**Cognizant Deep Nurture 4.0 Hands-on Exercise    
   
   
   
   
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:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

1. Create the Singleton Class Logger

// File: Logger.java

public class Logger {

// Private static variable to hold the single instance

private static Logger instance;

// Private constructor to prevent instantiation

private Logger() {

System.out.println("Logger instance created.");

}

// Public method to return the same instance every time

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

// Sample method to simulate logging

public void log(String message) {

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

}

}

2. Create the Test Class

// File: Main.java

public class Main {

public static void main(String[] args) {

// Get Logger instance

Logger logger1 = Logger.getInstance();

logger1.log("First log message.");

// Get Logger instance again

Logger logger2 = Logger.getInstance();

logger2.log("Second log message.");

// Check if both instances are the same

if (logger1 == logger2) {

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

} else {

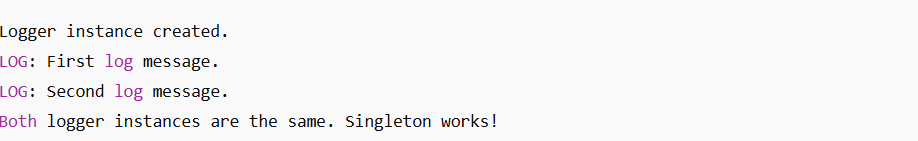
System.out.println("Logger instances are different. Singleton failed.");

}

}

}

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:**
   * Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   * Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   * Create an abstract class **DocumentFactory** with a method **createDocument()**.
   * Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   * Create a test class to demonstrate the creation of different document types using the factory method.

   
1.Define the Document Interface

// File: Document.java

public interface Document {

void open();

}

2. Create Concrete Document Classes

// File: WordDocument.java

public class WordDocument implements Document {

public void open() {

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

}

}

// File: PdfDocument.java

public class PdfDocument implements Document {

public void open() {

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

}

}

// File: ExcelDocument.java

public class ExcelDocument implements Document {

public void open() {

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

}

}

3. Create the Factory Classes

Abstract Factory

// File: DocumentFactory.java

public abstract class DocumentFactory {

public abstract Document createDocument();

}

Concrete Factories

// File: WordDocumentFactory.java

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

java

CopyEdit

// File: PdfDocumentFactory.java

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

java

CopyEdit

// File: ExcelDocumentFactory.java

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

4. Test the Factory Method Pattern

// File: Main.java

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

OUTPUT:

