**WEEK 1  
1.Design Patterns and Principles HandsOn**

**Exercise 1: Implementing the Singleton Pattern  
Code:**

**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(message);

    }

}

**Main.java**

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

        logger1.log("First logger message");

        logger2.log("Second logger message");

        if (logger1 == logger2) {

            System.out.println("Only one instance of Logger is used.");

        } else {

            System.out.println("Different instances exist (should not happen).");

        }

    }

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

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

**Code:  
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();

}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

    }

}

A screen shot of a computer

AI-generated content may be incorrect.**Output**:

**Exercise 3: Implementing the Builder Pattern**

**Code:**

**Computer.java**

public class Computer {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

}

public String getCPU() { return CPU; }

public String getRAM() { return RAM; }

public String getStorage() { return storage; }

public String getGraphicsCard() { return graphicsCard; }

@Override

public String toString() {

return "Computer Configuration:\n" +

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

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

"Storage: " + (storage != null ? storage : "Not included") + "\n" +

"Graphics Card: " + (graphicsCard != null ? graphicsCard : "Not included");

}

public static class Builder {

private String CPU;

private String RAM;

private String storage;

private String graphicsCard;

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 Computer build() {

return new Computer(this);

}

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

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

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

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.build();

System.out.println(basicComputer);

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

System.out.println(gamingComputer);

} }

A screen shot of a computer

AI-generated content may be incorrect.**Output:**

**Exercise 4: Implementing the Adapter Pattern**

**Code:**

**PaymentProcessor.java**

public interface PaymentProcessor {

void processPayment(double amount);

}

**StripeGateway.java**

public class StripeGateway {

public void makeStripePayment(double amount) {

System.out.println("Payment of $" + amount + " processed using Stripe.");

}

}

**PayPalGateway.java**

public class PayPalGateway {

public void sendPayPalPayment(double amount) {

System.out.println("Payment of $" + amount + " processed using PayPal.");

}

}

**StripeAdapter.java**

public class StripeAdapter implements PaymentProcessor {

private StripeGateway stripeGateway;

public StripeAdapter(StripeGateway stripeGateway) {

this.stripeGateway = stripeGateway;

}

public void processPayment(double amount) {

stripeGateway.makeStripePayment(amount);

}

}

**PayPalAdapter.java**

public class PayPalAdapter implements PaymentProcessor {

private PayPalGateway payPalGateway;

public PayPalAdapter(PayPalGateway payPalGateway) {

this.payPalGateway = payPalGateway;

}

public void processPayment(double amount) {

payPalGateway.sendPayPalPayment(amount);

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

StripeGateway stripe = new StripeGateway();

PaymentProcessor stripeProcessor = new StripeAdapter(stripe);

stripeProcessor.processPayment(500.0);

PayPalGateway paypal = new PayPalGateway();

PaymentProcessor paypalProcessor = new PayPalAdapter(paypal);

paypalProcessor.processPayment(750.0);

}

}

A screenshot of a computer program

AI-generated content may be incorrect.**Output**:

**Exercise 5: Implementing the Decorator Pattern**

**Code**:

**Notifier.java**

public interface Notifier {

void send(String message);

}

**EmailNotifier.java**

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

**NotifierDecorator.java**

public abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

public void send(String message) {

notifier.send(message);

}

}

**SMSNotifierDecorator.java**

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**SlackNotifierDecorator.java**

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

Notifier basicNotifier = new EmailNotifier();

Notifier smsNotifier = new SMSNotifierDecorator(basicNotifier);

Notifier slackNotifier = new SlackNotifierDecorator(smsNotifier);

System.out.println("Notification Output:");

slackNotifier.send("Server is down!");

} }

A screen shot of a computer

AI-generated content may be incorrect.**Output**:

**Exercise 6: Implementing the Proxy Pattern**

**Code:**

**Image.java**

public interface Image {

void display();

}

**RealImage.java**

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);

}

public void display() {

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

}

}

**ProxyImage.java**

public class ProxyImage implements Image {

private String fileName;

private RealImage realImage;

private boolean isLoaded = false;

public ProxyImage(String fileName) {

this.fileName = fileName;

}

public void display() {

if (realImage == null) {

realImage = new RealImage(fileName);

isLoaded = true;

} else {

System.out.println("Loaded from cache: " + fileName);

}

realImage.display();

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

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

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

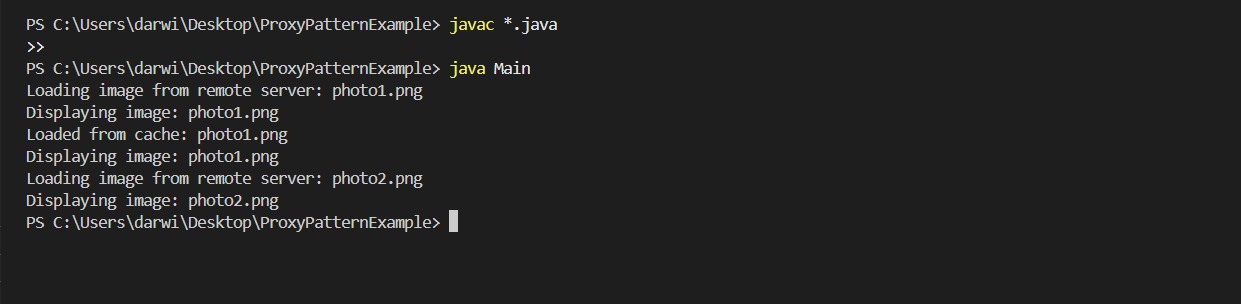
image1.display();

image1.display();

image2.display();

}

}

**Output:**

**Exercise 7: Implementing the Observer Pattern**

**Code:**

**Stock.java**

public interface Stock {

void register(Observer observer);

void deregister(Observer observer);

void notifyObservers();

}

**Observer.java**

public interface Observer {

void update(String stockName, double stockPrice);

}

**StockMarket.java**

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

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

private String stockName;

private double stockPrice;

public void setStockData(String stockName, double stockPrice) {

this.stockName = stockName;

this.stockPrice = stockPrice;

notifyObservers();

}

@Override

public void register(Observer observer) {

observers.add(observer);

}

@Override

public void deregister(Observer observer) {

observers.remove(observer);

}

@Override

public void notifyObservers() {

for (Observer observer : observers) {

observer.update(stockName, stockPrice);

}

}

}

**MobileApp.java**

public class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

@Override

public void update(String stockName, double stockPrice) {

System.out.println("MobileApp [" + name + "] - " + stockName + ": $" + stockPrice);

}

}

**WebApp.java**

public class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

@Override

public void update(String stockName, double stockPrice) {

System.out.println("WebApp [" + name + "] - " + stockName + ": $" + stockPrice);

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobile1 = new MobileApp("Client A");

Observer web1 = new WebApp("Client B");

stockMarket.register(mobile1);

stockMarket.register(web1);

stockMarket.setStockData("Cognizant", 3760.50);

stockMarket.setStockData("Sona", 1542.30);

stockMarket.deregister(web1);

stockMarket.setStockData("CTS", 415.75);

}

}

A screen shot of a computer

AI-generated content may be incorrect.**Output:**

**Exercise 8: Implementing the Strategy Pattern**

**Code:**

**PaymentStrategy.java**

public interface PaymentStrategy {

void pay(double amount);

}

**CreditCardPayment.java**

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);

}

}

**PayPalPayment.java**

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: " + email);

}

}

**PaymentContext.java**

public class PaymentContext {

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void payAmount(double amount) {

if (strategy != null) {

strategy.pay(amount);

} else {

System.out.println("No payment strategy selected.");

}

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456"));

context.payAmount(2500);

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

context.payAmount(1500);

}

}

**Output**:

A screen shot of a computer

AI-generated content may be incorrect.

**Exercise 9: Implementing the Command Pattern**

**Code:**

**Command.java**

public interface Command {

void execute();

}

**Light.java**

public class Light {

public void turnOn() {

System.out.println("The light is ON");

}

public void turnOff() {

System.out.println("The light is OFF");

}

}

**LightOnCommand.java**

public class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

**RemoteControl.java**

public class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

**Main.java**

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();

remote.setCommand(lightOn);

remote.pressButton();

remote.setCommand(lightOff);

remote.pressButton();

}

}

**Output:**

A black screen with white text

AI-generated content may be incorrect.

**Exercise 10: Implementing the MVC Pattern**

**Code:**

**Student.java**

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**

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**

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view) {

this.model = model;

this.view = view;

}

public void setStudentName(String name) {

model.setName(name);

}

public void setStudentId(String id) {

model.setId(id); }

public void setStudentGrade(String grade) {

model.setGrade(grade);

}

public void updateView() {

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

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

Student student = new Student("Newton", "S101", "B");

StudentView view = new StudentView();

StudentController controller = new StudentController(student, view);

controller.updateView();

controller.setStudentName("Darwin");

controller.setStudentGrade("A+");

System.out.println("\nAfter Update:");

controller.updateView();

}

}

A screen shot of a computer

AI-generated content may be incorrect.**Output:**

**Exercise 11: Implementing Dependency Injection**

**Code:**

**Customer.java**

public class Customer {

private String id;

private String name;

public Customer(String id, String name) {

this.id = id;

this.name = name;

}

public String getId() {

return id;

}

public String getName() {

return name;

}

}

**CustomerRepository.java**

public interface CustomerRepository {

Customer findCustomerById(String id);

}

**CustomerRepositoryImpl.java**

public class CustomerRepositoryImpl implements CustomerRepository {

@Override

public Customer findCustomerById(String id) {

return new Customer(id, "Darwin");

}

}

**CustomerService.java**

public class CustomerService {

private CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public void displayCustomer(String id) {

Customer customer = customerRepository.findCustomerById(id);

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

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

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

service.displayCustomer("123");

}

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.