DESIGN PATTERNS AND PRINCIPLES

Exercise 1: Singleton Pattern Implementation

Project Name: `SingletonPatternExample`

1. `Logger.java` (Singleton Class)

```
public class Logger {
    // Private static instance of Logger (Eager initialization)
    private static Logger instance = new Logger();

    // Private constructor to prevent instantiation
    private Logger() {
        System.out.println("Logger Initialized");
    }

    // Public static method to get the instance
    public static Logger getInstance() {
        return instance;
    }

    // Logging method for demonstration
    public void log(String message) {
        System.out.println("Log: " + message);
    }
}
```

2. `Main.java` (Test Class)

```
public class Main {
    public static void main(String[] args) {
        Logger logger1 = Logger.getInstance();
        Logger logger2 = Logger.getInstance();

        logger1.log("Application started");
        logger2.log("Another log message");
```

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

Exercise 2: Factory Method Pattern Implementation

Project Name: `FactoryMethodPatternExample`

1. `Document.java` (Interface)

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

2. Concrete Implementations

`WordDocument.java`

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

`PdfDocument.java`

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

`ExcelDocument.java`

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

```
}
```

3. `DocumentFactory.java` (Abstract Factory)

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

4. Concrete Factories

`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();
    }
}
```

5. `FactoryPatternTest.java` (Test Class)

```
public class FactoryPatternTest {
    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();
    }
}
```

Summary

- **Singleton Pattern:** Ensures a class (Logger) has only one instance with global access.
- **Factory Method Pattern:** Creates instances of different documents using a factory interface and concrete factory classes, promoting loose coupling and scalability.