Cognizant - DN 4.0 Deep Skilling Stage

Algorithms Data Structures

Shamyukthaa P

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

package com.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;

}

*@Override*

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

package com.cts;

public class SearchUtils {

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

for (Product product : products) {

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

return product;

}

}

return null;

}

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

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (compare == 0) {

return products[mid];

} else if (compare < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

}

package com.cts;

import java.util.Arrays;

import java.util.Comparator;

public class ECommerceSearchTest {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shampoo", "Personal Care"),

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

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

new Product(105, "Shoes", "Fashion")

};

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

Product found1 = SearchUtils.*linearSearch*(products, "Book");

System.out.println(found1 != null ? found1 : "Product not found");

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

System.out.println("Binary Search:");

Product found2 = SearchUtils.*binarySearch*(products, "Book");

System.out.println(found2 != null ? found2 : "Product not found");

}

}

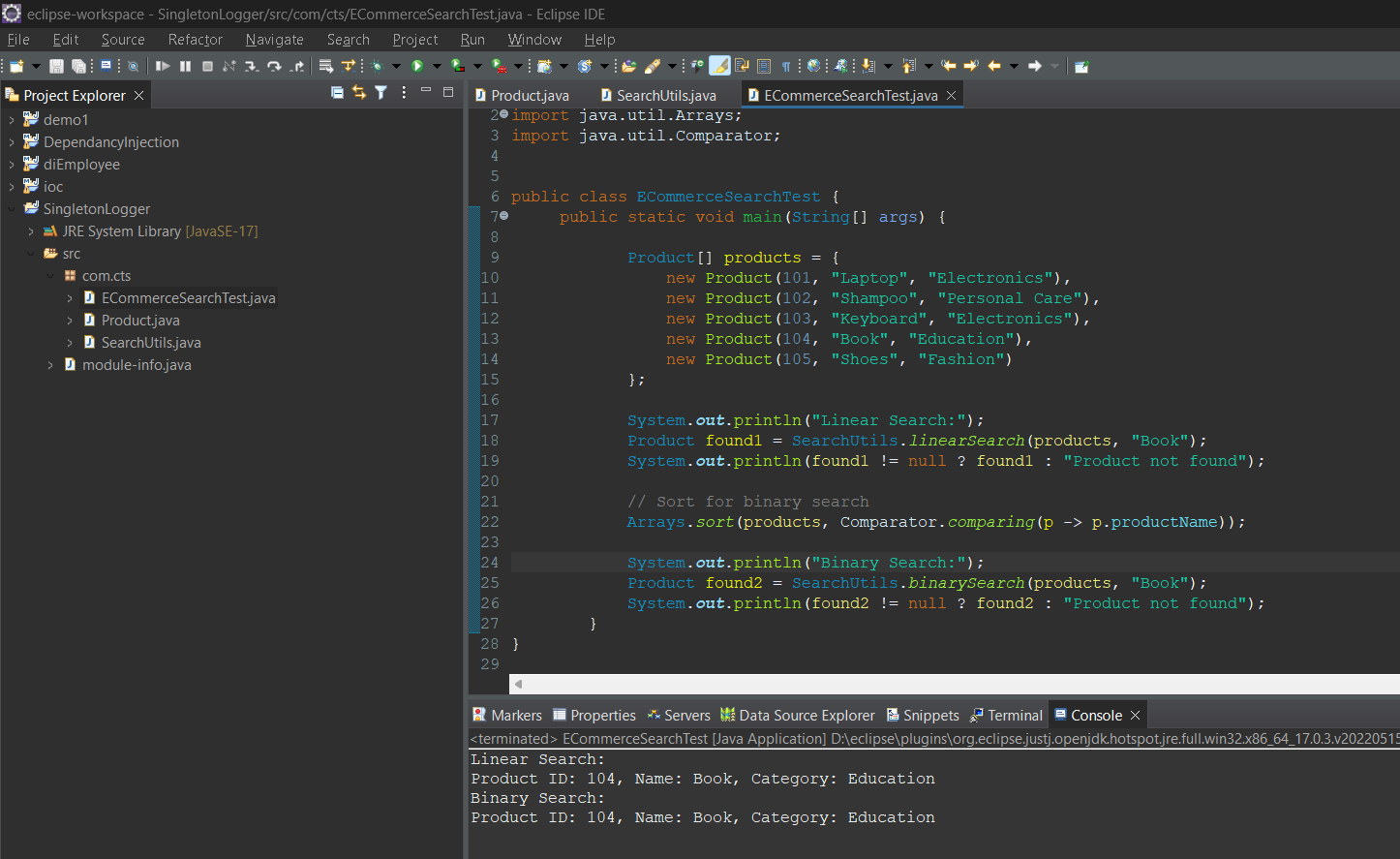
| **Factor** | **Linear Search** | **Binary Search** |
| --- | --- | --- |
| **Sorted Array** | Not needed | Required |
| **Time Complexity** | O(n) | O(log n) |
| **Performance** | Slower on large datasets | Much faster with large datasets |
| **Flexibility** | More flexible, no sorting | Needs maintenance of order |
| **Use Case** | Small or unsorted lists | Large, sorted, searchable lists |

| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| **Linear Search** | O(1) | O(n) | O(n) |
| **Binary Search** | O(1) | O(log n) | O(log n) |

**Use Binary Search** if:

* Your product list is **large**
* And you keep it **sorted by name or ID**

Otherwise, for small lists or no sorting, use **Linear Search.**

****

**Exercise 7: Financial Forecasting**

package com.cts;

public class FinacialForecast {

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

if (years == 0) {

return amount;

}

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

}

public static void main(String[] args) {

double presentValue = 10000;

double annualGrowthRate = 0.10;

int forecastYears = 5;

double futureValue = forecast(presentValue, annualGrowthRate, forecastYears);

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

}

}

Recursion is when a method calls itself to solve smaller parts of a problem.

Every recursive method must have.

A basecase (stopping condition)

A recursivecase (where it calls itself)

**Recursive Future Value Method:** FV(P, r, n) = FV(P × (1 + r), r, n - 1)

**Time Complexity:**

* Recursive Depth: n (number of years)
* Each recursive call does a simple multiplication.
* Time Complexity: O(n)
* Space Complexity: O(n) (because of the call stack)

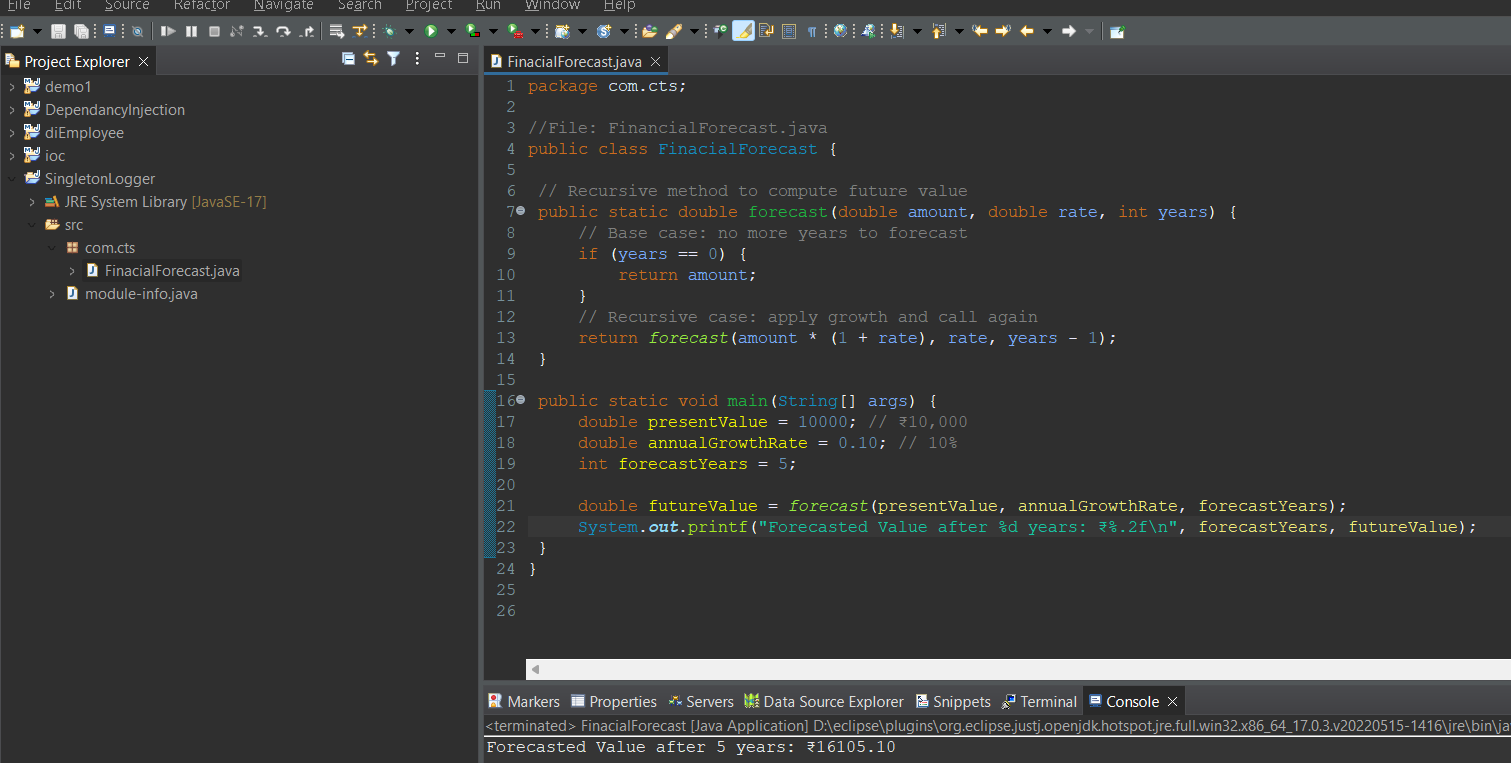
If number of years is large recursion will cause stack overflow and slow performance. We can use optimized iterative version.

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

for (int i = 0; i < years; i++) {

amount \*= (1 + rate)

}return amount;

}****