**DATA STRUCTURES AND ALGORITHMS**

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

**Product.java**

package ecommerce;

public class Product {

private int productId;

private String productName;

private String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

@Override

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

**SearchFunction.java**

package ecommerce;

import java.util.Arrays;

import java.util.Comparator;

public class SearchFunction {

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

for (Product p : products) {

if (p.getProductName().equalsIgnoreCase(targetName)) {

return p;

}

}

return null;

}

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

Arrays.sort(products, Comparator.comparing(Product::getProductName));

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int cmp = products[mid].getProductName().compareToIgnoreCase(targetName);

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

else if (cmp < 0) left = mid + 1;

else right = mid - 1;

}

return null;

}

}

**SearchTest.java**

package ecommerce;

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

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

new Product(2, "Shoes", "Fashion"),

new Product(3, "Book", "Education"),

new Product(4, "Phone", "Electronics"),

new Product(5, "Bag", "Fashion")

};

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

Product result1 = SearchFunction.linearSearch(products, "Book");

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

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

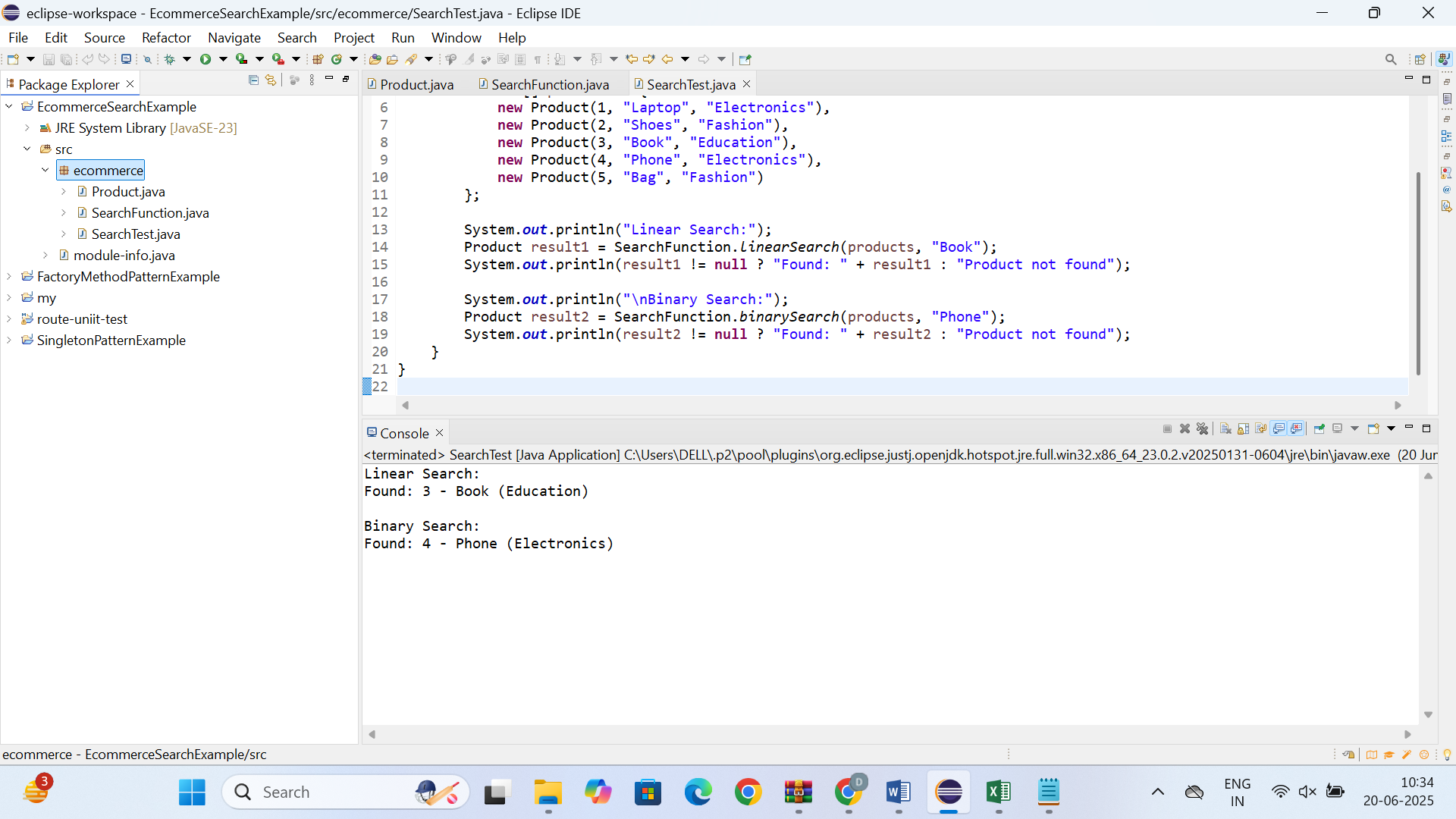
Product result2 = SearchFunction.binarySearch(products, "Phone");

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

}

}

**OUTPUT**



* **Exercise 7: Financial Forecasting**

**FinancialForecast.java**

package forecasting;

public class FinancialForecast {

public static double predictFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return predictFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

public static void main(String[] args) {

double initialValue = 10000;

double annualGrowthRate = 0.08;

int forecastYears = 5;

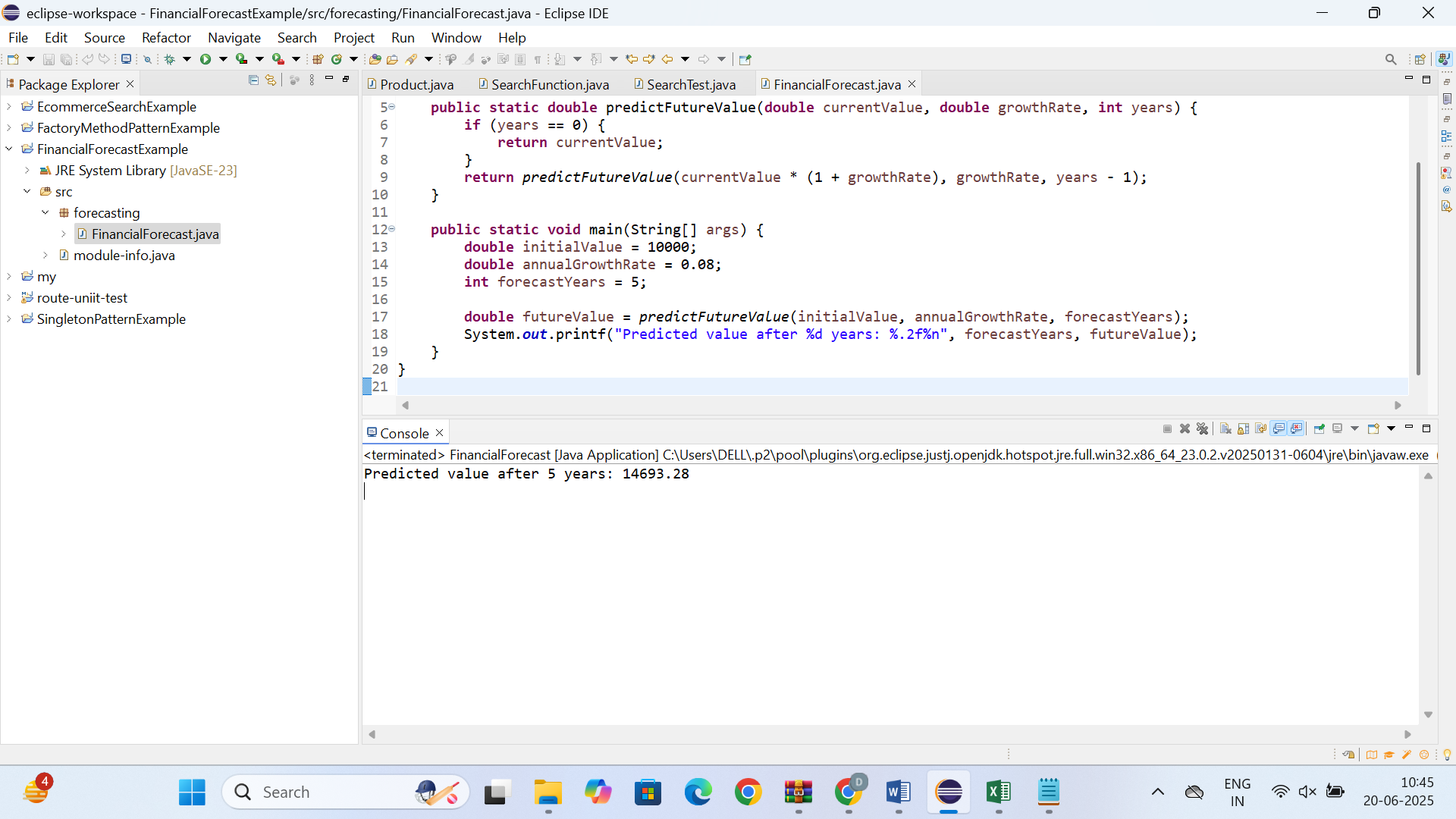
double futureValue = predictFutureValue(initialValue, annualGrowthRate, forecastYears);

System.out.printf("Predicted value after %d years: %.2f%n", forecastYears, futureValue);

}

}

**OUTPUT**



**EXPLANATION:**

**This is how it computes year by year:**

Year 0: 10000

Year 1: 10000 \* 1.08 = 10800

Year 2: 10800 \* 1.08 = 11664

Year 3: 11664 \* 1.08 = 12597.12

Year 4: 12597.12 \* 1.08 = 13604.89

Year 5: 13604.89 \* 1.08 = 14693.28

**Explain the concept of recursion and how it can simplify certain problems.**

 **Recursion** is a technique where a method calls itself to solve smaller instances of a problem.

 It consists of two main parts:

* **Base case** – the condition under which the recursion stops.
* **Recursive case** – the function calls itself with a smaller or simpler input.

 Recursion is useful for solving problems that can be broken down into similar sub-problems.

 It simplifies code for problems with **repetitive or hierarchical structure**, such as:

* Factorial calculation
* Fibonacci series
* Tree traversal
* Directory/file search

 Recursive algorithms are often **easier to understand and write** compared to iterative solutions for complex tasks.

 However, they must be used carefully to avoid **infinite loops** or **stack overflow**.

 Recursion can be optimized using techniques like **memoization** or **tail recursion** in some languages.