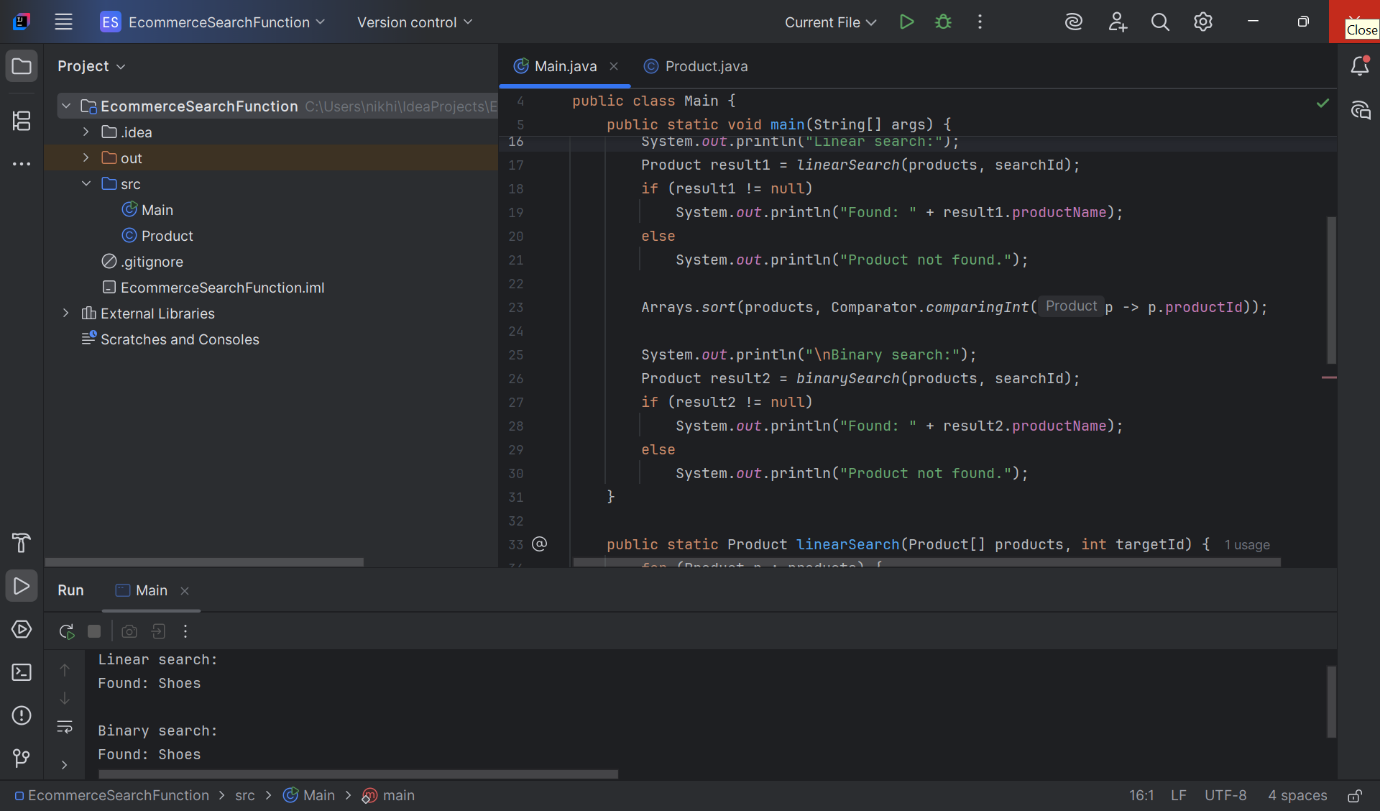
**Cognizant Java FSE Program**

**Week 1 : Data structures and Algorithms**

1. **E-commerce Platform Search Function**



Code :

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;

}

}

* import java.util.Arrays;

import java.util.Comparator;

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(102, "Phone", "Electronics"),

new Product(205, "Shirt", "Apparel"),

new Product(315, "Laptop", "Electronics"),

new Product(123, "Shoes", "Footwear"),

new Product(412, "Book", "Stationery")

};

int searchId = 123;

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

Product result1 = linearSearch(products, searchId);

if (result1 != null)

System.out.println("Found: " + result1.productName);

else

System.out.println("Product not found.");

Arrays.sort(products, Comparator.comparingInt(p -> p.productId));

System.out.println("\n Binary search:");

Product result2 = binarySearch(products, searchId);

if (result2 != null)

System.out.println("Found: " + result2.productName);

else

System.out.println("Product not found.");

}

public static Product linearSearch(Product[] products, int targetId) {

for (Product p : products) {

if (p.productId == targetId)

return p;

}

return null;

}

public static Product binarySearch(Product[] products, int targetId) {

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

while (left <= right) {

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

if (products[mid].productId == targetId)

return products[mid];

else if (products[mid].productId < targetId)

left = mid + 1;

else

right = mid - 1;

}

return null;

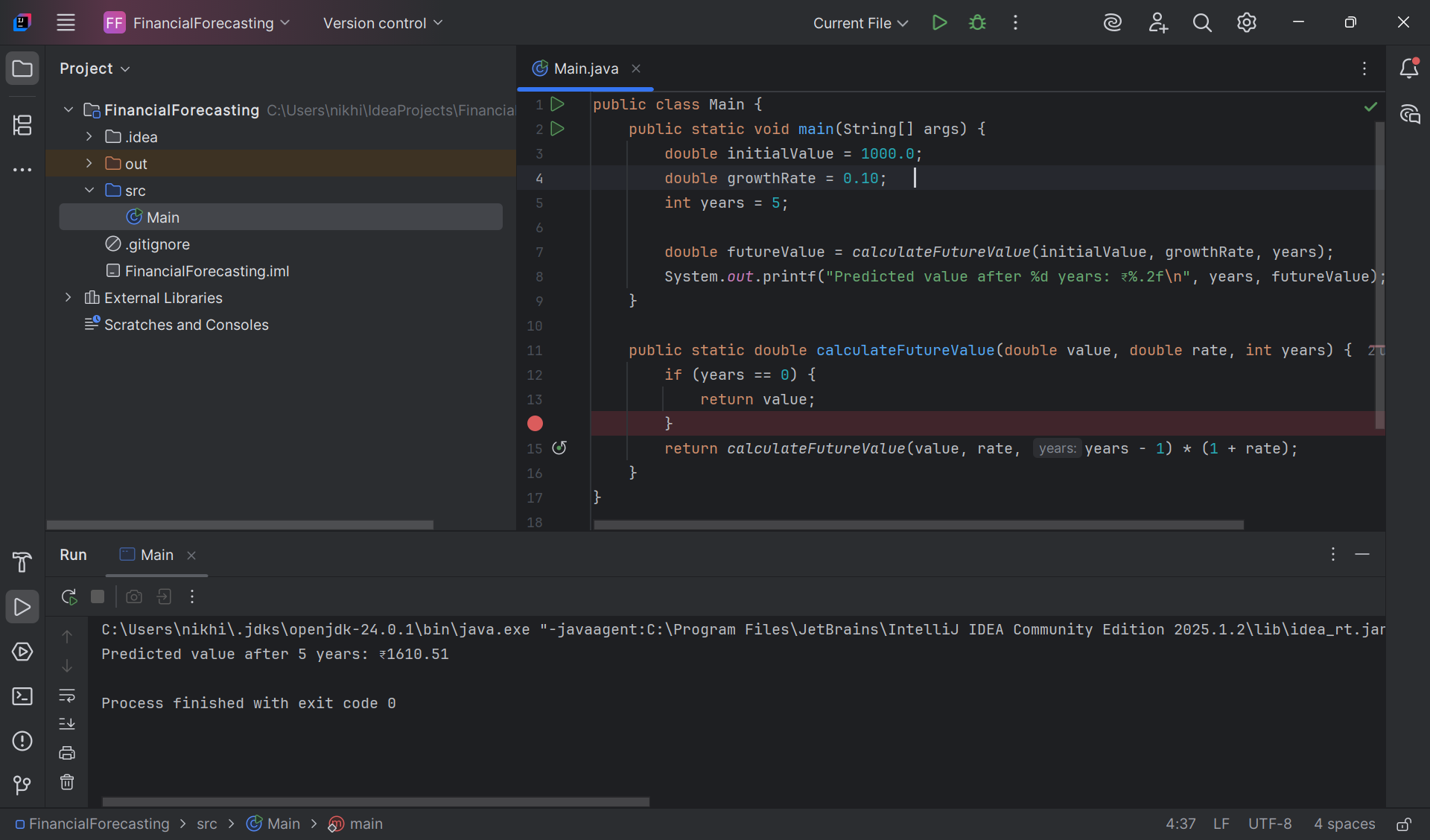
}

}

Analysis:

Linear Search is easier but slower O(n). Binary Search is faster O(log n) and better for large sorted lists. For small data or unsorted lists, linear search is fine. But for real-world e-commerce platforms that deal with a large, sorted product list, binary search is the better and more scalable solution because it is faster and more efficient, so it’s the right choice for an e-commerce platform.

1. **Financial Forecasting**



Code:

public class Main {  
 public static void main(String[] args) {  
 double initialValue = 1000.0;  
 double growthRate = 0.10;   
 int years = 5;  
  
 double futureValue = calculateFutureValue(initialValue, growthRate, years);  
 System.out.printf("Predicted value after %d years: ₹%.2f\n", years, futureValue);  
 }  
  
 public static double calculateFutureValue(double value, double rate, int years) {  
 if (years == 0) {  
 return value;  
 }  
 return calculateFutureValue(value, rate, years - 1) \* (1 + rate);  
 }  
}

Time Complexity

* Each call reduces years by 1 → T(n) = T(n - 1) + O(1)
* So, total complexity is: O(n)

Optimization Tips

Recursion is clean but can be inefficient. We can optimize it:

1.Convert to Iteration

public static double calculateFutureValueIterative(double value, double rate, int years) {

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

value \*= (1 + rate);

}

return value;

}

2.Use Memoization

* Store intermediate results in an array to avoid recalculating.
* Not needed for simple linear recursion like this, but good for Fibonacci-like problems.

Final Summary

Recursion helps simplify problems like financial forecasting. In this case, it repeatedly multiplies the value by the growth rate each year. While it's clean and easy to write, recursion can be replaced with iteration to improve performance for large input sizes.