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

**Code:**

import java.util.Arrays;

import java.util.Comparator;

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 "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;

    }

}

public class ProductSearch {

    // Linear Search

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

        for (Product p : products) {

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

                return p;

            }

        }

        return null;

    }

    // Binary Search (requires sorted array)

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

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

        while (left <= right) {

            int mid = (left + right) / 2;

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

            if (comparison == 0) {

                return products[mid];

            } else if (comparison < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        // Sample product list

        Product[] products = {

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

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

            new Product(103, "Mobile", "Electronics"),

            new Product(104, "Watch", "Accessories"),

            new Product(105, "Backpack", "Travel")

        };

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

        String searchTerm1 = "Watch";

        Product result1 = linearSearch(products, searchTerm1);

        if (result1 != null) {

            System.out.println("Product found: " + result1);

        } else {

            System.out.println("Product not found.");

        }

        // Sort products by productName for binary search

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

        System.out.println("\n=== Binary Search ===");

        String searchTerm2 = "Mobile";

        Product result2 = binarySearch(products, searchTerm2);

        if (result2 != null) {

            System.out.println("Product found: " + result2);

        } else {

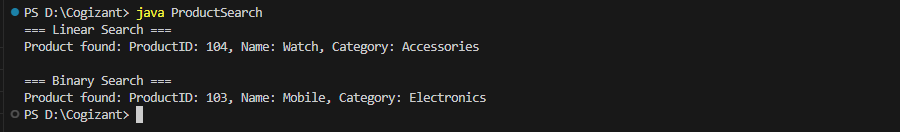
            System.out.println("Product not found.");

        }

    }

}

**Output:**



**Exercise 7: Financial Forecasting**

**Code:**

public class FinancialForecasting {

    // Recursive method to calculate future value

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

        if (years == 0) {

            return principal;

        } else {

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

        }

    }

    // Optimized recursive approach using memoization (optional)

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

        if (years == 0) return principal;

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

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

        return memo[years];

    }

    public static void main(String[] args) {

        double principal = 10000;  // Initial amount

        double rate = 0.1;         // 10% growth rate

        int years = 5;             // Forecasting for 5 years

        // Basic recursion

        double result = futureValue(principal, rate, years);

        System.out.printf("Future value (recursive): %.2f%n", result);

        // Optimized recursion with memoization

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

        double resultMemo = futureValueMemo(principal, rate, years, memo);

        System.out.printf("Future value (memoized): %.2f%n", resultMemo);

    }

}

**Output:**

