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

**1: 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.

**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;

}

}

**SearchUtility.java**

package EcommerceSearch;

public class SearchUtility {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

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

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

else right = mid - 1;

}

return null;

}

}

**SearchTest.java**

package EcommerceSearch;

import java.util.\*;

public class SearchTest {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Product[] products = {

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

new Product(2, "Shirt", "Clothing"),

new Product(3, "Phone", "Electronics"),

new Product(4, "Shoes", "Footwear"),

new Product(5, "Watch", "Accessories")

};

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

String searchName = scanner.nextLine();

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

if (result1 != null) {

System.out.println("Found (Linear):" + result1.productId +

"," + result1.productName +

"," + result1.category);

} else {

System.out.println("Not Found (Linear)");

}

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

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

if (result2 != null) {

System.out.println("Found (Binary):" + result2.productId +

"," + result2.productName +

"," + result2.category);

} else {

System.out.println("Not Found (Binary)");

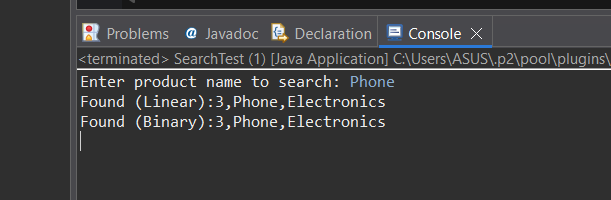
}

scanner.close();

}

}

**Output:**



**Explanation:**

1. Product Class

Defines a product with:

productId ,productName,category

This is the data model used in the array.

1. linearSearch() Method

Searches through each product in the array.

If found, returns the product.

If not, returns null.

1. binarySearch() Method

Performs binary search on a sorted array.

Compares the middle element.

Narrows the search to either half.

Returns the product if found.

1. Takes product name from the user.

Runs both linear and binary search.

Prints product details (ID, Name, Category).

**2: Financial Forecasting**

**Scenario:**

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

**Code:**

**FinancialForecast.java**

package financialForecast;

import java.util.\*;

public class FinancialForecast {

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

if (years == 0) {

return amount;

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter initial amount: ");

double amount = scanner.nextDouble();

System.out.print("Enter annual growth rate (e.g., 0.10 for 10%): ");

double rate = scanner.nextDouble();

System.out.print("Enter number of years: ");

int years = scanner.nextInt();

double futureValue = calculateFutureValue(amount, rate, years);

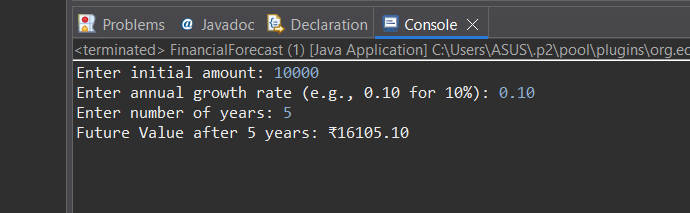
System.out.printf("Future Value after %d years: ₹%.2f\n", years, futureValue);

scanner.close();

}

}

**Output:**



**Explanation:**

1. User Input

The user enters:

Initial amount (e.g., ₹1000),Growth rate (e.g., 0.10 for 10%),Number of years

1. Recursive Method: calculateFutureValue()

This method calculates the future value year by year:

If years == 0, it returns the final amount.

Otherwise, it multiplies the amount by (1 + rate) and calls itself for years - 1.

1. Recursive Calls in Action

For 5 years, the method calls itself 5 times.

1. After all recursive steps, the final future value is printed using: