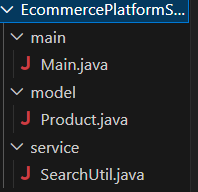
**Data Structures and Algorithms**

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

****

**Product.java**

package EcommercePlatformSearch.model;

public class Product {

    public int productId;

    public String productName;

    public String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    @Override

    public String toString() {

        return productId + " - " + productName + " (" + category + ")";

    }

}

**SearchUtil.java**

package EcommercePlatformSearch.service;

import EcommercePlatformSearch.model.Product;

import java.util.Arrays;

import java.util.Comparator;

public class SearchUtil {

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

        for (Product p : products) {

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

                return p;

            }

        }

        return null;

    }

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

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

        while (low <= high) {

            int mid = (low + high) / 2;

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

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

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

            else high = mid - 1;

        }

        return null;

    }

    public static void sortProductsByName(Product[] products) {

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

    }

}

**Main.java**

package EcommercePlatformSearch.main;

import EcommercePlatformSearch.model.Product;

import EcommercePlatformSearch.service.SearchUtil;

public class Main {

    public static void main(String[] args) {

        Product[] products = {

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

            new Product(102, "Shoes", "Footwear"),

            new Product(103, "Watch", "Accessories"),

            new Product(104, "Mobile", "Electronics")

        };

        // Linear Search

        Product result1 = SearchUtil.linearSearch(products, "Watch");

        System.out.println("Linear Search Result: " + result1);

        // Binary Search (sort first)

        SearchUtil.sortProductsByName(products);

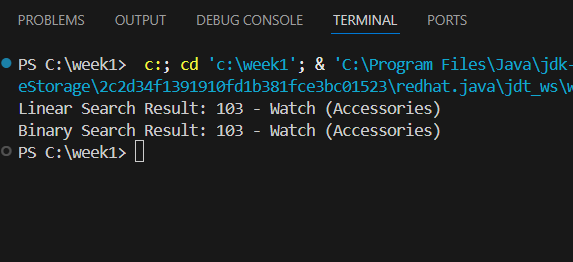
        Product result2 = SearchUtil.binarySearch(products, "Watch");

        System.out.println("Binary Search Result: " + result2);

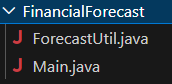
    }

}

**OUTPUT**



**Exercise 7: Financial Forecasting**

****

**ForecastUtil.java**

package FinancialForecast;

public class ForecastUtil {

    // Recursive function to calculate future value

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

        if (years == 0) {

            return presentValue;

        }

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

    }

    // Optimized with Memoization (optional, for large n)

    public static double futureValueMemo(double presentValue, double rate, int years, Double[] memo) {

        if (years == 0) return presentValue;

        if (memo[years] != null) return memo[years];

       memo[years] = (1 + rate) \* futureValueMemo(presentValue, rate, years - 1, memo);

        return memo[years];

    }

}

**Main.java**

package FinancialForecast;

public class Main {

    public static void main(String[] args) {

        double presentValue = 10000;  // Rs. 10,000

        double annualRate = 0.08;     // 8% annual growth

        int years = 5;

        // Simple recursive

        double result = ForecastUtil.futureValue(presentValue, annualRate, years);

        System.out.println("Future Value (recursive): Rs. " + result);

        // Memoized recursive

        Double[] memo = new Double[years + 1];

        double optimizedResult = ForecastUtil.futureValueMemo(presentValue, annualRate, years, memo);

        System.out.println("Future Value (memoized): Rs. " + optimizedResult);

    }

}

OUTPUT

