Exercise 1: Implementing the Singleton Pattern

Scenario:

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

Steps:

1. Create a New Java Project:

o Create a new Java project named SingletonPatternExample.

2. Define a Singleton Class:

o Create a class named Logger that has a private static instance of itself.

o Ensure the constructor of Logger is private.

o Provide a public static method to get the instance of the Logger class.

3. Implement the Singleton Pattern:

o Write code to ensure that the Logger class follows the Singleton design pattern.

4. Test the Singleton Implementation:

o Create a test class to verify that only one instance of Logger is created and used across the application.

**CODE:**

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);

}

public void error(String message) {

System.out.println("[ERROR] " + message);

}

public void warn(String message) {

System.out.println("[WARN] " + message);

}

}

// this is a testing class

public class SingletonTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

System.out.println("Are both logger instances the same? " + (logger1 == logger2));

logger1.log("Application started");

logger2.warn("This is a warning message");

new Thread(() -> {

Logger threadLogger = Logger.getInstance();

threadLogger.log("Message from thread");

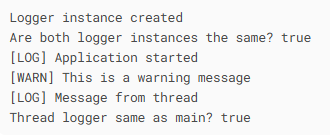
System.out.println("Thread logger same as main? " + (threadLogger == logger1));

}).start();

}

}

**OUTPUT:**



Exercise 2: Implementing the Factory Method Pattern

Scenario:

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

Steps:

1. Create a New Java Project:

o Create a new Java project named FactoryMethodPatternExample.

2. Define Document Classes:

o Create interfaces or abstract classes for different document types such as WordDocument, PdfDocument, and ExcelDocument.

3. Create Concrete Document Classes:

o Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.

4. Implement the Factory Method:

o Create an abstract class DocumentFactory with a method createDocument().

o Create concrete factory classes for each document type that extends DocumentFactory and implements the createDocument() method.

5. Test the Factory Method Implementation:

o Create a test class to demonstrate the creation of different document types using the factory method.

**CODE:**

//creating document classes and concrete document classes

interface Document {

void open();

}

public class WordDocument implements Document {

@Override

public void open() {

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

}

}

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

// implementing the factory method here

public class DocumentFactory {

public Document createDocument(String type) {

if (type.equalsIgnoreCase("word")) {

return new WordDocument();

} else if (type.equalsIgnoreCase("pdf")) {

return new PdfDocument();

}

throw new IllegalArgumentException("Invalid document type.");

}

}

**//** Testing the factory method implementation

public class Main {

public static void main(String[] args) {

DocumentFactory factory = new DocumentFactory();

Document wordDoc = factory.createDocument("word");

wordDoc.open();

Document pdfDoc = factory.createDocument("pdf");

pdfDoc.open();

}

}

**OUTPUT:**

