**ECOMMERCE PLATFORM SEARCH FUNCTION**

**CODE:**

**Main.java**

package EcommercePlatformSearchFunction;

public class Main {

    public static void main(String[] args) {

        Product[] products = {

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

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

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

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

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

        };

        String target = "Mobile";

        // Linear Search

        Product foundLinear = ProductSearch.linearSearch(products, target);

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

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

        // Binary Search (requires sorting first)

        ProductSearch.sortByName(products);

        Product foundBinary = ProductSearch.binarySearch(products, target);

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

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

    }

}

**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;

    }

    @Override

    public String toString() {

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

    }

}

**ProductSearch.java**

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

    // Linear Search by Product Name

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

        for (Product product : products) {

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

                return product;

            }

        }

        return null;

    }

    // Binary Search by Product Name (assumes 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 result = products[mid].getProductName().compareToIgnoreCase(targetName);

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

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

            else high = mid - 1;

        }

        return null;

    }

    // Helper method to sort products by name for binary search

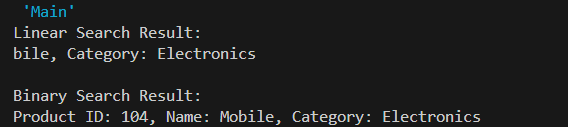
    public static void sortByName(Product[] products) {

        Arrays.sort(products, Comparator.comparing(Product::getProductName, String.CASE\_INSENSITIVE\_ORDER));

    }

}

**OUTPUT:**



**FINANCIAL FORECASTING**

**CODE:**

**Main.java**

public class Main {

    public static void main(String[] args) {

        double presentValue = 10000.0;

        double rate = 0.10;

        int years = 5;

        System.out.println("Using recursive function:");

        double futureValue = ForecastCalculator.calculateFutureValue(presentValue, rate, years);

        System.out.printf("Future Value after %d years: Rs %.2f%n", years, futureValue);

        System.out.println("\nUsing memoized recursive function:");

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

        double futureValueMemo = ForecastCalculator.calculateFutureValueMemo(presentValue, rate, years, memo);

        System.out.printf("Future Value after %d years (memoized): Rs %.2f%n", years, futureValueMemo);

    }

}

**ForecastCalculator.java**

public class ForecastCalculator {

    // Recursive method to calculate future value

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

        if (years == 0) {

            return presentValue;

        }

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

    }

    // Optimized version using memoization (optional)

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

        if (years == 0) return presentValue;

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

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

        return memo[years];

    }

}

**OUTPUT:**

