**Digital Nurture 4.0 Deep Skilling - Java FSE**  
**WEEK –1 Hands-on Exercises**  
**Module 2 - Data Structures and Algorithms**

**1. 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.

**Solution:**

**Linear Search:**

Linear search checks each item in the list one by one until it finds the target or reaches the end.

**Binary Search:**

Binary search repeatedly divides the sorted list in half to quickly find the target item.

**Code:**

**Product.java**

package EcommerceSearch;

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

}

}

**ProductSearch.java**  
package EcommerceSearch;

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void sortProducts(Product[] products) {

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

}

}

**SearchTest.java**

package EcommerceSearch;

import java.util.Scanner;

public class SearchTest {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Product[] products = {

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

new Product(2, "Laptop", "Electronics"),

new Product(3, "Shoes", "Footwear"),

new Product(4, "Watch", "Accessories"),

new Product(5, "T-shirt", "Clothing")

};

System.out.print("Enter product name to search: ");

String searchName = scanner.nextLine();

Product result1 = ProductSearch.linearSearch(products, searchName);

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

System.out.println(result1 != null ? result1 : "Product not found");

ProductSearch.sortProducts(products);

Product result2 = ProductSearch.binarySearch(products, searchName);

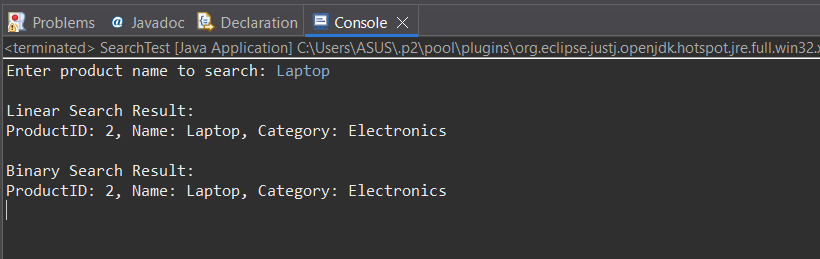
System.out.println("\nBinary Search Result:");

System.out.println(result2 != null ? result2 : "Product not found");

}

}

**Output:**



**Explanation:**

1. Product Class

A blueprint for product objects.

* ProductId
* ProductName
* Category

1. A fixed list of products is stored in an array.

Example: Phone, Laptop, Watch, etc.

1. The user types the name of the product they want to search.

Example: Laptop

1. Linear Search Method

* Goes through each product in the array, one by one.
* If it finds a matching productName, it returns that product.

1. Binary Search Method

* Searches faster using divide and conquer.
* But it works only after sorting the array by product name.
* If it finds the name, it returns the matching product.
* Sorting Before Binary Search Before using binary search, products are sorted by name using Arrays.sort().

1. Output Shows the result of both searches:

* Linear Search Result
* Binary Search Result
* If found, shows the product details.
* If not found, shows “Product not found”.

**2. Exercise 7: Financial Forecasting**

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

**Solution:**

**Recursive Method:**

A recursive method is a function that calls itself to solve a problem in smaller steps.It continues calling itself until it reaches a base condition where it stops.

**Code:**

**FinancialForecast.java**

package finance;

import java.util.\*;

public class FinancialForecast {

public static double predictFutureValue(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

} else {

return predictFutureValue(presentValue, growthRate, years - 1) \* (1 + growthRate);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter Present Value: ");

double presentValue = scanner.nextDouble();

System.out.print("Enter Annual Growth Rate (in %): ");

double ratePercent = scanner.nextDouble();

double growthRate = ratePercent / 100;

System.out.print("Enter Number of Years: ");

int years = scanner.nextInt();

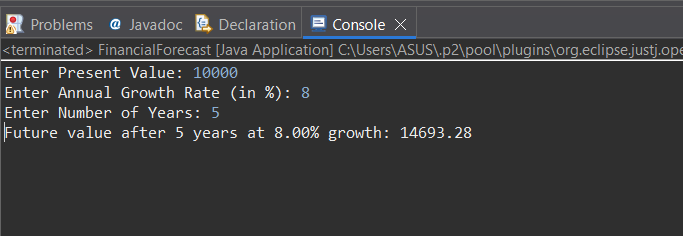
double futureValue = predictFutureValue(presentValue, growthRate, years);

System.out.printf("Future value after %d years at %.2f%% growth: %.2f\n", years, ratePercent, futureValue);

}

}

**Output:**

  
  
**Explanation:**

1. The user is asked to enter:

* Present Value (e.g.10,000)
* Annual Growth Rate (e.g.8%)
* Number of Years (e.g.5)

1. predictFutureValue() This method calculates how much the value will grow year after year.

It calls itself again and again, each time decreasing the number of years by 1.

1. When years become 0, it returns the original present value.

This stops the recursion from continuing forever.

1. Recursive Case

For every year, it multiplies the previous year’s result by (1 + growth rate).

Example:

Year 1: 10,000 × 1.08 = 10,800

Year 2: 10,800 × 1.08 = 11,664

1. After all recursive calls finish, the final future value is returned to the main method.
2. The program prints the future value after the entered number of years.

Example: Future value after 5 years at 8.00% growth: 14693.28