**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Code:**

package dsa;

public class FinancialForecast {

public static double futureValue(double pv, double r, int n) {

if (n == 0) {

return pv;

}

return *futureValue*(pv \* (1 + r), r, n - 1);

}

public static double predictNextValue(double[] pastValues, int periods) {

if (pastValues.length < 2) {

throw new IllegalArgumentException("At least two past values are required to predict.");

}

double lastValue = pastValues[pastValues.length - 1];

double previousValue = pastValues[pastValues.length - 2];

double growthRate = (lastValue - previousValue) / previousValue;

return *futureValue*(lastValue, growthRate, 1);

}

public static void main(String[] args) {

double[] pastValues = {1000, 1100, 1240, 1332};

double ans = *predictNextValue*(pastValues, pastValues.length);

System.*out*.printf("Predicted Next Year Value (Recursive): %.2f\n", ans);

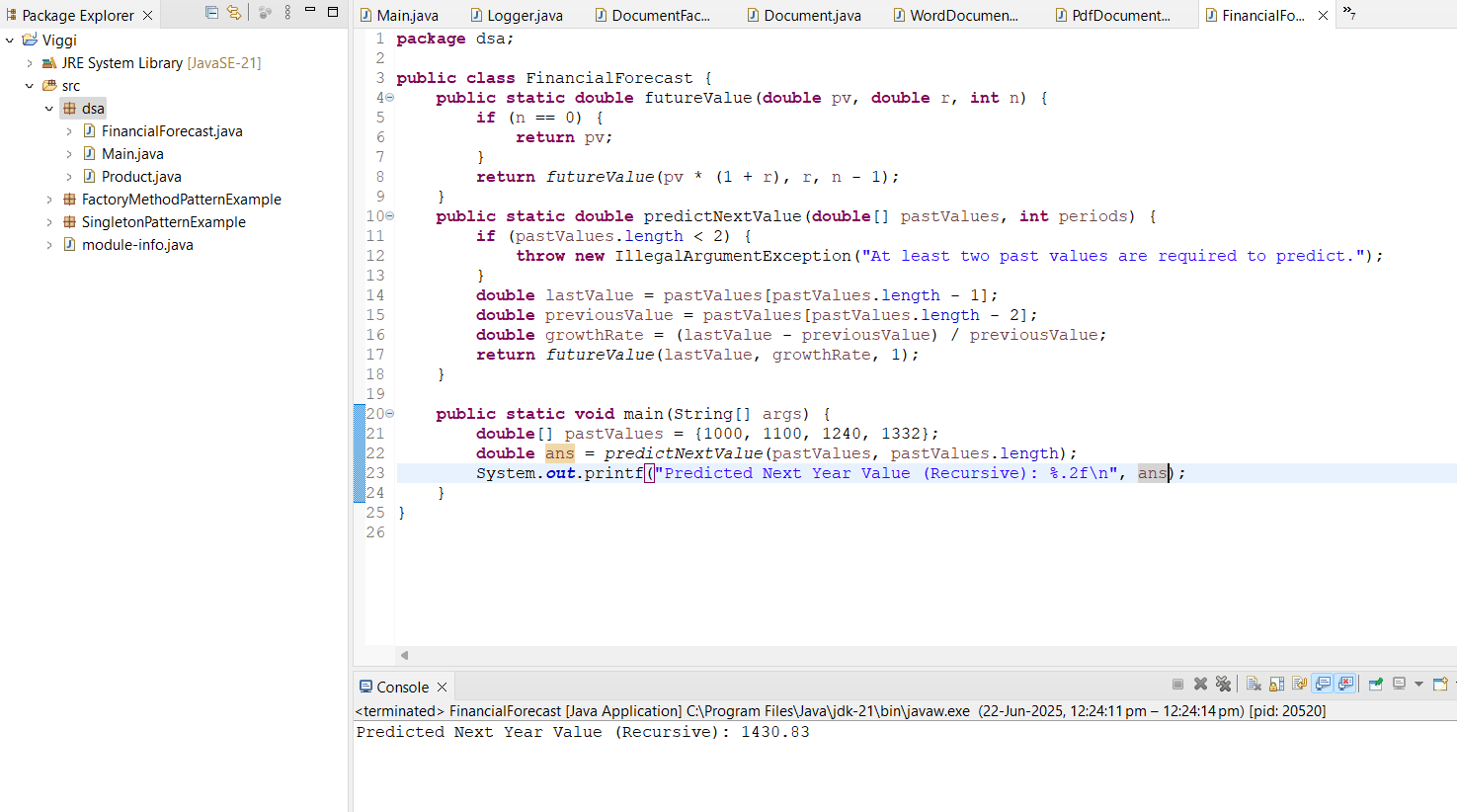
}

}

**Output:**

**Time Complexity:**The time complexity of the recursive algorithm is “O(n)” , where n is the number of periods. This is because the function makes a single recursive call for each period, leading to n calls in total.

**Optimization**: To optimize the recursive solution and avoid excessive computation, we can use memoization. This technique involves storing the results of expensive function calls and reusing them when the same inputs occur again. This can significantly reduce the number of recursive calls and improve performance.

****