Week-1 Hands-on Exercise

Design Pattern and Principles File

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

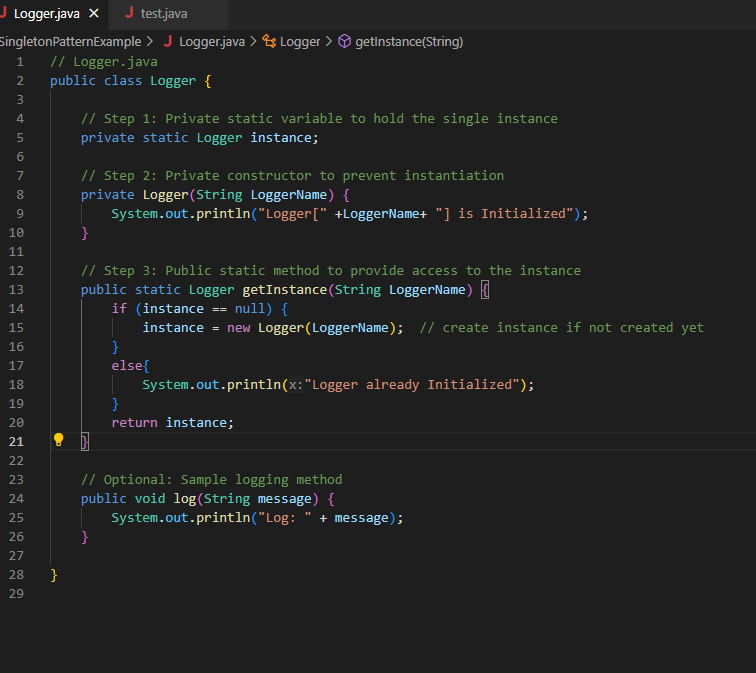
You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

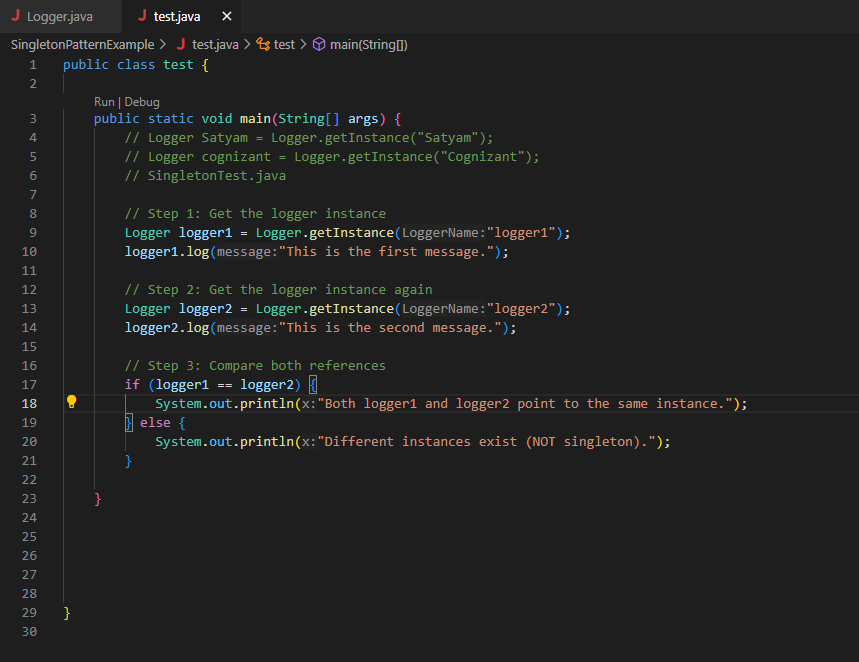
1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to 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.

Code Screenshot:-

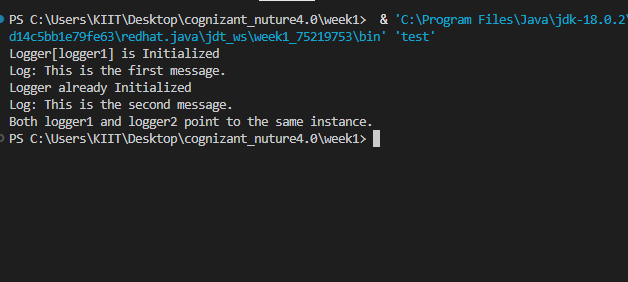
Logger class



Test class



The output of Test class (implementation of Singleton Pattern) in terminal:



**Exercise 2: Implementing the Factory Method Pattern**

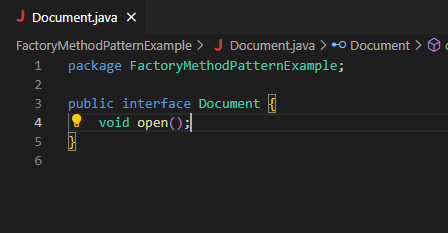
**Scenario:**

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:**

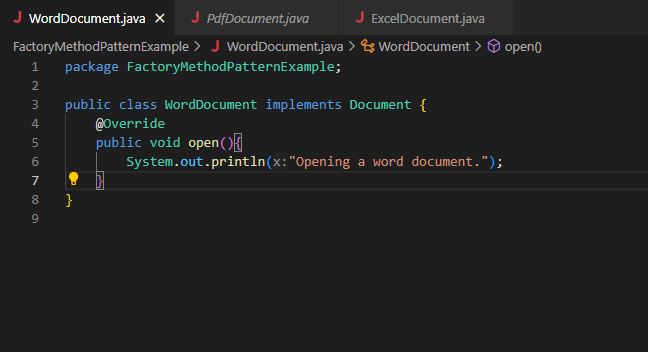
1. **Create a New Java Project:**
   * Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   * Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document 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.

Document interface:

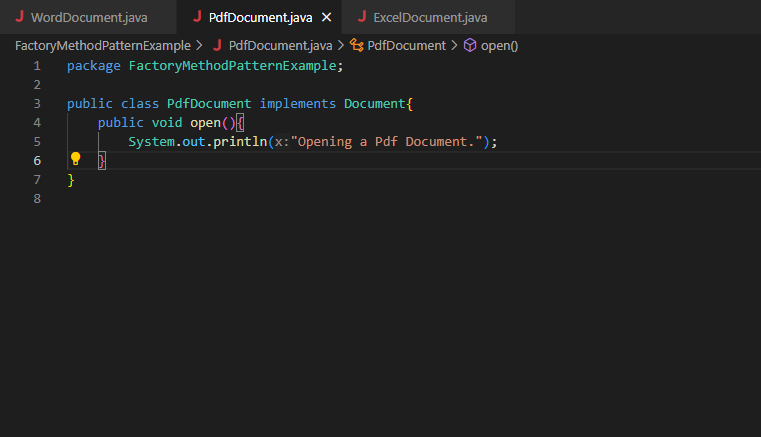


Concrete Document Class:-

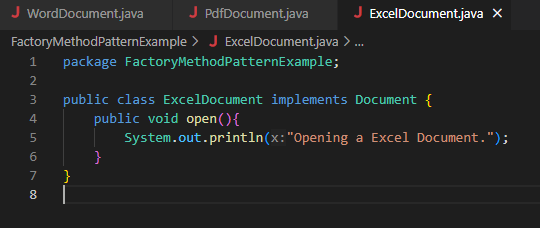
1. WordDocumnet class



1. PdfDocument class

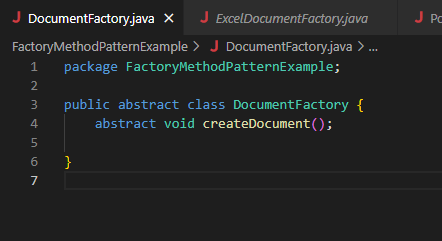


1. ExcelDocument

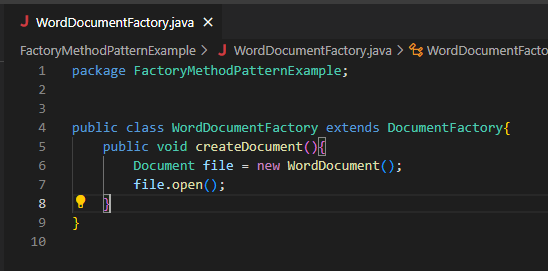


Step - 4:

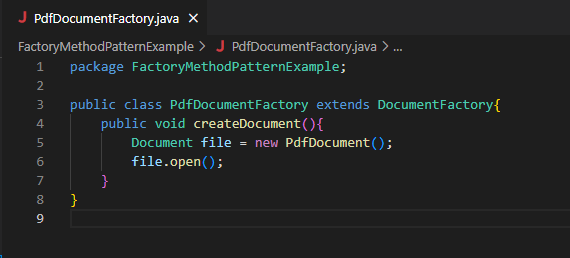
DocumentFactory class:



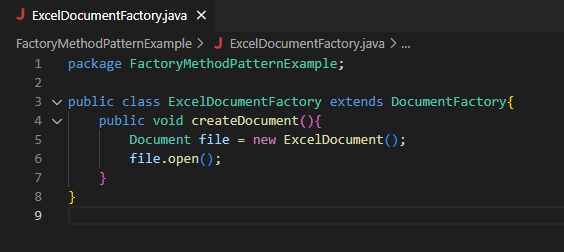
WordDocumentFactory class:



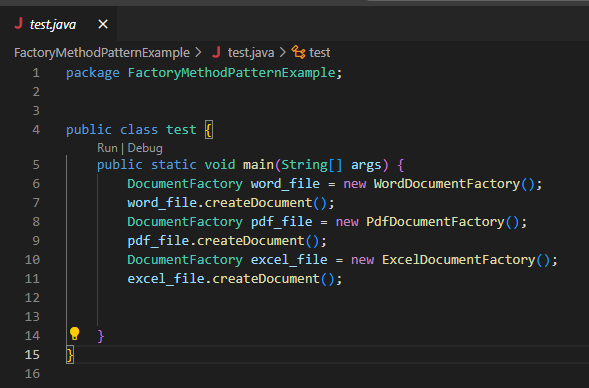
PdfDocumentFactory class:



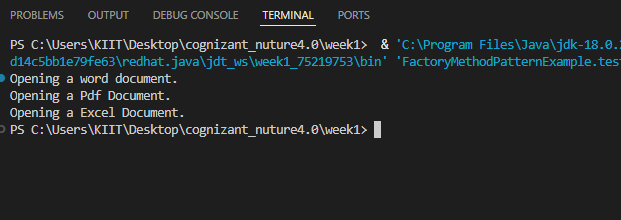
ExcelDocumentFactory class:



Step-5: Test the Factory Method Implementation:



Output of Test.java



Algorithms Data Structure file

**Exercise 2: E-commerce Platform 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.

Step-1: Understand Asymptotic Notation:

Q) Explain Big O notation and how it helps in analyzing algorithms?

-> Big O notation describes the upper bound of an algorithm's running time in terms of the input size n. It helps us understand how the performance scales as the data grows.

1. Describe the best, average, and worst-case scenarios for search operations?

-> For linear Search

Best case = O(1)

Average case = O(n)

Worst case = O(n)

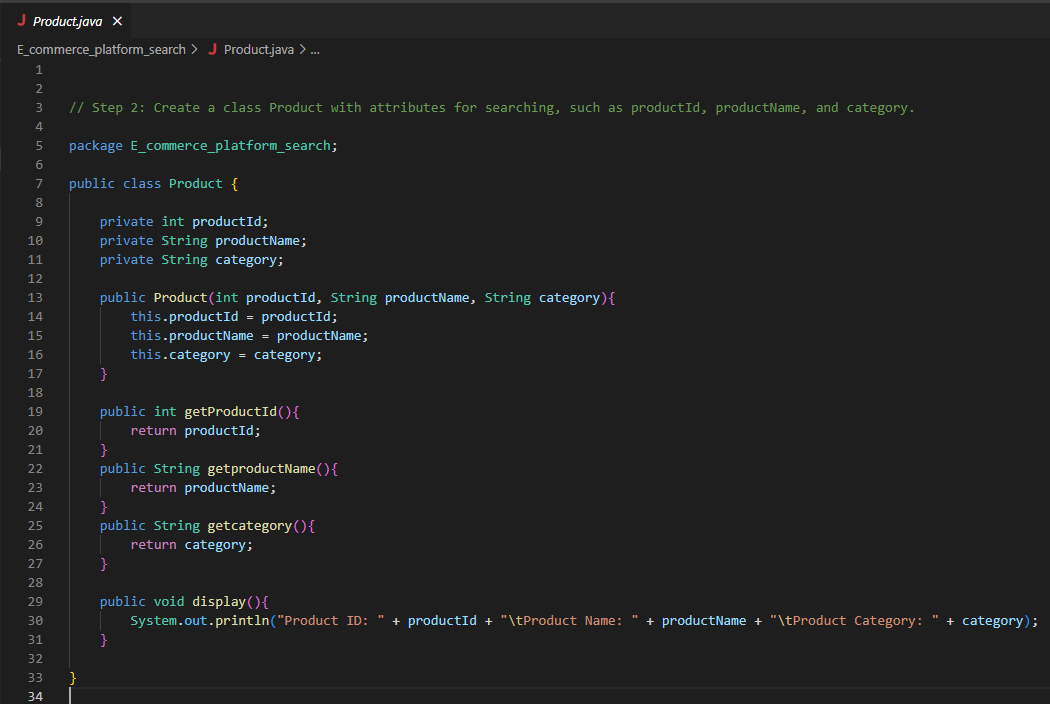
For Binary Search

Best case = O(1)

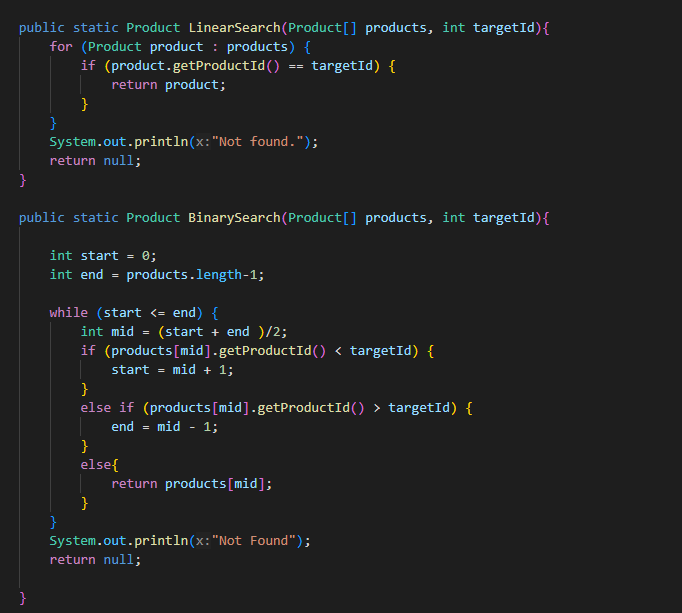
Average case = O(log n)

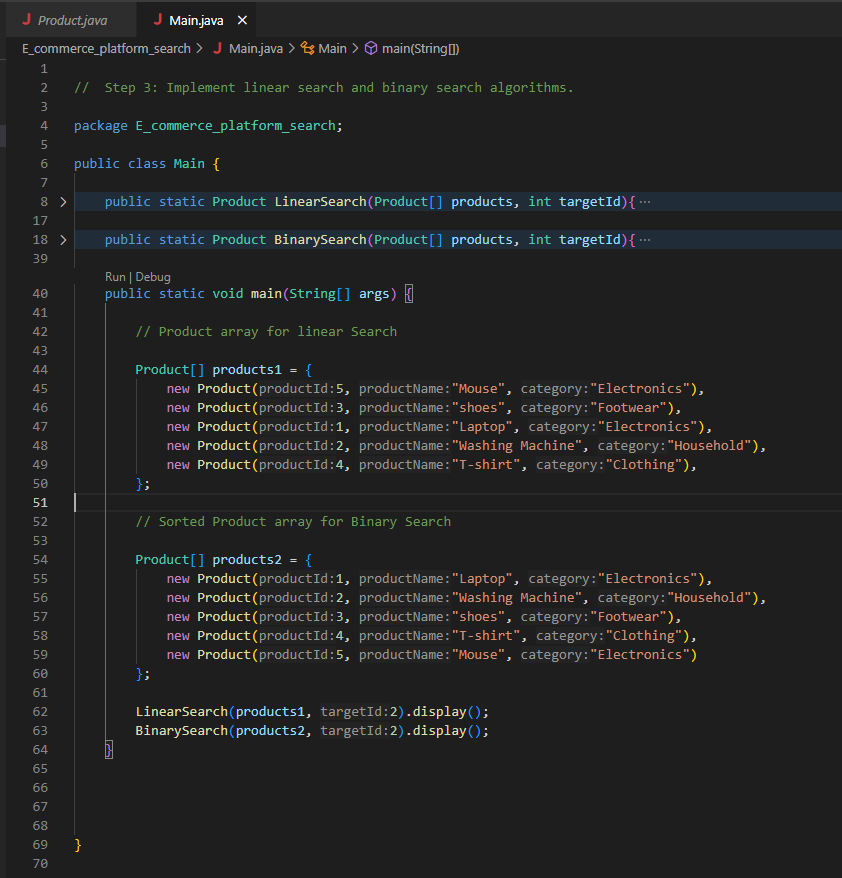
Worst case = O(n)

Step-2: Product class

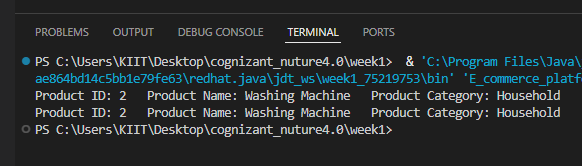


Step-3: Implementation of Product class via Main class





Output of Main.java



Step-4: Analysis

Q) Compare the time complexity of linear and binary search algorithms.

-> Time Complexity for linear Search is O(n) where as for Binary Search is O(log n)

Q) Discuss which algorithm is more suitable for your platform and why?

-> Binary Search is more suitable for the platform as it is faster and have better performance for large datasets.

**Exercise 7: Financial Forecasting**

**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.

Step-1: Understanding Recursive Algorithms:

Q) Explain the concept of recursion and how it can simplify certain problems?

-> Recursion is a programming technique where a method calls itself to solve a problem by breaking it down into smaller subproblems.

    To understand that how it simplifies the certain program, we use factorial example.

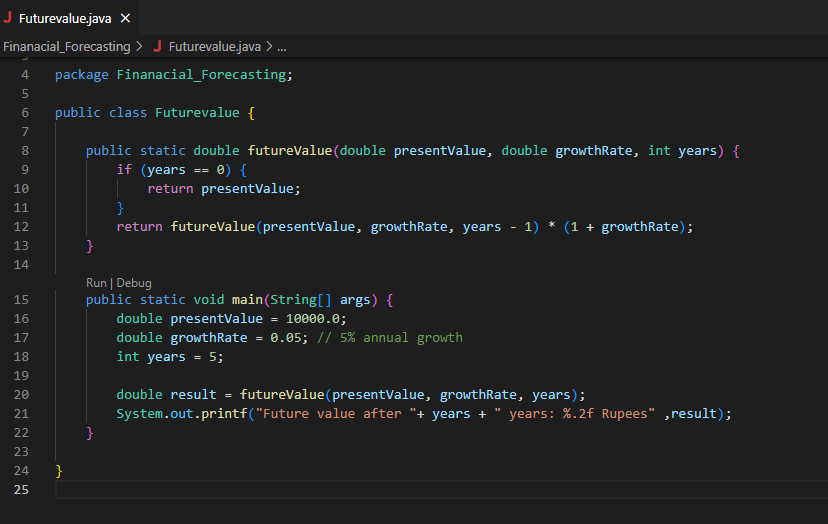
    Factorial of n = n \* (n-1) \* (n-2) \* (n-3) \* .... \* (1).

    In Recursion, we breakdown this problem into simpler ones,

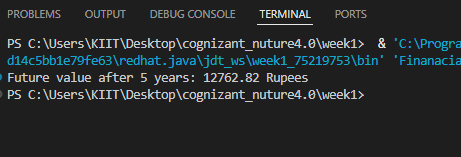
        factorial(n) = n \* factorial(n-1)

    like this factorial(n) calls factorial(n-1) and so on till n = 1.

Step-2: Creating Futurevalue class



Step-3: Implementation



Step 4: Analysis

Q) Discuss the time complexity of your recursive algorithm?

-> Time complexity of Recursive algorithms is O(n).