**DIGITAL\_NURTURE\_4.0**

**Exercise 2 & 7**

**Superset ID – 6404833**

**Exercise 2: E-commerce Platform Search Function**

**Code**

import java.util.Arrays;

import java.util.Comparator;

class Product {

    int productId;

    String productName;

    String category;

    public Product(int productId, String productName, String category) {

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    @Override

    public String toString() {

        return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

    }

}

public class EcommerceSearch {

    public static void main(String[] args) {

        Product[] products = {

            new Product(101, "Laptop", "Electronics"),

            new Product(102, "Phone", "Electronics"),

            new Product(103, "Shirt", "Clothing"),

            new Product(104, "Shoes", "Footwear"),

            new Product(105, "Book", "Stationery")

        };

        System.out.println(" Linear Search:");

        Product result1 = linearSearch(products, "Shoes");

        System.out.println(result1 != null ? result1 : "Product not found");

        System.out.println("\n Binary Search:");

        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

        Product result2 = binarySearch(products, "Shoes");

        System.out.println(result2 != null ? result2 : "Product not found");

}

    // Linear Search

    public static Product linearSearch(Product[] products, String targetName) {

        for (Product product : products) {

            if (product.productName.equalsIgnoreCase(targetName)) {

                return product;

            }

        }

        return null;

    }

    // Binary Search (on sorted array)

    public static Product binarySearch(Product[] products, String targetName) {

        int low = 0, high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int compare = products[mid].productName.compareToIgnoreCase(targetName);

            if (compare == 0) return products[mid];

            else if (compare < 0) low = mid + 1;

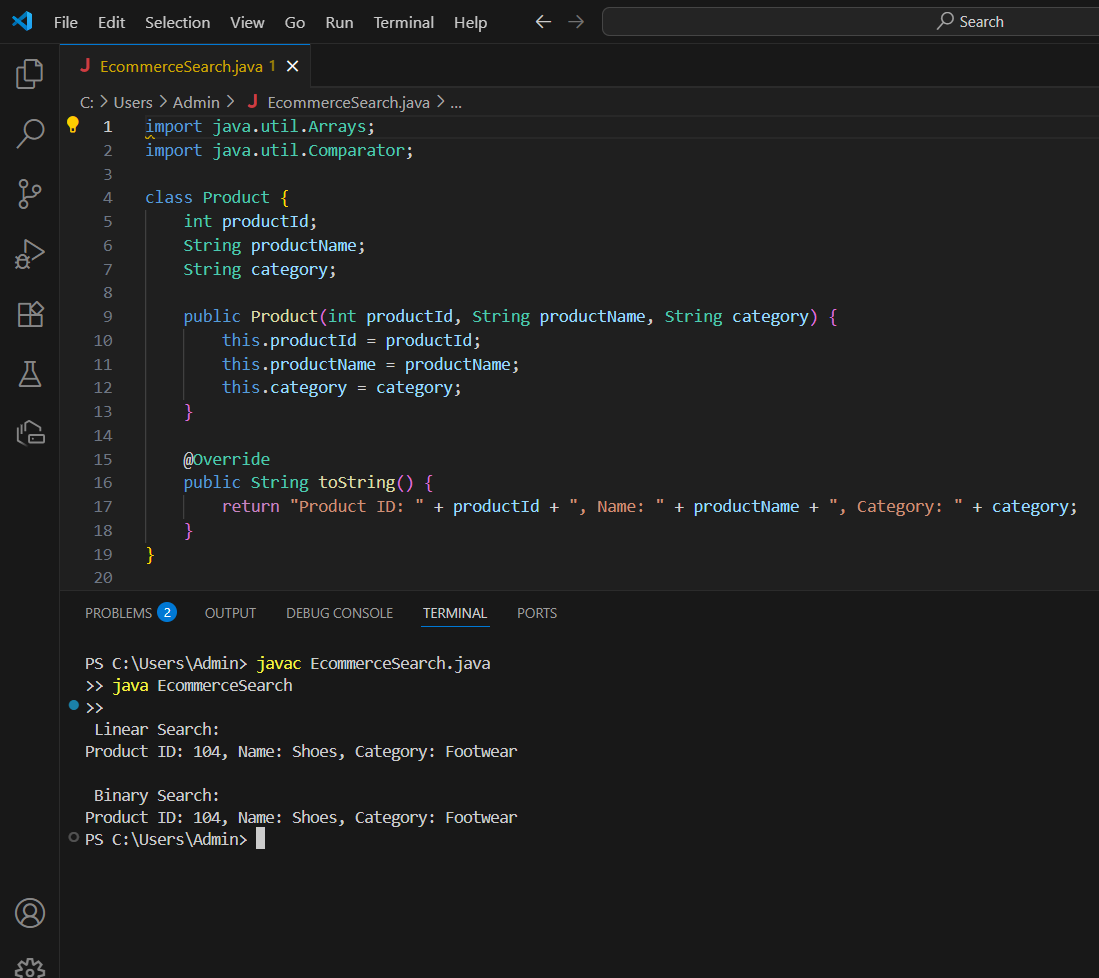
            else high = mid - 1;

        }

        return null;

    }

}

**Output**

**Analysis**

**1.Time Complexity**

|  |  |  |
| --- | --- | --- |
| **Search Type** | **Time Complexity** | **Suitable for** |
| Linear Search | O(n) | Small or unsorted data |
| Binary Search | O(log n) | Large sorted datasets |

**2. Which is Better?**

* **Binary Search** is much faster for large, sorted product lists.
* **Linear Search** is useful when the dataset is small or **unsorted**.
* For an e-commerce platform with **thousands of products**, **Binary Search** is the best choice for performance.

**Exercise 7: Financial Forecasting**

**Code**

public class FinancialForecast {

    // Recursive method to calculate future value

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

        if (years == 0) {

            return presentValue;

        }

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

    }

    public static void main(String[] args) {

        double presentValue = 10000.0; // Initial value (e.g., revenue)

        double growthRate = 0.10;      // 10% annual growth

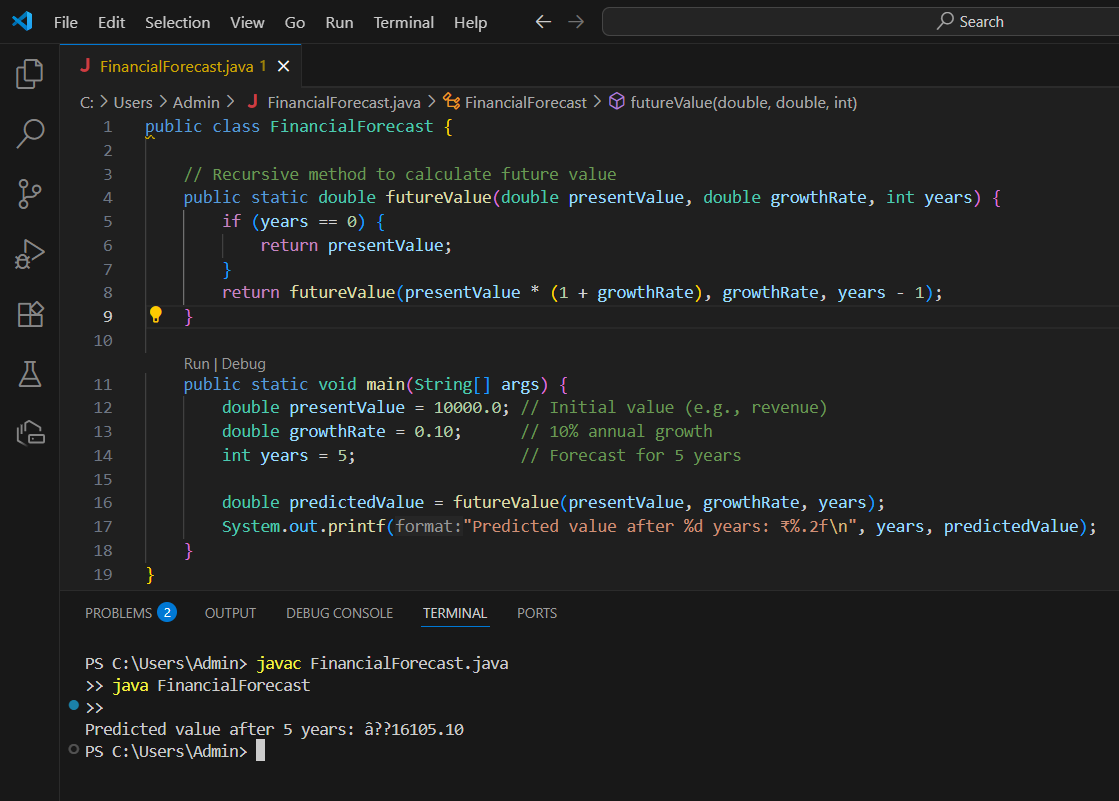
        int years = 5;                 // Forecast for 5 years

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

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

    }

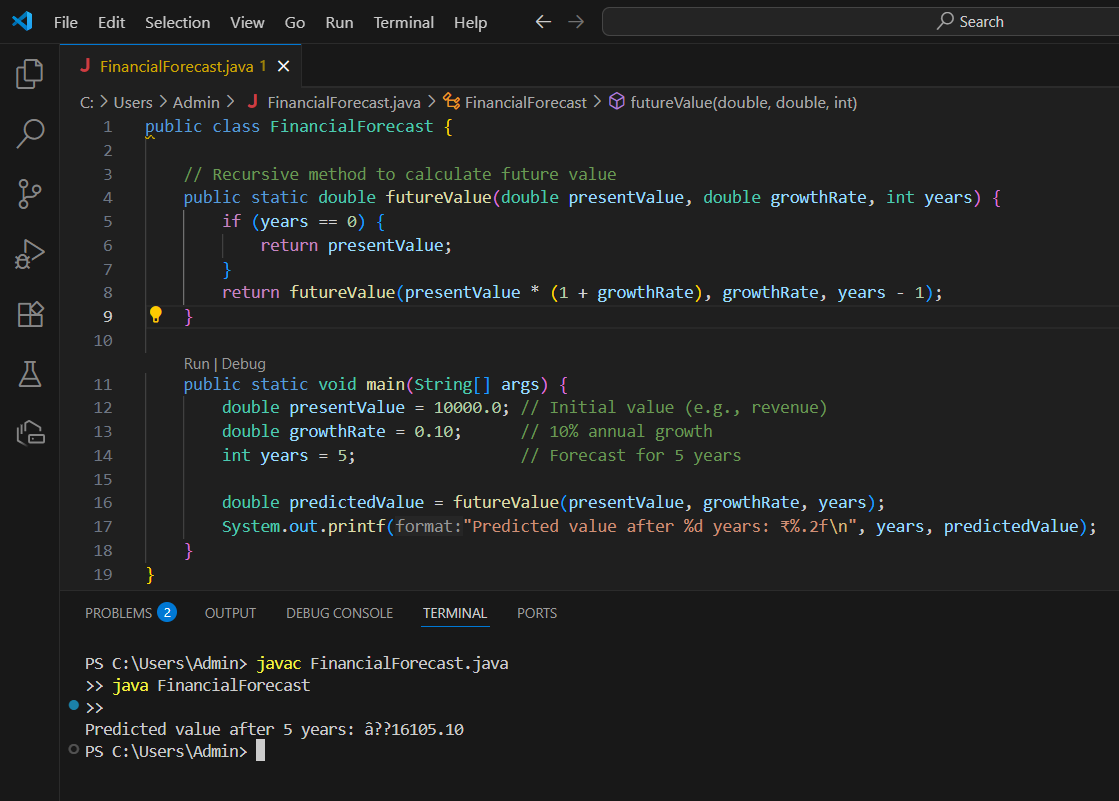
}

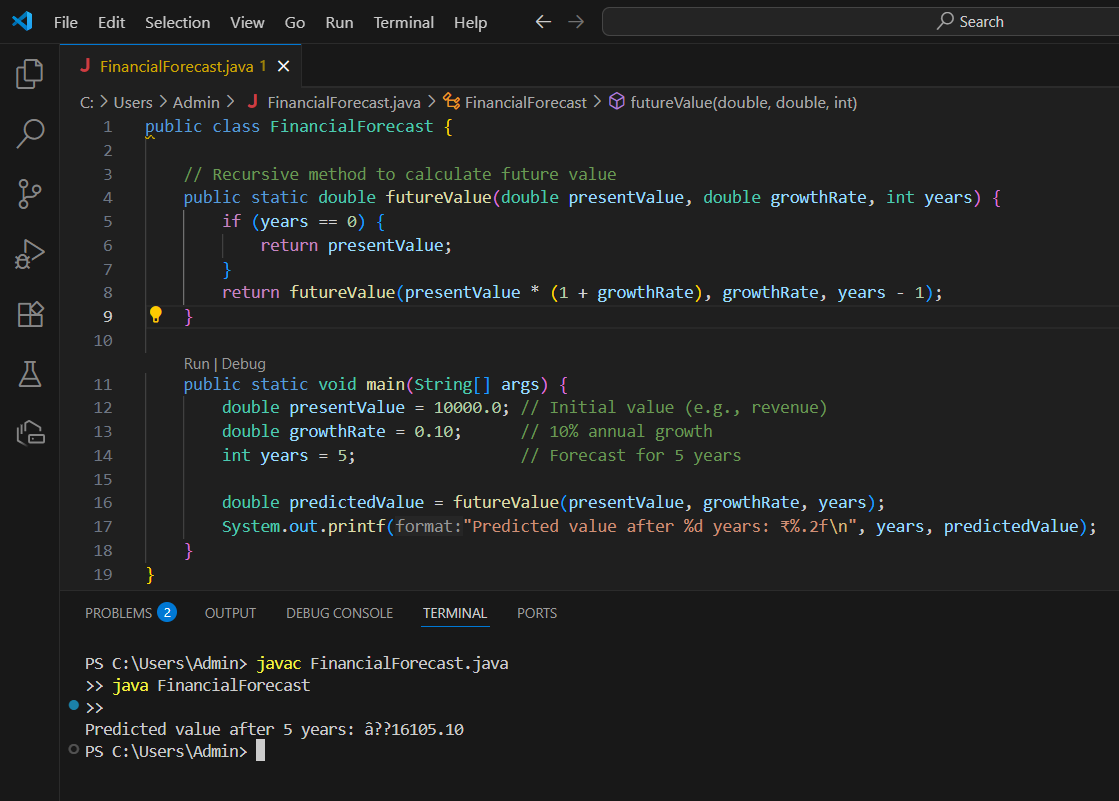
**OUTPUT**

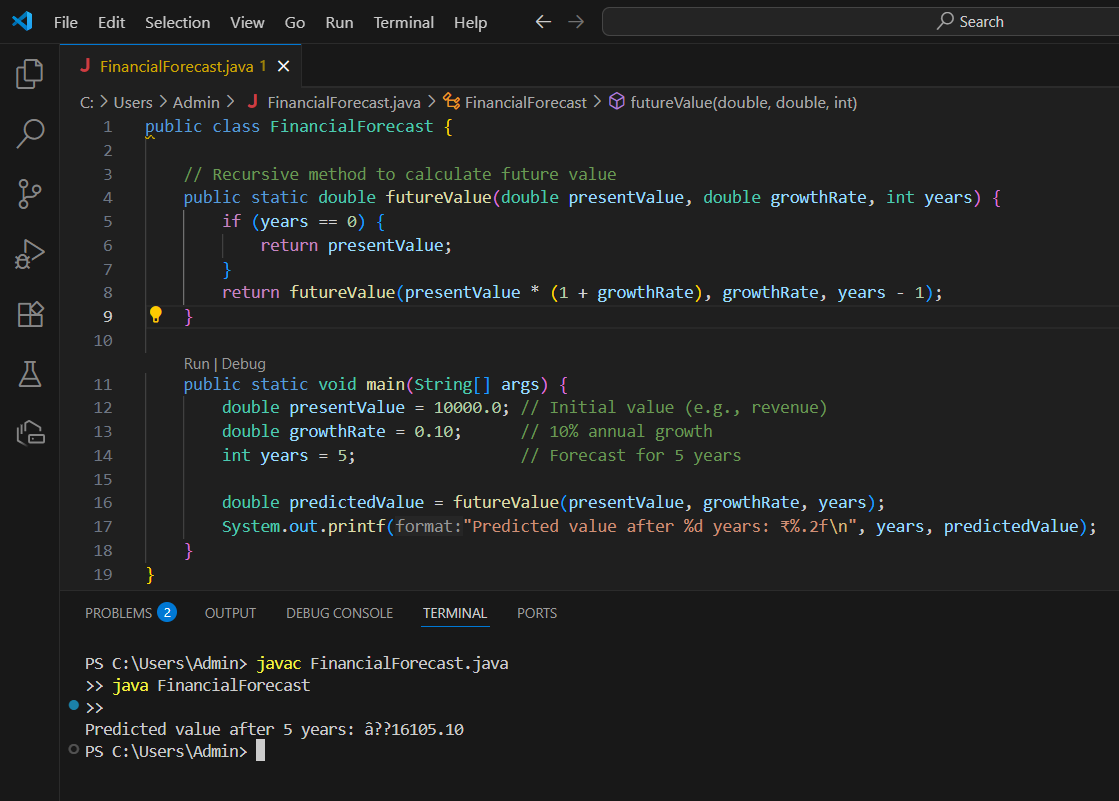
**Analysis**

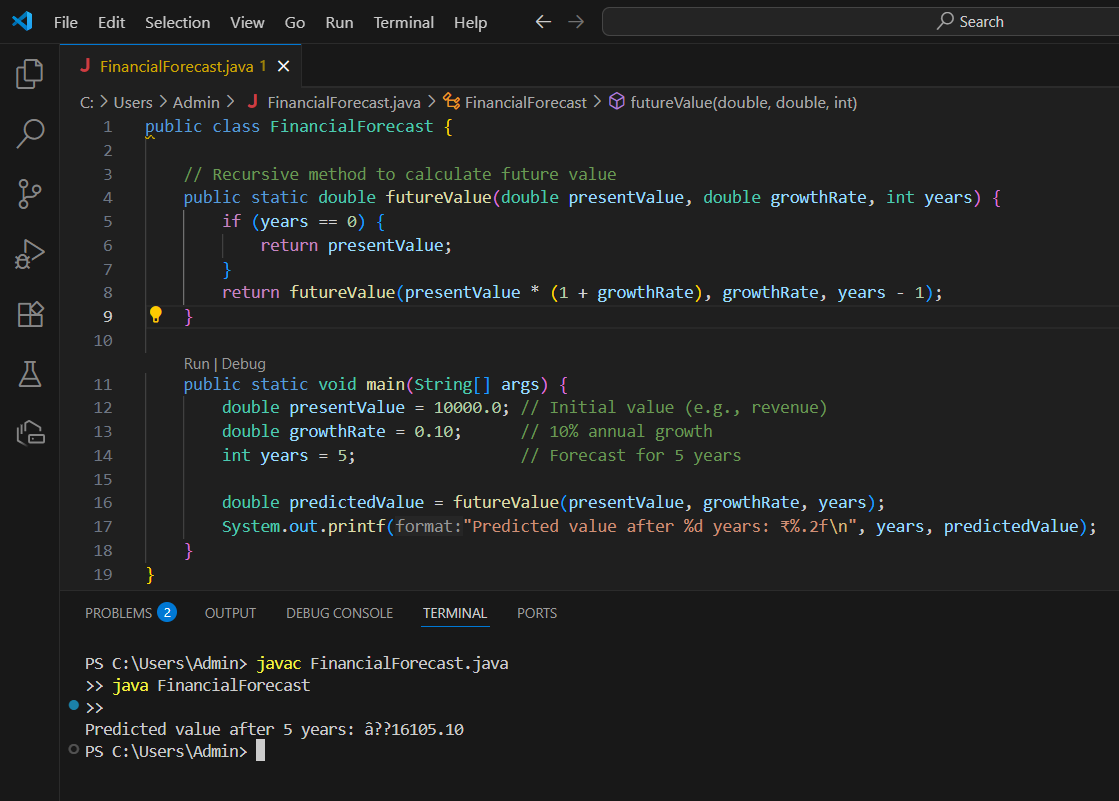
This version uses O(1) space instead of recursive call stack — preferred for **large n**.

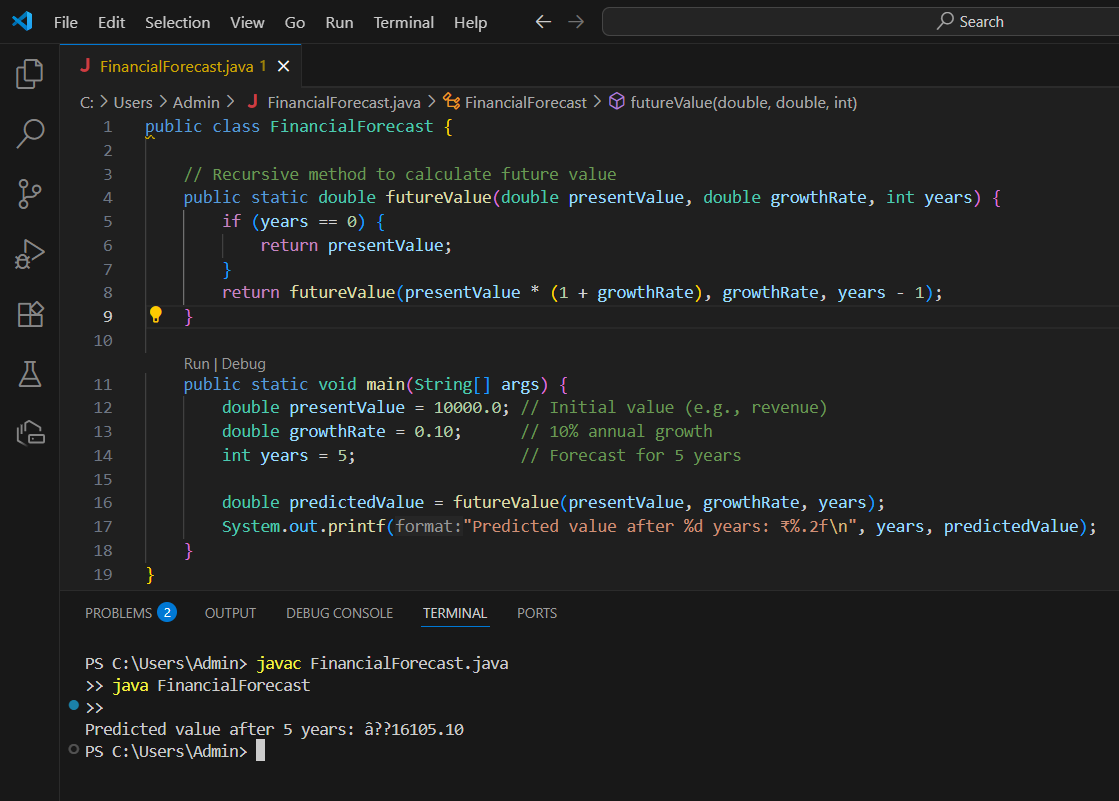
|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Time Complexity** | **Space Complexity** | **Suitable For** |
| Recursion | O(n) | O(n) | Small time spans |
| Iteration | O(n) | O(1) | Large time spans |

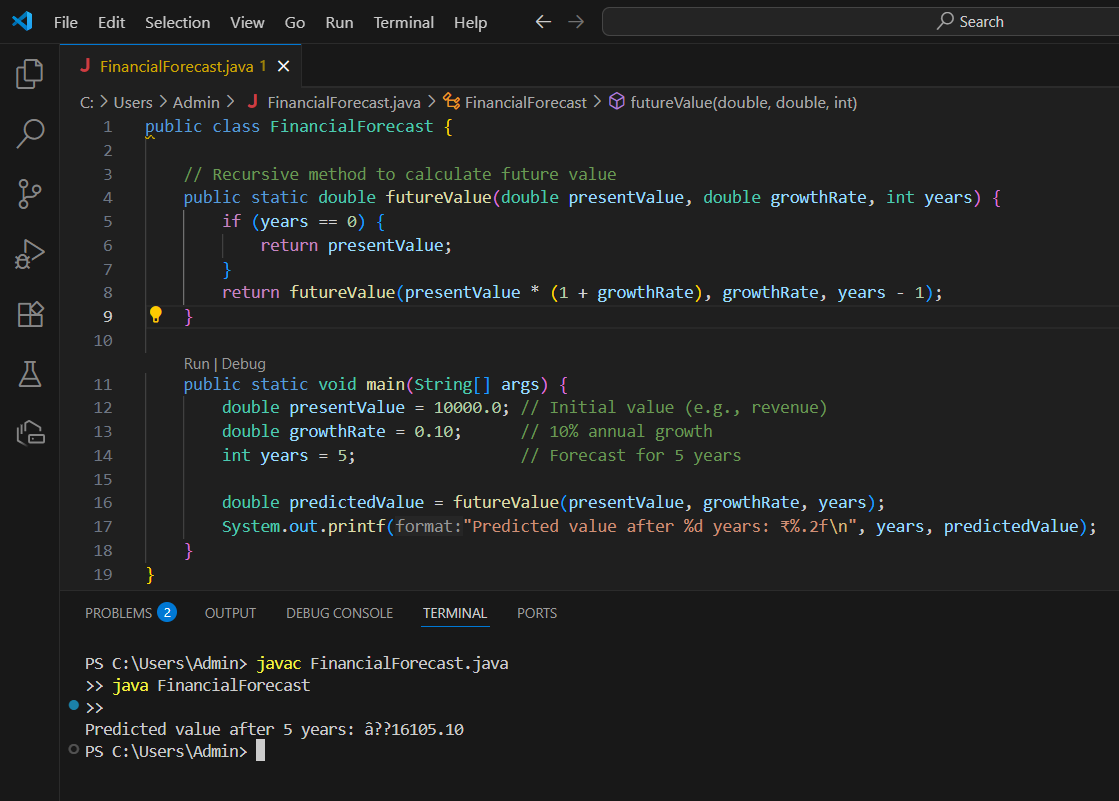


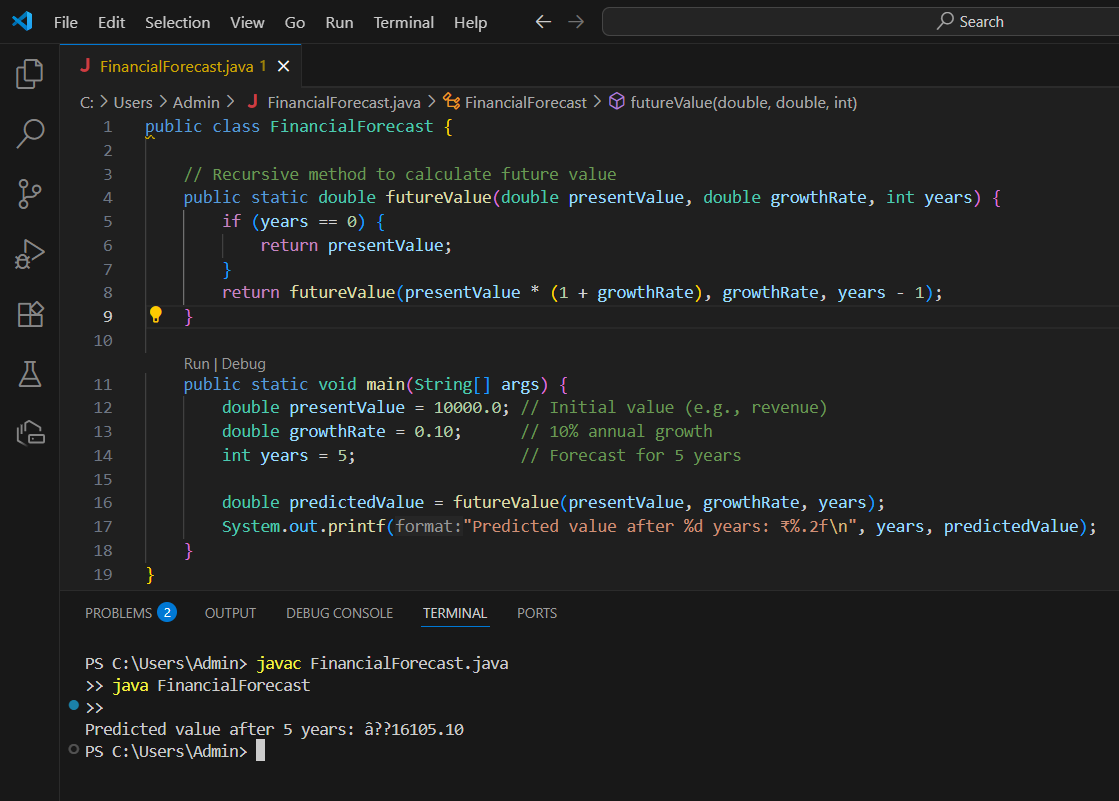
****



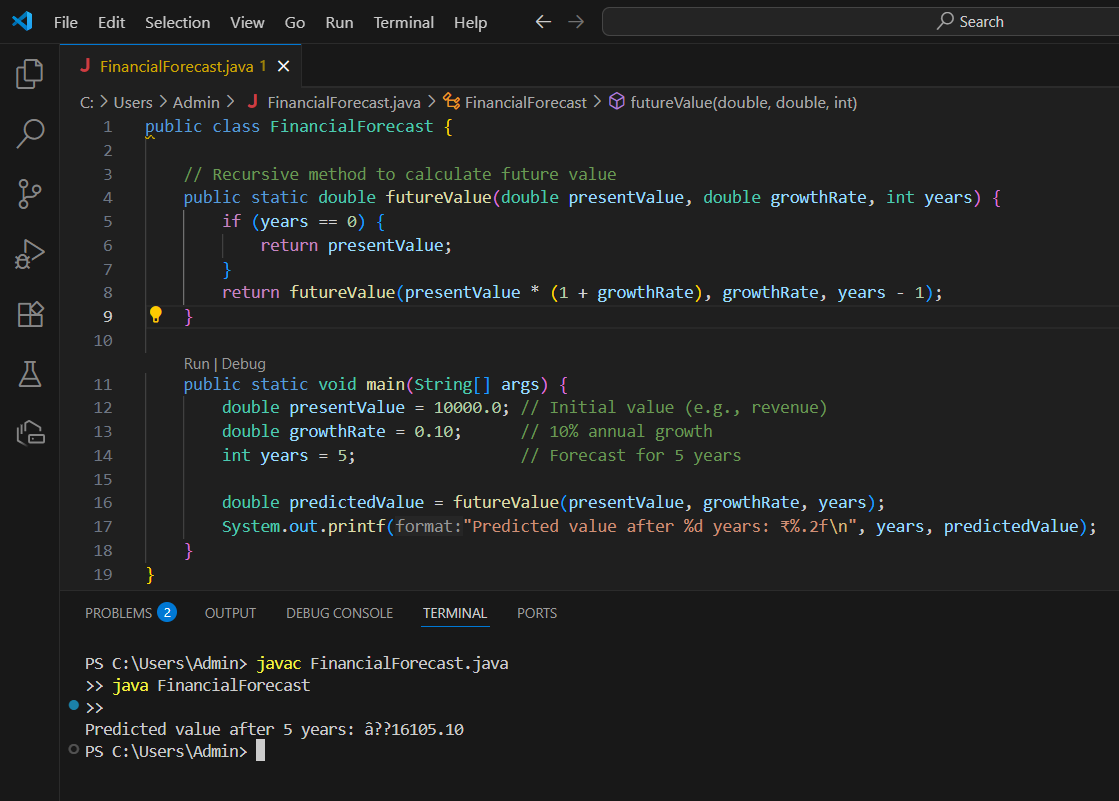


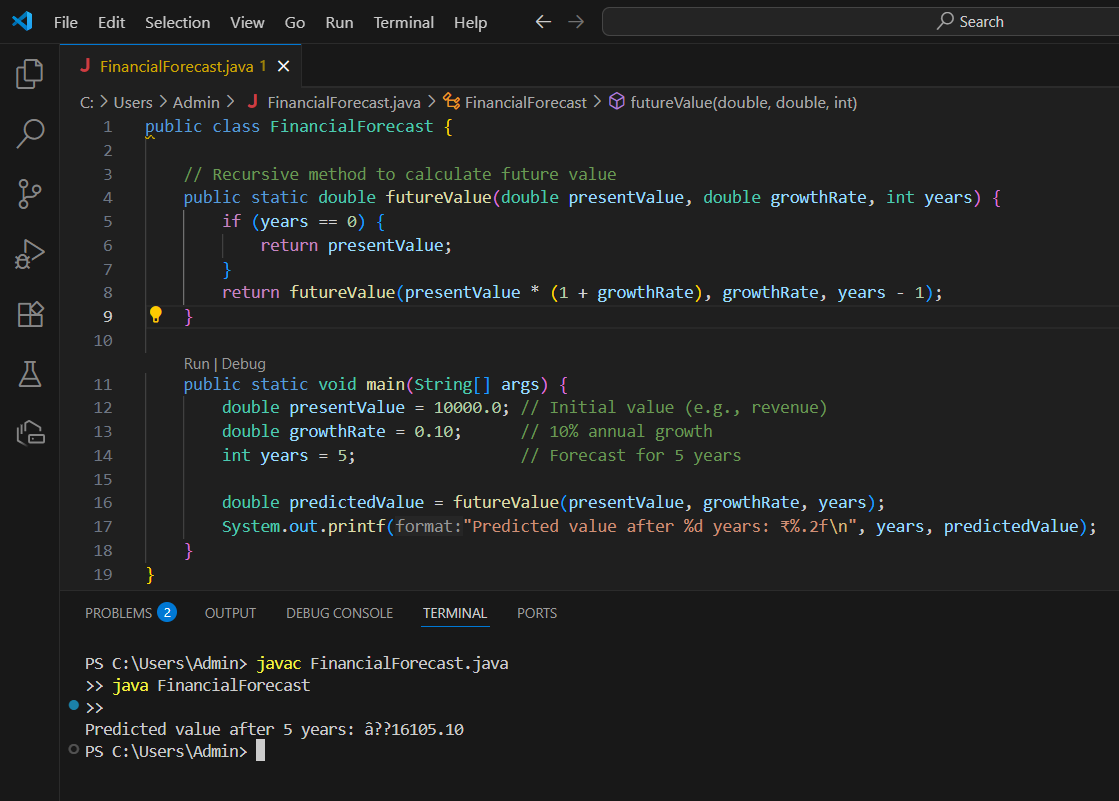


****

**Outpu** ****

**t**



****