**Week 1- Hands on**

**Name- Adi Jain**

**Superset ID- 6390400**

**Design Patterns & Principles**

**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:**

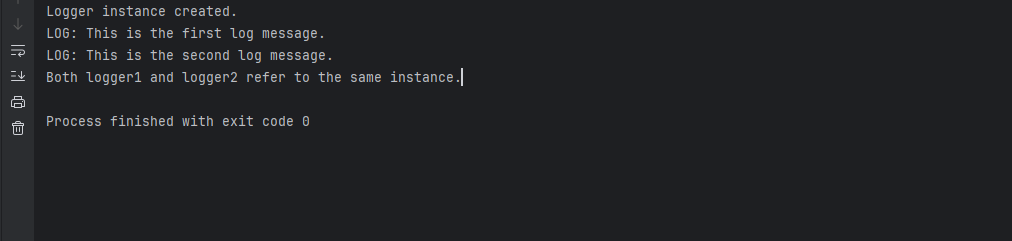
1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to 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.

**Code**

package Singleton;  
  
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(); // Lazy initialization  
 }  
 return *instance*;  
 }  
   
 public void log(String message) {  
 System.*out*.println("LOG: " + message);  
 }  
}

package Singleton;  
  
public class TestLogger {  
 public static void main(String[] args) {  
 logger logger1 = logger.*getInstance*();  
 logger1.log("This is the first log message.");  
   
 logger logger2 = logger.*getInstance*();  
 logger2.log("This is the second log message.");  
  
   
 if (logger1 == logger2) {  
 System.*out*.println("Both logger1 and logger2 refer to the same instance.");  
 } else {  
 System.*out*.println("Different instances exist. 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.

**Code**

package factorymethod;  
  
public interface document {  
 void open();  
 void close();  
 void save();  
}

package factorymethod;  
  
public abstract class DocumentFactory {  
 public abstract document createDocument();  
}

package factorymethod;  
public class exceldocument implements document {  
  
 @Override  
 public void open() {  
 System.*out*.println("Opening Excel document...");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing Excel document...");  
 }  
  
 @Override  
 public void save() {  
 System.*out*.println("Saving Excel document...");  
 }  
}

package factorymethod;  
  
public class excelDocumentFactory extends DocumentFactory {  
  
 @Override  
 public document createDocument() {  
 return new exceldocument();  
 }  
}

package factorymethod;  
  
public class pdfdocument implements document {  
  
 @Override  
 public void open() {  
 System.*out*.println("Opening PDF document...");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing PDF document...");  
 }  
  
 @Override  
 public void save() {  
 System.*out*.println("Saving PDF document...");  
 }  
}

package factorymethod;  
  
public class pdfdocumentfactory extends DocumentFactory {  
  
 @Override  
 public document createDocument() {  
 return new pdfdocument();  
 }  
}

package factorymethod;  
  
  
public class worddocument implements document {  
  
 @Override  
 public void open() {  
 System.*out*.println("Opening Word document...");  
 }  
  
 @Override  
 public void close() {  
 System.*out*.println("Closing Word document...");  
 }  
  
 @Override  
 public void save() {  
 System.*out*.println("Saving Word document...");  
 }  
}

package factorymethod;  
  
public class WordDocumentFactory extends DocumentFactory {  
  
 @Override  
 public document createDocument() {  
 return new worddocument();  
 }  
}

package factorymethod;  
  
public class TestFactoryMethod {  
  
 public static void main(String[] args) {  
 DocumentFactory factory;  
  
 factory = new WordDocumentFactory();  
 document wordDoc = factory.createDocument();  
 wordDoc.open();  
 wordDoc.save();  
 wordDoc.close();  
  
 System.*out*.println();  
  
 factory = new pdfdocumentfactory();  
 document pdfDoc = factory.createDocument();  
 pdfDoc.open();  
 pdfDoc.save();  
 pdfDoc.close();  
  
 System.*out*.println();  
  
 factory = new excelDocumentFactory();  
 document excelDoc = factory.createDocument();  
 excelDoc.open();  
 excelDoc.save();  
 excelDoc.close();  
 }  
}

**Output**

