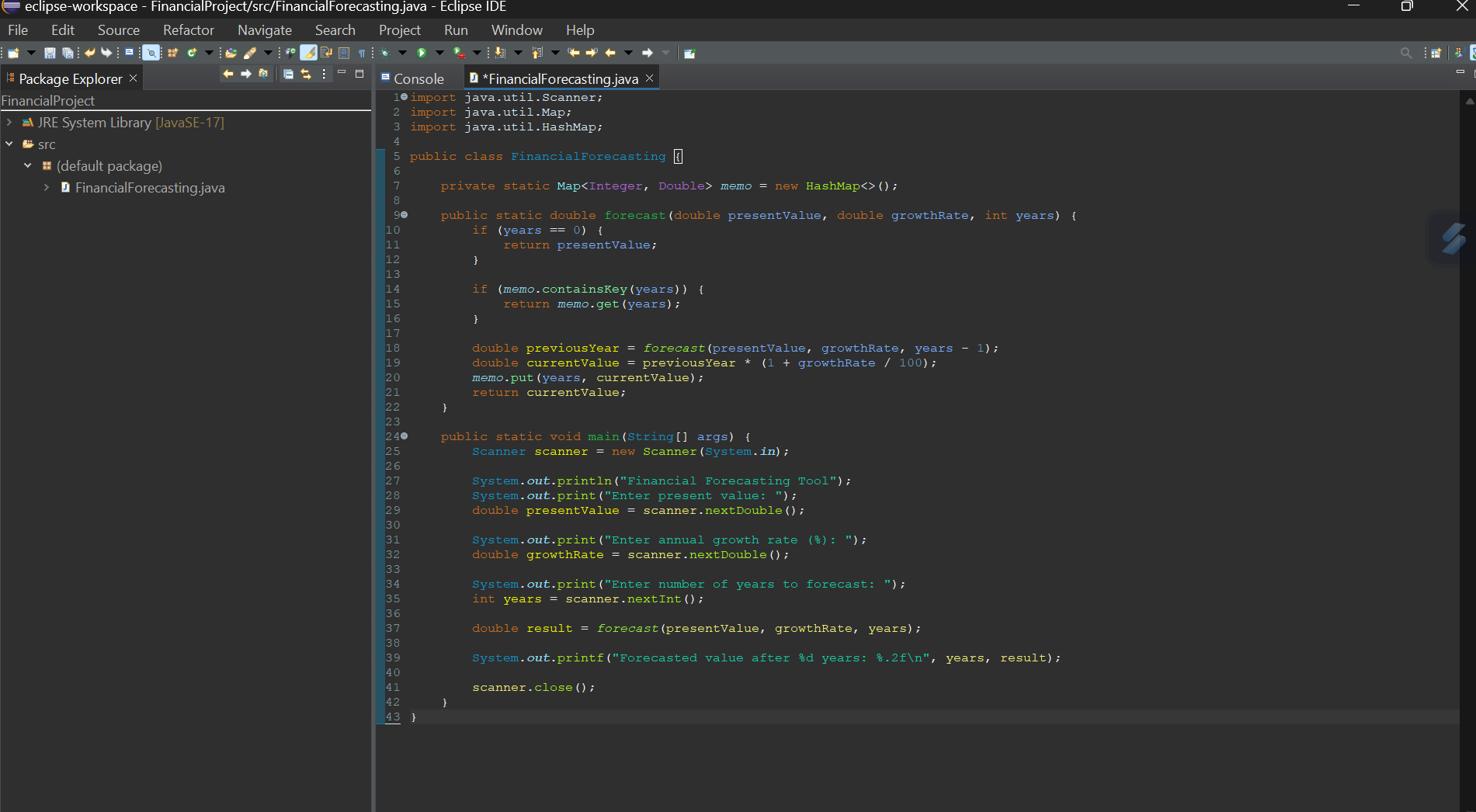
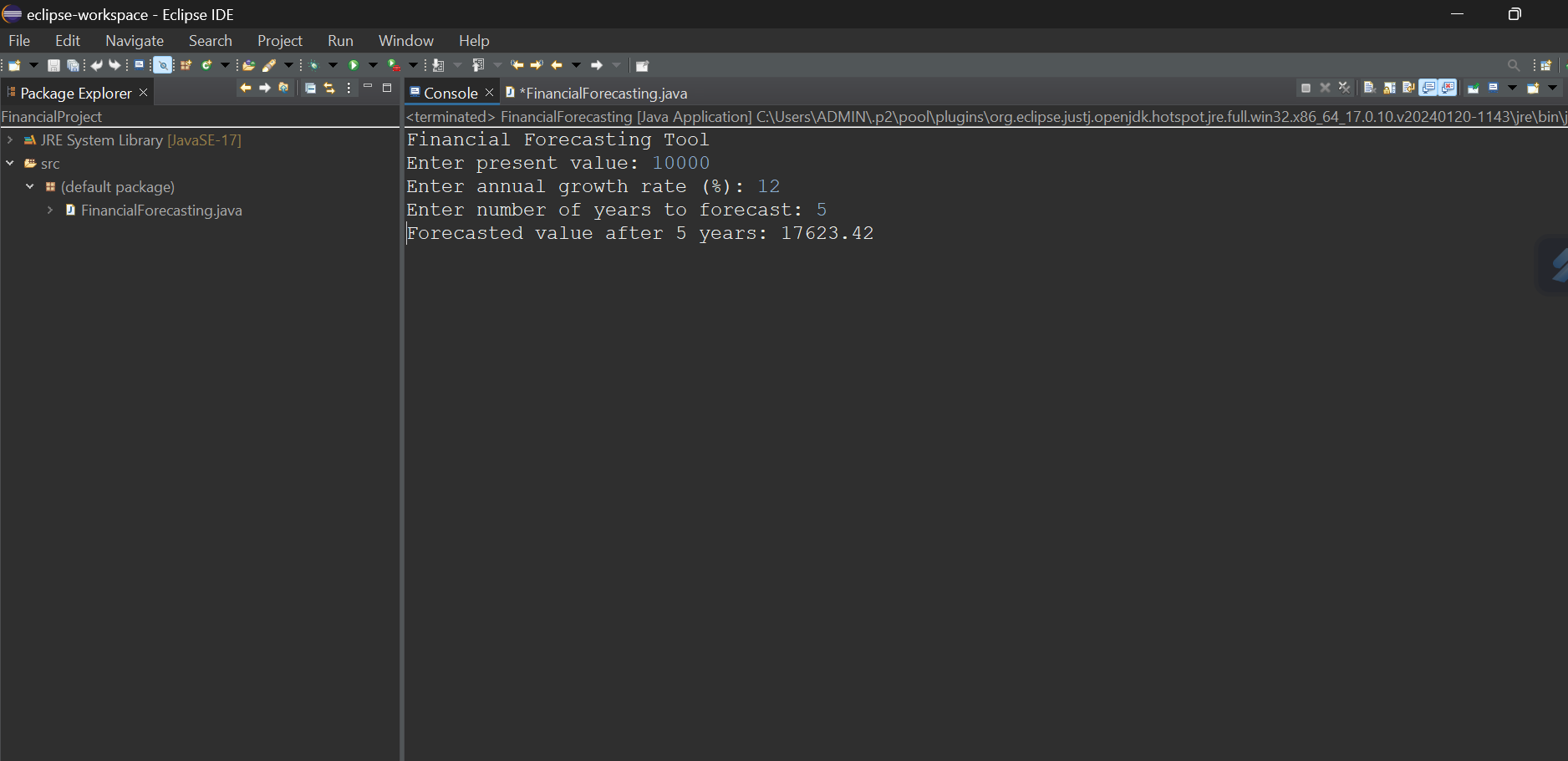
*DATA STRUCTURES AND ALGORITHMS*

***Exercise 01: Financial Forecasting***

**

*I developed a recursive financial forecasting tool in Java that calculates future investment value based on a given growth rate and time period. The core logic uses recursion to repeatedly apply the compound interest formula, and I optimized the solution using memoization to avoid redundant computations, significantly improving performance. The time complexity of the unoptimized recursive approach is exponential (O(2^n)), but with memoization, it reduces to linear time (O(n)). This recursive method simplifies the logic and ensures efficient handling of forecasting over large time spans*.

***CODE:***

*import java.util.Scanner;*

*import java.util.Map;*

*import java.util.HashMap;*

*public class FinancialForecasting {*

*private static Map<Integer, Double> memo = new HashMap<>();*

*public static double forecast(double presentValue, double growthRate, int years) {*

*if (years == 0) {*

*return presentValue;*

*}*

*if (memo.containsKey(years)) {*

*return memo.get(years);*

*}*

*double previousYear = forecast(presentValue, growthRate, years - 1);*

*double currentValue = previousYear \* (1 + growthRate / 100);*

*memo.put(years, currentValue);*

*return currentValue;*

*}*

*public static void main(String[] args) {*

*Scanner scanner = new Scanner(System.in);*

*System.out.println("Financial Forecasting Tool");*

*System.out.print("Enter present value: ");*

*double presentValue = scanner.nextDouble();*

*System.out.print("Enter annual growth rate (%): ");*

*double growthRate = scanner.nextDouble();*

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

*int years = scanner.nextInt();*

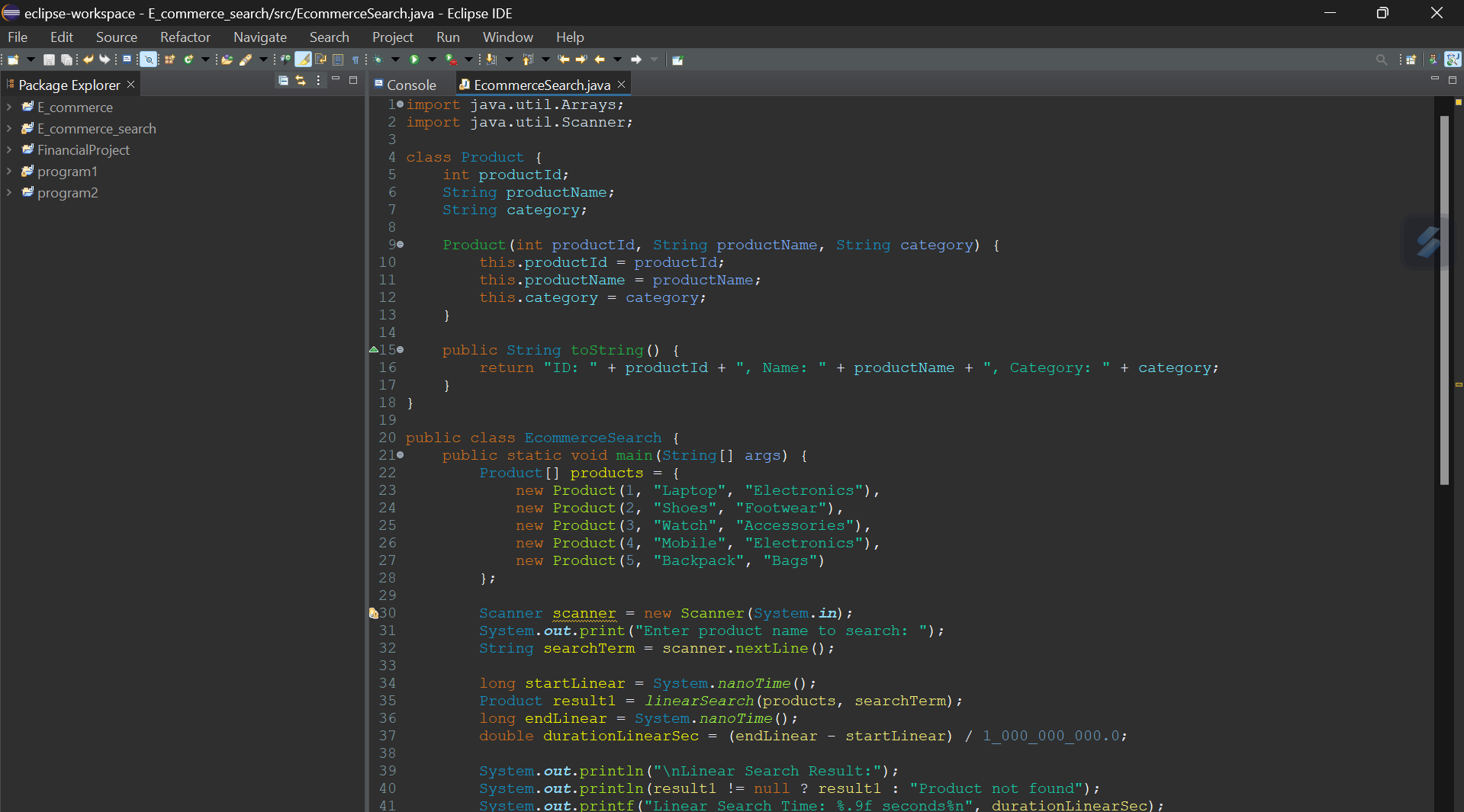
*double result = forecast(presentValue, growthRate, years);*

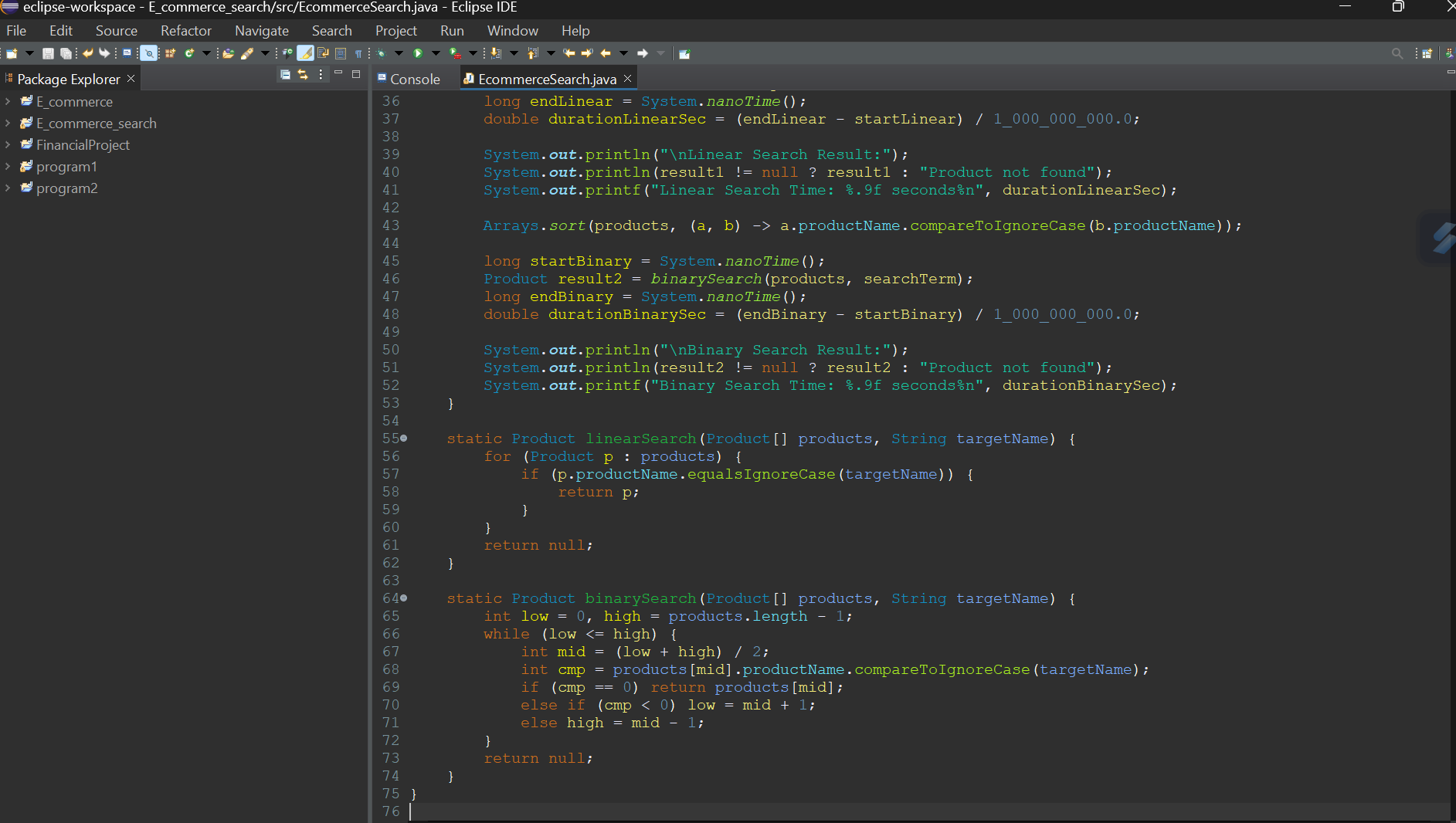
*System.out.printf("Forecasted value after %d years: %.2f\n", years, result);*

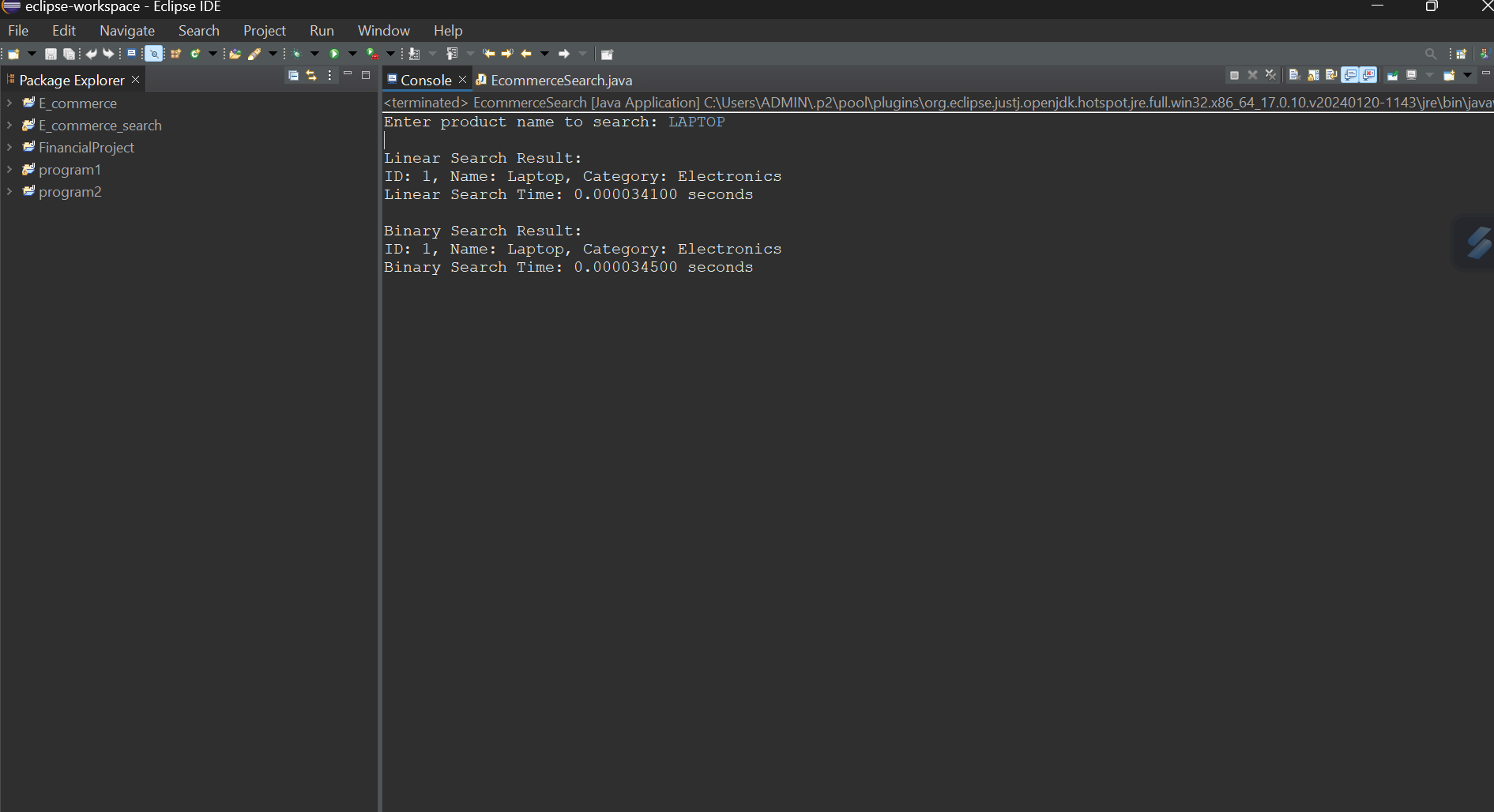
*scanner.close();*

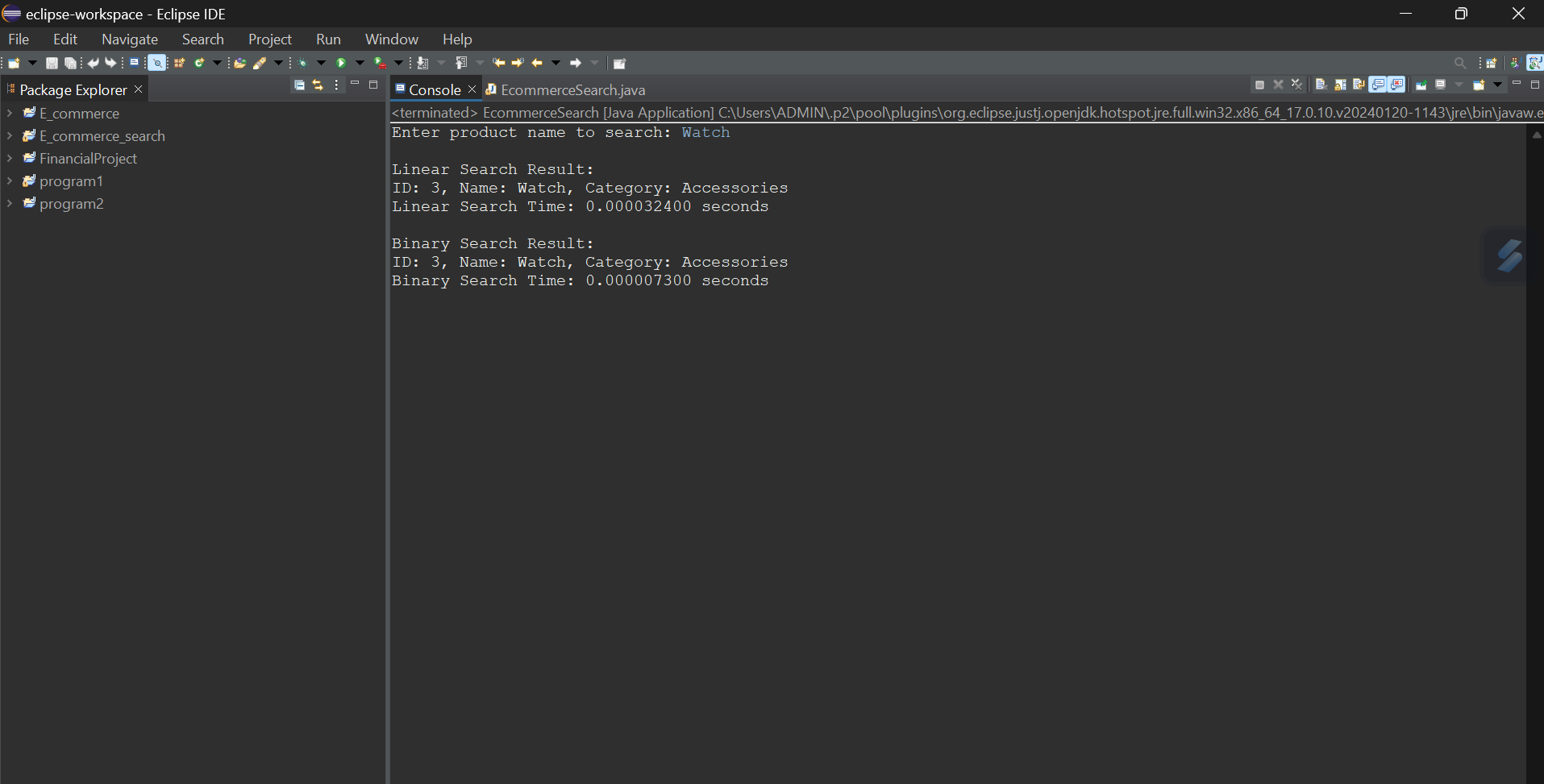
*}*

*}*

***Exercise 02: E-commerce Platform Search Function***

******

******

******

*I implemented a search functionality for an e-commerce platform in Java using both linear and binary search algorithms. I created a Product class with attributes like productId, productName, and category to simulate realistic data. Products are stored in an array, and the search feature allows users to input a product name. Linear search scans the array sequentially, while binary search operates on a sorted array using a divide-and-conquer strategy. To demonstrate efficiency, I measured the runtime for both searches in seconds using System.nanoTime(). The linear search has a time complexity of O(n), making it slower on large datasets, while binary search operates in O(log n), showing significantly faster performance. This comparison clearly shows how algorithm choice impacts real-world performance and helps in choosing the most efficient method for large-scale platforms.*

***CODE:***

*import java.util.Arrays;*

*import java.util.Scanner;*

*class Product {*

*int productId;*

*String productName;*

*String category;*

*Product(int productId, String productName, String category) {*

*this.productId = productId;*

*this.productName = productName;*

*this.category = category;*

*}*

*public String toString() {*

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

*}*

*}*

*public class EcommerceSearch {*

*public static void main(String[] args) {*

*Product[] products = {*

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

*new Product(2, "Shoes", "Footwear"),*

*new Product(3, "Watch", "Accessories"),*

*new Product(4, "Mobile", "Electronics"),*

*new Product(5, "Backpack", "Bags")*

*};*

*Scanner scanner = new Scanner(System.in);*

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

*String searchTerm = scanner.nextLine();*

*long startLinear = System.nanoTime();*

*Product result1 = linearSearch(products, searchTerm);*

*long endLinear = System.nanoTime();*

*double durationLinearSec = (endLinear - startLinear) / 1\_000\_000\_000.0;*

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

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

*System.out.printf("Linear Search Time: %.9f seconds%n", durationLinearSec);*

*Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));*

*long startBinary = System.nanoTime();*

*Product result2 = binarySearch(products, searchTerm);*

*long endBinary = System.nanoTime();*

*double durationBinarySec = (endBinary - startBinary) / 1\_000\_000\_000.0;*

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

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

*System.out.printf("Binary Search Time: %.9f seconds%n", durationBinarySec);*

*}*

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

*for (Product p : products) {*

*if (p.productName.equalsIgnoreCase(targetName)) {*

*return p;*

*}*

*}*

*return null;*

*}*

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

*int low = 0, 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;*

*}*

*}*