**EXERCISE 1 IMPLEMENTING THE SINGLETON PATTERN**

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("Both logger1 and logger2 are the same instance.");

        } else {

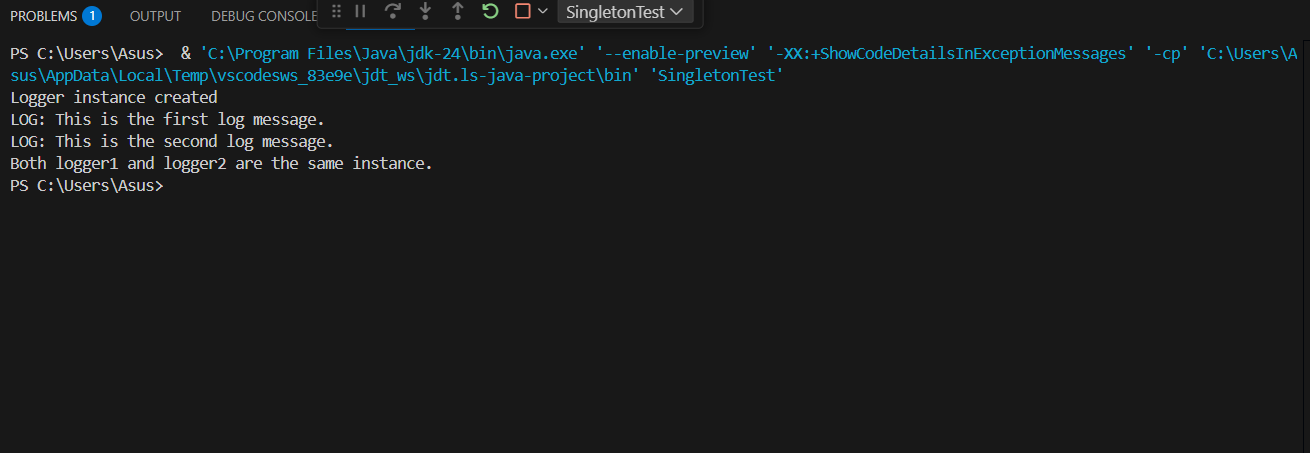
            System.out.println("logger1 and logger2 are different instances.");

        }

    }

}

**OUTPUT:**

****

**EXERCISE 2(IMPLEMENTING THE FACTORY METHOD PATTERN)**

public class FactoryMethodPatternExample {

    public interface Document {

        void open();

        void save();

        void close();

    }

    public static class WordDocument implements Document {

        @Override

        public void open() {

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

        }

        @Override

        public void save() {

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

        }

        @Override

        public void close() {

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

        }

    }

    public static class PdfDocument implements Document {

        @Override

        public void open() {

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

        }

        @Override

        public void save() {

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

        }

        @Override

        public void close() {

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

        }

    }

    public static class ExcelDocument implements Document {

        @Override

        public void open() {

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

        }

        @Override

        public void save() {

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

        }

        @Override

        public void close() {

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

        }

    }

    public abstract static class DocumentFactory {

        public abstract Document createDocument();

        public void performDocumentOperations() {

            Document doc = createDocument();

            doc.open();

            doc.save();

            doc.close();

        }

    }

    public static class WordDocumentFactory extends DocumentFactory {

        @Override

        public Document createDocument() {

            return new WordDocument();

        }

    }

    public static class PdfDocumentFactory extends DocumentFactory {

        @Override

        public Document createDocument() {

            return new PdfDocument();

        }

    }

    public static class ExcelDocumentFactory extends DocumentFactory {

        @Override

        public Document createDocument() {

            return new ExcelDocument();

        }

    }

    public static class FactoryMethodTest {

        public static void main(String[] args) {

            DocumentFactory wordFactory = new WordDocumentFactory();

            System.out.println("Using Word Document Factory:");

            wordFactory.performDocumentOperations();

            System.out.println();

            DocumentFactory pdfFactory = new PdfDocumentFactory();

            System.out.println("Using PDF Document Factory:");

            pdfFactory.performDocumentOperations();

            System.out.println();

            DocumentFactory excelFactory = new ExcelDocumentFactory();

            System.out.println("Using Excel Document Factory:");

            excelFactory.performDocumentOperations();

        }

    }

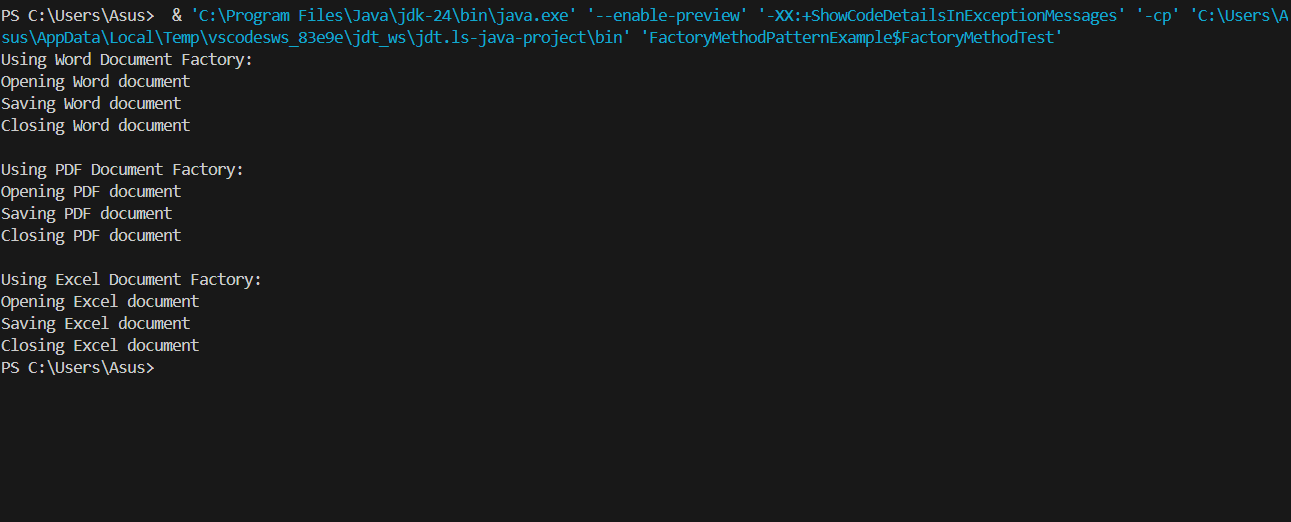
    public static void main(String[] args) {

        FactoryMethodTest.main(args);

    }

}

**OUTPUT:**

****

**EXERCISE 3**

public class BuilderPatternExample {

    static class Computer {

        private final String CPU;

        private final String RAM;

        private final String storage;

        private final String GPU;

        private final String powerSupply;

        private final String coolingSystem;

        private final String caseType;

        private Computer(Builder builder) {

            this.CPU = builder.CPU;

            this.RAM = builder.RAM;

            this.storage = builder.storage;

            this.GPU = builder.GPU;

            this.powerSupply = builder.powerSupply;

            this.coolingSystem = builder.coolingSystem;

            this.caseType = builder.caseType;

        }

        public String getCPU() { return CPU; }

        public String getRAM() { return RAM; }

        public String getStorage() { return storage; }

        public String getGPU() { return GPU; }

        public String getPowerSupply() { return powerSupply; }

        public String getCoolingSystem() { return coolingSystem; }

        public String getCaseType() { return caseType; }

        @Override

        public String toString() {

            return "Computer{" +

                    "CPU='" + CPU + '\'' +

                    ", RAM='" + RAM + '\'' +

                    ", storage='" + storage + '\'' +

                    ", GPU='" + GPU + '\'' +

                    ", powerSupply='" + powerSupply + '\'' +

                    ", coolingSystem='" + coolingSystem + '\'' +

                    ", caseType='" + caseType + '\'' +

                    '}';

        }

        public static class Builder {

         private final String CPU;

            private final String RAM;

            private String storage = "No storage specified";

            private String GPU = "No GPU specified";

            private String powerSupply = "No power supply specified";

            private String coolingSystem = "No cooling system specified";

            private String caseType = "No case type specified";

            public Builder(String CPU, String RAM) {

                this.CPU = CPU;

                this.RAM = RAM;

            }

            public Builder storage(String storage) {

                this.storage = storage;

                return this;

            }

            public Builder GPU(String GPU) {

                this.GPU = GPU;

                return this;

            }

            public Builder powerSupply(String powerSupply) {

                this.powerSupply = powerSupply;

                return this;

            }

            public Builder coolingSystem(String coolingSystem) {

                this.coolingSystem = coolingSystem;

                return this;

            }

            public Builder caseType(String caseType) {

                this.caseType = caseType;

                return this;

            }

            public Computer build() {

                return new Computer(this);

            }

        }

    }

    static class ComputerTest {

        public static void main(String[] args) {

            Computer basicComputer = new Computer.Builder("Intel i3", "8GB")

                    .build();

            System.out.println("Basic Computer: " + basicComputer);

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

                    .storage("1TB SSD")

                    .GPU("NVIDIA RTX 3080")

                    .powerSupply("750W")

                    .coolingSystem("Liquid cooling")

                    .caseType("Mid-tower")

                    .build();

            System.out.println("\nGaming Computer: " + gamingComputer);

            Computer workstation = new Computer.Builder("AMD Ryzen 9", "64GB")

                    .storage("2TB NVMe SSD")

                    .GPU("NVIDIA RTX 3090")

                    .powerSupply("1000W")

                    .build();

            System.out.println("\nWorkstation: " + workstation);

            Computer customComputer = new Computer.Builder("Intel i7", "16GB")

                    .storage("512GB SSD + 2TB HDD")

                    .coolingSystem("Air cooling")

                    .caseType("Full-tower")

                    .build();

            System.out.println("\nCustom Computer: " + customComputer);

        }

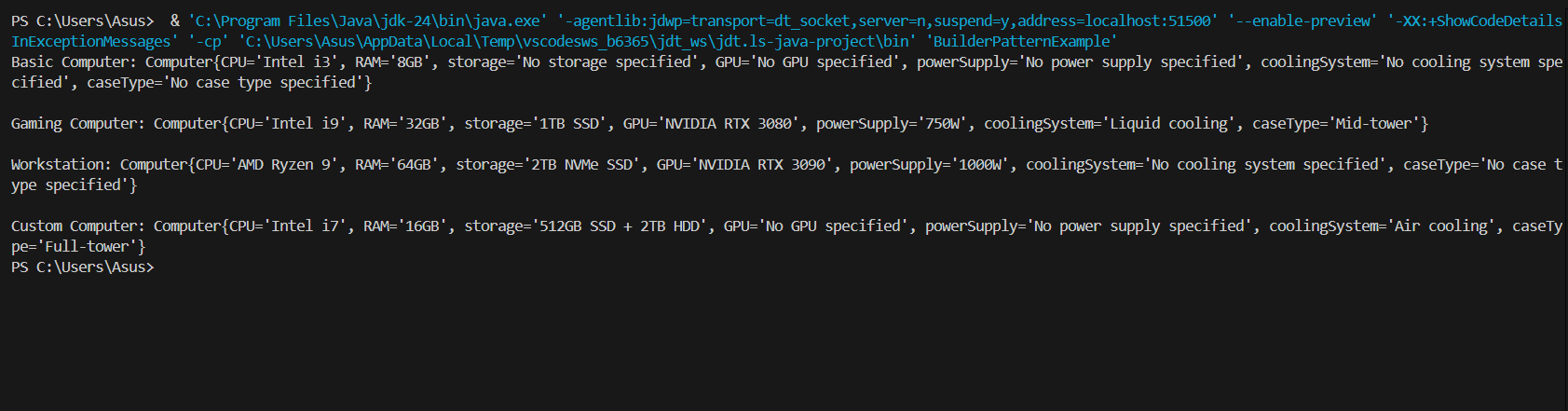
    }

    public static void main(String[] args) {

        ComputerTest.main(args);

    } }

**OUTPUT:**

****

**EXERCISE 4**

public class AdapterPatternExample {

    public interface PaymentProcessor {

        void processPayment(double amount);

        String getProcessorName();

    }

        public static class PayPalGateway {

        public void makePayment(double amount) {

            System.out.println("Processing payment of $" + amount + " through PayPal");

        }

        public String getName() {

            return "PayPal";

        }

    }

    public static class StripeGateway {

        public void charge(double amount) {

            System.out.println("Charging $" + amount + " through Stripe");

        }

        public String getProviderName() {

            return "Stripe";

        }

    }

    public static class AmazonPayGateway {

        public void executePayment(double amount) {

            System.out.println("Executing payment of $" + amount + " through Amazon Pay");

        }

        public String getGatewayName() {

            return "Amazon Pay";

        }

    }

        public static class PayPalAdapter implements PaymentProcessor {

        private final PayPalGateway payPalGateway;

        public PayPalAdapter(PayPalGateway payPalGateway) {

            this.payPalGateway = payPalGateway;

        }

        @Override

        public void processPayment(double amount) {

            payPalGateway.makePayment(amount);

        }

        @Override

        public String getProcessorName() {

            return payPalGateway.getName();

        }

    }

    public static class StripeAdapter implements PaymentProcessor {

        private final StripeGateway stripeGateway;

        public StripeAdapter(StripeGateway stripeGateway) {

            this.stripeGateway = stripeGateway;

        }

        @Override

        public void processPayment(double amount) {

            stripeGateway.charge(amount);

        }

        @Override

        public String getProcessorName() {

            return stripeGateway.getProviderName();

        }

    }

    public static class AmazonPayAdapter implements PaymentProcessor {

        private final AmazonPayGateway amazonPayGateway;

        public AmazonPayAdapter(AmazonPayGateway amazonPayGateway) {

            this.amazonPayGateway = amazonPayGateway;

        }

        @Override

        public void processPayment(double amount) {

            amazonPayGateway.executePayment(amount);

        }

        @Override

        public String getProcessorName() {

            return amazonPayGateway.getGatewayName();

        }

    }

        public static class PaymentProcessorTest {

        public static void main(String[] args) {

            PayPalGateway payPalGateway = new PayPalGateway();

            StripeGateway stripeGateway = new StripeGateway();

            AmazonPayGateway amazonPayGateway = new AmazonPayGateway();

            PaymentProcessor payPalAdapter = new PayPalAdapter(payPalGateway);

            PaymentProcessor stripeAdapter = new StripeAdapter(stripeGateway);

            PaymentProcessor amazonPayAdapter = new AmazonPayAdapter(amazonPayGateway);

            System.out.println("=== Processing Payments with Different Gateways ===");

            payPalAdapter.processPayment(100.50);

            System.out.println("Processor: " + payPalAdapter.getProcessorName());

            System.out.println("\n");

            stripeAdapter.processPayment(75.25);

            System.out.println("Processor: " + stripeAdapter.getProcessorName());

            System.out.println("\n");

            amazonPayAdapter.processPayment(200.00);

            System.out.println("Processor: " + amazonPayAdapter.getProcessorName());

            System.out.println("\n=== Unified Payment Processing ===");

            processPayment(payPalAdapter, 150.75);

            processPayment(stripeAdapter, 50.00);

            processPayment(amazonPayAdapter, 300.50);

        }

        private static void processPayment(PaymentProcessor processor, double amount) {

            System.out.println("\nProcessing payment of $" + amount + " using " + processor.getProcessorName());

            processor.processPayment(amount);

        }

    }

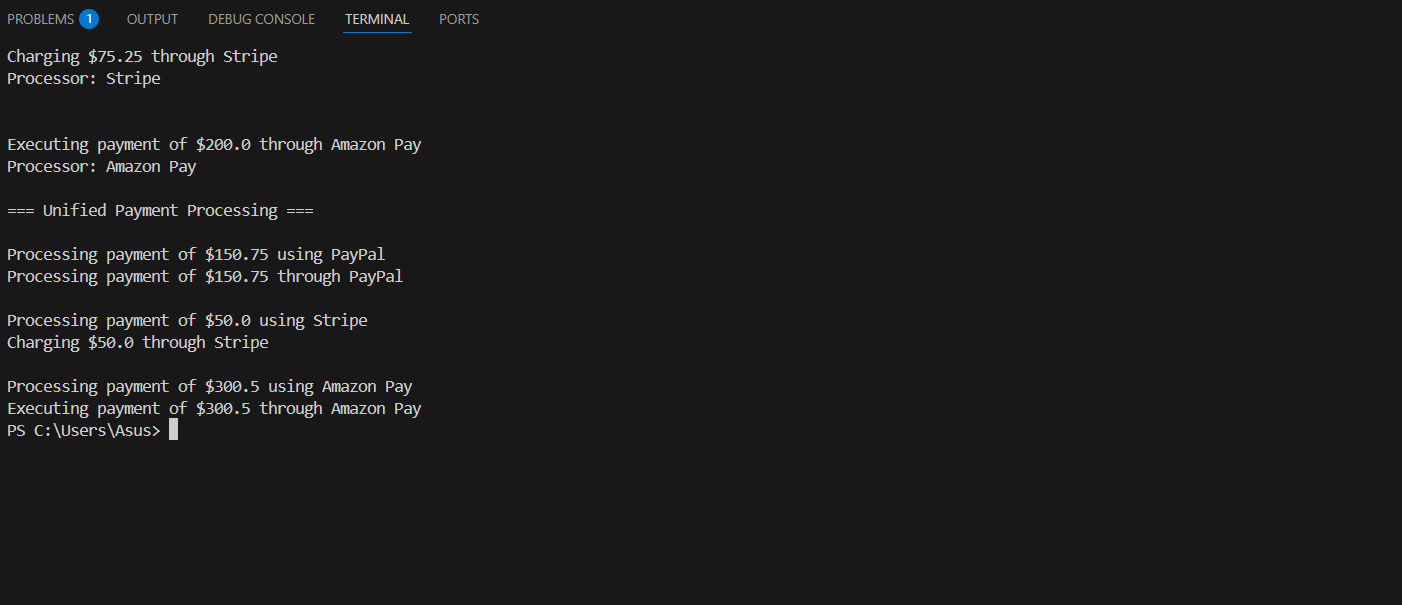
    public static void main(String[] args) {

        PaymentProcessorTest.main(args);

    }

}

**OUTPUT:**

****

**EXERCISE 5**

public class AdapterPatternExample {

    public interface PaymentProcessor {

        void processPayment(double amount);

        String getProcessorName();

    }

    public static class PayPalGateway {

        public void makePayment(double amount) {

            System.out.println("Processing payment of $" + amount + " through PayPal");

        }

        public String getName() {

            return "PayPal";

        }

    }

    public static class StripeGateway {

        public void charge(double amount) {

            System.out.println("Charging $" + amount + " through Stripe");

        }

        public String getProviderName() {

            return "Stripe";

        }

    }

    public static class AmazonPayGateway {

        public void executePayment(double amount) {

            System.out.println("Executing payment of $" + amount + " through Amazon Pay");

        }

        public String getGatewayName() {

            return "Amazon Pay";

        }

    }

    public static class PayPalAdapter implements PaymentProcessor {

        private final PayPalGateway payPalGateway;

        public PayPalAdapter(PayPalGateway payPalGateway) {

            this.payPalGateway = payPalGateway;

        }

        @Override

        public void processPayment(double amount) {

            payPalGateway.makePayment(amount);

        }

        @Override

        public String getProcessorName() {

            return payPalGateway.getName();

        }

    }

    public static class StripeAdapter implements PaymentProcessor {

        private final StripeGateway stripeGateway;

        public StripeAdapter(StripeGateway stripeGateway) {

            this.stripeGateway = stripeGateway;

        }

        @Override

        public void processPayment(double amount) {

            stripeGateway.charge(amount);

        }

        @Override

        public String getProcessorName() {

            return stripeGateway.getProviderName();

        }

    }

    public static class AmazonPayAdapter implements PaymentProcessor {

        private final AmazonPayGateway amazonPayGateway;

        public AmazonPayAdapter(AmazonPayGateway amazonPayGateway) {

            this.amazonPayGateway = amazonPayGateway;

        }

        @Override

        public void processPayment(double amount) {

            amazonPayGateway.executePayment(amount);

        }

        @Override

        public String getProcessorName() {

            return amazonPayGateway.getGatewayName();

        }

    }

    public static class PaymentProcessorTest {

        public static void main(String[] args) {

            PayPalGateway payPalGateway = new PayPalGateway();

            StripeGateway stripeGateway = new StripeGateway();

            AmazonPayGateway amazonPayGateway = new AmazonPayGateway();

            PaymentProcessor payPalAdapter = new PayPalAdapter(payPalGateway);

            PaymentProcessor stripeAdapter = new StripeAdapter(stripeGateway);

            PaymentProcessor amazonPayAdapter = new AmazonPayAdapter(amazonPayGateway);

            System.out.println("=== Processing Payments with Different Gateways ===");

            payPalAdapter.processPayment(100.50);

            System.out.println("Processor: " + payPalAdapter.getProcessorName());

            System.out.println("\n");

            stripeAdapter.processPayment(75.25);

            System.out.println("Processor: " + stripeAdapter.getProcessorName());

            System.out.println("\n");

            amazonPayAdapter.processPayment(200.00);

            System.out.println("Processor: " + amazonPayAdapter.getProcessorName());

            System.out.println("\n=== Unified Payment Processing ===");

            processPayment(payPalAdapter, 150.75);

            processPayment(stripeAdapter, 50.00);

            processPayment(amazonPayAdapter, 300.50);

        }

        private static void processPayment(PaymentProcessor processor, double amount) {

            System.out.println("\nProcessing payment of $" + amount + " using " + processor.getProcessorName());

            processor.processPayment(amount);

        }

    }

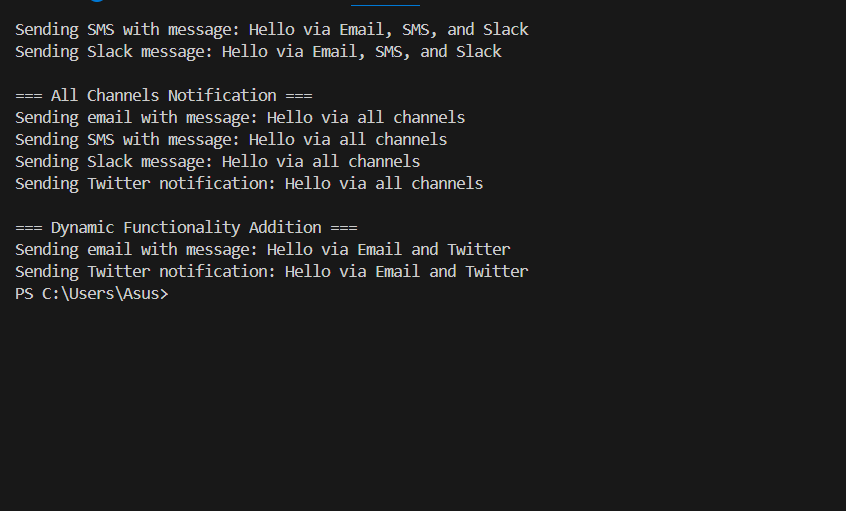
    public static void main(String[] args) {

        PaymentProcessorTest.main(args);

    }

}

**OUTPUT:**

****

**EXERCISE 6**

public class ProxyPatternExample {

    public interface Image {

        void display();

    }

    public static class RealImage implements Image {

        private final String filename;

        public RealImage(String filename) {

            this.filename = filename;

            loadFromServer();

        }

        private void loadFromServer() {

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

            try {

                Thread.sleep(2000);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            System.out.println("Image " + filename + " loaded successfully!");

        }

        @Override

        public void display() {

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

        }

    }

    public static class ProxyImage implements Image {

        private RealImage realImage;

        private final String filename;

        private boolean isCached = false;

        public ProxyImage(String filename) {

            this.filename = filename;

        }

        @Override

        public void display() {

            if (isCached) {

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

                return;

            }

            if (realImage == null) {

                realImage = new RealImage(filename);

            }

            realImage.display();

            isCached = true;

        }

        public void clearCache() {

            this.realImage = null;

            this.isCached = false;

            System.out.println("Cache cleared for image " + filename);

        }

    }

    public static class ImageTest {

        public static void main(String[] args) {

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

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

            Image image3 = new ProxyImage("image3.jpg");

            System.out.println("\n=== First display of image1 ===");

            image1.display();

            System.out.println("\n=== Second display of image1 (should use cache) ===");

            image1.display();

            System.out.println("\n=== First display of image2 ===");

            image2.display();

            System.out.println("\n=== First display of image3 ===");

            image3.display();

            System.out.println("\n=== Clearing cache for image1 ===");

            ((ProxyImage)image1).clearCache();

            System.out.println("\n=== Displaying image1 again after clearing cache ===");

            image1.display();

        }

    }

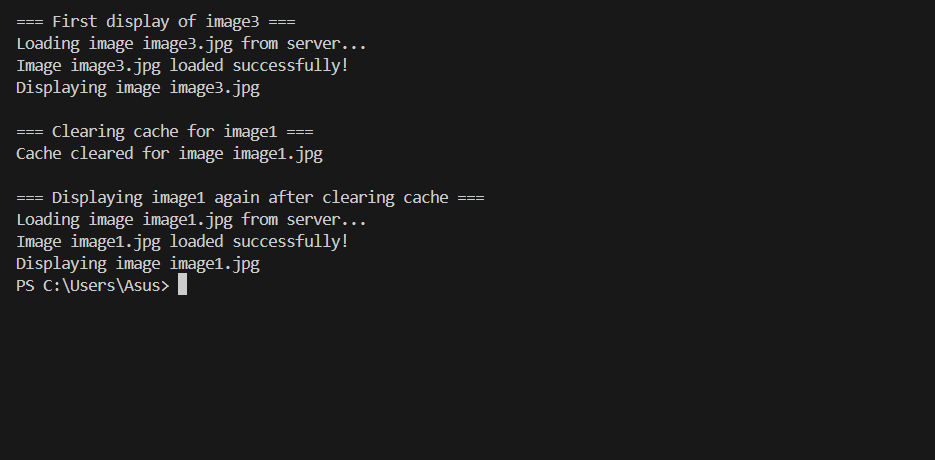
    public static void main(String[] args) {

        ImageTest.main(args);

    }

}

**OUTPUT:**

****

**EXERCISE 7**

import java.util.ArrayList;

import java.util.List;

interface Observer {

    void update(double stockPrice);

}

interface Stock {

    void registerObserver(Observer observer);

    void deregisterObserver(Observer observer);

    void notifyObservers();

    void setStockPrice(double price);

    double getStockPrice();

}

class MobileApp implements Observer {

    private String appName = "Mobile App";

    @Override

    public void update(double stockPrice) {

        System.out.println(appName + " notified. New stock price: $" + stockPrice);

    }

}

class WebApp implements Observer {

    private String appName = "Web App";

    @Override

    public void update(double stockPrice) {

        System.out.println(appName + " notified. New stock price: $" + stockPrice);

    }

}

class StockMarket implements Stock {

    private List<Observer> observers;

    private double stockPrice;

    public StockMarket() {

        observers = new ArrayList<>();

    }

    @Override

    public void registerObserver(Observer observer) {

        observers.add(observer);

    }

    @Override

    public void deregisterObserver(Observer observer) {

        observers.remove(observer);

    }

    @Override

    public void notifyObservers() {

        for (Observer observer : observers) {

            observer.update(stockPrice);

        }

    }

    @Override

    public void setStockPrice(double price) {

        this.stockPrice = price;

        notifyObservers();

    }

    @Override

    public double getStockPrice() {

        return stockPrice;

    }

}

public class ObserverPatternTest {

    public static void main(String[] args) {

        StockMarket stockMarket = new StockMarket();

        Observer mobileApp = new MobileApp();

        Observer webApp = new WebApp();

        stockMarket.registerObserver(mobileApp);

        stockMarket.registerObserver(webApp);

        System.out.println("Setting stock price to $100");

        stockMarket.setStockPrice(100.0);

        stockMarket.deregisterObserver(webApp);

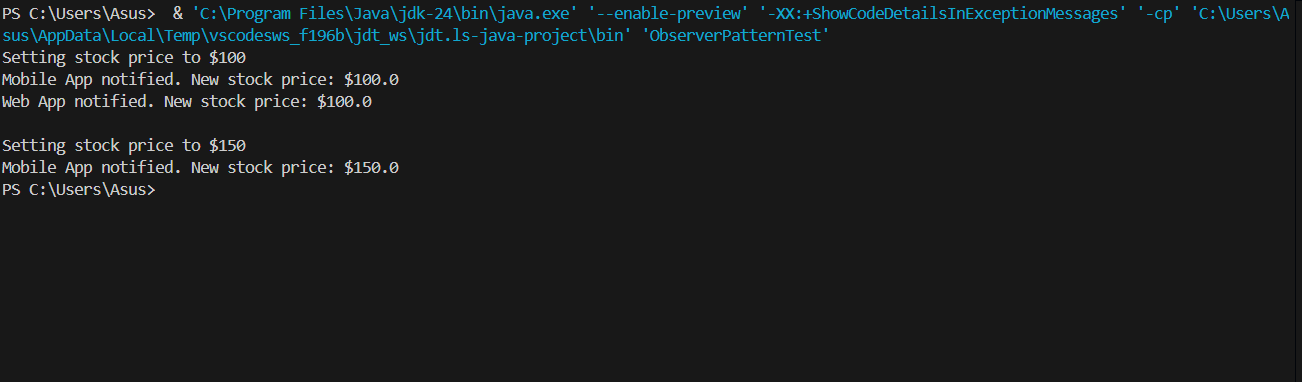
        System.out.println("\nSetting stock price to $150");

        stockMarket.setStockPrice(150.0);

    }

}

**OUTPUT:**

****

**EXERCISE 8**

interface PaymentStrategy {

    void pay(int amount);

}

class CreditCardPayment implements PaymentStrategy {

    private String cardNumber;

    private String name;

    private String cvv;

    private String dateOfExpiry;

    public CreditCardPayment(String cardNumber, String name, String cvv, String dateOfExpiry) {

        this.cardNumber = cardNumber;

        this.name = name;

        this.cvv = cvv;

        this.dateOfExpiry = dateOfExpiry;

    }

    @Override

    public void pay(int amount) {

        System.out.println(amount + " paid with credit card.");

    }

}

class PayPalPayment implements PaymentStrategy {

    private String email;

    private String password;

    public PayPalPayment(String email, String password) {

        this.email = email;

        this.password = password;

    }

    @Override

    public void pay(int amount) {

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

    }

}

class PaymentContext {

    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {

        this.paymentStrategy = paymentStrategy;

    }

    public void executePayment(int amount) {

        paymentStrategy.pay(amount);

    }

}

public class StrategyPatternExample {

    public static void main(String[] args) {

        PaymentContext context = new PaymentContext();

        System.out.println("Payment using Credit Card:");

        context.setPaymentStrategy(new CreditCardPayment("1234567890123456", "John Doe", "123", "12/23"));

        context.executePayment(100);

        System.out.println("\nPayment using PayPal:");

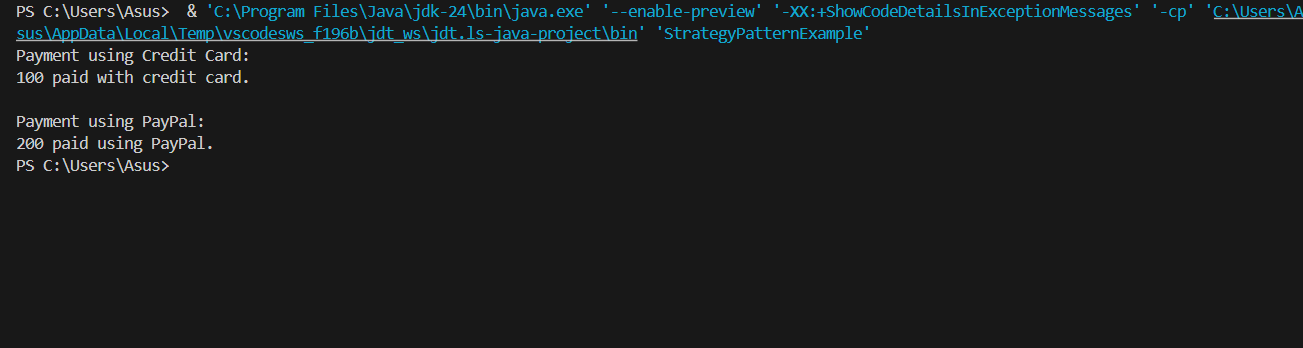
        context.setPaymentStrategy(new PayPalPayment("john@example.com", "password123"));

        context.executePayment(200);

    }

}

**OUTPUT:**

****

**EXERCISE 9**

import java.util.Scanner;

interface Command {

    void execute();

}

class Light {

    public void turnOn() {

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

    }

    public void turnOff() {

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

    }

}

class LightOnCommand implements Command {

    private Light light;

    public LightOnCommand(Light light) {

        this.light = light;

    }

    @Override

    public void execute() {

        light.turnOn();

    }

}

class LightOffCommand implements Command {

    private Light light;

    public LightOffCommand(Light light) {

        this.light = light;

    }

    @Override

    public void execute() {

        light.turnOff();

    }

}

class RemoteControl {

    private Command command;

    public void setCommand(Command command) {

        this.command = command;

    }

    public void pressButton() {

        command.execute();

    }

}

public class CommandPatternTest {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        Light light = new Light();

        Command lightOn = new LightOnCommand(light);

        Command lightOff = new LightOffCommand(light);

        RemoteControl remote = new RemoteControl();

        System.out.println("Home Automation System");

        System.out.println("Enter 'on' to turn light on");

        System.out.println("Enter 'off' to turn light off");

        System.out.println("Enter 'exit' to quit");

        while (true) {

            System.out.print("\nEnter command: ");

            String input = scanner.nextLine().trim().toLowerCase();

            switch (input) {

                case "on":

                    remote.setCommand(lightOn);

                    remote.pressButton();

                    break;

                case "off":

                    remote.setCommand(lightOff);

                    remote.pressButton();

                    break;

                case "exit":

                    System.out.println("Exiting system...");

                    scanner.close();

                    return;

                default:

                    System.out.println("Invalid command. Please try again.");

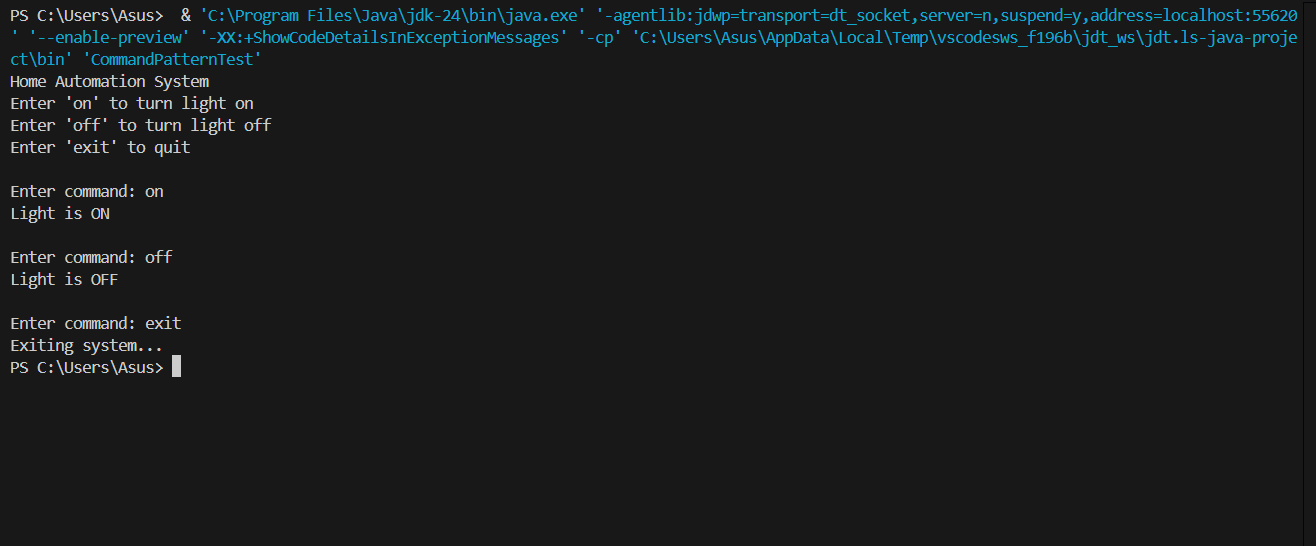
            }

        }

    }

}

**OUTPUT:**

****

**EXERCISE 10**

public class MVCPatternExample {

    static 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;

        }

    }

    static class StudentView {

        public void displayStudentDetails(String studentName, String studentId, String studentGrade) {

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

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

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

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

        }

    }

    static 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 String getStudentName() {

            return model.getName();

        }

        public void setStudentId(String id) {

            model.setId(id);

        }

        public String getStudentId() {

            return model.getId();

        }

        public void setStudentGrade(String grade) {

            model.setGrade(grade);

        }

        public String getStudentGrade() {

            return model.getGrade();

        }

        public void updateView() {

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

        }

    }

    public static void main(String[] args) {

        Student model = new Student("John Doe", "S1001", "A");

        StudentView view = new StudentView();

        StudentController controller = new StudentController(model, view);

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

        controller.updateView();

        System.out.println();

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

        controller.setStudentName("Jane Smith");

        controller.setStudentId("S1002");

        controller.setStudentGrade("B+");

               controller.updateView();

    }

}

**OUTPUT:**

****

**EXERCISE 11**

import java.util.Scanner;

import java.util.HashMap;

import java.util.Map;

class Customer {

    private String id;

    private String name;

    private String email;

    public Customer(String id, String name, String email) {

        this.id = id;

        this.name = name;

        this.email = email;

    }

        public String getId() {

        return id;

    }

    public String getName() {

        return name;

    }

    public String getEmail() {

        return email;

    }

    @Override

    public String toString() {

        return "Customer{" +

                "id='" + id + '\'' +

                ", name='" + name + '\'' +

                ", email='" + email + '\'' +

                '}';

    }

}

interface CustomerRepository {

    Customer findCustomerById(String id);

}

class CustomerRepositoryImpl implements CustomerRepository {

    private Map<String, Customer> customers = new HashMap<>();

    public CustomerRepositoryImpl() {

        customers.put("C001", new Customer("C001", "John Doe", "john@example.com"));

        customers.put("C002", new Customer("C002", "Jane Smith", "jane@example.com"));

        customers.put("C003", new Customer("C003", "Bob Johnson", "bob@example.com"));

    }

    @Override

    public Customer findCustomerById(String id) {

        return customers.get(id);

    }

}

class CustomerService {

    private CustomerRepository customerRepository;

    public CustomerService(CustomerRepository customerRepository) {

        this.customerRepository = customerRepository;

    }

    public Customer getCustomerDetails(String customerId) {

        Customer customer = customerRepository.findCustomerById(customerId);

        if (customer == null) {

            throw new IllegalArgumentException("Customer not found with ID: " + customerId);

        }

        return customer;

    }

}

public class DependencyInjectionTest {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        CustomerRepository customerRepository = new CustomerRepositoryImpl();

        CustomerService customerService = new CustomerService(customerRepository);

        System.out.println("Customer Management System");

        System.out.println("Enter customer ID to search (or 'exit' to quit)");

        while (true) {

            System.out.print("\nEnter customer ID: ");

            String input = scanner.nextLine().trim();

            if (input.equalsIgnoreCase("exit")) {

                System.out.println("Exiting system...");

                scanner.close();

                return;

            }

            try {

                Customer customer = customerService.getCustomerDetails(input);

                System.out.println("Customer found: " + customer);

            } catch (IllegalArgumentException e) {

                System.out.println("Error: " + e.getMessage());

            }

        }

    }

}

**OUTPUT:**

****