Data Structures and Algorithms - Hands-On Assignment

# Problem 1: E-commerce Platform Search Function

# 1.1 Code Implementation

### Product Class (Base Model)

package search;  
  
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;  
 }  
   
 public String toString() {  
 return productId + " " + productName + " " + category;  
 }  
}

### Linear Search Implementation

package search;  
  
public class LinearSearch {  
 public Product ls(Product[] products, String Name) {  
 for(Product p : products) {  
 if(p.productName.equalsIgnoreCase(Name)) {  
 return p;  
 }  
 }  
 return null;  
 }  
}

### Binary Search Implementation

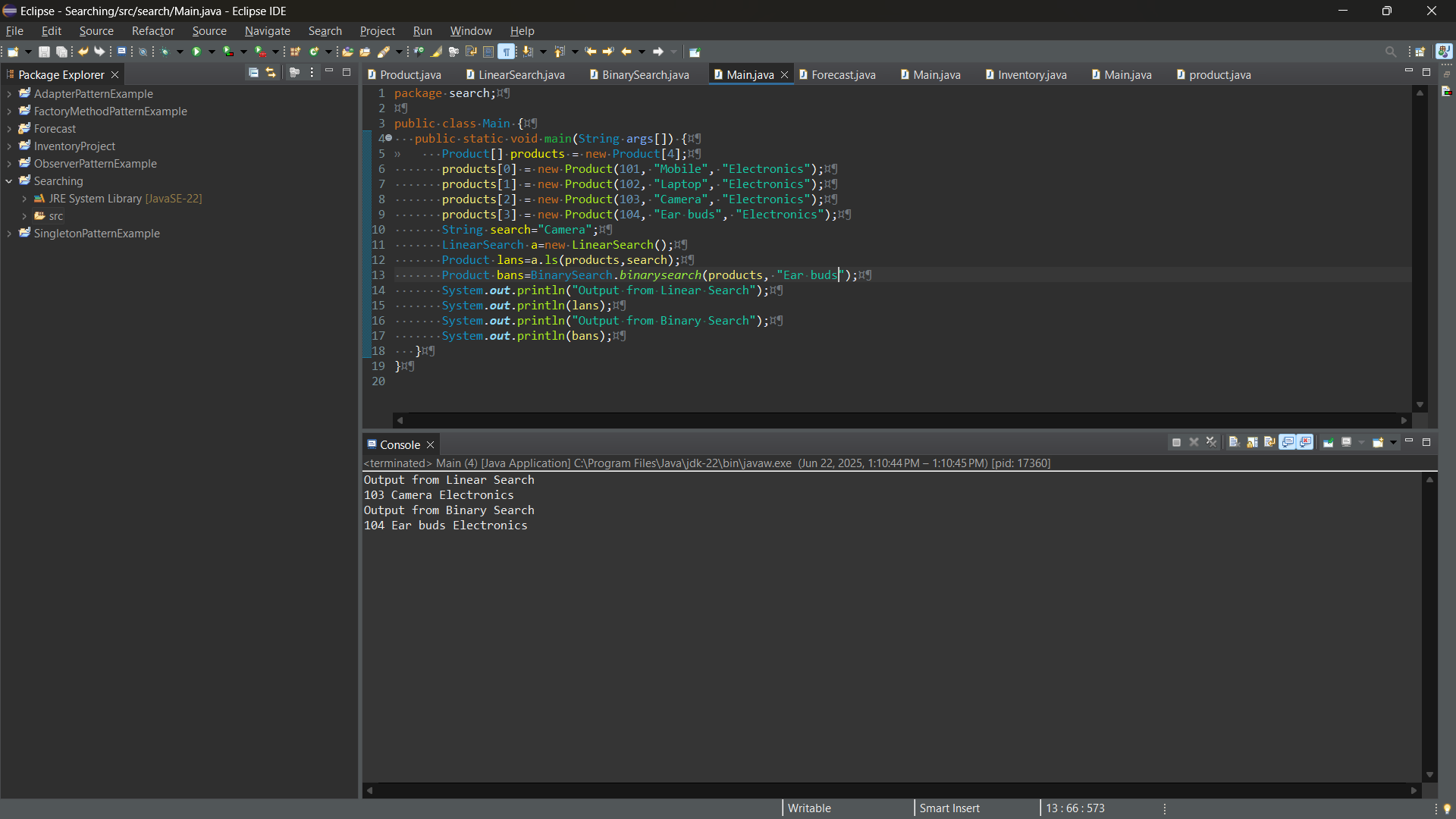
package search;  
  
import java.util.Arrays;  
import java.util.Comparator;  
  
public class BinarySearch {  
 public static Product binarysearch(Product[] products, String Name) {  
 Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));  
 int left = 0;  
 int right = products.length - 1;  
 while(left <= right) {  
 int mid = (left + right) / 2;  
 int comp = products[mid].productName.compareToIgnoreCase(Name);  
 if(comp == 0) {  
 return products[mid];  
 } else if(comp < 0) {  
 left = mid + 1;  
 } else {  
 right = mid - 1;  
 }  
 }  
 return null;  
 }  
}

### Main Class (Driver Code)

package search;  
  
public class Main {  
 public static void main(String args[]) {  
 Product[] products = new Product[4];  
 products[0] = new Product(101, "Mobile", “Electronics ");  
 products[1] = new Product(102, "Laptop", "Electronics");  
 products[2] = new Product(103, "Camera", "Electronics");  
 products[3] = new Product(104, "Ear buds", " Electronics ");  
   
 String search = "Camera";  
 LinearSearch a = new LinearSearch();  
 Product lans = a.ls(products, search);  
 Product bans = BinarySearch.binarysearch(products, "Ear buds");  
   
 System.out.println("Output from Linear Search");  
 System.out.println(lans);  
 System.out.println("Output from Binary Search");  
 System.out.println(bans);  
 }  
}

## Expected Output

Output from Linear Search  
103 Camera Electronics  
Output from Binary Search  
104 Ear buds Electronics



## Analysis

Algorithm Time Complexity Space Complexity Best Use Case  
Linear Search O(n) O(1) Small or unsorted datasets  
Binary Search O(log n) O(1) Large sorted datasets  
  
Key Notes:  
- Binary Search requires sorting first (O(n log n) preprocessing)  
- Linear Search works on unsorted data  
- Both methods are case-insensitive  
- Both methods are case-insensitive

# Problem 7: Financial Forecasting

## Code Implementation

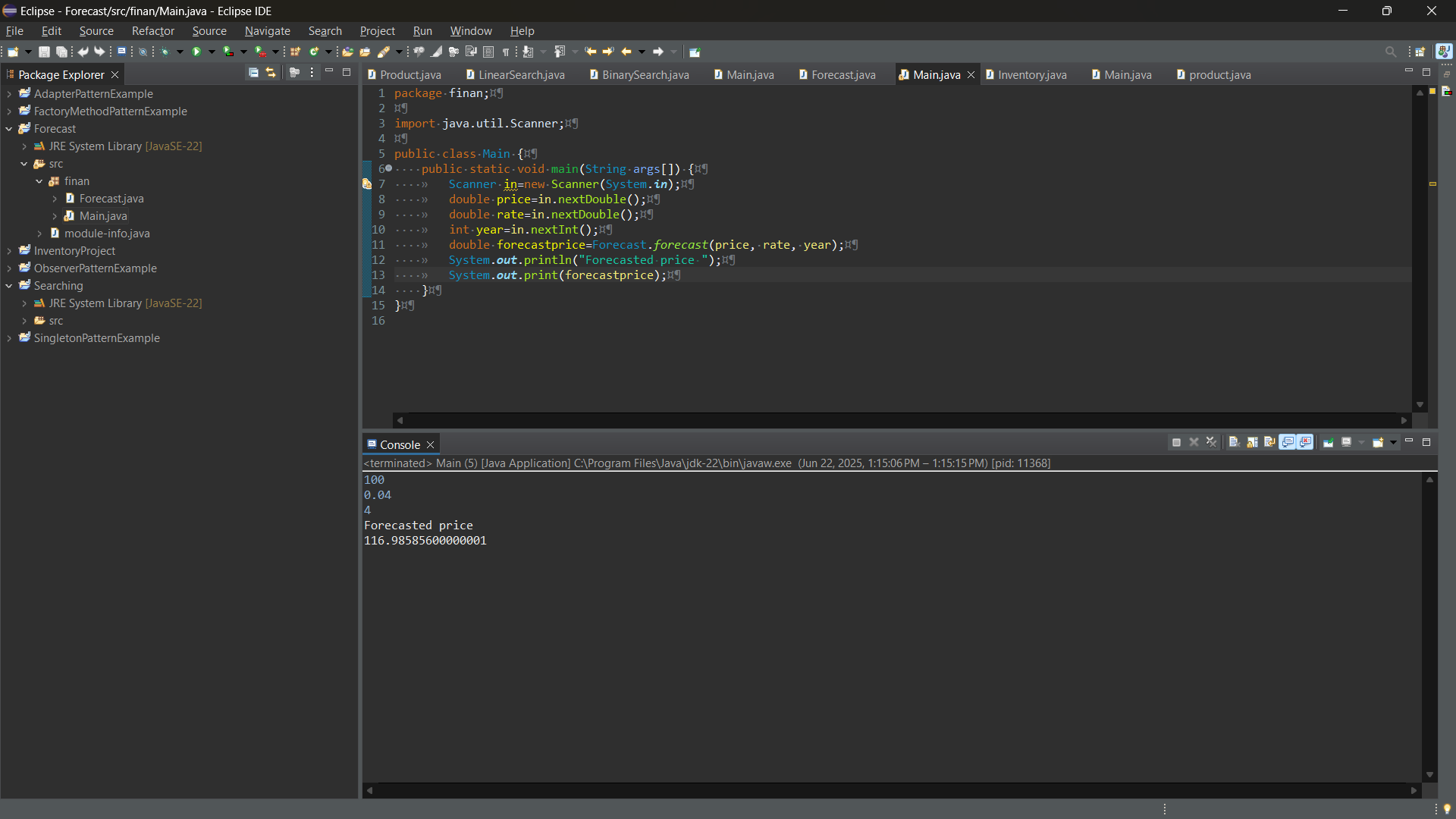
### Forecast Class

package finan;  
  
public class Forecast {  
 public static double forecast(double amount, double rate, int year) {  
 if(year == 0) {  
 return amount;  
 }  
 return forecast(amount, rate, year - 1) \* (1 + rate);  
 }  
}

### Main Class (Driver Code)

package finan;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String args[]) {  
 Scanner in = new Scanner(System.in);  
 double price = in.nextDouble();  
 double rate = in.nextDouble();  
 int year = in.nextInt();  
   
 Forecast fo = new Forecast();  
 double forecastprice = fo.forecast(price, rate, year);  
 System.out.print(forecastprice);  
 }  
}

## Output

For input: 1000 0.05 3  
Output: 1157.625  
  


## Analysis

Aspect Details  
Algorithm Recursive compound interest  
Time Complexity O(n)  
Space Complexity O(n) (call stack)  
Input Example Principal: 1000, Rate: 5%, Years: 3  
Output Example 1157.63