// 1. Understand Asymptotic Notation:

// Big O notation describes the upper bound of an algorithm's running time, indicating its efficiency as input size grows.

// It helps analyze performance by focusing on the worst-case scenario, ignoring constants and lower-order terms.

// Best case: Minimum time taken (e.g., O(1) if the element is found at the first position).

// Average case: Expected time over all possible inputs (e.g., O(n/2) for linear search, simplified to O(n)).

// Worst case: Maximum time taken (e.g., O(n) for linear search, O(log n) for binary search).

import java.util.Arrays;

import java.util.Scanner;

class Product implements Comparable<Product> {

    private String productId;

    private String productName;

    private String category;

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

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

    public String getProductId() { return productId; }

    public String getProductName() { return productName; }

    public String getCategory() { return category; }

    @Override

    public int compareTo(Product other) {

        return this.productName.compareToIgnoreCase(other.productName);

    }

    @Override

    public String toString() {

        return "Product{ID=" + productId + ", Name=" + productName + ", Category=" + category + "}";

    }

}

public class EcommerceSearch {

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

        for (int i = 0; i < products.length; i++) {

            if (products[i].getProductName().equalsIgnoreCase(searchKey) ||

                products[i].getProductId().equalsIgnoreCase(searchKey) ||

                products[i].getCategory().equalsIgnoreCase(searchKey)) {

                return i;

            }

        }

        return -1;

    }

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

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

        while (left <= right) {

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

            int compare = products[mid].getProductName().compareToIgnoreCase(searchKey);

            if (compare == 0) return mid;

            if (compare < 0) left = mid + 1;

            else right = mid - 1;

        }

        return -1;

    }

    public static void main(String[] args) {

        Product[] unsortedProducts = {

            new Product("P001", "Laptop", "Electronics"),

            new Product("P002", "Mouse", "Electronics"),

            new Product("P003", "Shirt", "Clothing")

        };

        Product[] sortedProducts = unsortedProducts.clone();

        Arrays.sort(sortedProducts);

        Scanner scanner = new Scanner(System.in);

        boolean running = true;

        while (running) {

            System.out.println("\nE-commerce Search Menu:");

            System.out.println("1. Linear Search");

            System.out.println("2. Binary Search");

            System.out.println("3. Exit");

            System.out.print("Enter your choice (1-3): ");

            int choice;

            try {

                choice = Integer.parseInt(scanner.nextLine());

            } catch (NumberFormatException e) {

                System.out.println("Invalid input. Please enter a number between 1 and 3.");

                continue;

            }

            switch (choice) {

                case 1:

                    System.out.print("Enter search term (ID, Name, or Category): ");

                    String linearKey = scanner.nextLine();

                    int linearIndex = linearSearch(unsortedProducts, linearKey);

                    System.out.println("Linear Search Result: " +

                        (linearIndex != -1 ? unsortedProducts[linearIndex] : "Not found"));

                    break;

                case 2:

                    System.out.print("Enter search term (Name only, case-insensitive): ");

                    String binaryKey = scanner.nextLine();

                    int binaryIndex = binarySearch(sortedProducts, binaryKey);

                    System.out.println("Binary Search Result: " +

                        (binaryIndex != -1 ? sortedProducts[binaryIndex] : "Not found"));

                    break;

                case 3:

                    running = false;

                    System.out.println("Exiting E-commerce Search System.");

                    break;

                default:

                    System.out.println("Invalid choice. Please enter a number between 1 and 3.");

            }

        }

        scanner.close();

    }

}

// Analysis (unchanged):

// Time Complexity:

// - Linear Search: O(n) in all cases.

// - Binary Search: O(log n) in all cases.

// Comparison and Suitability remain as previously described.

// 4. Analysis:

// Time Complexity:

// Linear Search: O(n) in all cases (best, average, worst) as it checks each element sequentially.

// Binary Search: O(log n) in best, average, and worst cases, as it halves the search space each step.

// Comparison:

// Linear search is simpler and works on unsorted data, making it suitable for small or frequently changing inventories.

// Binary search requires a sorted array, offering better performance for large, static datasets.

// Suitability:

// For an e-commerce platform with a large, relatively stable product catalog (e.g., millions of items), binary search is more suitable due to its O(log n) efficiency.

// Pre-sorting can be done offline, and updates can trigger re-sorting, balancing performance and practicality.

// Linear search may be used for small subsets or real-time additions before a full sort.

OUTPUT:

A screen shot of a computer program

AI-generated content may be incorrect.