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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Product.java**

public 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;

}

public String toString() {

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

}

}

**SearchUtility.java**

public class SearchUtility {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

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

while (left <= right) {

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

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

if (comparison == 0)

return products[mid];

else if (comparison < 0)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

}

**Main.java**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Shoes", "Footwear"),

new Product(102, "Watch", "Accessories"),

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

new Product(104, "Shirt", "Clothing"),

new Product(105, "Phone", "Electronics")

};

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

Product linearResult = SearchUtility.linearSearch(products, "Watch");

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

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

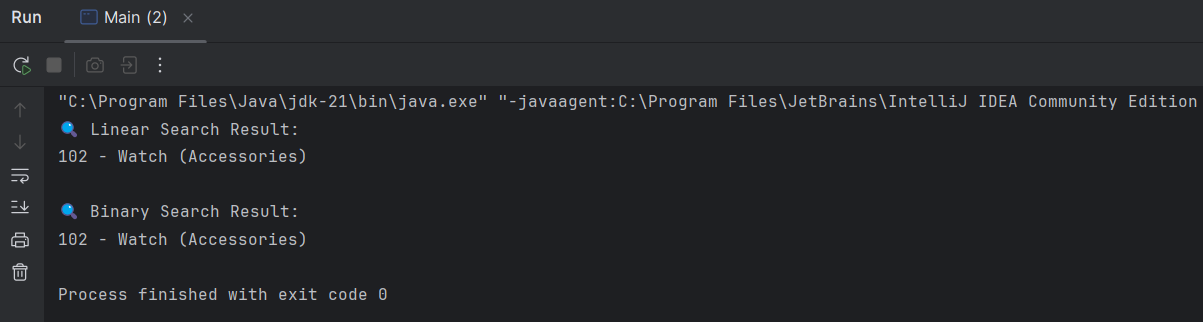
System.out.println("\n🔍 Binary Search Result:");

Product binaryResult = SearchUtility.binarySearch(products, "Watch");

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

}

}



**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Program:**

**Forecast.java**

public class Forecast {

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

if (years == 0) {

return principal;

}

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

}

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

return memo[years];

}

}

**Main.java**

public class Main {

public static void main(String[] args) {

double principal = 10000;

double rate = 0.05;

int years = 10;

System.out.println("Recursive Future Value (No Optimization):");

double result1 = Forecast.futureValue(principal, rate, years);

System.out.printf("Future value after %d years: ₹%.2f%n", years, result1);

System.out.println("\n Optimized Recursive Future Value (Memoized):");

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

double result2 = Forecast.futureValueMemo(principal, rate, years, memo);

System.out.printf("Future value after %d years: ₹%.2f%n", years, result2);

}

}

