**Exercise 7: Financial Forecasting**

**Recursion**

Recursion is a technique where a method calls **itself** to solve a problem by breaking it into **smaller subproblems**.

Recursive methods have two parts:

1. **Base Case** – condition to stop recursion.
2. **Recursive Case** – continues breaking the problem down.

Recursion simplifies problems like Fibonacci sequence, Factorials, Tree traversals and Financial predictions based on repeated patterns .

**Implementation**

**FinancialForecast.java**

public class FinancialForecast {

    // Recursive method to calculate future value

    public static double forecast(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        }

        // Recursive case: apply growth rate and call again

        return forecast(presentValue \* (1 + growthRate), growthRate, years - 1);

    }

    public static void main(String[] args) {

        double presentValue = 1000.0;

        double growthRate = 0.10;

        int years = 5;

        double futureValue = forecast(presentValue, growthRate, years);

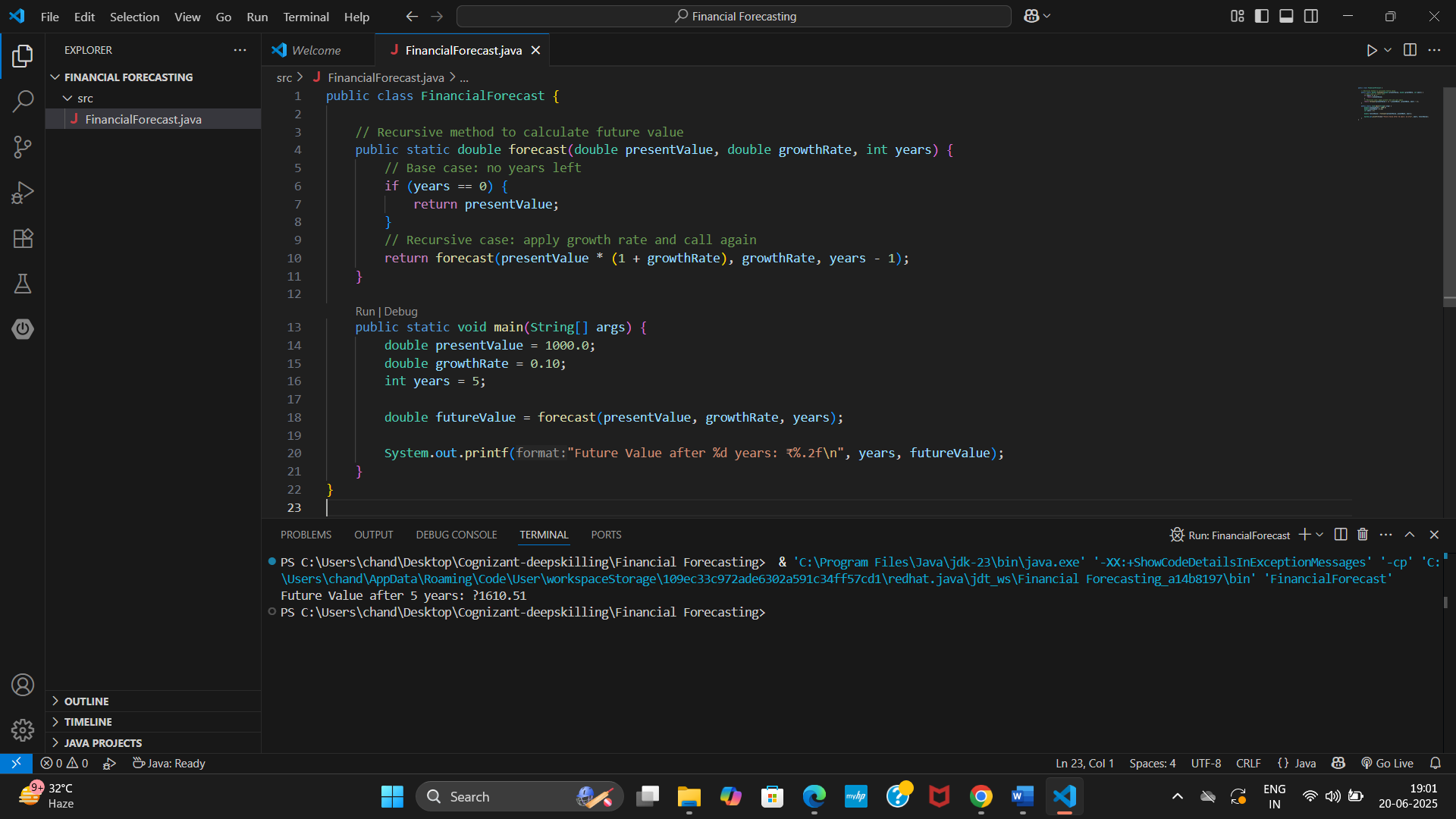
        System.out.printf("Future Value after %d years: ₹%.2f\n", years, futureValue);

    }

}

**Output**

Future Value after 5 years: ₹1610.51

****

**Analysis**

* **Time Complexity**

**Time Complexity**: O(n), where n is the number of years.

Each recursive call decreases years by 1 → leads to **n recursive calls**.

* **Space Complexity**

**Space Complexity**: O(n), due to the **call stack** storing each recursive call.

**Optimization**

Recursive algorithms may become inefficient if:

* There's **no memoization** (especially in overlapping subproblems like Fibonacci).
* The **depth of recursion** is large (risk of StackOverflowError).

**Optimized version:**

public static double forecastIterative(double presentValue, double growthRate, int years) {

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

presentValue \*= (1 + growthRate);

}

return presentValue;

}

This version uses **O(1)** space and is safer for large years.