DESIGN PRINCIPLES AND PATTERNS

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.

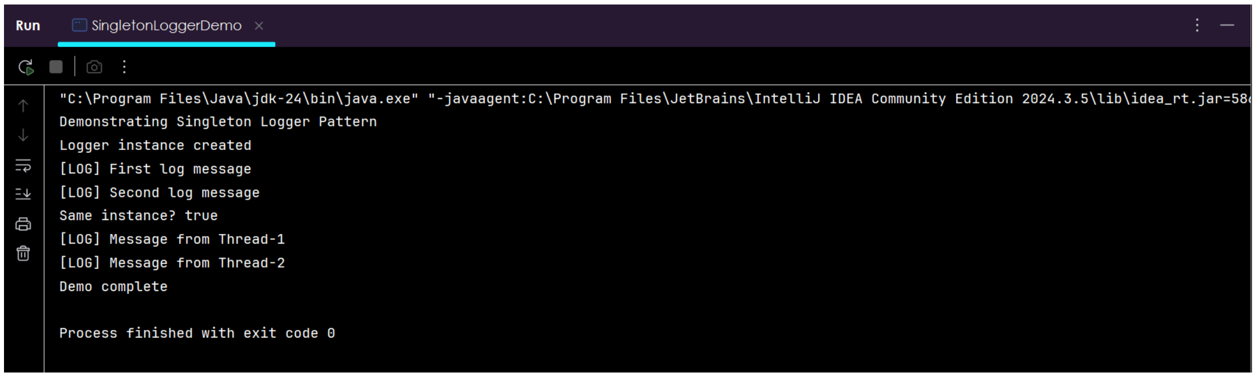
Steps:

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:

```
public class SingletonLoggerDemo {


  static class Logger {
    private static volatile Logger instance;

    private Logger() {
      System.out.println("Logger instance created");
    }

    public static Logger getInstance() {
      if (instance == null) {
        synchronized (Logger.class) {
          if (instance == null) {
            instance = new Logger();
          }
        }
      }
```

```
            return instance;
        }

        public void log(String message) {
            System.out.println("[LOG] " + message);
        }
    }

    public static void main(String[] args) {
        System.out.println("Demonstrating Singleton Logger Pattern");

        Logger logger1 = Logger.getInstance();
        Logger logger2 = Logger.getInstance();

        logger1.log("First log message");
        logger2.log("Second log message");

        System.out.println("Same instance? " + (logger1 == logger2));

        Runnable task = () -> {
            Logger threadLogger = Logger.getInstance();
            threadLogger.log("Message from " + Thread.currentThread().getName());
        };

        Thread thread1 = new Thread(task, "Thread-1");
        Thread thread2 = new Thread(task, "Thread-2");

        thread1.start();
        thread2.start();

        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Demo complete");
    }
}
```
Output:



```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.5\lib\idea_rt.jar=58
Demonstrating Singleton Logger Pattern
Logger instance created
[LOG] First log message
[LOG] Second log message
Same instance? true
[LOG] Message from Thread-1
[LOG] Message from Thread-2
Demo complete

Process finished with exit code 0
```

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.

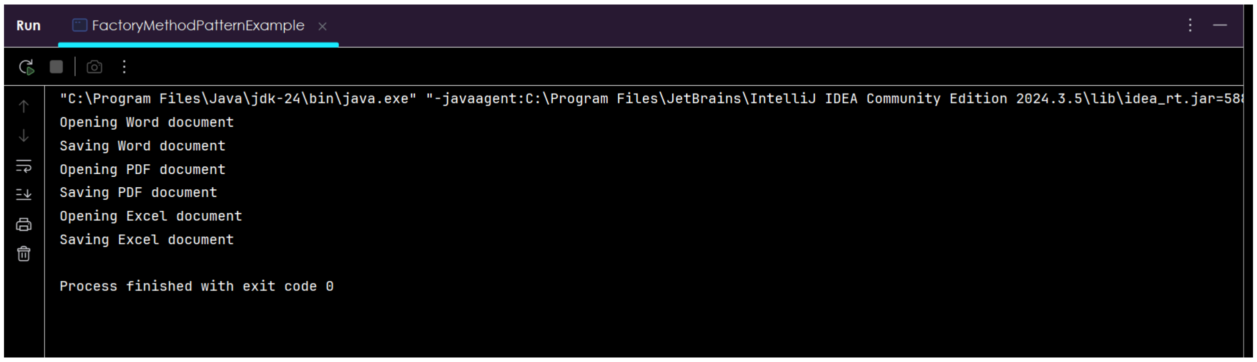
Code:

```
interface Document {

    void open();

    void save();
```

```
}

// Step 3: Create concrete document classes
class WordDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening Word document");
    }

    @Override
    public void save() {
        System.out.println("Saving Word document");
    }
}

class PdfDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening PDF document");
    }

    @Override
    public void save() {
        System.out.println("Saving PDF document");
    }
}

class ExcelDocument implements Document {
    @Override
    public void open() {
        System.out.println("Opening Excel document");
    }

    @Override
    public void save() {
        System.out.println("Saving Excel document");
    }
}


abstract class DocumentFactory {
    public abstract Document createDocument();



}
```

```
class WordDocumentFactory extends DocumentFactory {
   @Override
   public Document createDocument() {
      return new WordDocument();
   }
}

class PdfDocumentFactory extends DocumentFactory {
   @Override
   public Document createDocument() {
      return new PdfDocument();
   }
}

class ExcelDocumentFactory extends DocumentFactory {
   @Override
   public Document createDocument() {
      return new ExcelDocument();
   }
}


public class FactoryMethodPatternExample {
   public static void main(String[] args) {


      DocumentFactory wordFactory = new WordDocumentFactory();
      Document wordDoc = wordFactory.createDocument();
      wordDoc.open();
      wordDoc.save();


      DocumentFactory pdfFactory = new PdfDocumentFactory();
      Document pdfDoc = pdfFactory.createDocument();
      pdfDoc.open();
      pdfDoc.save();



      DocumentFactory excelFactory = new ExcelDocumentFactory();
      Document excelDoc = excelFactory.createDocument();
      excelDoc.open();
      excelDoc.save();
   }
}
```

Output:



```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.5\lib\idea_rt.jar=58
Opening Word document
Saving Word document
Opening PDF document
Saving PDF document
Opening Excel document
Saving Excel document

Process finished with exit code 0
```