**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.

Here's a complete response for **Exercise 7: Financial Forecasting**, covering all the required steps.

**Exercise 7: Financial Forecasting**

**1. Understand Recursive Algorithms**

**Recursion** is a programming concept where a method calls itself to solve a smaller version of the problem.  
It is useful for problems that can be broken down into smaller, similar subproblems.

Example: Calculating factorial:

int factorial(int n) {

if (n == 0) return 1;

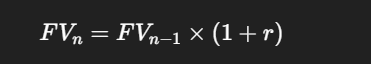
return n \* factorial(n - 1);

}

**In financial forecasting**, recursion can help model future values based on previous years' data using repeated application of a growth formula.

**2. Setup**

Suppose we are forecasting the **future value (FV)** of an investment using this formula:



Where:

* FVnFV\_n is the value at year n
* rr is the annual growth rate
* FV0FV\_0 is the initial investment

**3. Implementation (Java Code)**

public class FinancialForecast {

    // Recursive method to calculate future value

    public static double forecastValue (double currentValue, double rate, int years) {

        if (years == 0) return currentValue; // Base case

        return forecastValue (currentValue \* (1 + rate), rate, years - 1); // Recursive call

    }

    public static void main (String [] args) {

        double initialValue = 10000; // Initial investment

        double growthRate = 0.08;    // 8% growth rate

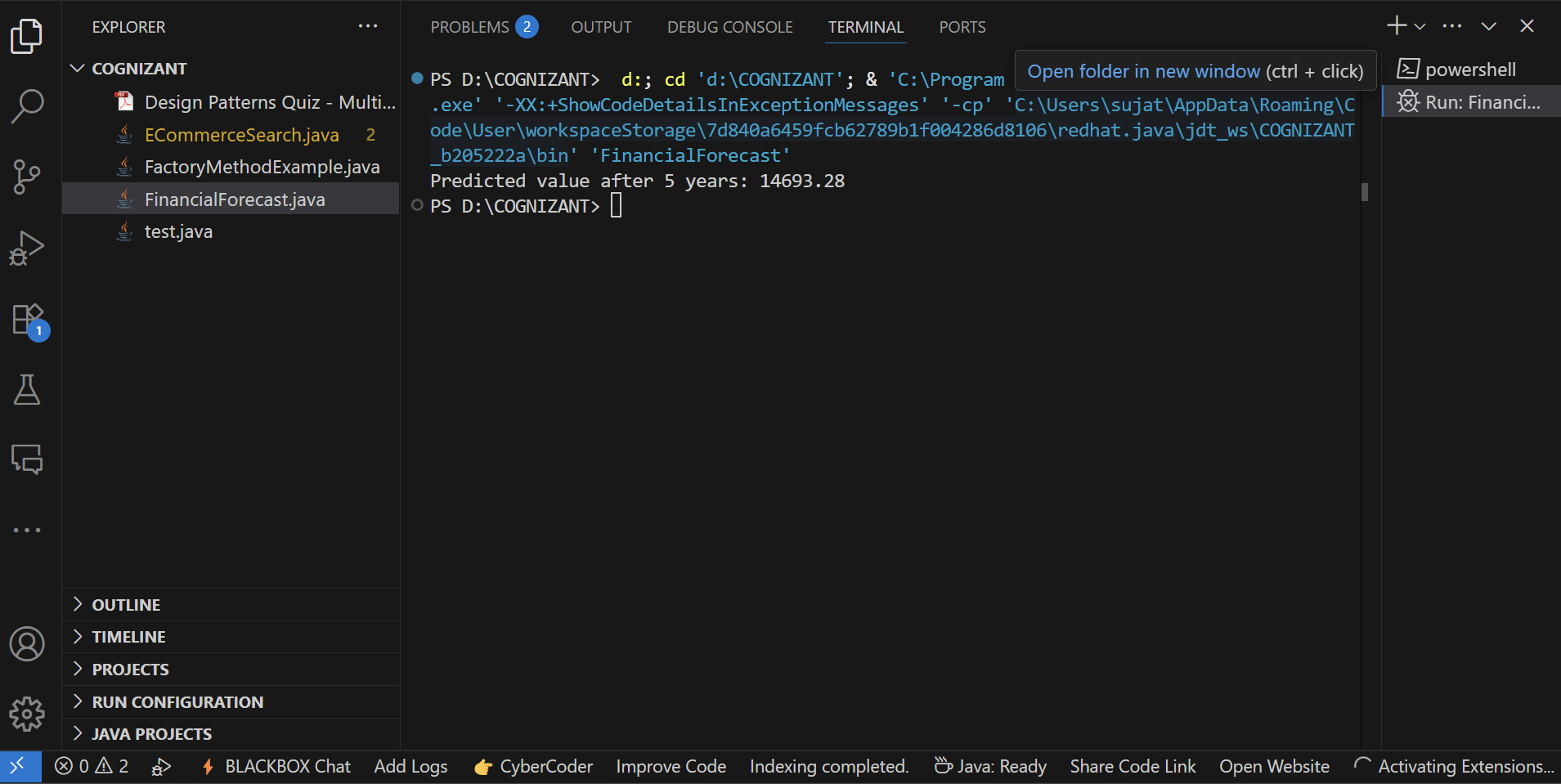
        int futureYears = 5;

        double predictedValue = forecastValue (initialValue, growthRate, futureYears);

        System.out.printf("Predicted value after %d years: %.2f\n", futureYears, predictedValue);

    }

}

**Output:-**

**4. Analysis**

**Time Complexity**

The time complexity of the recursive function is **O(n)**, where n is the number of years, as each call reduces the number of years by 1 until reaching 0.

**Optimization**

**Problem with recursion**:

* Recursive calls can be inefficient for large n due to deep call stacks.

**Optimized approach (using iteration)**:

public static double forecastValueIterative (double currentValue, double rate, int years) {

for (int i = 0; i < years; i++) {

currentValue \*= (1 + rate);

}

return currentValue;

}

* This avoids recursion overhead and stack overflow issues.
* Time complexity remains **O(n)**, but with better space efficiency (no call stack growth).