**Week-1 Design principles & Patterns**

**Exercise 1: Implementing the Singleton Pattern**

**//Logger.java**

public class Logger {

  private static Logger instance;

  private Logger(){

    System.out.println("Logger object created!");

  }

  public static Logger getInstance(){

    if(instance==null){

        instance= new Logger();

    }

    return instance;

  }

  public void log(String msg){

    System.out.println("LOG : "+ msg);

  }

}

**//Main.java**

public class Main {

    public static void main(String[] args) {

      Logger logger1= Logger.getInstance();

      logger1.log("First Log!");

      Logger logger2= Logger.getInstance();

      logger2.log("Second Log!");

      if(logger1 == logger2){

        System.out.println("Both are same instances");

      }else{

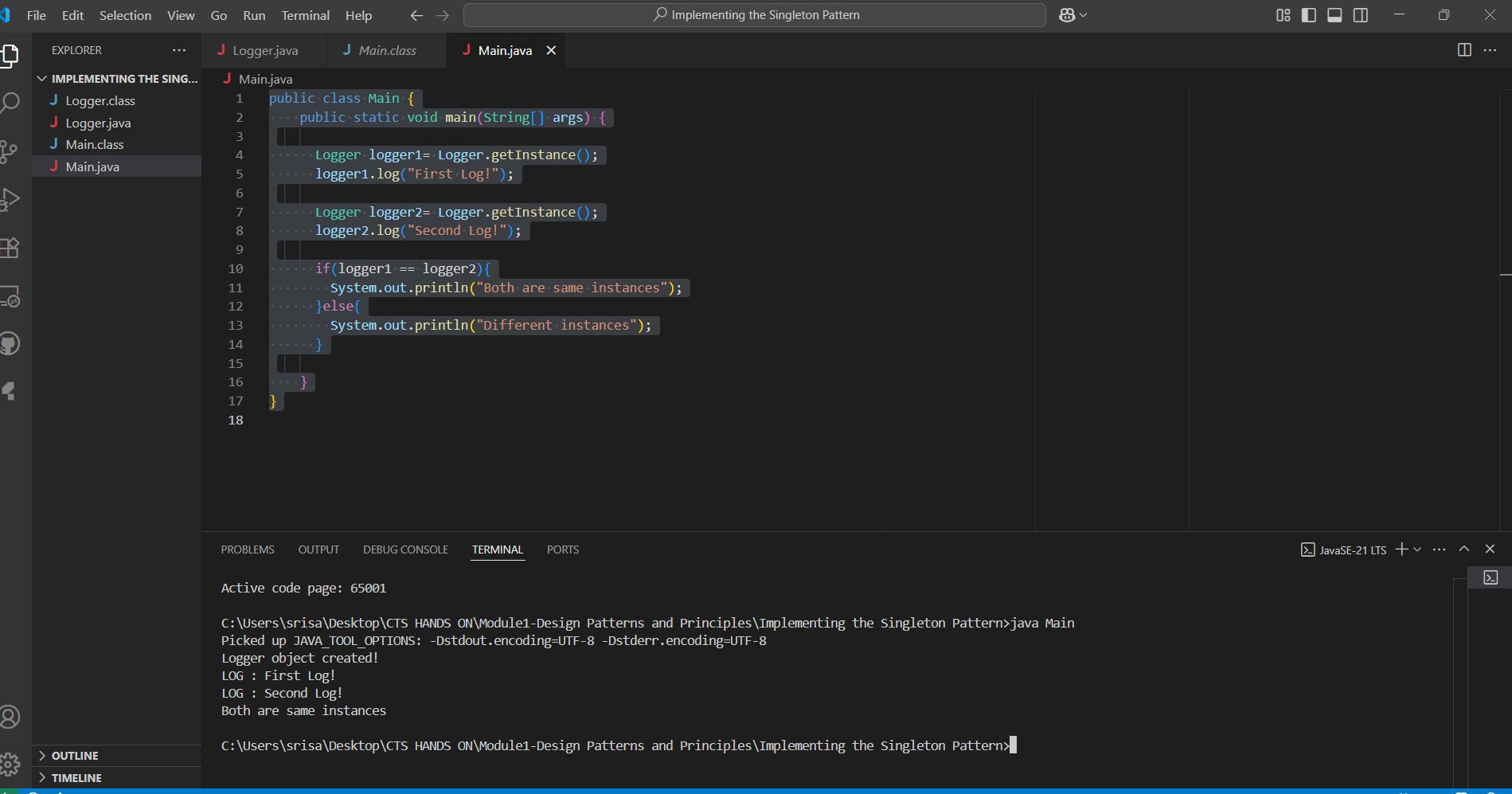
        System.out.println("Different instances");

      }

    }

}

**Output :**



**Exercise 2: Implementing the Factory Method Pattern**

**// Document.java**

public interface Document{

    void open();

}

**//WordDoc.java**

public class WordDoc implements Document{

    public void open(){

        System.out.println("opening a Word Document!");

    }

}

**//PdfDoc.java**

public class PdfDoc implements Document{

    public void open(){

        System.out.println("Opening Pdf Document!");

    }

}

**//ExcelDoc.java**

public class ExcelDoc implements Document{

    public void open(){

        System.out.println("Opening Excel Document!");

    }

}

**//DocumentFactory.java**

public abstract class DocumentFactory{

    public abstract Document createDocument();

}

**//WordDocFactory.java**

public class WordDocFactory extends DocumentFactory{

    public Document createDocument(){

        return new WordDoc();

    }

}

**//PdfDocFactory.java**

public class PdfDocFactory extends DocumentFactory{

    public Document createDocument(){

        return new PdfDoc();

    }

}

**//ExcelDocFactory.java**

public class ExcelDocFactory extends DocumentFactory{

    public Document createDocument(){

        return new ExcelDoc();

    }

}

**//Main.java**

public class Main {

    public static void main(String[] args) {

        DocumentFactory wordFactory = new WordDocFactory();

        Document word = wordFactory.createDocument();

        word.open();

        DocumentFactory pdfFactory = new PdfDocFactory();

        Document pdf = pdfFactory.createDocument();

        pdf.open();

        DocumentFactory excelFactory = new ExcelDocFactory();

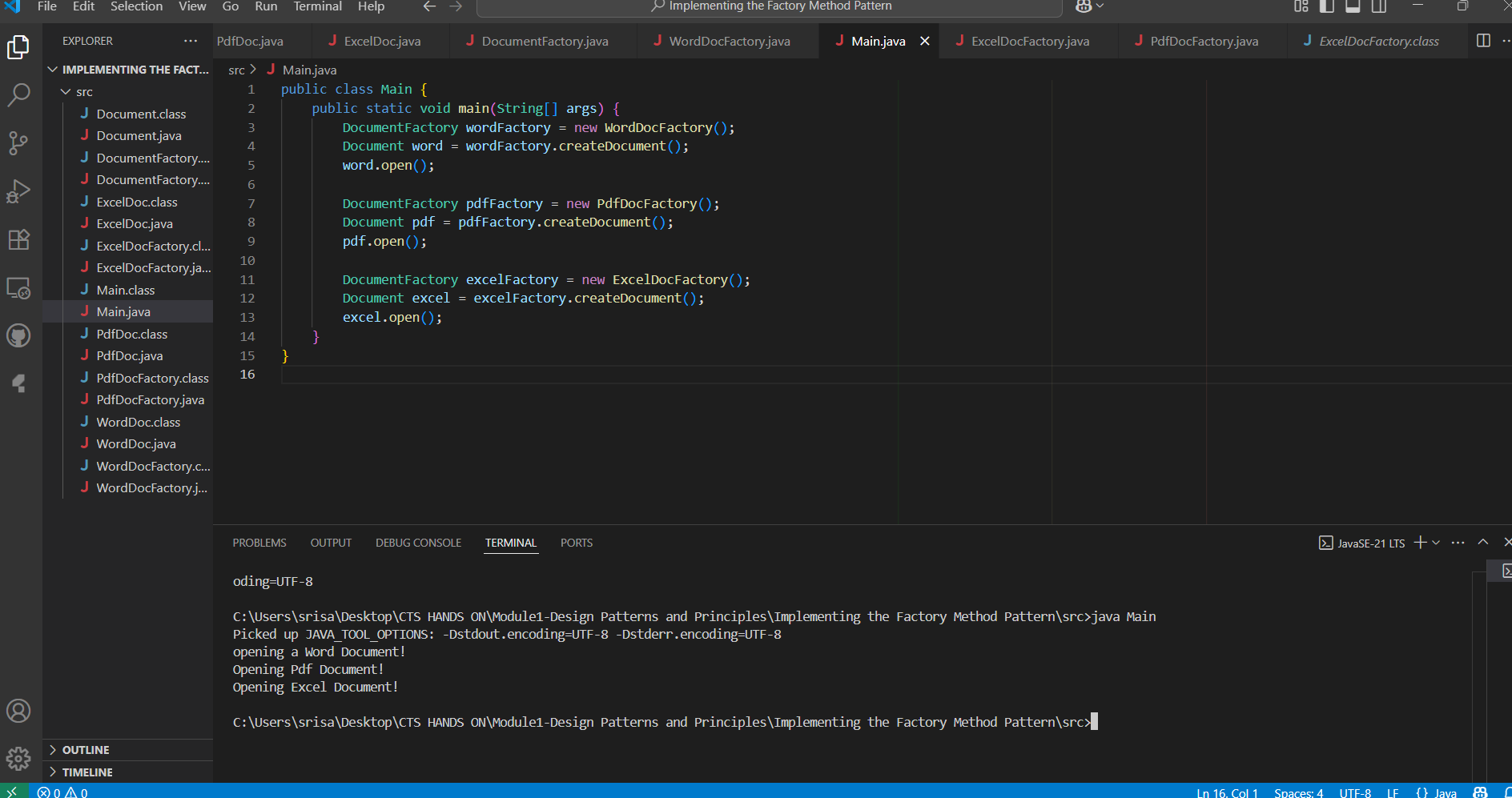
        Document excel = excelFactory.createDocument();

        excel.open();

    }

}

**Output :**



**Week-1 Data structures and Algorithms**

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

**// Product.java**

public 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;

    }

    public String toString() {

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

    }

}

**// BinarySearch.java**

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

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

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

        int left = 0, right = products.length - 1;

        while (left <= right) {

            int mid = (left + right) / 2;

            int cmp = products[mid].productName.compareToIgnoreCase(key);

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

            else if (cmp < 0) left = mid + 1;

            else right = mid - 1;

        }

        return null;

    }

}

**// linearSearch.java**

public class LinearSearch {

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

        for (Product p : products) {

            if (p.productName.equalsIgnoreCase(key)) {

                return p;

            }

        }

        return null;

    }

}

**//Main.java**

public class Main {

    public static void main(String[] args) {

        Product[] products = {

            new Product(1, "Laptop", "Electronics"),

            new Product(2, "Shoes", "Footwear"),

            new Product(3, "Watch", "Accessories"),

            new Product(4, "Book", "Education"),

            new Product(5, "Mobile", "Electronics")

        };

        Product found1 = LinearSearch.linearSearch(products, "Watch");

        System.out.println("Linear Search Result: " + (found1 != null ? found1 : "Not Found"));

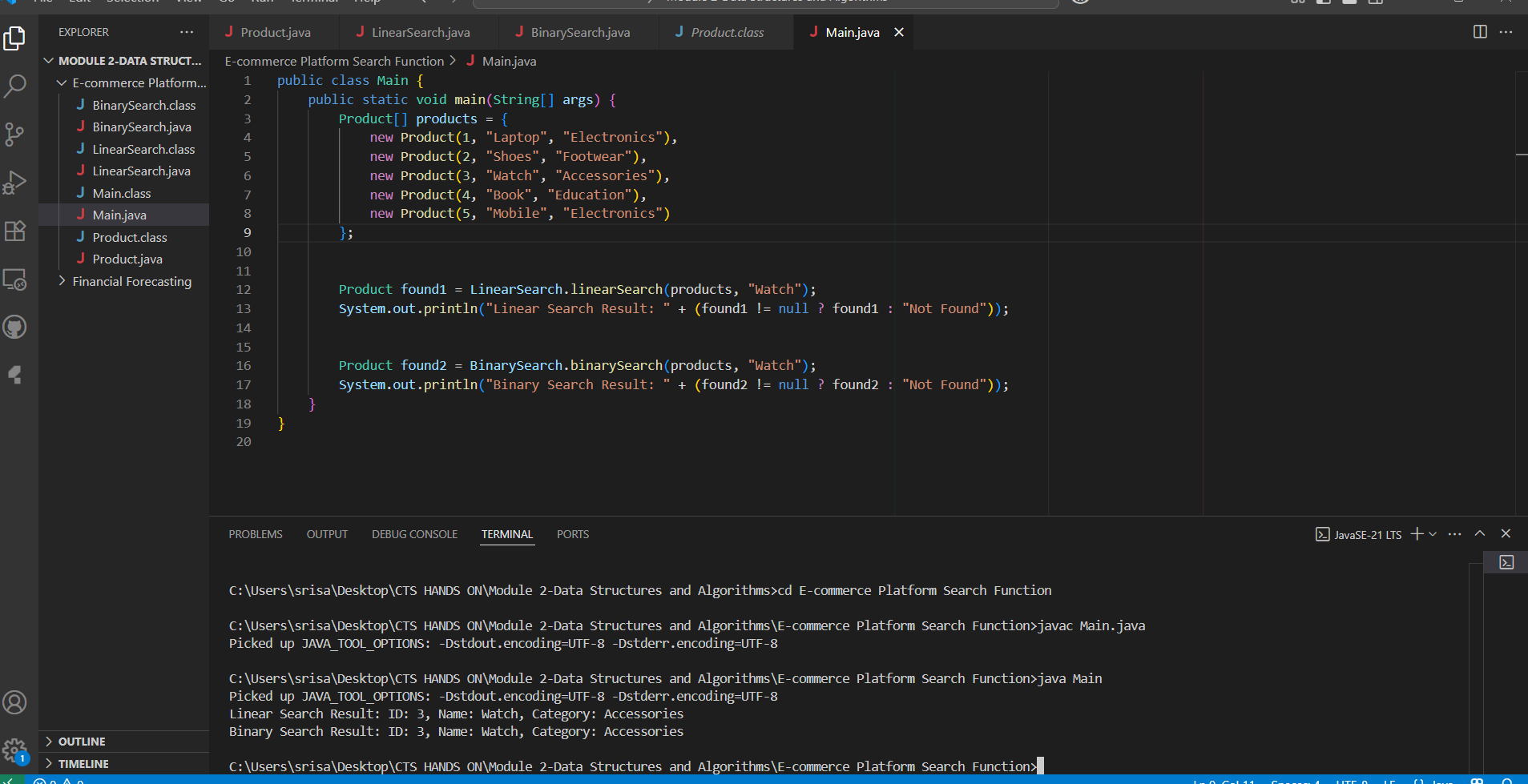
        Product found2 = BinarySearch.binarySearch(products, "Watch");

        System.out.println("Binary Search Result: " + (found2 != null ? found2 : "Not Found"));

    }

}

**Output :**

****

**Exercise 7: Financial Forecasting**

**//** **FinancialForecast.java**

public class FinancialForecast {

    public static double predictFutureValue(double initialAmount, double rate, int years) {

        if (years == 0) {

            return initialAmount;

        }

        return predictFutureValue(initialAmount, rate, years - 1) \* (1 + rate);

    }

    public static void main(String[] args) {

        double initial = 10000;

        double growthRate = 0.10;

        int years = 5;

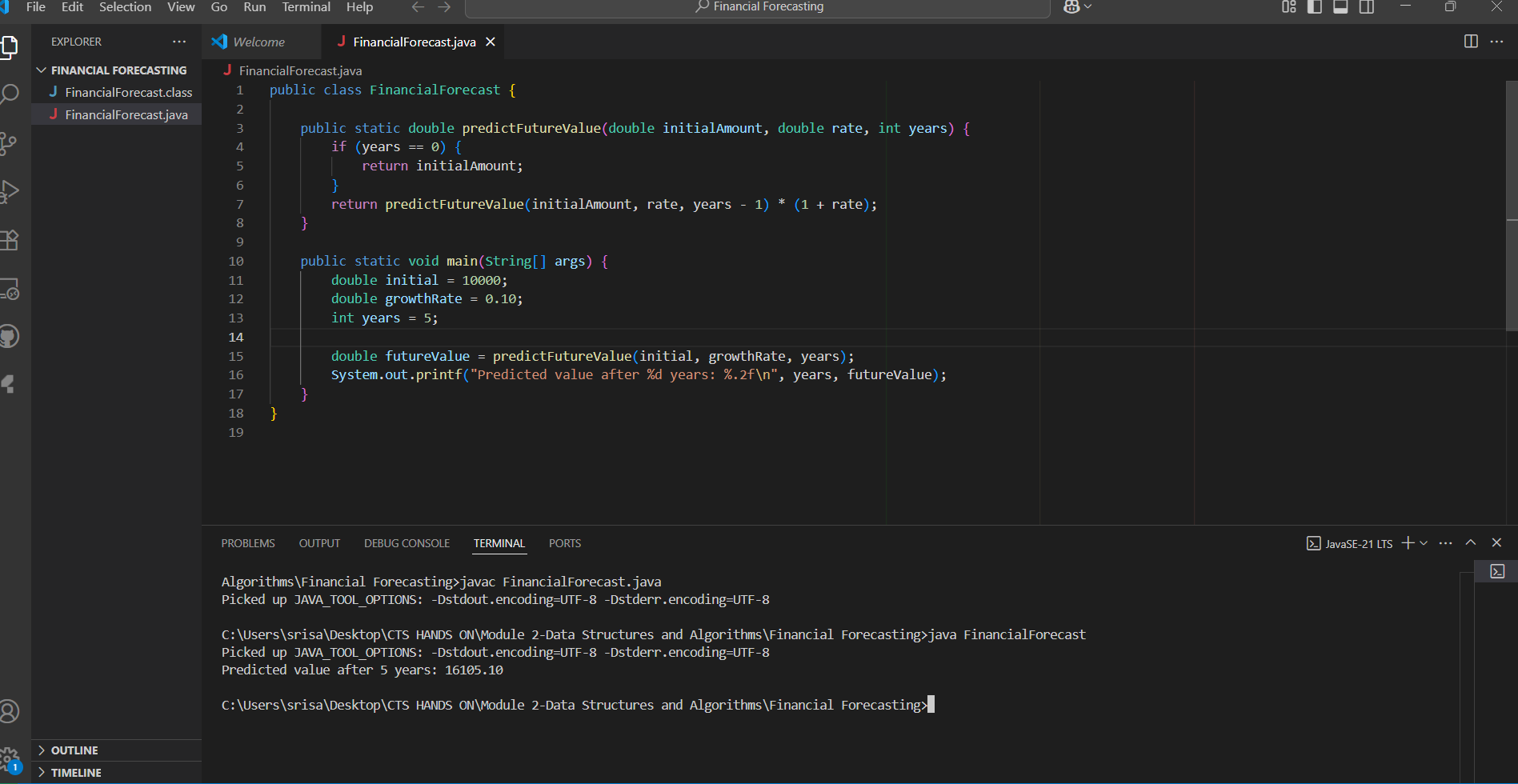
        double futureValue = predictFutureValue(initial, growthRate, years);

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

    }

}

**Output :**

****