**Exercise 1: Implementing the Singleton Pattern**

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: " + message);

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("First log message");

logger2.log("Second log message");

if (logger1 == logger2) {

System.out.println("Logger is a Singleton. Both references are the same.");

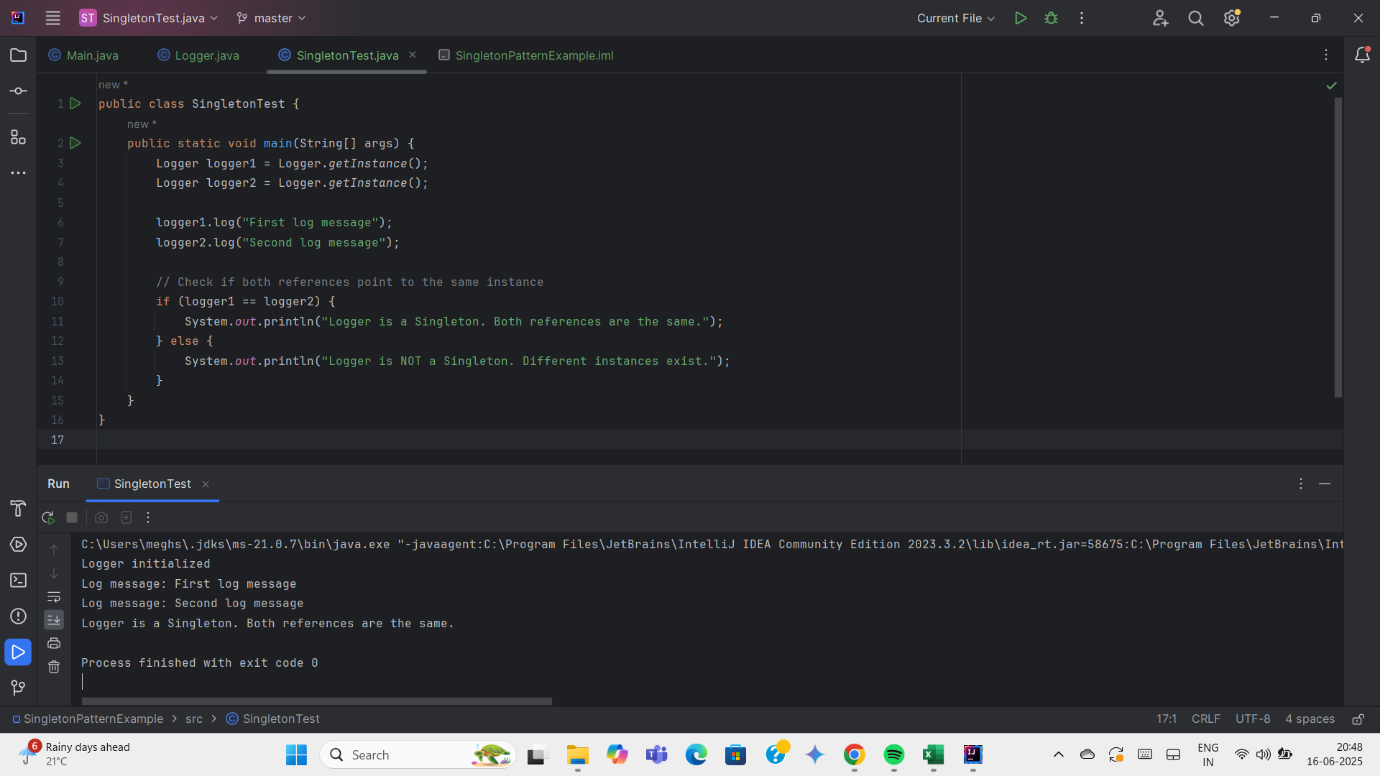
} else {

System.out.println("Logger is NOT a Singleton. Different instances exist.");

}

}

}



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

abstract class Document {

public abstract void open();

}

class WordDocument extends Document {

@Override

public void open() {

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

}

}

class PdfDocument extends Document {

@Override

public void open() {

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

}

}

class ExcelDocument extends Document {

@Override

public void open() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory factory;

factory = new WordDocumentFactory();

Document wordDoc = factory.createDocument();

wordDoc.open();

factory = new PdfDocumentFactory();

Document pdfDoc = factory.createDocument();

pdfDoc.open();

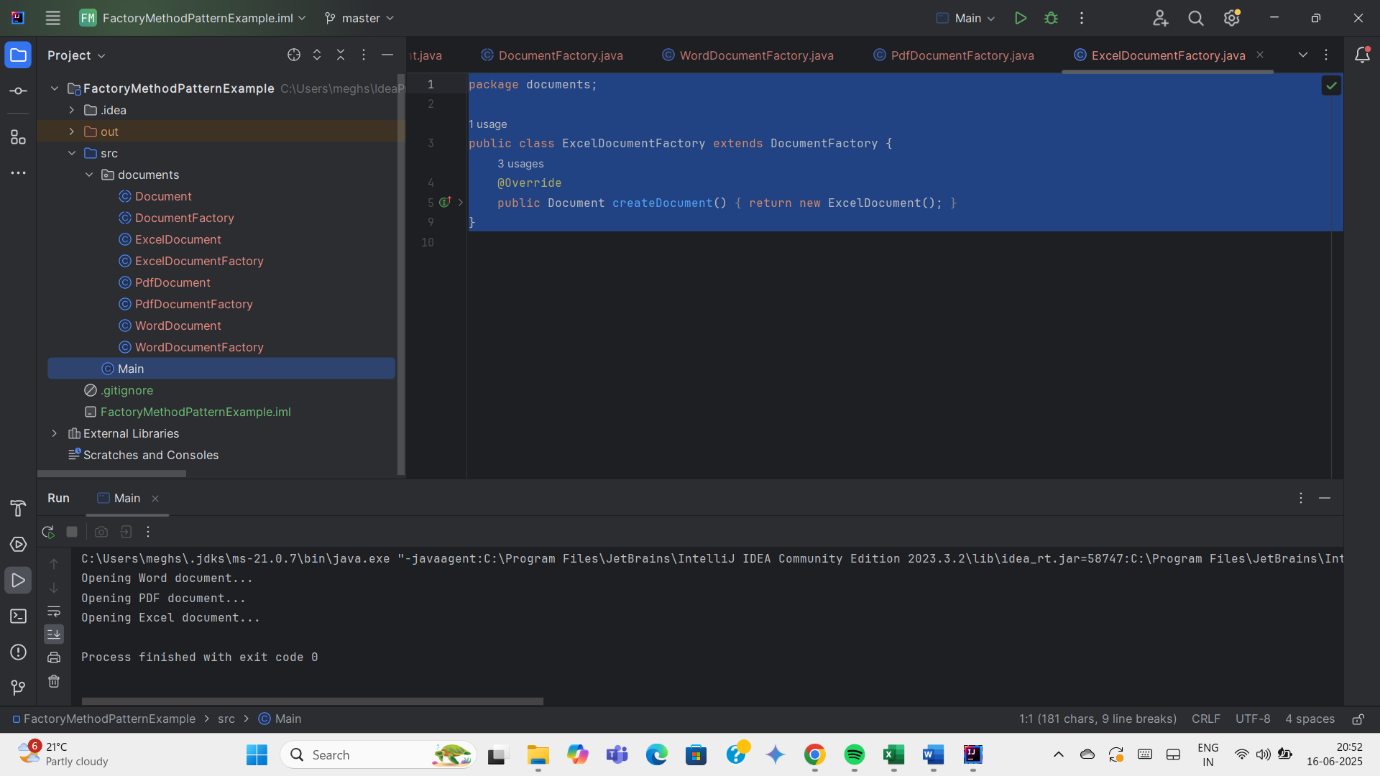
factory = new ExcelDocumentFactory();

Document excelDoc = factory.createDocument();

excelDoc.open();

}

}



**Exercise 3: Implementing the Builder Pattern**

class Computer {

private String CPU;

private String RAM;

private String storage;

private Computer(Builder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

}

public void displayConfiguration() {

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

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

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

}

public static class Builder {

private String CPU;

private String RAM;

private String storage;

public Builder setCPU(String CPU) {

this.CPU = CPU;

return this;

}

public Builder setRAM(String RAM) {

this.RAM = RAM;

return this;

}

public Builder setStorage(String storage) {

this.storage = storage;

return this;

}

public Computer build() {

return new Computer(this);

}

}

}

public class Main {

public static void main(String[] args) {

// Create different configurations of Computer using Builder

Computer gamingPC = new Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.build();

Computer officePC = new Computer.Builder()

.setCPU("Intel i5")

.setRAM("16GB")

.setStorage("512GB SSD")

.build();

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

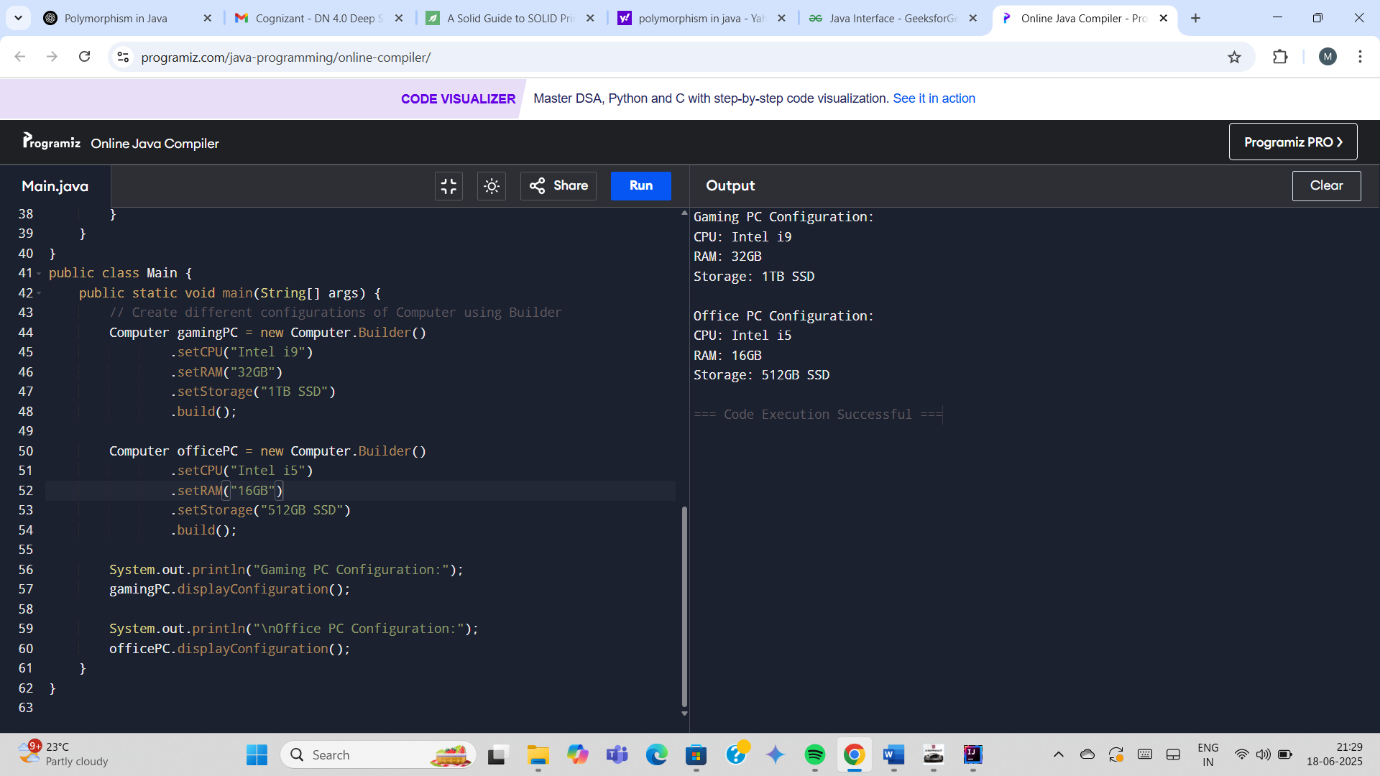
gamingPC.displayConfiguration();

System.out.println("\nOffice PC Configuration:");

officePC.displayConfiguration();

}

}



**Exercise 5: Implementing the Decorator Pattern**

interface Notifier {

void send(String message);

}

class EmailNotifier implements Notifier {

@Override

public void send(String message) {

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

}

}

abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

@Override

public void send(String message) {

notifier.send(message);

}

}

// Concrete decorator for SMS

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

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

}

}

// Concrete decorator for Slack

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);

}

}

public class Main {

public static void main(String[] args) {

// Base Notifier: Email

Notifier notifier = new EmailNotifier();

// Add SMS Decorator

notifier = new SMSNotifierDecorator(notifier);

// Add Slack Decorator

notifier = new SlackNotifierDecorator(notifier);

// Send message

notifier.send("Hello Meghzzz, your payment is successful!");

}

}

