**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

**Java Project Name : SingletonPatternExample**

**Logger.java**

package com.singleton.example;

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("LOG: " + message);

}

}

**SingletonTest.java**

package com.singleton.example;

public class SingletonTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First log message");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message");

if (logger1 == logger2) {

System.out.println("Both references point to the same instance");

} else {

System.out.println("Different instances created");

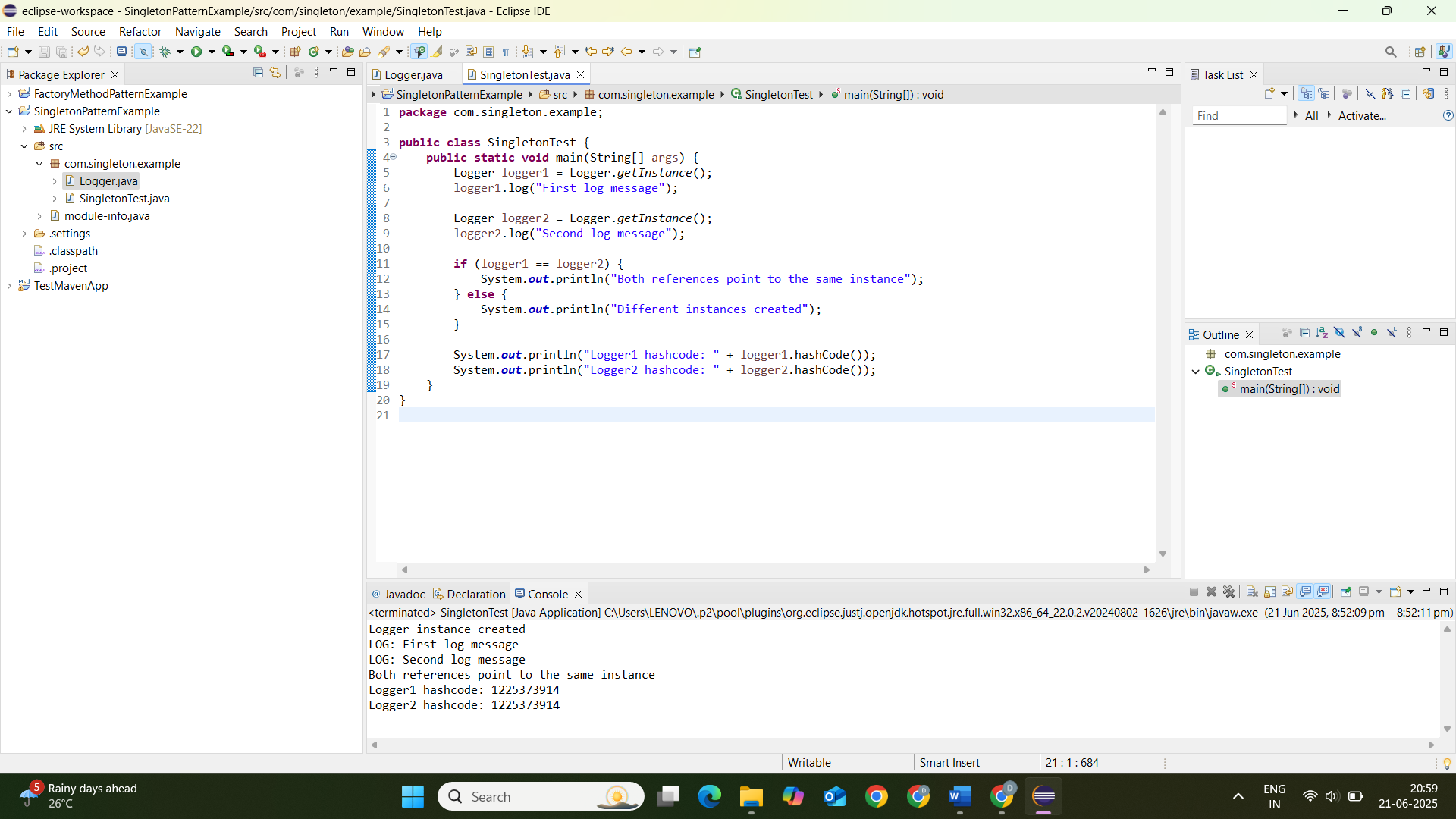
}

System.out.println("Logger1 hashcode: " + logger1.hashCode());

System.out.println("Logger2 hashcode: " + logger2.hashCode());

}

}



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

**Java Project Name : FactoryMethodPatternExample**

**Document.java**

package com.factorymethod.example;

public interface Document {

void open();

void save();

void close();

}

**WordDocument.java**

package com.factorymethod.example;

public 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");

}

}

**PdfDocument.java**

package com.factorymethod.example;

public 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");

}

}

**ExcelDocument.java**

package com.factorymethod.example;

public 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");

}

}

**DocumentFactory.java**

package com.factorymethod.example;

public abstract class DocumentFactory {

public abstract Document createDocument();

public void processDocument() {

Document doc = createDocument();

doc.open();

doc.save();

doc.close();

}

}

**WordDocumentFactory.java**

package com.factorymethod.example;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

package com.factorymethod.example;

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

package com.factorymethod.example;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**DocumentManagementTest.java**

package com.factorymethod.example;

public class DocumentManagementTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

System.out.println("Processing Word Document:");

wordFactory.processDocument();

System.out.println("\nProcessing PDF Document:");

pdfFactory.processDocument();

System.out.println("\nProcessing Excel Document:");

excelFactory.processDocument();

System.out.println("\nCreating documents directly:");

Document wordDoc = wordFactory.createDocument();

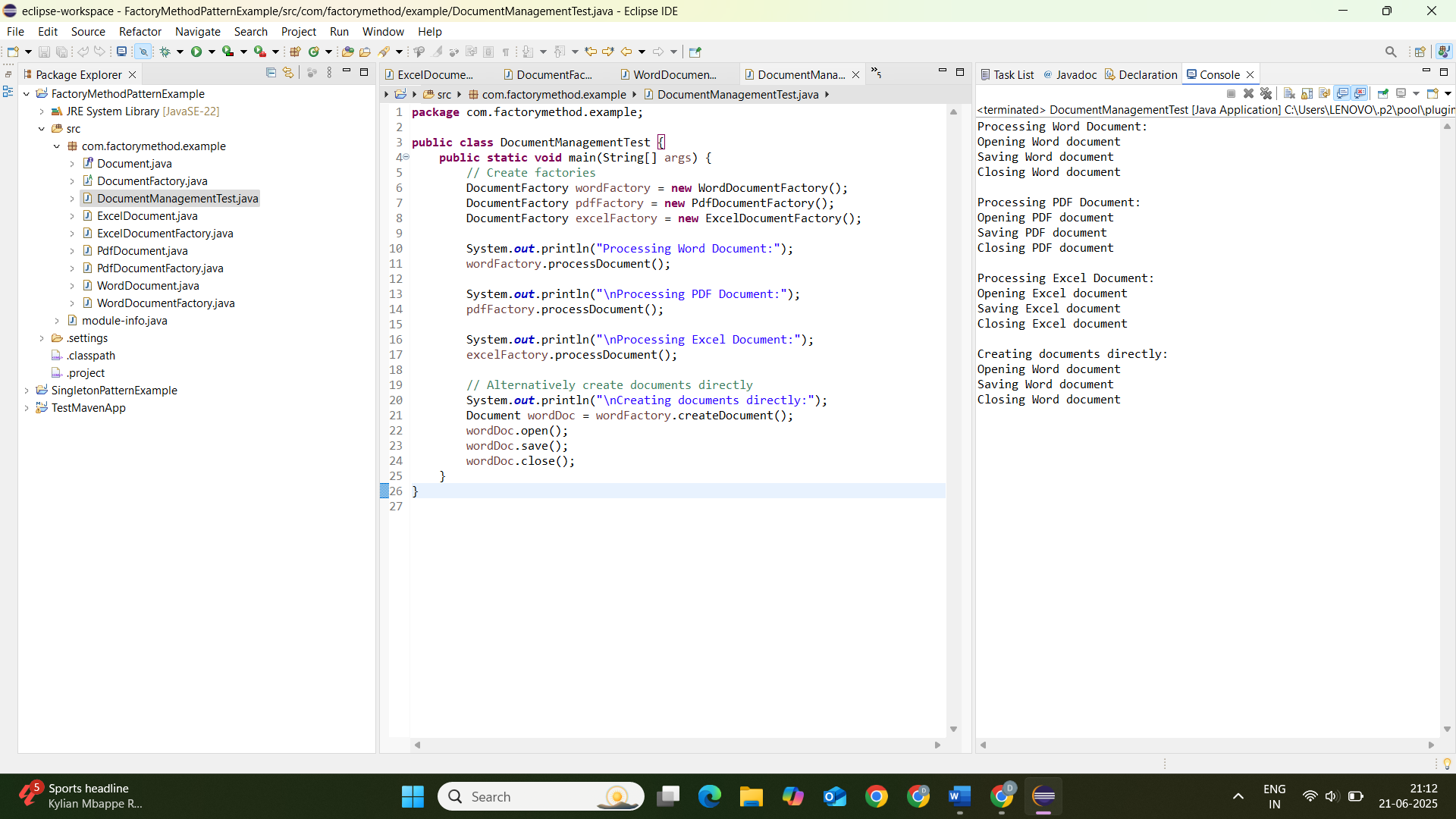
wordDoc.open();

wordDoc.save();

wordDoc.close();

}

}



**Exercise 3: Implementing the Builder Pattern**

**Java Project Name : BuilderPatternExample**

**Computer.java**

package com.example.builder;

public class Computer {

private final String CPU;

private final String RAM;

private final String storage;

private final String graphicsCard;

private final String operatingSystem;

private final boolean hasBluetooth;

private final boolean hasWiFi;

private Computer(ComputerBuilder builder) {

this.CPU = builder.CPU;

this.RAM = builder.RAM;

this.storage = builder.storage;

this.graphicsCard = builder.graphicsCard;

this.operatingSystem = builder.operatingSystem;

this.hasBluetooth = builder.hasBluetooth;

this.hasWiFi = builder.hasWiFi;

}

public String getCPU() {

return CPU;

}

public String getRAM() {

return RAM;

}

public String getStorage() {

return storage;

}

public String getGraphicsCard() {

return graphicsCard;

}

public String getOperatingSystem() {

return operatingSystem;

}

public boolean hasBluetooth() {

return hasBluetooth;

}

public boolean hasWiFi() {

return hasWiFi;

}

@Override

public String toString() {

return "Computer{" +

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

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

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

", graphicsCard='" + graphicsCard + '\'' +

", operatingSystem='" + operatingSystem + '\'' +

", hasBluetooth=" + hasBluetooth +

", hasWiFi=" + hasWiFi +

'}';

}

public static class ComputerBuilder {

private final String CPU;

private final String RAM;

private String storage = "1TB HDD";

private String graphicsCard = "Integrated";

private String operatingSystem = "Windows 10";

private boolean hasBluetooth = false;

private boolean hasWiFi = true;

public ComputerBuilder(String CPU, String RAM) {

this.CPU = CPU;

this.RAM = RAM;

}

public ComputerBuilder storage(String storage) {

this.storage = storage;

return this;

}

public ComputerBuilder graphicsCard(String graphicsCard) {

this.graphicsCard = graphicsCard;

return this;

}

public ComputerBuilder operatingSystem(String operatingSystem) {

this.operatingSystem = operatingSystem;

return this;

}

public ComputerBuilder hasBluetooth(boolean hasBluetooth) {

this.hasBluetooth = hasBluetooth;

return this;

}

public ComputerBuilder hasWiFi(boolean hasWiFi) {

this.hasWiFi = hasWiFi;

return this;

}

public Computer build() {

return new Computer(this);

}

}

}

**ComputerBuilderTest.java**

package com.example.builder;

public class ComputerBuilderTest {

public static void main(String[] args) {

Computer basicComputer = new Computer.ComputerBuilder("Intel i5", "8GB")

.build();

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

Computer gamingComputer = new Computer.ComputerBuilder("AMD Ryzen 9", "32GB")

.storage("1TB SSD")

.graphicsCard("NVIDIA RTX 3080")

.operatingSystem("Windows 11")

.hasBluetooth(true)

.build();

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

Computer developerComputer = new Computer.ComputerBuilder("Intel i7", "16GB")

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

.operatingSystem("Ubuntu 22.04")

.hasWiFi(false) // Developer prefers wired connection

.build();

System.out.println("\nDeveloper Computer: " + developerComputer);

Computer customComputer = new Computer.ComputerBuilder("Apple M1 Max", "64GB")

.storage("2TB SSD")

.graphicsCard("Apple 32-core GPU")

.operatingSystem("macOS Monterey")

.hasBluetooth(true)

.hasWiFi(true)

.build();

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

}

