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:

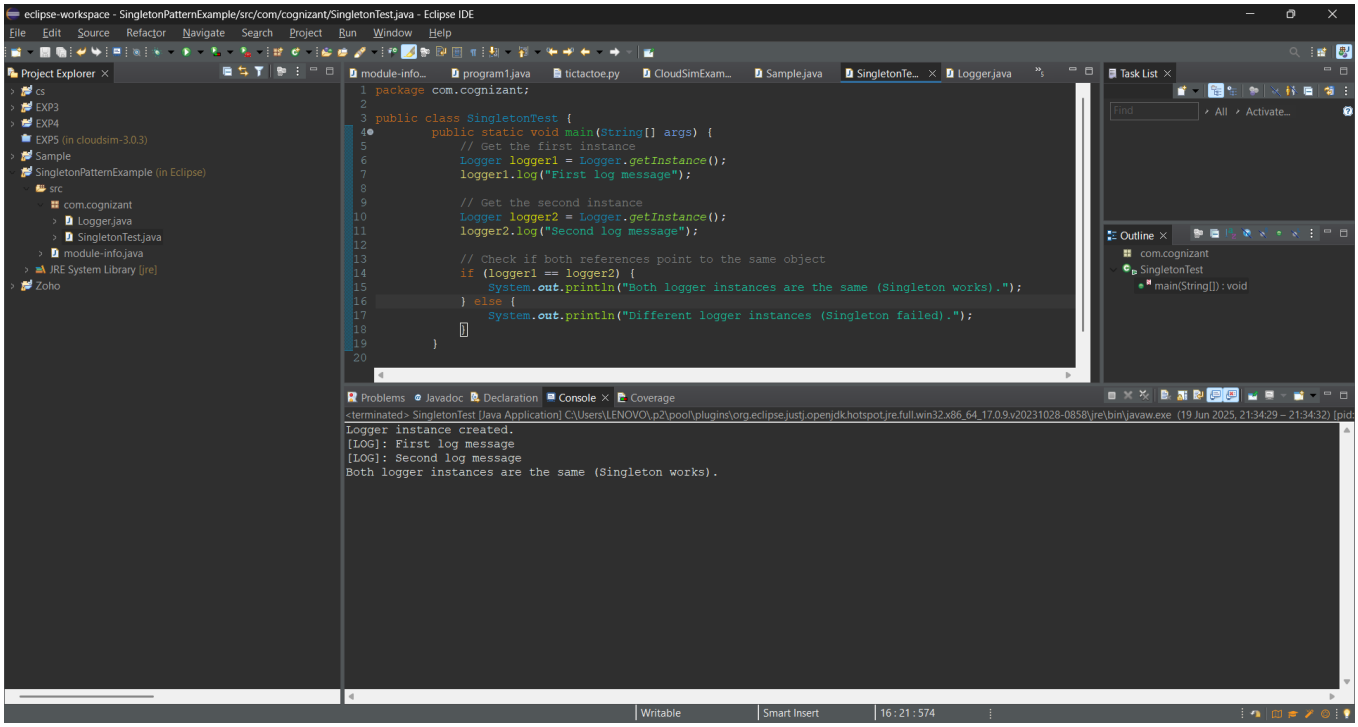
Logger Class:

```
package com.cognizant;

public class Logger {
          // Private static instance of the same class
          private static Logger instance;

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

          // Public static method to return the single instance
          public static Logger getInstance() {
              if (instance == null) {
                  instance = new Logger();
              }
              return instance;
          }

          // Sample logging method
          public void log(String message) {
              System.out.println("[LOG]: " + message);
          }

}
```

SingletonTest Class:

```
package com.cognizant;

public class SingletonTest {
          public static void main(String[] args) {
               // Get the first instance
               Logger logger1 = Logger.getInstance();
               logger1.log("First log message");

               // Get the second instance
               Logger logger2 = Logger.getInstance();
               logger2.log("Second log message");

               // Check if both references point to the same object
               if (logger1 == logger2) {
                    System.out.println("Both logger instances are the same
(Singleton works).");
               } else {
                    System.out.println("Different logger instances (Singleton
failed).");
               }
          }

}
```

Output Screenshot:



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

```
package com.FactoryMethodPatternExample;

public interface Document {
      void open();
}
```

WordDocument Class:

```
package com.FactoryMethodPatternExample;

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


}
```

PdfDocument Class:

```
package com.FactoryMethodPatternExample;

public class PdfDocument implements Document{
          private static int count = 0;
          private int id;

          public PdfDocument() {
              id = ++count;
          }

          @Override
          public void open() {
              System.out.println("Opening PDF Document #" + id);
          }


}
```

ExcelDocument Class:

```
package com.FactoryMethodPatternExample;

public class ExcelDocument implements Document{
          private static int count = 0;
          private int id;

          public ExcelDocument() {
              id = ++count;
          }

          @Override
          public void open() {
              System.out.println("Opening Excel Document #" + id);
          }


}
```

Abstract DocumentFactory Class:

```
package com.FactoryMethodPatternExample;

public abstract class DocumentFactory {
       public abstract Document createDocument();

}
```

WordDocumentFactory Class:

```
package com.FactoryMethodPatternExample;

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


}
```

PdfDocumentFactory Class:

```
package com.FactoryMethodPatternExample;

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


}
```

ExcelDocumentFactory Class:

```
package com.FactoryMethodPatternExample;

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

}
```

FactoryPatternTest Class:

```
package com.FactoryMethodPatternExample;

public class FactoryPatternTest {
          public static void main(String[] args) {
              DocumentFactory wordFactory = new WordDocumentFactory();
              DocumentFactory pdfFactory = new PdfDocumentFactory();
              DocumentFactory excelFactory = new ExcelDocumentFactory();

              Document word1 = wordFactory.createDocument();
              Document word2 = wordFactory.createDocument();
              Document pdf1 = pdfFactory.createDocument();
```

```
                Document excel1 = excelFactory.createDocument();
                Document pdf2 = pdfFactory.createDocument();

                word1.open();
                word2.open();
                pdf1.open();
                excel1.open();
                pdf2.open();          }


}
```

Output Screenshot: