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

**Code:**

import java.util.Arrays;

import java.util.Comparator;

public class SearchDemo {

    static 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 + "]";

        }

    }

    // Linear Search implementation

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

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

    // Binary Search implementation

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

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

        while (left <= right) {

            int mid = (left + right) / 2;

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

            if (compareResult == 0) {

                return products[mid];

            } else if (compareResult < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        Product[] products = {

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

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

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

                new Product(4, "T-shirt", "Fashion"),

                new Product(5, "Watch", "Accessories")

        };

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

        Product foundLinear = linearSearch(products, "Laptop");

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

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

        System.out.println("\nBinary Search Result:");

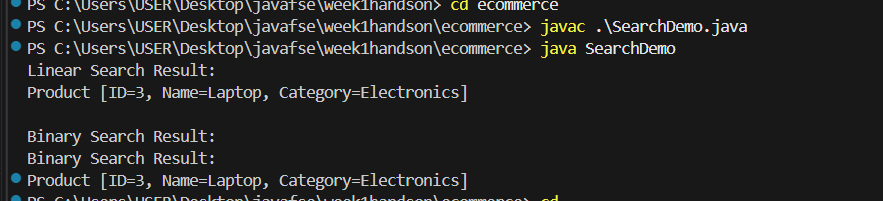
        Product foundBinary = binarySearch(products, "Laptop");

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

    }

}

**Output:**



**Exercise 7: Financial Forecasting**

**Code:**

public class FinancialForecasting {

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

        if (years == 0) {

            return currentValue;

        }

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

    }

    public static double futureValueMemo(double currentValue, double growthRate, int years, double[] memo) {

        if (years == 0)

            return currentValue;

        if (memo[years] != 0)

            return memo[years];

        memo[years] = futureValueMemo(currentValue, growthRate, years - 1, memo) \* (1 + growthRate);

        return memo[years];

    }

    public static void main(String[] args) {

        double currentValue = 1000.0;

        double growthRate = 0.10;

        int years = 5;

        double futureValue = futureValueRecursive(currentValue, growthRate, years);

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

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

        double futureValueOpt = futureValueMemo(currentValue, growthRate, years, memo);

        System.out.printf("Future Value (Optimized with Memoization): %.2f\n", futureValueOpt);

    }

}

**Output:**

