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.

ANSWER

```
public class SingletonTest {
    // Singleton Logger class
    static 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);
        }
    }
    // Main method to test Logger
    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("Both logger instances are the same.");
        } else {
            System.out.println("Logger instances are different.");
        }
    }
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POLYGLOT NOTEBOOK                    powershell

PS C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\DesignPatternsAssignments\SingletonPatternEg> javac SingletonTes
t.java
PS C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\DesignPatternsAssignments\SingletonPatternEg> java SingletonTest

Logger initialized
Log: First log message
Log: Second log message
Both logger instances are same
PS C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\DesignPatternsAssignments\SingletonPatternEg>
```

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.

ANSWER

```
public class FactoryMethodTest {
    // Document interface
    interface Document {
        void open();
    }
    // Concrete document types
    static class WordDocument implements Document {
        public void open() {
            System.out.println("Opening Word document.");
        }
    }
    static class PdfDocument implements Document {
        public void open() {
            System.out.println("Opening PDF document.");
```

```
    }
}
static class ExcelDocument implements Document {
    public void open() {
        System.out.println("Opening Excel document.");
    }
}
// Abstract factory
abstract static class DocumentFactory {
    public abstract Document createDocument();
}
// Concrete factories
static class WordFactory extends DocumentFactory {
    public Document createDocument() {
        return new WordDocument();
    }
}
static class PdfFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}
static class ExcelFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}
// Main method to test factory
public static void main(String[] args) {
    DocumentFactory wordFactory = new WordFactory();
    Document wordDoc = wordFactory.createDocument();
    wordDoc.open();
    DocumentFactory pdfFactory = new PdfFactory();
    Document pdfDoc = pdfFactory.createDocument();
    pdfDoc.open();
    DocumentFactory excelFactory = new ExcelFactory();
```

```
        Document excelDoc = excelFactory.createDocument();
        excelDoc.open();
    }
}
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POLYGLOT NOTEBOOK                    powershell

PS C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\DesignPatternsAssignments\FactoryMethodPatternEg>
javac FactoryMethodTest.java
PS C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\DesignPatternsAssignments\FactoryMethodPatternEg> java Fa
ctoryMethodTest
Opening Word document.
Opening PDF document.
Opening Excel document.
PS C:\Users\KIIT\Documents\Digital-Nurture-4.0-DotNetFSE-solution\week1\DesignPatternsAssignments\FactoryMethodPatternEg>
```