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 of Exercise 1

Code

Logger.java

```
public class Logger {
    private static Logger instance;

    private Logger() {

    }

    public static Logger getInstance() {
        if(instance == null) {
            instance = new Logger();
        }
        return instance;
    }
}
```

Test.java

```
public class Test {
    public static void main(String[] args) {
       Logger loggerObj1 = Logger.getInstance();
       Logger loggerObj2 = Logger.getInstance();
       Logger loggerObj3 = Logger.getInstance();
       // Logger loggerObj4 = new Logger();  // compilation error


       System.out.println("Address of first object: " + loggerObj1);
       System.out.println("Address of second object: " + loggerObj2);
       System.out.println("Address of third object: " + loggerObj3);
    }
}
```

Output Screenshots

```
[Running] cd "d:\DN-4.0-Java\C1-Engineering concepts\solutions\" && javac Test.java && java Test
Address of first object: Logger@2c7b84de
Address of second object: Logger@2c7b84de
Address of third object: Logger@2c7b84de

[Done] exited with code=0 in 1.51 seconds
```

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 of Exercise 2

Code

Document.java

```
package FactoryMethodPatternExample;

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

WordDocument.java

```
package FactoryMethodPatternExample;

public class WordDocument implements Document{
    public void open() {
```

```
        System.out.println("opening word doc");
    }

    public void close() {
        System.out.println("closing word doc");
    }
}
```

PdfDocument.java

```
package FactoryMethodPatternExample;

public class PdfDocument implements Document{
    public void open() {
        System.out.println("opening pdf doc");
    }

    public void close() {
        System.out.println("closing pdf doc");
    }
}
```

ExcelDocument.java

```
package FactoryMethodPatternExample;
public class ExcelDocument implements Document{
    public void open() {
        System.out.println("opening excel doc");
    }

    public void close() {
        System.out.println("closing excel doc");
    }
}
```

DocumentFactory.java

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

WordFactory.java

```
package FactoryMethodPatternExample;

public class WordFactory extends DocumentFactory{
    public Document createDocument() {
        return new WordDocument();
    }
}
```

PdfFactory.java

```
package FactoryMethodPatternExample;

public class PdfFactory extends DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

ExcelFactory.java

```
package FactoryMethodPatternExample;

public class ExcelFactory extends DocumentFactory{
    public Document createDocument() {
        return new ExcelDocument();
    }
}
```

Test.java

```
package FactoryMethodPatternExample;

public class Test {
    public static void main(String[] args) {
        DocumentFactory word = new WordFactory();
        Document worddoc = word.createDocument();
        worddoc.open();


        DocumentFactory pdf = new PdfFactory();
        Document pdfdoc = pdf.createDocument();
        pdfdoc.open();
```

```
        DocumentFactory excel = new ExcelFactory();
        Document exceldoc = excel.createDocument();
        exceldoc.open();


        worddoc.close();
        pdfdoc.close();
        exceldoc.close();
    }
}
```

Output Screenshots

```
opening word doc
opening pdf doc
opening excel doc
closing word doc
closing pdf doc
closing excel doc
PS D:\DN-4.0-Java>
```