**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;

}

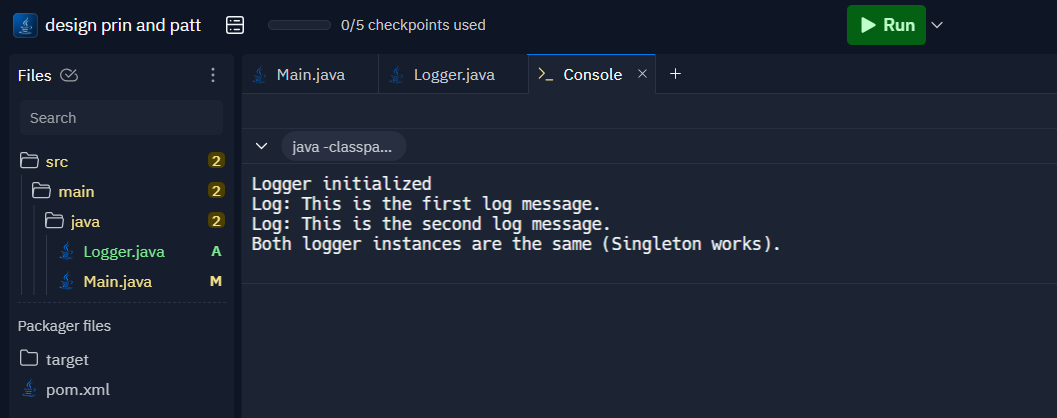
public void log(String message) {

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();

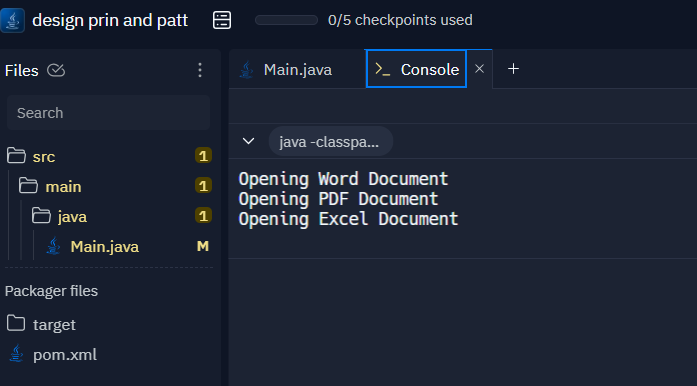
Document excel = excelFactory.createDocument();

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.