp = **new** WebApp("Chrome");

stockMarket.registerObserver(mobileApp);

stockMarket.registerObserver(webApp);

stockMarket.setStockPrice(120.75);

stockMarket.setStockPrice(125.00);

// Remove one observer

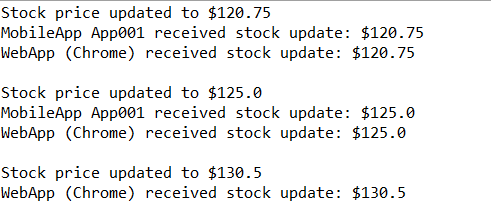
stockMarket.removeObserver(mobileApp);

stockMarket.setStockPrice(130.50);

}

}

**Output:**

****

**Exercise 8: Implementing the Strategy Pattern**

**Code:**

**=>**PaymentStrategy.java

**package** ex\_no8;

**public** **interface** PaymentStrategy {

**void** pay(**double** amount);

}

=>CreditCardPayment.java

**package** ex\_no8;

**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 ending with " + cardNumber.substring(cardNumber.length() - 4));

}

}

=>PayPalPayment.java

**package** ex\_no8;

**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);

}

}

=>PaymentContext.java

**package** ex\_no8;

**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("No payment strategy selected.");

} **else** {

strategy.pay(amount);

}

}

}

=>Main.java

**package** ex\_no8;

**public** **class** Main {

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

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

// Using Credit Card Payment

context.setPaymentStrategy(**new** CreditCardPayment("1234567890123456"));

context.executePayment(250.00);

// Switching to PayPal Payment

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

context.executePayment(180.50);

}

}

**Output:**

****

**Exercise 9: Implementing the Command Pattern**

**Coding:**

=>Command.java

**package** ex\_no9;

**public** **interface** Command {

**void** execute();

}

=>LightOnCommand.java

**package** ex\_no9;

**public** **class** LightOnCommand **implements** Command {

**private** Light light;

**public** LightOnCommand(Light light) {

**this**.light = light;

}

@Override

**public** **void** execute() {

light.turnOn();

}

}

=>LightOffCommand.java

**package** ex\_no9;

**public** **class** LightOffCommand **implements** Command {

**private** Light light;

**public** LightOffCommand(Light light) {

**this**.light = light;

}

@Override

**public** **void** execute() {

light.turnOff();

}

}

=>Light.java

**package** ex\_no9;

**public** **class** Light {

**public** **void** turnOn() {

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

}

**public** **void** turnOff() {

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

}

}

=>RemoteControl.java

**package** ex\_no9;

**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 set.");

}

}

}

=>Main.java

**package** ex\_no9;

**public** **class** Main {

**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("Press ON button:");

remote.setCommand(lightOn);

remote.pressButton();

System.***out***.println("Press OFF button:");

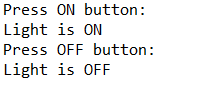
remote.setCommand(lightOff);

remote.pressButton();

}

}

**Output:**

****

**Exercise 10: Implementing the MVC Pattern**

**Coding:**

**=>**Student.java

**package** ex\_no10;

**public** **class** Student {

**private** String name;

**private** String id;

**private** String grade;

// Constructor

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

**this**.name = name;

**this**.id = id;

**this**.grade = grade;

}

// Getters and Setters

**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** ex\_no10;

**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);

}

}

=>StudentController.java

**package** ex\_no10;

**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** setStudentGrade(String grade) {

model.setGrade(grade);

}

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

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

}

}

=>Main.java

**package** ex\_no10;

**public** **class** Main {

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

// Model

Student student = **new** Student("Alice", "STU101", "A");

// View

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

// Controller

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

// Initial display

controller.updateView();

// Update student data via controller

controller.setStudentName("Alice Johnson");

controller.setStudentGrade("A+");

// Display updated data

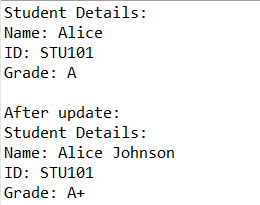
System.***out***.println("\nAfter update:");

controller.updateView();

}

}

**Output:**

****

**Exercise 11: Implementing Dependency Injection**

**Coding:**

**=>** CustomerRepository.java

**package** ex\_no11;

**public** **interface** CustomerRepository {

String findCustomerById(String customerId);

}

=> CustomerRepositoryImpl.java

**package** ex\_no11;

**public** **class** CustomerRepositoryImpl **implements** CustomerRepository {

@Override

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

// Simulated data retrieval

**return** "Customer [ID=" + customerId + ", Name=John Doe]";

}

}

=> CustomerService.java

**package** ex\_no11;

**public** **class** CustomerService {

**private** CustomerRepository customerRepository;

// Constructor Injection

**public** CustomerService(CustomerRepository customerRepository) {

**this**.customerRepository = customerRepository;

}

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

String customer = customerRepository.findCustomerById(customerId);

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

}

}

=> Main.java

**package** ex\_no11;

**public** **class** Main {

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

// Manually inject dependency

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

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

// Use service to fetch customer

service.displayCustomer("CUST001");

}

}

**Output:**

****