DESIGN PATTERNS AND 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:

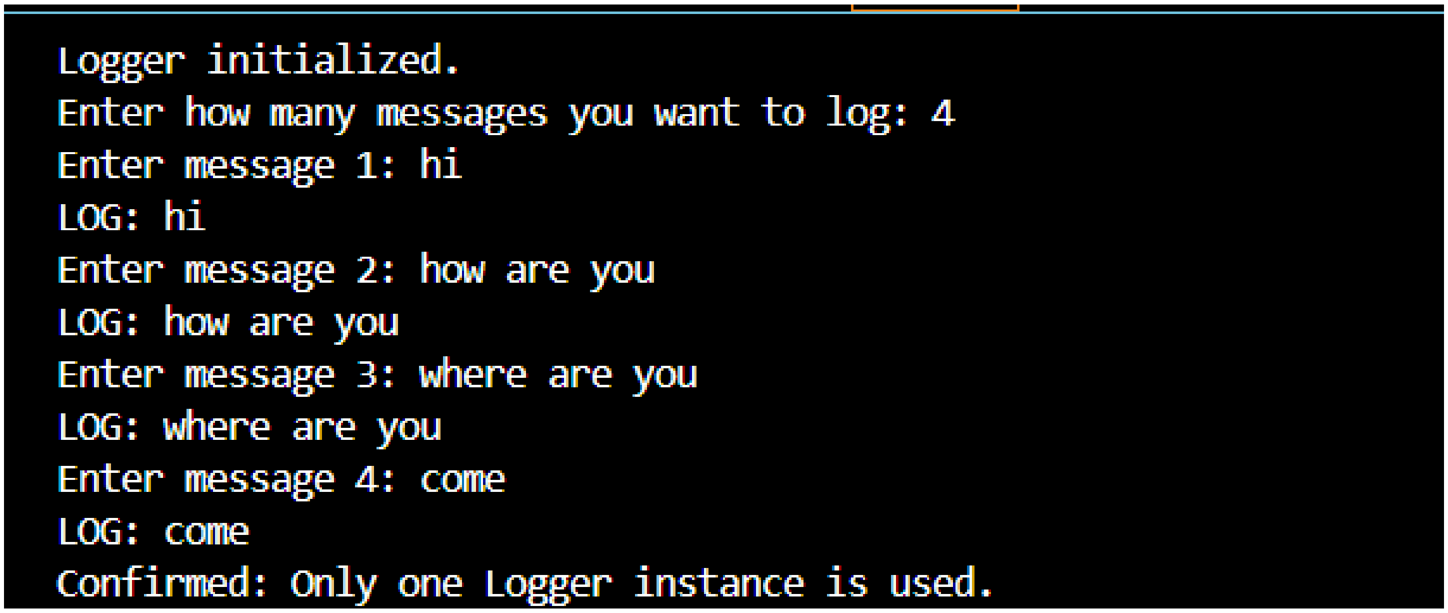
1. **Create a New Java Project:**
    - **Create a new Java project named SingletonPatternExample.**
2. **Define a Singleton Class:**
    - **Create a class named Logger that has a private static instance of itself.**
    - **Ensure the constructor of Logger is private.**
    - **Provide a public static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

SOLUTION:

```
//Logger.java
public 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.java
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Logger logger = Logger.getInstance();
        System.out.print("Enter how many messages you want to log: ");
        int count = scanner.nextInt();
        scanner.nextLine();
        for (int i = 1; i <= count; i++) {
            System.out.print("Enter message " + i + ": ");
            String message = scanner.nextLine();
            logger.log(message);
        }
        Logger anotherLogger = Logger.getInstance();
        if (logger == anotherLogger) {
            System.out.println("Confirmed: Only one Logger instance is used.");
        } else {
            System.out.println("Different Logger instances found.");
        }
        scanner.close();
    }
}
```

OUTPUT:



```
Logger initialized.
Enter how many messages you want to log: 4
Enter message 1: hi
LOG: hi
Enter message 2: how are you
LOG: how are you
Enter message 3: where are you
LOG: where are you
Enter message 4: come
LOG: come
Confirmed: Only one Logger instance is used.
```

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.**

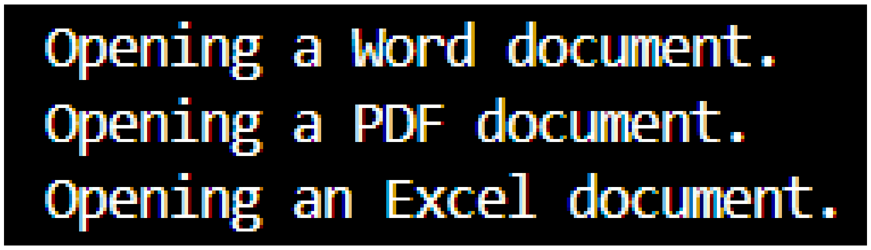
SOLUTION:

```
//Document.java
public interface Document {
   void open();
}

//WordDocument.java
public class WordDocument implements Document {
   public void open() {
      System.out.println("Opening a Word document.");
   }
}

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

//ExcelDocument.java
```

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

//DocumentFactory.java
public abstract class DocumentFactory {
    public abstract Document createDocument();
}

//WordDocumentFactory.java
public class WordDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new WordDocument();
    }
}

//PdfDocumentFactory.java
public class PdfDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}

//ExcelDocumentFactory.java
public class ExcelDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}

//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:



```
Opening a Word document.
Opening a PDF document.
Opening an Excel document.
```

NAME-RITIKA KUMARI

SUPERSET ID- 6392654