Exercise 4: 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.

1. Understand Recursive Algorithms:

✓ What is Recursion?

- **Recursion** is when a method **calls itself** with smaller inputs until reaching a **base case**.
- It simplifies problems that can be broken into smaller versions of the same problem.

✓ Why is recursion useful here?

- Financial projections often follow a pattern:

  futureValue(year) = futureValue(year-1) * (1 + growthRate)

- Instead of using loops, recursion lets us directly state the math:

"Future value at year depends on the value of year-1"

2. Setup:

Create a method:

double calculateFutureValue(double presentValue, double growthRate, int years)

Present Value: Starting amount.

growthRate: Yearly rate (e.g. 0.05 for 5%).

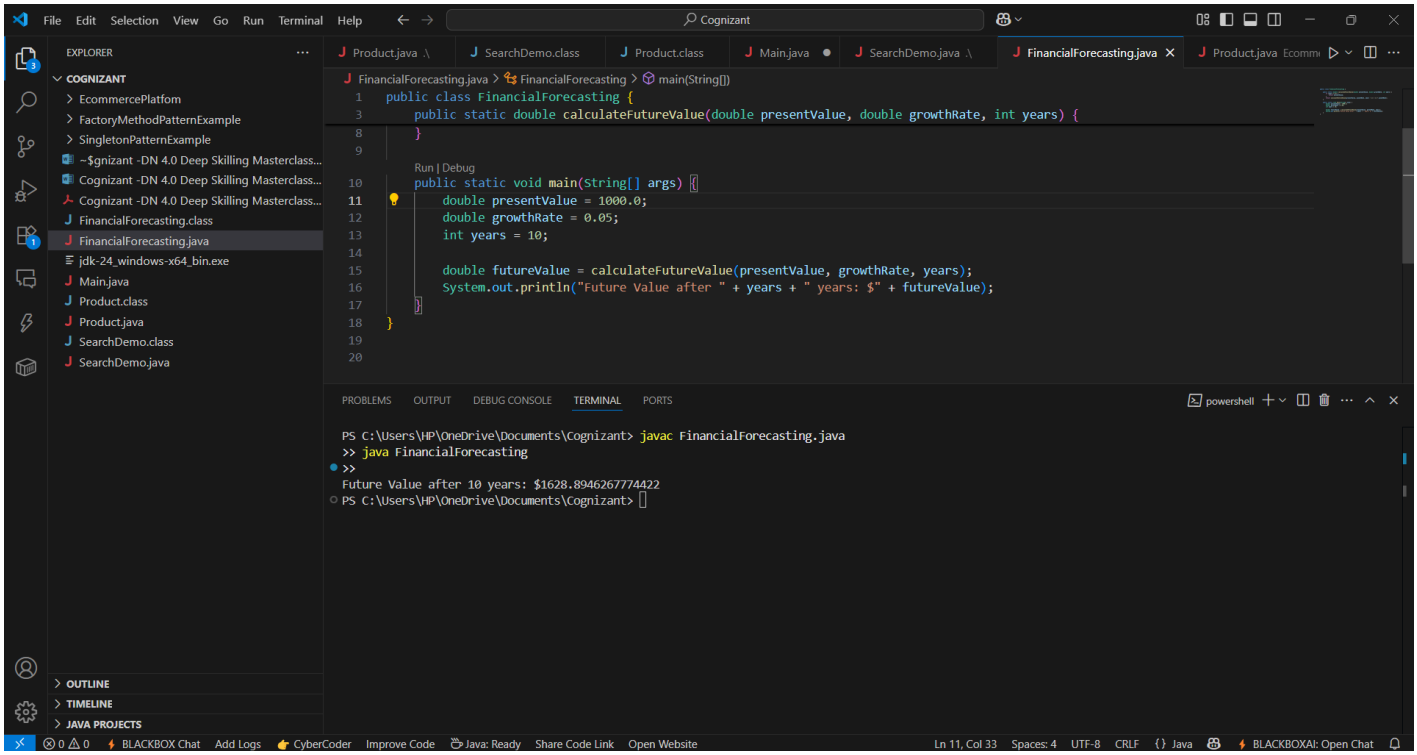
years: Time horizon.

3. Implementation:

Financial Forecasting.java

```
public class FinancialForecasting {
    public static double calculateFutureValue(double presentValue, double growthRate, int years) {
        if (years == 0) {
            return presentValue;
        }
        return calculateFutureValue(presentValue, growthRate, years - 1) * (1 + growthRate);
    }
    public static void main(String[] args) {
        double presentValue = 1000.0;
        double growthRate = 0.05;
        int years = 10;
        double futureValue = calculateFutureValue(presentValue, growthRate, years);
        System.out.println("Future Value after " + years + " years: $" + futureValue);
    }
}
```

OUTPUT:



4. Analysis:

Time Complexity

Recursive call depth = years.

Time complexity = **O(years)** because the method is invoked once per year.

Space complexity = **O(years)** due to the recursive call stack.

4. Optimize:

Iterative Approach

Replace recursion with a simple for loop to save stack space:

```
double value = presentValue;for (int i = 0; i < years; i++) {
        value *= (1 + growthRate);
    }
```

Time complexity = O(years)

Space complexity = O(1)

Direct formula (Closed-form)

Financial math formula:

Future value = presentValue * (1 + growthRate)^years

Calculate this in **O(1)** time:

```
double futureValue = presentValue * Math.pow(1 + growthRate, years);
```

Approach	Time Complexity	Space Complexity	Best Use
Recursion	O(years)	O(years)	Learning
Iteration	O(years)	O(1)	Practical
Formula	O(1)	O(1)	Optimal