**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.**

**A screenshot of a computer program

AI-generated content may be incorrect. Implementation:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.A screenshot of a computer screen

AI-generated content may be incorrect.  
  
  
A black screen with a black background

AI-generated content may be incorrect.**

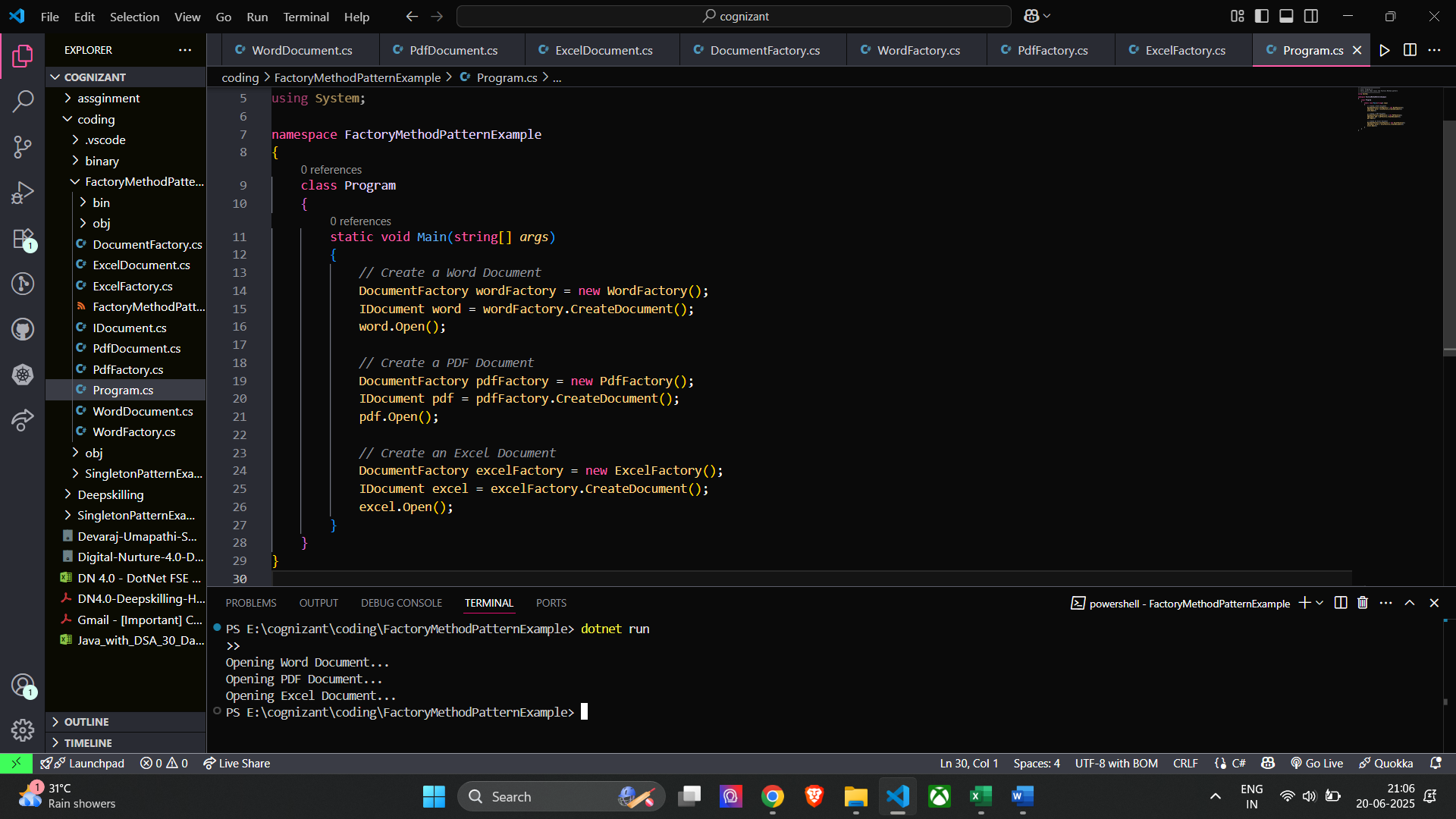
**Exercise 2: Implementing the Factory Method Pattern**

**Scenario:**

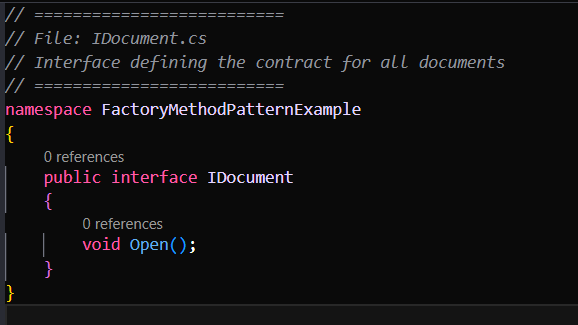
**You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.**

**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.**

**Implementation:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of text

AI-generated content may be incorrect.**

**A computer screen with white text

AI-generated content may be incorrect.A computer screen shot of a program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.**

**Algorithms Data Structures**

**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**

**A computer screen shot of a black screen

AI-generated content may be incorrect.Implementation:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Time Complexity Comparison**

**Linear Search:**

* **Best Case: O(1) → Item is found at the beginning.**
* **Worst Case: O(n) → Item is at the end or not present.**
* **Performance: Slower for large product lists because it checks each item one by one.**

**Binary Search:**

* **Best Case: O(1) → Item is found in the middle.**
* **Worst Case: O(log n) → Efficient even for large data because it cuts the list in half with each step.**
* **Performance: Much faster for large, sorted data.**

**In a real-world e-commerce platform, where thousands or even millions of products may exist, Binary Search is the better choice because:**

* **It is much faster when searching through large amounts of data.**
* **It reduces the number of comparisons needed by continuously dividing the search range.**
* **It is ideal when the product list is sorted, like by product ID or price.**

**On the other hand, Linear Search is:**

* **Simple and works on unsorted data.**
* **Useful for very small product lists or when you need to check every product manually (like for keyword matches or filters).**

**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.**

**A screenshot of a computer program

AI-generated content may be incorrect.Implementation :**

**A screenshot of a computer program

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.**

**The concept of recursion and how it can simplify certain problems:**

**Recursion is a programming technique where a function calls itself to solve a smaller or simpler version of the same problem. It continues until it reaches a base case, which is a condition that stops the recursion.**

**It makes code simpler and cleaner, especially for problems that follow a repetitive or self-similar pattern (like calculating interest, traversing a tree, solving factorials, etc.).**

**In financial forecasting, recursion allows us to calculate future values year-by-year, based on the previous year’s result — exactly how compound growth works.**

**Time Complexity of Your Recursive Algorithm:**

**Time Complexity: O(n)**

**Because it makes one recursive call for each year until it reaches year 0. So, for 10 years, it performs 10 calls.**

**Space Complexity: O(n)**

**Optimize the Recursive Solution:**

**Recursion can become inefficient when it repeats calculations for the same values multiple times especially for large inputs.**

**Optimization Technique: Memoization**

* **Store results of each year’s calculation in a dictionary or cache.**
* **If the function is called again for the same input, return the result directly from memory instead of recalculating.**

**This reduces unnecessary work and makes the algorithm much faster.**

**Time Complexity with Memoization:**

* **O(n) — same as before, but much faster because each result is computed only once.**
* **Space Complexity: O(n) — for the memoization dictionary.**