**Exercise 2: E-Commerce Application**

**Code:**

***Product.java***

package assignments.cts;

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 + ")";

}

}

***SearchAlgorithms.java***

package assignments.cts;

import java.util.Arrays;

import java.util.Comparator;

public class SearchAlgorithms {

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0)

return products[mid];

else if (cmp < 0)

low = mid + 1;

else

high = mid - 1;

}

return null;

}

public static void sortProductsByName(Product[] products) {

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

}

}

***Main.java***

package assignments.cts;

public class App

{

public static void main( String[] args )

{

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Chair", "Furniture"),

new Product(103, "Shoes", "Footwear"),

new Product(104, "Book", "Stationery"),

};

Product found1 = SearchAlgorithms.linearSearch(products, "Chair");

System.out.println("Linear Search Result: " + (found1 != null ? found1 : "Not Found"));

SearchAlgorithms.sortProductsByName(products);

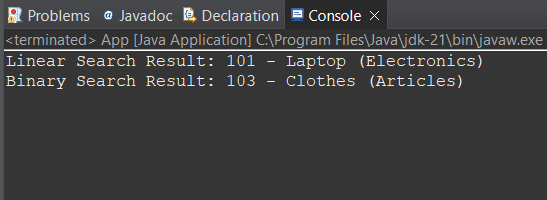
Product found2 = SearchAlgorithms.binarySearch(products, "Chair");

System.out.println("Binary Search Result: " + (found2 != null ? found2 : "Not Found"));

} }

}

**Output:**



**Analysing Time Complexity for Search Algorithms:**

**Big O Notation:**

* Big O notation describes the **upper bound** of an algorithm's running time as input size increases.
* It helps evaluate **scalability** and **performance** of algorithms without running them.

**Searching Algorithms - Big O complexities:**

* **O(1)** – Constant time
* **O(log n)** – Logarithmic time (e.g., binary search)
* **O(n)** – Linear time (e.g., linear search)
* **O(n log n)** – Log-linear (e.g., merge sort)
* **O(n²)** – Quadratic (e.g., bubble sort)

|  |  |  |  |
| --- | --- | --- | --- |
| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| Linear Search | O(1) (first element) | O(n/2) ≈ O(n) | O(n) |
| Binary Search | O(1) (middle element) | O(log n) | O(log n) |