EngineeringConcepts

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.

Folder:

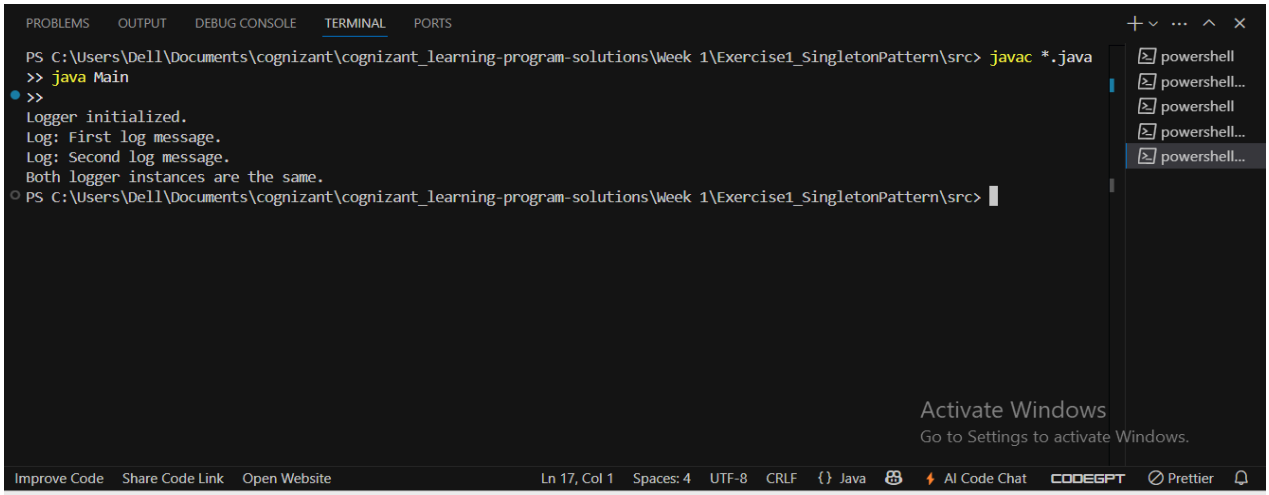
```
Unset
Exercise1_SingletonPattern/src
```

How to Run:

```
Unset

### ▶️ How to Run:
```bash
cd Exercise1_SingletonPattern/src
javac *.java
java Main
```
```

Output:



Exercise 2 – 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()**.

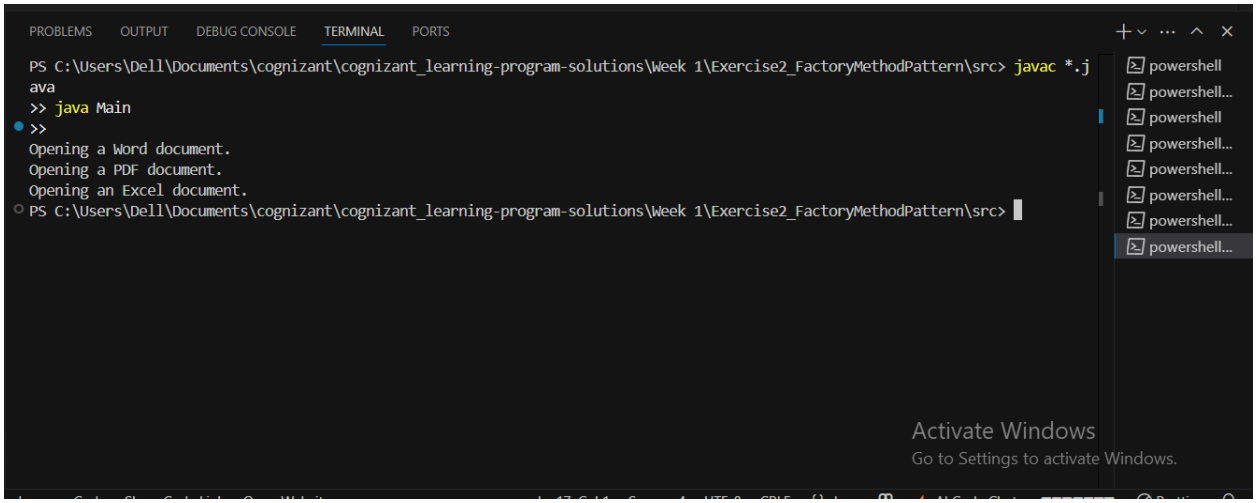
Folder:

```
Unset
Exercise2_FactoryMethodPattern/src
```

How to Run:

```shell
cd Exercise2_FactoryMethodPattern/src
javac *.java
java Main
```

Output:



Exercise 3 – E-Commerce Search Function

Scenario:

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

Steps:

1. Understand Asymptotic Notation:
    - Explain Big O notation and how it helps in analyzing algorithms.
    - Describe the best, average, and worst-case scenarios for search operations.
2. Setup:

    - Create a class Product with attributes for searching, such as productId, productName, and category.
3. Implementation:
    - Implement linear search and binary search algorithms.
    - Store products in an array for linear search and a sorted array for binary search.
4. Analysis:
    - Compare the time complexity of linear and binary search algorithms.
    - Discuss which algorithm is more suitable for your platform and why.

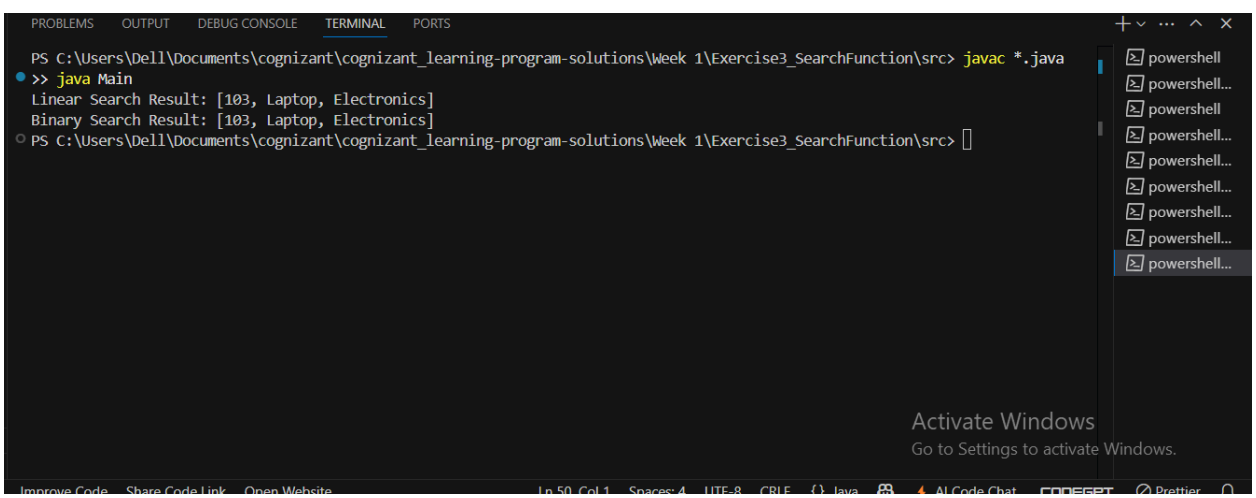
Folder:

```
Unset
Exercise3_SearchFunction/src
```

How to Run:

```
Shell
cd Exercise3_SearchFunction/src
javac *.java
java Main
```

Output:



Time Complexity:

- Linear Search: O(n)
- Binary Search: O(log n)

Exercise 4 – Financial Forecasting (Recursion)

Scenario:

You are developing a financial forecasting tool that predicts future values based on past data.

Steps:

1. **Understand Recursive Algorithms:**
   - Explain the concept of recursion and how it can simplify certain problems.
2. **Setup:**
   - Create a method to calculate the future value using a recursive approach.
3. **Implementation:**
   - Implement a recursive algorithm to predict future values based on past growth rates.
4. **Analysis:**
   - Discuss the time complexity of your recursive algorithm.
   - Explain how to optimize the recursive solution to avoid excessive computation

Folder:

```
Unset
Exercise4_FinancialForecasting
```

How to Run:

```
Shell
cd Exercise4_FinancialForecasting
javac Main.java
java Main
```

Output:

```
Unset
Future value after 5 years: ₹14693.28
```

General Note

- All programs are written in pure Java.
- Compile using `javac *.java` and run using `java Main` from the respective `src` folder.
- Tested in VS Code on Windows using PowerShell.
