**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Explain why data structures and algorithms are essential in handling large inventories.**

Large inventories contain thousands of products which require efficient methods to create, retrieve, update, and delete. So, it is crucial to use appropriate data structures that allow to perform operations efficiently.

**Discuss the types of data structures suitable for this problem.**

HashMap can be used to store the product with their unique Product ID. The get() method in hashmaps retrieves the Product in O(1) time complexity. Adding Products using the put() method is also done in O(1) time complexity. So HashMap can be considered suitable for this problem.

To store different products in the inventory, ArrayList can also be used. But there are a few limitations when using arraylist for inventory management. Adding products to the arraylist can be done in O(1) time complexity. But to retrieve and update or delete the product, the retrieval process can take O(n) time complexity. This is not generally considered good in terms of efficiency of the program.

Other data structures such as LinkedList can also be used to store products in the inventory, but the time complexity for adding, retrieving and deleting the products would be high.

**Project: InventoryManagementSystem**

**Product.java**

public class Product {  
 private int productId;  
 private String productName;  
 private int quantity;  
 private double price;  
  
 public Product(int productId, String productName, int quantity, double price){  
 this.productId = productId;  
 this.productName = productName;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 public int getProductId() {  
 return productId;  
 }  
  
 public void setProductId(int productId) {  
 this.productId = productId;  
 }  
  
 public String getProductName() {  
 return productName;  
 }  
  
 public void setProductName(String productName) {  
 this.productName = productName;  
 }  
  
 public int getQuantity() {  
 return quantity;  
 }  
  
 public void setQuantity(int quantity) {  
 this.quantity = quantity;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 public void setPrice(double price) {  
 this.price = price;  
 }  
  
 public void printProduct(){  
 System.*out*.println("Product Id: " + productId + "\nProduct Name: " + productName + "\nProduct Quantity: " + quantity + "\nProduct Price: " + price + "\n");  
 }  
}

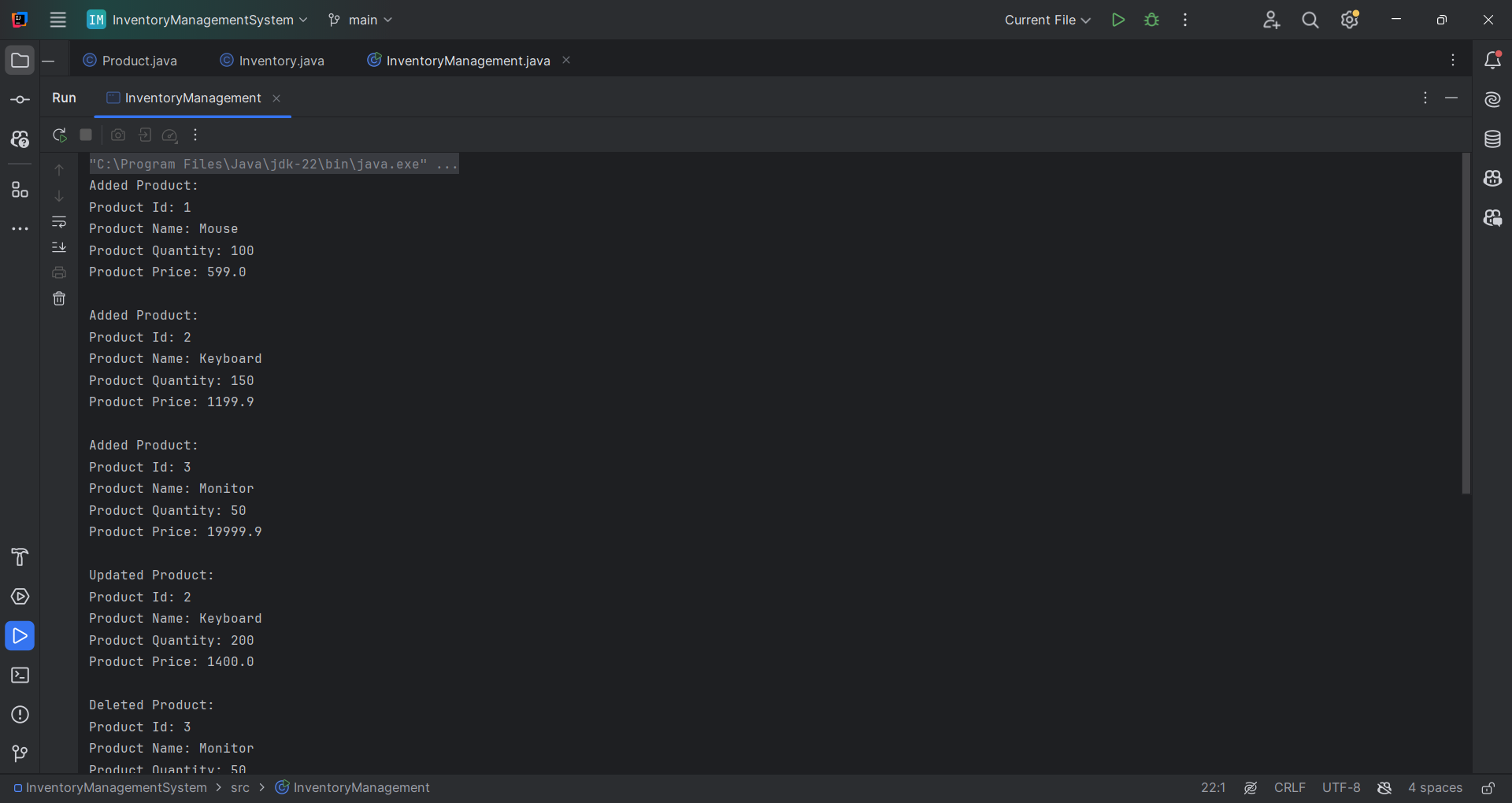
**Inventory.java**

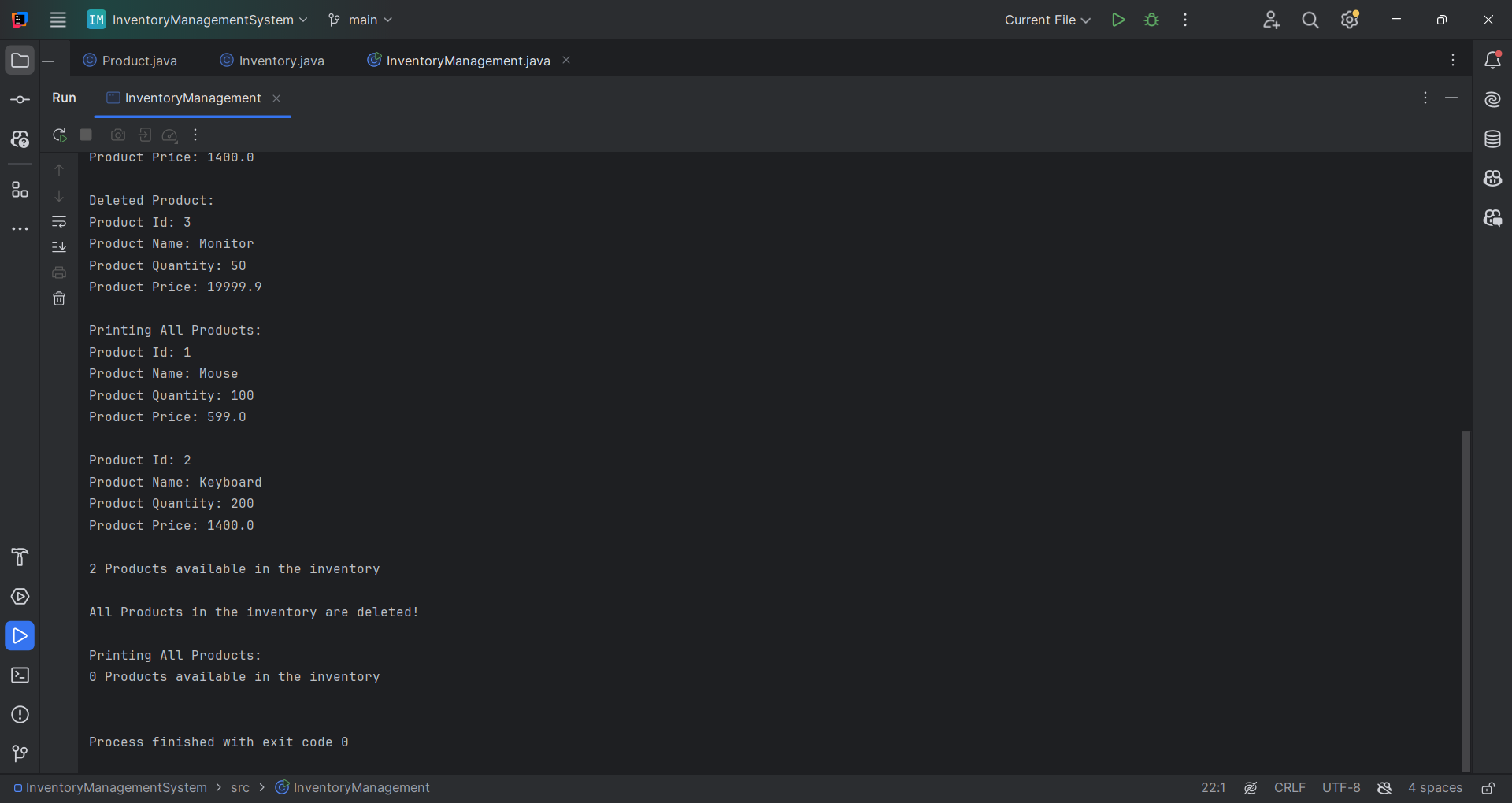
import java.util.\*;  
  
public class Inventory {  
 private HashMap<Integer, Product> inventory;  
  
 public Inventory(){  
 inventory = new HashMap<>();  
 }  
  
 public void addProduct(Product product){  
 inventory.put(product.getProductId(), product);  
 System.*out*.println("Added Product:");  
 product.printProduct();  
 }  
  
 public void updateProduct(int productId, int newQuantity, double newPrice){  
 if(inventory.containsKey((productId))){  
 Product product = inventory.get(productId);  
 product.setQuantity(newQuantity);  
 product.setPrice(newPrice);  
 System.*out*.println("Updated Product:");  
 product.printProduct();  
 }else{  
 System.*out*.println("Product ID not found! Please enter valid Product ID");  
 }  
 }  
  
 public void deleteProduct(int productId){  
 if(inventory.containsKey(productId)){  
 Product deleted = inventory.remove(productId);  
 System.*out*.println("Deleted Product:");  
 deleted.printProduct();  
 }else{  
 System.*out*.println("Product ID not found! Please enter valid Product ID");  
 }  
 }  
  
 public void deleteAllProducts(){  
 inventory.clear();  
 System.*out*.println("All Products in the inventory are deleted!\n");  
 }  
  
 public void printAllProducts(){  
 int count = 0;  
 System.*out*.println("Printing All Products:");  
 for(Product product : inventory.values()){  
 product.printProduct();  
 count++;  
 }  
 System.*out*.println(count + " Products available in the inventory\n");  
 }  
}

**InventoryManagement.java**

public class InventoryManagement {  
 public static void main(String[] args){  
 Inventory inventory = new Inventory();  
 Product product1 = new Product(1, "Mouse", 100, 599.0);  
 Product product2 = new Product(2, "Keyboard", 150, 1199.9);  
 Product product3 = new Product(3, "Monitor", 50, 19999.9);  
  
 inventory.addProduct(product1);  
 inventory.addProduct(product2);  
 inventory.addProduct(product3);  
  
 inventory.updateProduct(2, 200, 1400.00);  
  
 inventory.deleteProduct(3);  
  
 inventory.printAllProducts();  
  
 inventory.deleteAllProducts();  
 inventory.printAllProducts();  
 }  
}

**Output:**





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

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Explain Big O notation and how it helps in analyzing algorithms.**

Big O Notation is a mathematical notation used to describe the performance or complexity of an algorithm. It calculates the performance of the algorithm in terms of Time Complexity and Space complexity. It helps developers know the worst possible time and space that the algorithm can use. For example, if we perform a linear search in an array, the worst case is when the element is at the end of the array. So, the time complexity would be O(n) for n elements.

**Describe the best, average, and worst-case scenarios for search operations.**

Best-case scenario: Best-case scenario is when the desired element is found immediately. For example, when performing linear search on an array and the desired element is the first element. This is called best case scenario.

Average-case scenario: Average-case scenario is when the desired element is located somewhere in the middle. For example, when performing linear search on an array and the desired element is the middle element. This is called average-case scenario.

Worst-case scenario: Worst-case scenario is when the desired element is not found or it is the last element. For example, when performing linear search on an array, and the desired element is at the end of the array.

**Project: E-commerce Platform Search Function**

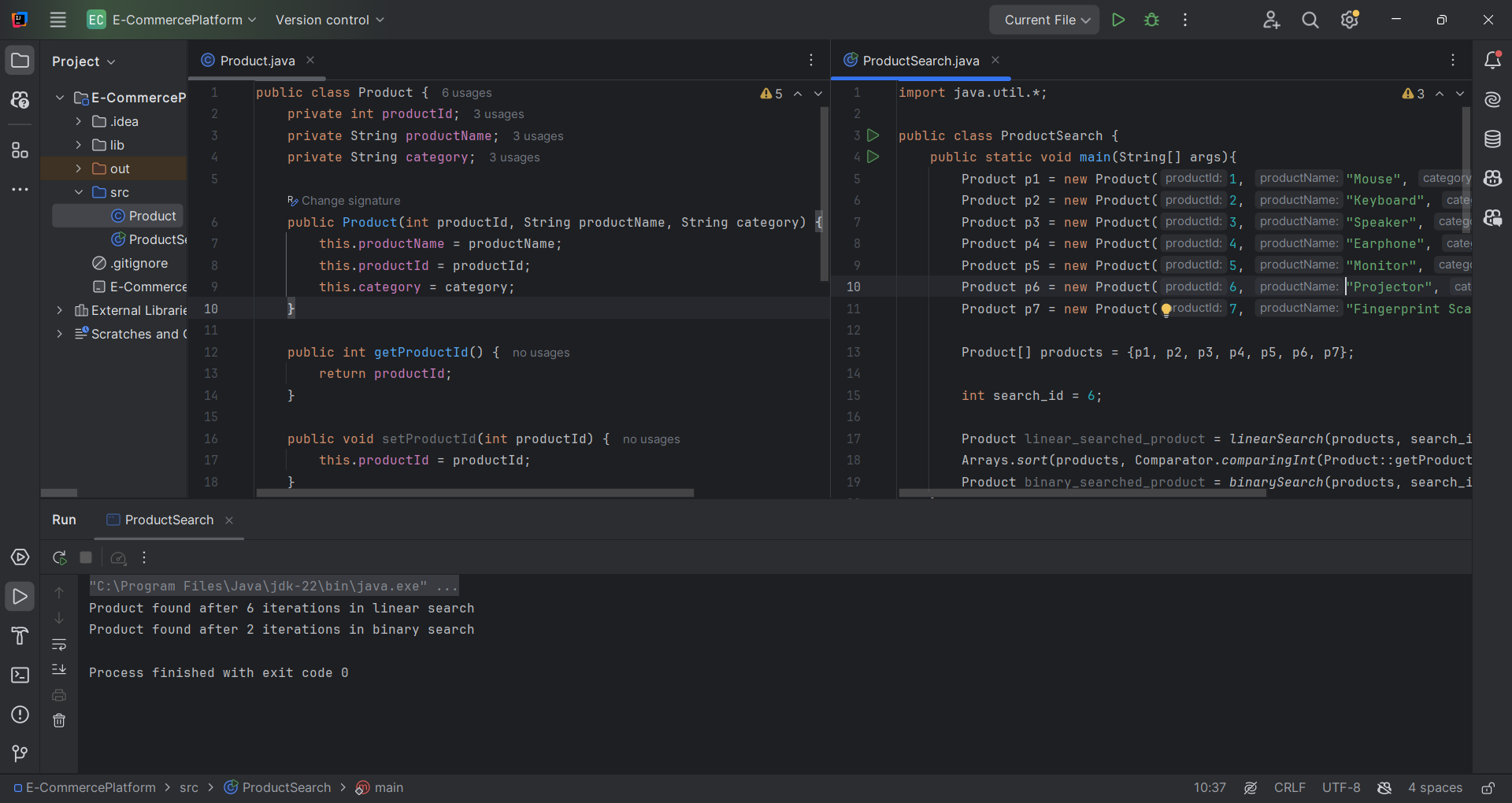
**Product.java**

public class Product {  
 private int productId;  
 private String productName;  
 private String category;  
  
 public Product(int productId, String productName, String category) {  
 this.productName = productName;  
 this.productId = productId;  
 this.category = category;  
 }  
  
 public int getProductId() {  
 return productId;  
 }  
  
 public void setProductId(int productId) {  
 this.productId = productId;  
 }  
  
 public String getProductName() {  
 return productName;  
 }  
  
 public void setProductName(String productName) {  
 this.productName = productName;  
 }  
  
 public String getCategory() {  
 return category;  
 }  
  
 public void setCategory(String category) {  
 this.category = category;  
 }  
}

**ProductSearch.java**

import java.util.\*;  
  
public class ProductSearch {  
 public static void main(String[] args){  
 Product p1 = new Product(1, "Mouse", "Input devices");  
 Product p2 = new Product(2, "Keyboard", "Input devices");  
 Product p3 = new Product(3, "Speaker", "Output devices");  
 Product p4 = new Product(4, "Earphone", "Output devices");  
 Product p5 = new Product(5, "Monitor", "Output devices");  
 Product p6 = new Product(6, "Projector", "Output device");  
 Product p7 = new Product(7, "Fingerprint Scanner", "Input device");  
  
 Product[] products = {p1, p2, p3, p4, p5, p6, p7};  
  
 int search\_id = 6;  
  
 Product linear\_searched\_product = *linearSearch*(products, search\_id);  
 Arrays.*sort*(products, Comparator.*comparingInt*(Product::getProductId));  
 Product binary\_searched\_product = *binarySearch*(products, search\_id);  
 }  
 public static Product linearSearch(Product[] products, int search\_id){  
 for(int i = 0; i < products.length; i++){  
 if(products[i].getProductId() == search\_id){  
 System.*out*.println("Product found after " + (i + 1) + " iterations in linear search");  
 return products[i];  
 }  
 }  
 System.*out*.println("Product not found");  
 return null;  
 }  
  
 public static Product binarySearch(Product[] products, int search\_id){  
 int i = 0;  
 int low = 0;  
 int high = products.length - 1;  
 int mid = (low + high)/2;  
 while(low <= high){  
 i++;  
 mid = (low + high)/2;  
 if(products[mid].getProductId() == search\_id){  
 System.*out*.println("Product found after " + i + " iterations in binary search");  
 return products[mid];  
 }else if(products[mid].getProductId() < search\_id){  
 low = mid + 1;  
 }else{  
 high = mid - 1;  
 }  
 }  
 System.*out*.println("Product not found!!!");  
 return null;  
 }  
}

**Output:**



**Analysis:**

1. **Compare the time complexity of linear and binary search algorithms.**

**Linear Search:** Time Complexity - O(n)

**Binary Search:** Time Complexity – O(log n)

Linear search checks from the beginning to the end until the desired element is found. So the worst case time complexity would be O(n). In our program, we have searched for the element with the productId. Linear search takes 6 iterations to find the element.

Binary search checks whether the middle element is the desired element, else it checks whether the element is after the middle element or before the middle element as the array is sorted. This process is repeated until the desired element is found. Binary Search takes O(log n) time complexity. The same product with the productid 6 is found in just 2 iterations while using binary search.

1. **Discuss which algorithm is more suitable for your platform and why.**

Both search algorithms have their own advantages and disadvantages. In terms of search time complexity, Binary search is much faster. But whenever a new product is added to the inventory, the inventory must be sorted.

In conclusion for large number of products, binary search is more suitable when sorted by the productid.

**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

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

Recursion is a programming technique where a method calls itself to solve a problem.

For example, to print n natural numbers using recursion:

public static void printNumbers(int n) {

if (n == 0) return;

printNumbers(n - 1);

System.out.println(n);

}

The method keeps calling itself with a smaller value of n until n becomes 0. This repeated self-calling process is known as recursion.

**Project: FinancialForecasting**

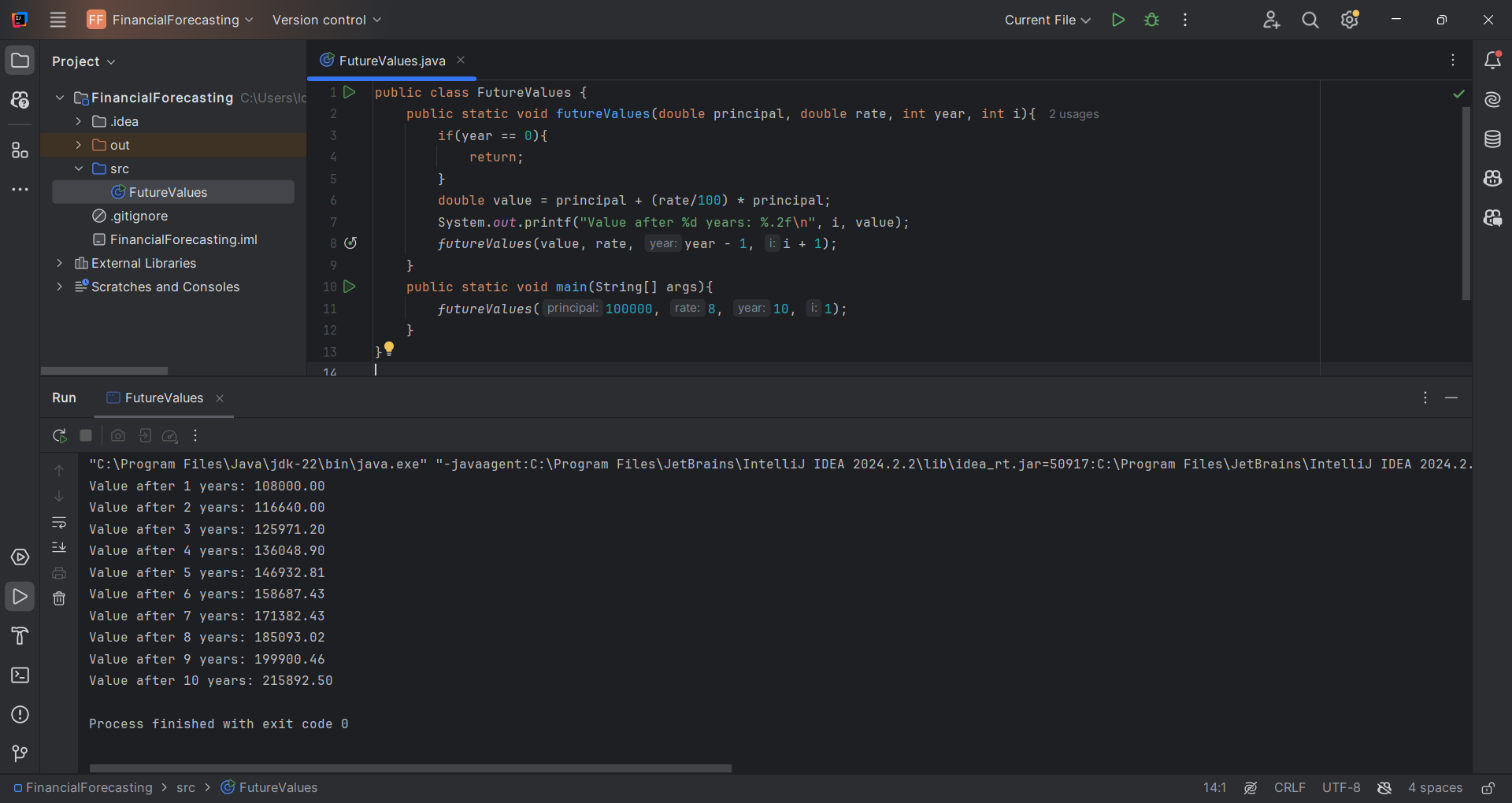
**FutureValues.java**

public class FutureValues {  
 public static void futureValues(double principal, double rate, int year, int i){  
 if(year == 0){  
 return;  
 }  
 double value = principal + (rate/100) \* principal;  
 System.*out*.printf("Value after %d years: %.2f\n", i, value);  
 *futureValues*(value, rate, year - 1, i + 1);  
 }  
 public static void main(String[] args){  
 *futureValues*(100000, 8, 10, 1);  
 }  
}

**Explanation:**

The futureValues() method is executed recursively until the remaining years become 0.  
Each time the method is called recursively, it prints the value of the principal amount after the corresponding year..

Output:



**Analysis:**

**Discuss the time complexity of your recursive algorithm.**

Time Complexity: O(n), where n is the number of years.

**Explain how to optimize the recursive solution to avoid excessive computation.**

An iterative loop does the same work without the overhead of function calls or stack usage

Optimize recursion using memoization by storing results of already computed subproblems in a cache. This problem doesn’t require memoization as it doesn’t require already computed values.