**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.  
  
**Code**

**Logger.java**

package DesignPrinciplesAndPatterns;  
public class Logger {  
  
 private static Logger *instance*;  
  
 private Logger() { }  
  
 public static synchronized Logger getInstance() {  
 if (*instance* == null) {  
 *instance* = new Logger();  
 }  
 return *instance*;  
 }  
  
 public void log(String message) {  
 System.*out*.println("[LOG] " + message);  
 }  
}

**LoggerTest.Java**

package DesignPrinciplesAndPatterns;  
  
public class LoggerTest {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.*getInstance*();  
 Logger logger2 = Logger.*getInstance*();  
  
 if (logger1 == logger2) {  
 System.*out*.println("Singleton works! Same instance.");  
 } else {  
 System.*out*.println("Singleton failed! Different instances.");  
 }  
  
 logger1.log("First message");  
 logger2.log("Second message");  
 }  
}

**Output**A screenshot of a computer

AI-generated content may be incorrect.

**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.

**Code**

package factorymethod;public interface Document {  
 void open();  
 void close();  
 void save(String content);  
}

package factorymethod;  
  
public class WordDocument implements Document {  
 @Override public void open() { System.*out*.println("Opening Word doc"); }  
 @Override public void close() { System.*out*.println("Closing Word doc"); }  
 @Override public void save(String content) {  
 System.*out*.println("Saving to Word doc: " + content);  
 }  
}

package factorymethod;  
  
public class PdfDocument implements Document {  
 @Override public void open() { System.*out*.println("Opening PDF"); }  
 @Override public void close() { System.*out*.println("Closing PDF"); }  
 @Override public void save(String content) {  
 System.*out*.println("Saving to PDF: " + content);  
 }  
}

package factorymethod;  
  
public class ExcelDocument implements Document {  
 @Override public void open() { System.*out*.println("Opening Excel sheet"); }  
 @Override public void close() { System.*out*.println("Closing Excel sheet"); }  
 @Override public void save(String content) {  
 System.*out*.println("Saving to Excel sheet: " + content);  
 }  
}

package factorymethod;  
  
*/\*\* Defines the factory method \*/*public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

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

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

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

**FactortmethodTest.java**

package factorymethod;  
  
public class FactoryMethodTest {  
 public static void main(String[] args) {  
   
 DocumentFactory wordFac = new WordFactory();  
 DocumentFactory pdfFac = new PdfFactory();  
 DocumentFactory excelFac = new ExcelFactory();  
  
  
 Document doc1 = wordFac.createDocument();  
 Document doc2 = pdfFac.createDocument();  
 Document doc3 = excelFac.createDocument();  
  
  
 doc1.open(); doc1.save("Hello WORD"); doc1.close();  
 doc2.open(); doc2.save("Hello PDF"); doc2.close();  
 doc3.open(); doc3.save("Hello EXCEL"); doc3.close();  
 }  
}

**Output**

A screenshot of a computer

AI-generated content may be incorrect.