**Design patterns and principles**

**Hands-on: Exercise 1: Implementing the Singleton Pattern**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

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

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

if (logger1 == logger2) {

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

} else {

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

}

}

}

Logger.java

public class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

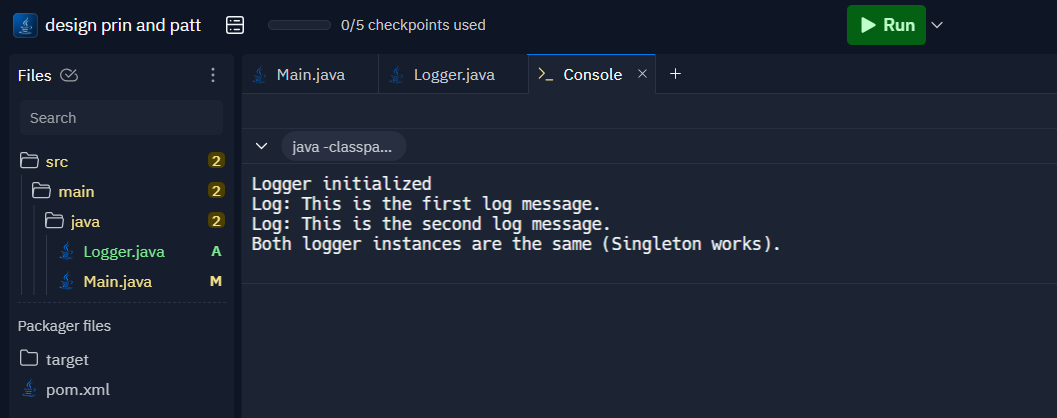
public void log(String message) {

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

}

}

OUTPUT



**Hands-on: Exercise 2: Implementing the Factory Method Pattern**

**Code-**

Main.java

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelFactory();

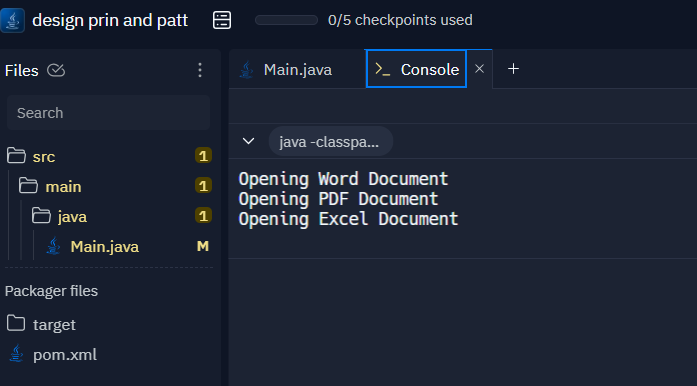
Document excel = excelFactory.createDocument();

excel.open();

}

}

OUTPUT



**Hands-on:Exercise 3 Implementing the Builder Pattern**

**Code-**

Main.java

class Computer {

private String CPU;

private int RAM;

private int storage;

private boolean hasGraphicsCard;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.hasGraphicsCard = builder.hasGraphicsCard;

}

public void displayConfig() {

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

System.out.println("RAM: " + RAM + "GB");

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

System.out.println("Graphics Card: " + (hasGraphicsCard ? "Yes" : "No"));

}

public static class Builder {

private String CPU;

private int RAM;

private int storage;

private boolean hasGraphicsCard;

public Builder setCPU(String CPU) {

this.CPU = CPU;

return this;

}

public Builder setRAM(int RAM) {

this.RAM = RAM;

return this;

}

public Builder setStorage(int storage) {

this.storage = storage;

return this;

}

public Builder setGraphicsCard(boolean hasGraphicsCard) {

this.hasGraphicsCard = hasGraphicsCard;

return this;

}

public Computer build() {

return new Computer(this);

}

}

}

public class Main {

public static void main(String[] args) {

Computer basicPC = new Computer.Builder().setCPU("Intel i3").setRAM(8).setStorage(256).build();

Computer gamingPC = new Computer.Builder().setCPU("AMD Ryzen 7").setRAM(16).setStorage(1024).setGraphicsCard(true).build();

System.out.println("Basic PC Configuration:");

basicPC.displayConfig();

System.out.println("Gaming PC Configuration:");

gamingPC.displayConfig();

}

}OUTPUT

A screenshot of a computer

AI-generated content may be incorrect.

**Hands-on Exercise 4: Implementing the Adapter Pattern**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());

stripeProcessor.processPayment(500.0);

PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPalGateway());

paypalProcessor.processPayment(750.0);

}

}

interface PaymentProcessor {

void processPayment(double amount);

}

class StripeGateway {

public void makeStripePayment(double amount) {

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

}

}

class PayPalGateway {

public void sendPayment(double amountInDollars) {

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

}

}

class StripeAdapter implements PaymentProcessor {

private StripeGateway stripeGateway;

public StripeAdapter(StripeGateway stripeGateway) {

this.stripeGateway = stripeGateway;

}

public void processPayment(double amount) {

stripeGateway.makeStripePayment(amount);

}

}

class PayPalAdapter implements PaymentProcessor {

private PayPalGateway payPalGateway;

public PayPalAdapter(PayPalGateway payPalGateway) {

this.payPalGateway = payPalGateway;

}

public void processPayment(double amount) {

payPalGateway.sendPayment(amount);

}

}

OUTPUT

A screenshot of a computer

AI-generated content may be incorrect.

**Hands-on: Exercise 5: Implementing the Decorator Pattern**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

Notifier notifier1 = new EmailNotifier();

notifier1.send("Hello via Email");

Notifier notifier2 = new SMSNotifierDecorator(new EmailNotifier());

notifier2.send("Hello via Email and SMS");

Notifier notifier3 = new SlackNotifierDecorator(new SMSNotifierDecorator(new EmailNotifier()));

notifier3.send("Hello via Email, SMS, and Slack");

}

}

interface Notifier {

void send(String message);

}

class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message);

}

}

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}OUTPUT

A screenshot of a computer

AI-generated content may be incorrect.

**Hands-on: Exercise 6: Implementing the Proxy Pattern**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

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

image1.display();

image1.display();

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

image2.display();

}

}

interface Image {

void display();

}

class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

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

}

public void display() {

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

}

}

class ProxyImage implements Image {

private RealImage realImage;

private String filename;

public ProxyImage(String filename) {

this.filename = filename;

}

public void display() {

if (realImage == null) {

realImage = new RealImage(filename);

}

realImage.display();

}

}

OUTPUT-

**A screenshot of a computer

AI-generated content may be incorrect.**

**Hands-on: Exercise 7: Implementing the Observer Pattern**

**Code-**

Main.java

import java.util.\*;

public class Main {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobile = new MobileApp("MobileApp1");

Observer web = new WebApp("WebApp1");

stockMarket.registerObserver(mobile);

stockMarket.registerObserver(web);

stockMarket.setPrice("AAPL", 150.5);

stockMarket.setPrice("GOOG", 2800.0);

stockMarket.removeObserver(web);

stockMarket.setPrice("AAPL", 152.0);

}

}

interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers(String stockName, double price);

}

interface Observer {

void update(String stockName, double price);

}

class StockMarket implements Stock {

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

private Map<String, Double> stockPrices = new HashMap<>();

public void registerObserver(Observer o) {

observers.add(o);

}

public void removeObserver(Observer o) {

observers.remove(o);

}

public void notifyObservers(String stockName, double price) {

for (Observer o : observers) {

o.update(stockName, price);

}

}

public void setPrice(String stockName, double price) {

stockPrices.put(stockName, price);

notifyObservers(stockName, price);

}

}

class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

public void update(String stockName, double price) {

System.out.println(name + " received update: " + stockName + " = " + price);

}

}

class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

public void update(String stockName, double price) {

System.out.println(name + " received update: " + stockName + " = " + price);

}

}

OUTPUT-

A screenshot of a computer program

AI-generated content may be incorrect.

**Hands-on: Exercise 8: Implementing the Strategy Pattern**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext();

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

context.pay(250.0);

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

context.pay(400.0);

}

}

interface PaymentStrategy {

void pay(double amount);

}

class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

public CreditCardPayment(String cardNumber) {

this.cardNumber = cardNumber;

}

public void pay(double amount) {

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

}

}

class PayPalPayment implements PaymentStrategy {

private String email;

public PayPalPayment(String email) {

this.email = email;

}

public void pay(double amount) {

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

}

}

class PaymentContext {

private PaymentStrategy strategy;

public void setStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void pay(double amount) {

strategy.pay(amount);

}

}

OUTPUT-

A screenshot of a computer

AI-generated content may be incorrect.

**Hands-on: Exercise 9: Implementing the Command Pattern**

**Code-**

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();

}

}

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;

}

public void execute() {

light.turnOn();

}

}

class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

OUTPUT-

A screenshot of a computer

AI-generated content may be incorrect.

**Hands-on: Exercise 10: Implementing the MVC Pattern**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

Student model = new Student("John Doe", "S101", "A");

StudentView view = new StudentView();

StudentController controller = new StudentController(model, view);

controller.updateView();

controller.setStudentName("Jane Smith");

controller.setStudentGrade("B+");

controller.updateView();

}

}

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;

}

}

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);

}

}

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 String getStudentName() {

return model.getName();

}

public String getStudentId() {

return model.getId();

}

public String getStudentGrade() {

return model.getGrade();

}

public void updateView() {

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

}

}

OUTPUT-

A screenshot of a computer

AI-generated content may be incorrect.

**Hands-on: Exercise 11: Implementing Dependency Injection**

**Code-**

Main.java

public class Main {

public static void main(String[] args) {

CustomerRepository repository = new CustomerRepositoryImpl();

CustomerService service = new CustomerService(repository);

Customer customer = service.getCustomerById(1);

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

}

}

class Customer {

private int id;

private String name;

public Customer(int id, String name) {

this.id = id;

this.name = name;

}

public int getId() {

return id;

}

public String getName() {

return name;

}

}

interface CustomerRepository {

Customer findCustomerById(int id);

}

class CustomerRepositoryImpl implements CustomerRepository {

public Customer findCustomerById(int id) {

return new Customer(id, "Customer" + id);

}

}

class CustomerService {

private CustomerRepository customerRepository;

public CustomerService(CustomerRepository customerRepository) {

this.customerRepository = customerRepository;

}

public Customer getCustomerById(int id) {

return customerRepository.findCustomerById(id);

}

}

OUTPUT-

A screenshot of a computer

AI-generated content may be incorrect.