**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:**

**i)Created a project folder named SingletonPattern Example**

**ii)Defining and Creating Singletonclass named Logger.java**

package SingletonPatternExample;

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);

    }

}

**iv)Testing SingletonPattern**

package SingletonPatternExample;

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

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

        Logger logger2 = Logger.getInstance();

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

        if (logger1 == logger2) {

            System.out.println("Same logger instance used.");

        } else {

            System.out.println("Different instances! Singleton failed.");

        }

    }

}

**Output:**

**A screen shot 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.

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:**

**i)**Created a new Project Folder FactroyPattern

ii)Defined Document Class consistingdifferent document types such as WordDocument, PdfDocument, and ExcelDocument.

public interface Document {

    void open();

    void close();

}

public class ExcelDocument implements Document {

    public void open() {

        System.out.println("Opening Excel Document");

    }

    public void close() {

        System.out.println("Closing Excel Document");

    }

}

public class WordDocument implements Document {

    public void open() {

        System.out.println("Opening Word Document");

    }

    public void close() {

        System.out.println("Closing Word Document");

    }

}

public class PdfDocument implements Document {

    public void open() {

        System.out.println("Opening PDF Document");

    }

    public void close() {

        System.out.println("Closing PDF Document");

    }

}

Iii,iv)

public abstract class DocumentFactory {

    public abstract Document createDocument();

}

public class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

public class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

public class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

v)

public class Main {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        Document wordDoc = wordFactory.createDocument();

        wordDoc.open();

        wordDoc.close();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        Document pdfDoc = pdfFactory.createDocument();

        pdfDoc.open();

        pdfDoc.close();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

        Document excelDoc = excelFactory.createDocument();

        excelDoc.open();

        excelDoc.close();

    }

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.