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

**Product.java**

public class Product {

    private int productId;

    private String productName;

    private String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    public int getProductId() {

        return productId;

    }

    public String getProductName() {

        return productName;

    }

    public String getCategory() {

        return category;

    }

}

**ProductSearch.java**

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

    // Linear search by productId

    public static Product linearSearch(Product[] products, int targetId) {

        for (Product product : products) {

            if (product.getProductId() == targetId) {

                return product;

            }

        }

        return null;

    }

    // Binary search by productId (array must be sorted by productId)

    public static Product binarySearch(Product[] products, int targetId) {

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

        while (left <= right) {

            int mid = left + (right - left) / 2;

            int midId = products[mid].getProductId();

            if (midId == targetId) {

                return products[mid];

            } else if (midId < targetId) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

            new Product(3, "Laptop", "Electronics"),

            new Product(1, "Shirt", "Clothing"),

            new Product(5, "Book", "Stationery"),

            new Product(2, "Phone", "Electronics"),

            new Product(4, "Shoes", "Footwear")

        };

        // Linear search (unsorted)

        int searchId = 2;

        Product foundLinear = linearSearch(products, searchId);

        System.out.println("Linear Search: " + (foundLinear != null ? foundLinear.getProductName() : "Not found"));

        // Sort array by productId for binary search

        Product[] sortedProducts = Arrays.copyOf(products, products.length);

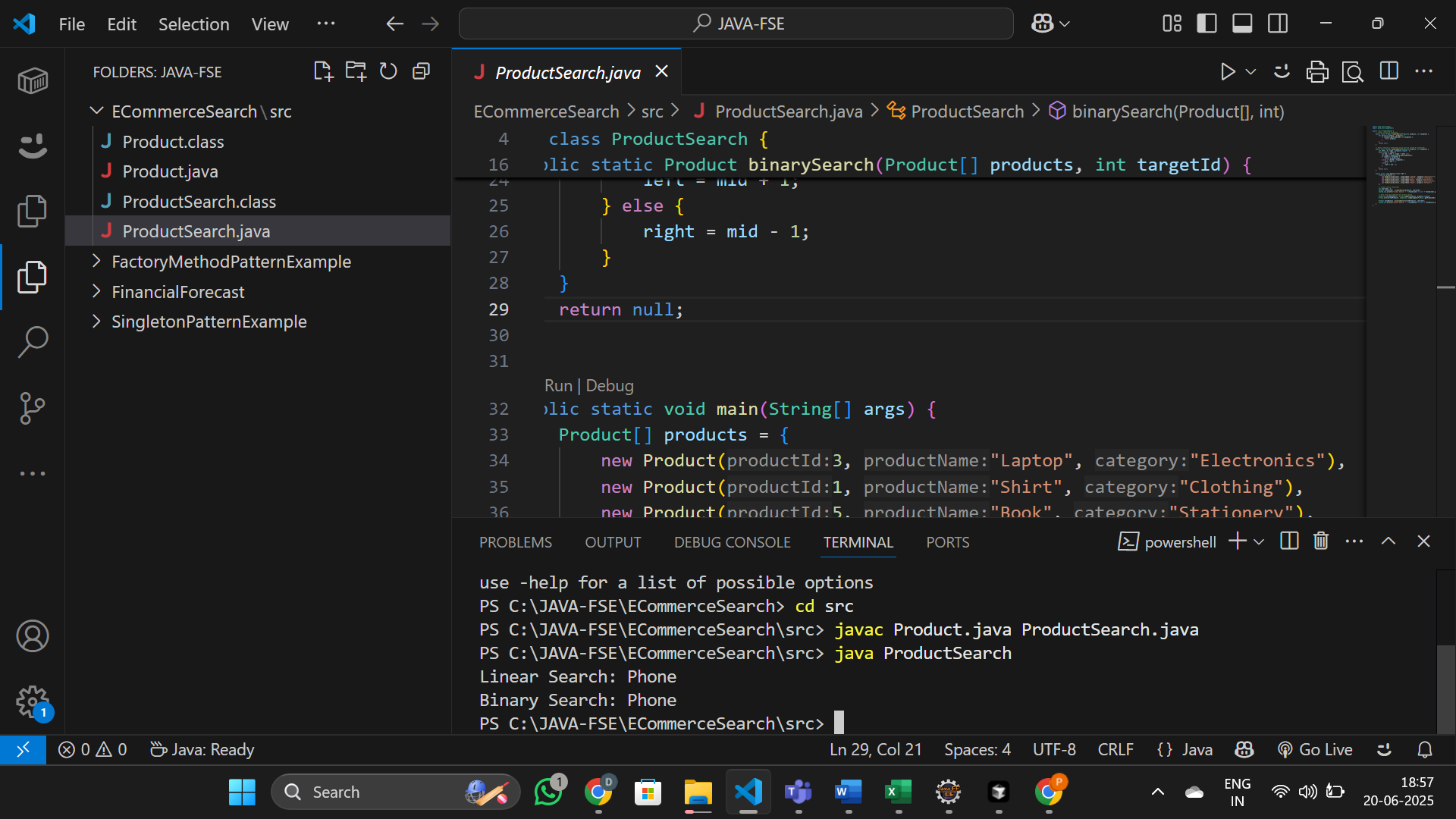
        Arrays.sort(sortedProducts, Comparator.comparingInt(Product::getProductId));

        Product foundBinary = binarySearch(sortedProducts, searchId);

        System.out.println("Binary Search: " + (foundBinary != null ? foundBinary.getProductName() : "Not found"));

    }

}

****

**Exercise 7: Financial Forecasting**

**FinancialForecast.java**

import java.util.HashMap;

import java.util.Map;

public class FinancialForecast {

    // Recursive method to calculate future value

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

        if (years == 0) {

            return presentValue;

        }

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

    }

    // Optimized recursive method with memoization

    public static double futureValueMemo(double presentValue, double growthRate, int years, Map<Integer, Double> memo) {

        if (years == 0) {

            return presentValue;

        }

        if (memo.containsKey(years)) {

            return memo.get(years);

        }

        double result = (1 + growthRate) \* futureValueMemo(presentValue, growthRate, years - 1, memo);

        memo.put(years, result);

        return result;

    }

    public static void main(String[] args) {

        double presentValue = 1000.0;

        double growthRate = 0.05; // 5% annual growth

        int years = 10;

        // Simple recursion

        double futureValue1 = futureValueRecursive(presentValue, growthRate, years);

        System.out.printf("Future Value (Recursive): %.2f\n", futureValue1);

        // Optimized recursion with memoization

        Map<Integer, Double> memo = new HashMap<>();

        double futureValue2 = futureValueMemo(presentValue, growthRate, years, memo);

        System.out.printf("Future Value (Memoized): %.2f\n", futureValue2);

    }

**}**

