**COGNIZANT DIGI NURTURE 4.0**

**WEEK 1 MANDATORY HANDS ON BY (6377625)**

**Data structures and Algorithm:**

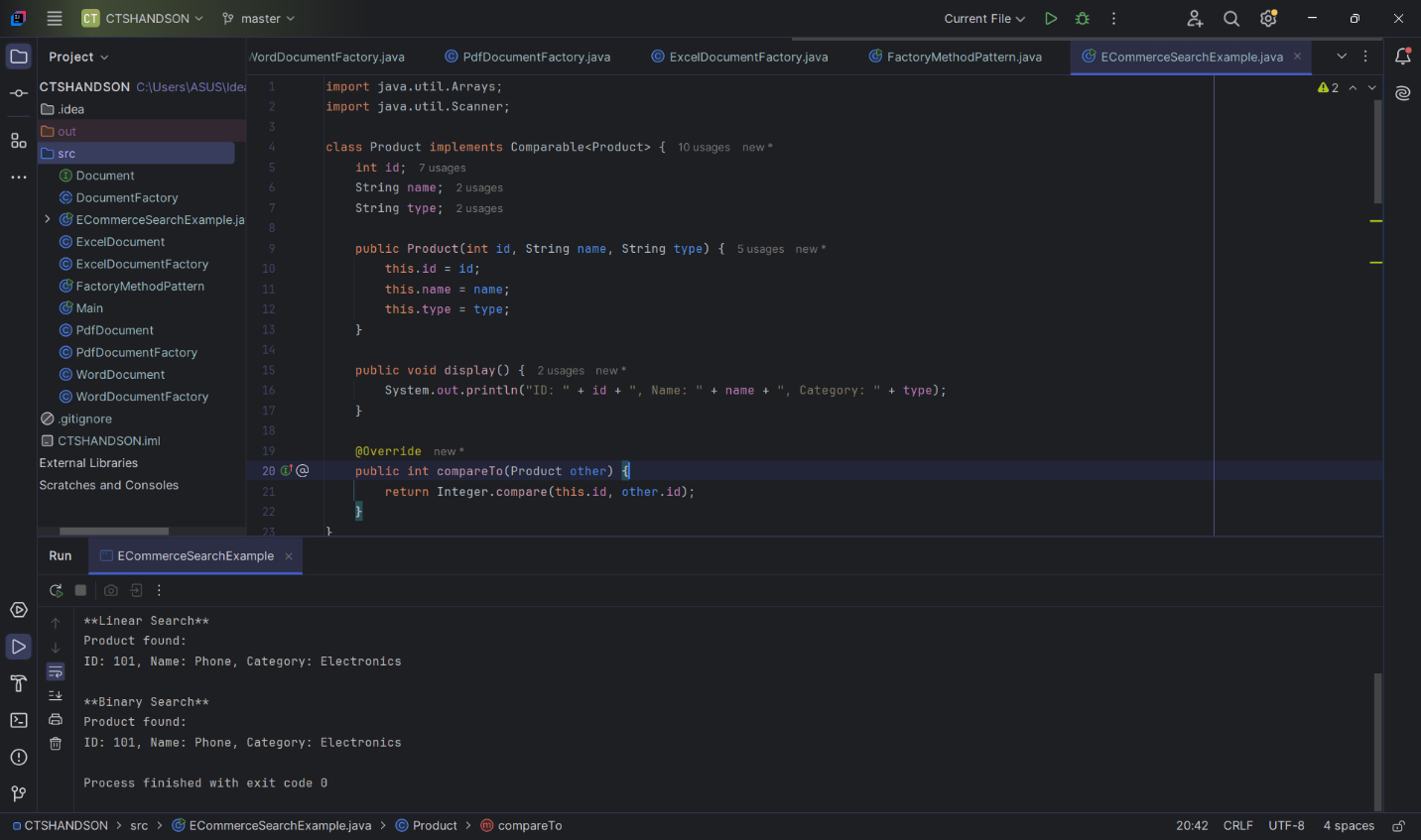
**Exercise 2: (E-commerce Platform Search Function)**

I developed a Product class that includes fields like productId, productName, and category. I implemented two search techniques: **Linear Search** and **Binary Search**, both used to locate products by their ID. For linear search, the products are placed in a regular array, while for binary search, the same array is first sorted. The program highlights the difference in performance between these two search methods using basic **asymptotic time complexity concepts**.

**ECommerceSearchExample.java:**

import java.util.Arrays;  
import java.util.Scanner;  
  
class Product implements Comparable<Product> {  
 int id;  
 String name;  
 String type;  
  
 public Product(int id, String name, String type) {  
 this.id = id;  
 this.name = name;  
 this.type = type;  
 }  
  
 public void display() {  
 System.out.println("ID: " + id + ", Name: " + name + ", Category: " + type);  
 }  
  
 @Override  
 public int compareTo(Product other) {  
 return Integer.compare(this.id, other.id);  
 }  
}  
  
public class ECommerceSearchExample {  
  
 public static int linearSearch(Product[] items, int keyId) {  
 for (int index = 0; index < items.length; index++) {  
 if (items[index].id == keyId) {  
 return index;  
 }  
 }  
 return -1;  
 }  
  
 public static int binarySearch(Product[] items, int keyId) {  
 int left = 0;  
 int right = items.length - 1;  
  
 while (left <= right) {  
 int mid = (left + right) / 2;  
  
 if (items[mid].id == keyId) {  
 return mid;  
 } else if (items[mid].id < keyId) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
  
 return -1;  
 }  
  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(105, "Shoes", "Footwear"),  
 new Product(101, "Phone", "Electronics"),  
 new Product(103, "Laptop", "Electronics"),  
 new Product(102, "T-shirt", "Clothing"),  
 new Product(104, "Watch", "Accessories")  
 };  
  
 Scanner scanner = new Scanner(System.in);  
 System.out.print("Enter Product ID to search: ");  
 int inputId = scanner.nextInt();  
  
 int linearIndex = linearSearch(products, inputId);  
 System.out.println("\n\*\*Linear Search\*\*");  
 if (linearIndex != -1) {  
 System.out.println("Product found:");  
 products[linearIndex].display();  
 } else {  
 System.out.println("Product not found!");  
 }  
  
 Arrays.sort(products);  
  
 int binaryIndex = binarySearch(products, inputId);  
 System.out.println("\n\*\*Binary Search\*\*");  
 if (binaryIndex != -1) {  
 System.out.println("Product found:");  
 products[binaryIndex].display();  
 } else {  
 System.out.println("Product not found!");  
 }  
 }  
}

**OUTPUT**:



**Exercise 7:** Financial Forecasting

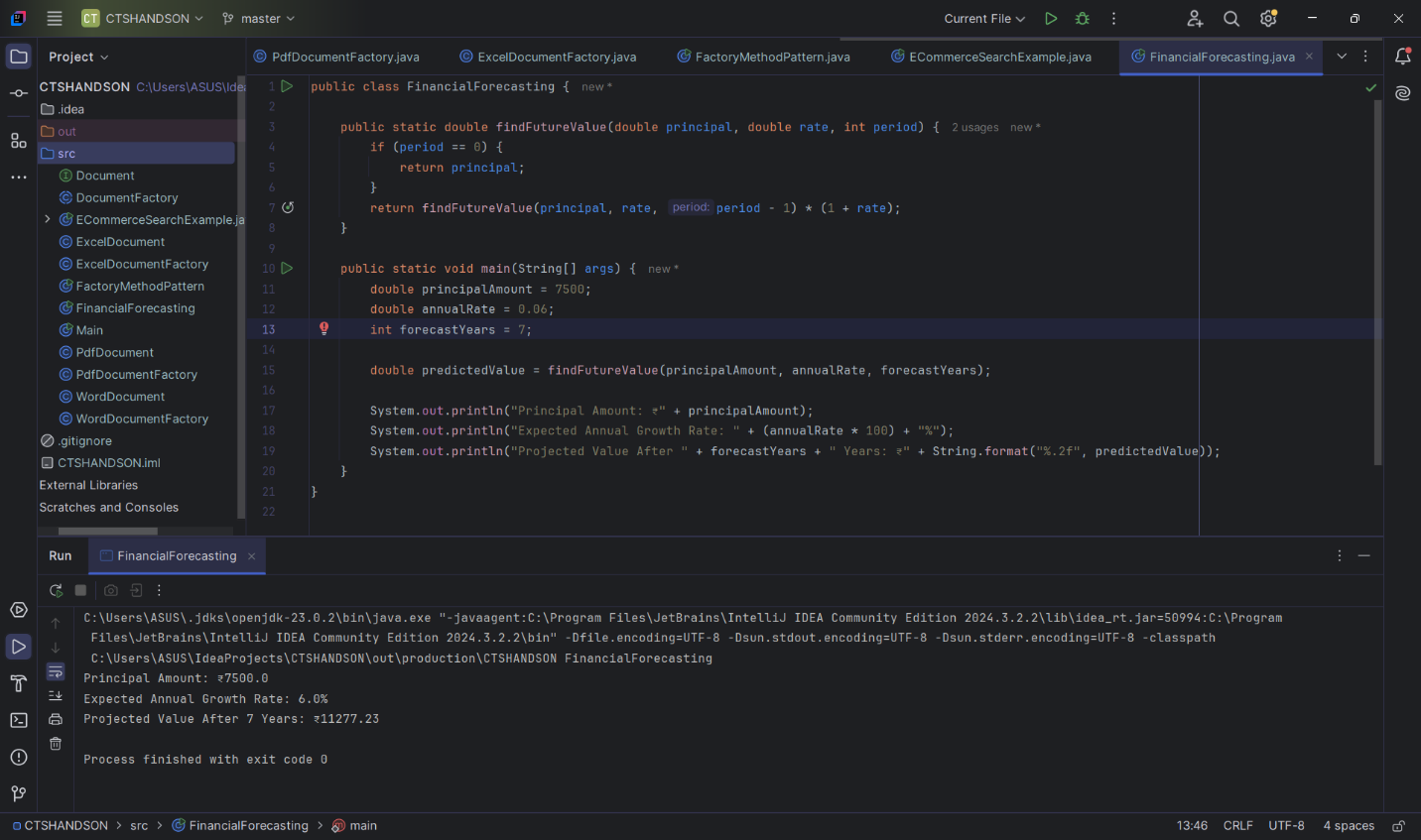
I created a recursive method to calculate future financial values based on initial amount and annual growth rate using recursion. This program demonstrates the concept of recursion and explores how it simplifies forecasting calculations over multiple years.

**CODE:**

**FinancialForecasting.java**

**public class FinancialForecasting {  
  
 public static double findFutureValue(double principal, double rate, int period) {  
 if (period == 0) {  
 return principal;  
 }  
 return findFutureValue(principal, rate, period - 1) \* (1 + rate);  
 }  
  
 public static void main(String[] args) {  
 double principalAmount = 7500;  
 double annualRate = 0.06;  
 int forecastYears = 7;  
  
 double predictedValue = findFutureValue(principalAmount, annualRate, forecastYears);  
  
 System.out.println("Principal Amount: ₹" + principalAmount);  
 System.out.println("Expected Annual Growth Rate: " + (annualRate \* 100) + "%");  
 System.out.println("Projected Value After " + forecastYears + " Years: ₹" + String.format("%.2f", predictedValue));  
 }  
}**

**OUTPUT:**

****