**WEEK 1 SOLUTONS**

**Design Patterns :**

**Exercise 1: Implementing the Singleton Pattern:**

**Java code:**

Logger.java :

package SingletonPattern;

public class Logger {

private static Logger *log*;

private Logger() {}

public static Logger getInstance() {

if(*log*==null) {

*log* = new Logger();

}

return *log*;

}

public static void printLog(String msg) {

System.***out***.println(msg);

}

}

Test.java:

package SingletonPattern;

public class Test {

public static void main(String[] args) {

Logger log1 = Logger.*getInstance*();

log1.*printLog*("Printing First Log");

Logger log2 = Logger.*getInstance*();

log2.*printLog*("Printing Second Log");

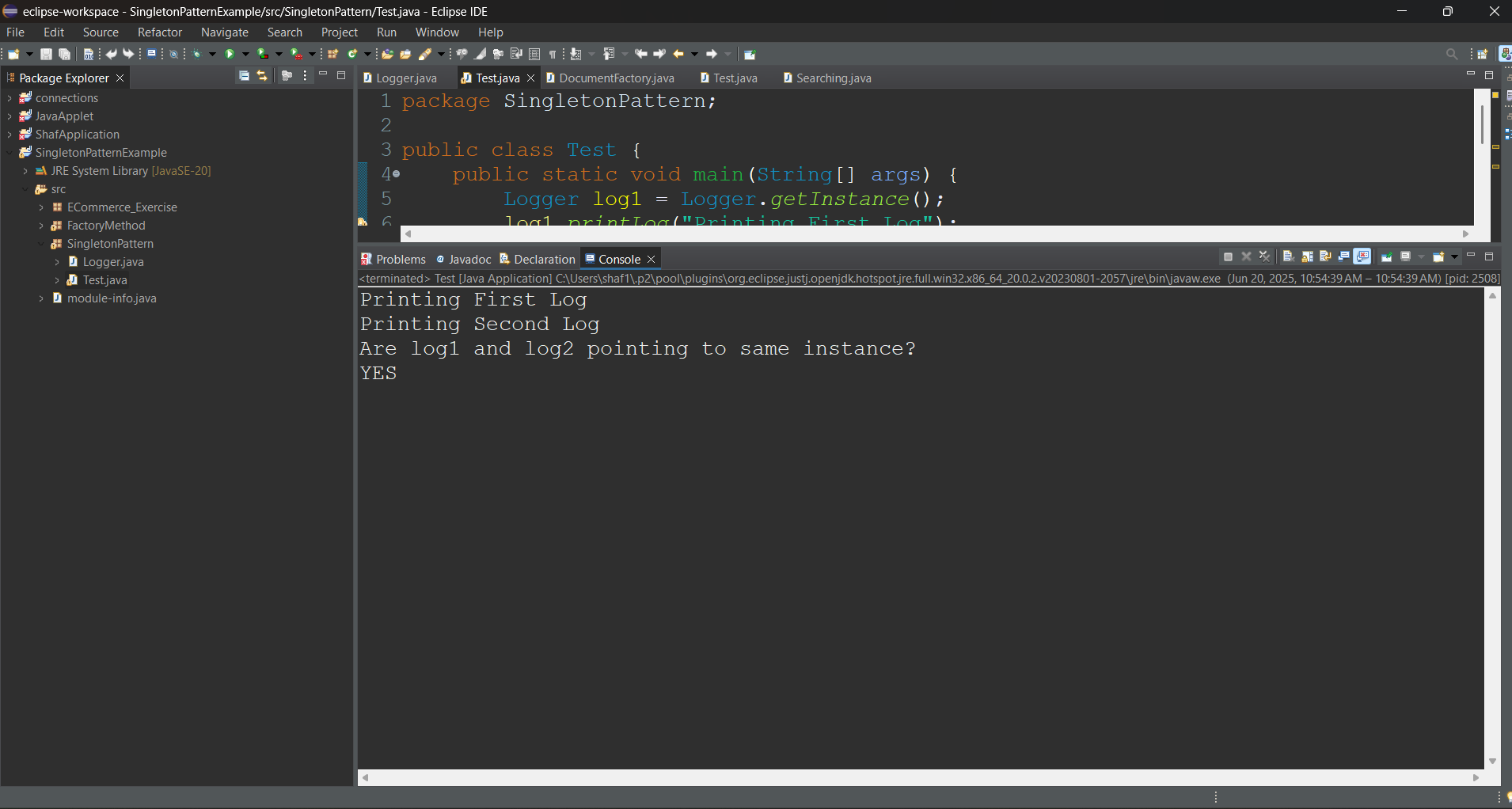
System.***out***.println("Are log1 and log2 pointing to same instance?");

System.***out***.println((log1==log2)==true?"YES":"NO");

}

}

Output:



**Exercise 2: Implementing the Factory Method Pattern**

**Java code:**

DocumentFactory.java:

package FactoryMethod;

interface Document{

String getType();

void setName(String name);

void save();

}

class WordDocument implements Document{

private String Filename;

*@Override*

public String getType() {

return "docx";

}

public void setName(String name) {

Filename = name;

}

public void save() {

System.***out***.println("Saving as "+Filename+"."+getType());

}

}

class PDFDocument implements Document{

private String Filename;

*@Override*

public String getType() {

return "pdf";

}

public void setName(String name) {

Filename = name;

}

public void save() {

System.***out***.println("Saving as "+Filename+"."+getType());

}

}

class ExcelDocument implements Document{

private String Filename;

*@Override*

public String getType() {

return "xls";

}

public void setName(String name) {

Filename = name;

}

public void save() {

System.***out***.println("Saving as "+Filename+"."+getType());

}

}

public class DocumentFactory {

public Document createDocument(String type) {

if(type.equals("docx")||type.equals("doc")) {

return new WordDocument();

}else if(type.equals("pdf")) {

return new PDFDocument();

}else if(type.equals("xls")||type.equals("xlsx")) {

return new ExcelDocument();

}

return null;

}

}

Test.java:

package FactoryMethod;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

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

System.***out***.println("Enter File Type: ");

String fileType = sc.nextLine();

DocumentFactory df = new DocumentFactory();

Document dc = df.createDocument(fileType);

if(dc==null) {

System.***out***.println("ERROR : Not Such Type. File Not Created");

}else {

System.***out***.println("Enter File Name: ");

String fileName = sc.nextLine();

dc.setName(fileName);

dc.save();

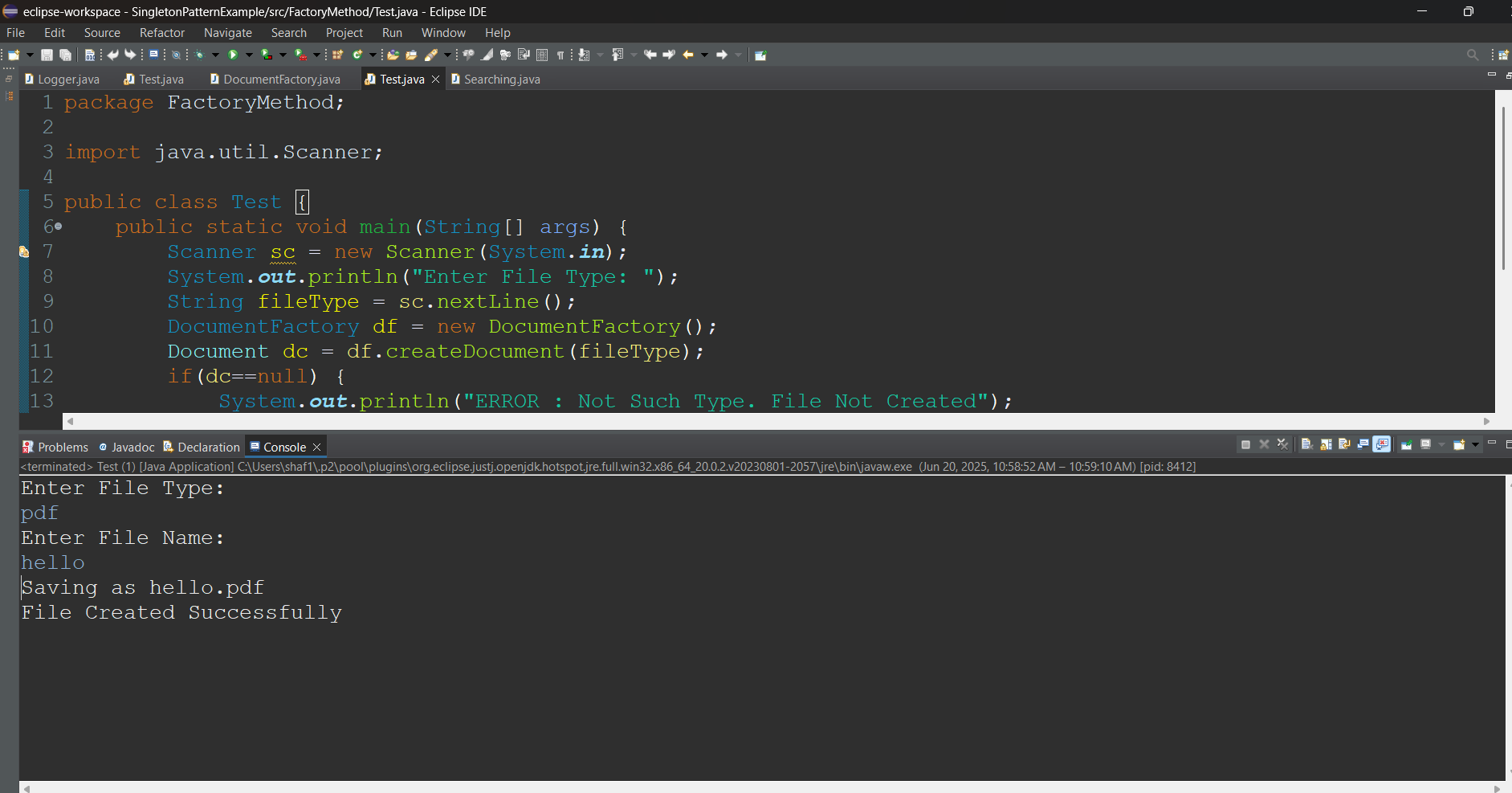
System.***out***.println("File Created Successfully");

}

}

}

Output:



**Algorithms & Data Structures :**

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

* *Big O notation describes the worst-case time or space complexity of an algorithm as the input size grows. It helps in comparing the efficiency of algorithms regardless of hardware or programming language.*
* *Search operations :*

*Linear search Binary Search*

*Best O(1) O(1)*

*Avg O(n) O(logn)*

*Worst O(n) O(logn)*

**Java code:**

package ECommerce\_Exercise;

class Product {

String productID;

String productName;

String category;

public Product(String pi,String pn,String c) {

this.productID=pi;

this.productName=pn;

this.category=c;

}

}

class List{

public Product[] productList(int n) {

Product[] p\_list = new Product[5];

p\_list[0] = new Product("WLE01","WireLess EarPhones","Electronic");

p\_list[1]= new Product("MON03", "27-inch Monitor", "Electronic");

p\_list[2]= new Product("MOU07", "Wireless Mouse", "Accessories");

p\_list[3]= new Product("KEY10", "Mechanical Keyboard", "Accessories");

p\_list[4]= new Product("CHA13", "Ergo Chair", "Furniture");

return p\_list;

}

}

public class Searching{

public static void main(String[] args) {

List ls = new List();

Product[] arr = ls.productList(5);

//Linear Search

System.***out***.print("Linearly Searching CHA13 : ");

if(*linearSearch*(arr,"CHA13")) { System.***out***.println("FOUND");}

else{System.***out***.println("NOT FOUND");}

// Binary Search

//array is already in sorted form

System.***out***.print("Binary Search KEY9 : ");

if(*binarySearch*(arr,"KEY9")) { System.***out***.println("FOUND");}

else{System.***out***.println("NOT FOUND");}

}

static boolean linearSearch(Product[] arr,String product\_id\_to\_be\_searched){

for(Product product: arr) {

if(product.productID.equals(product\_id\_to\_be\_searched)) {

return true;

}

}

return false;}

static boolean binarySearch(Product[] arr,String product\_id\_to\_be\_searched) {

int start =0,end = arr.length-1;

while(start<=end) {

int mid = start+(end-start)/2;

if(arr[mid].productID.equals(product\_id\_to\_be\_searched)) {

return true;

}

else if(arr[mid].productID.compareTo(product\_id\_to\_be\_searched)>0) {

end = mid-1;

}else {

start=mid+1;

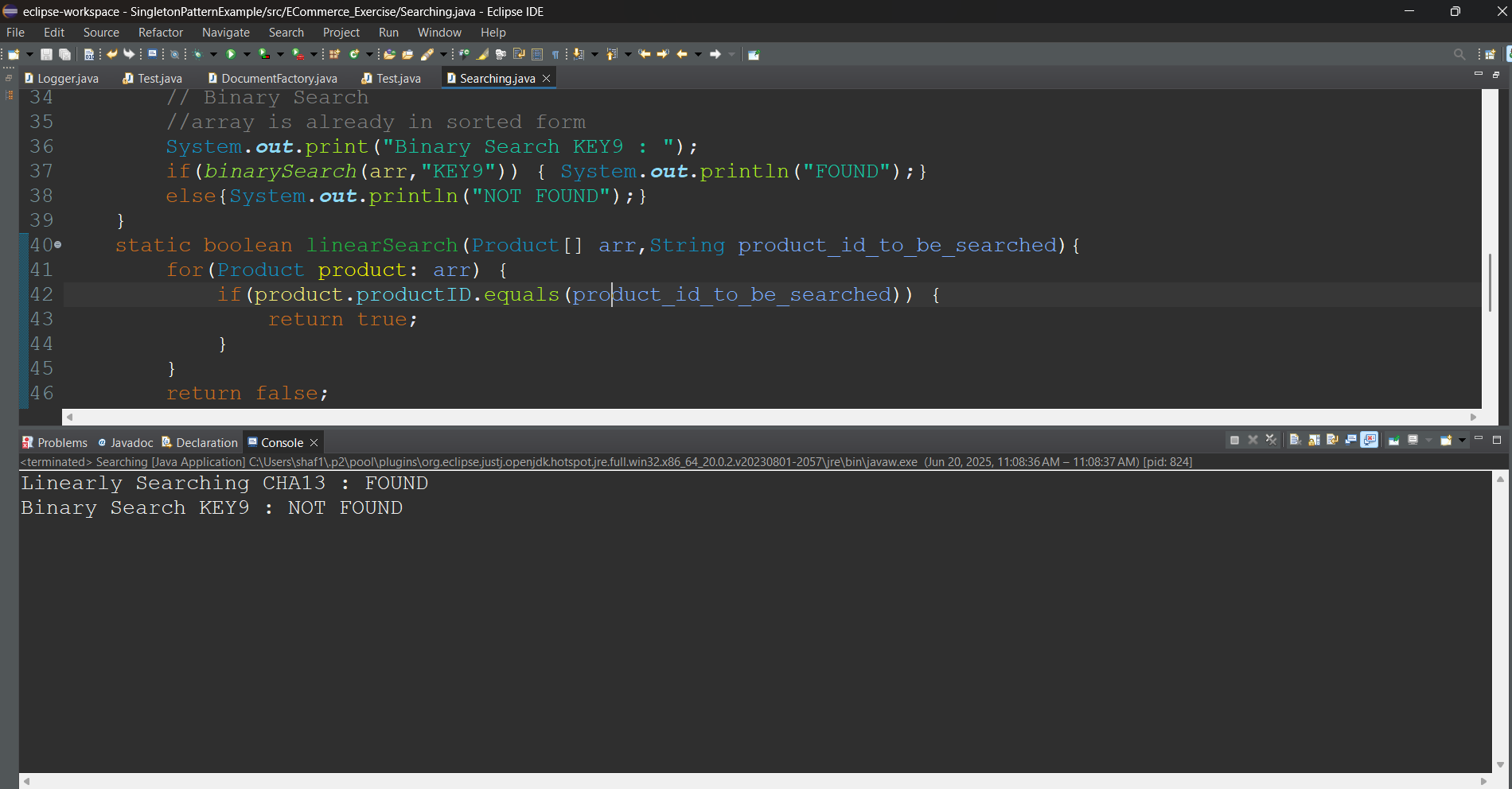
}

}

return false;

}

} Output:



**Exercise 7: Financial Forecasting**

* *Recursion is a programming technique where the function calls itself to solve smaller chunks or parts of same problem. It can solve problems where a task is repetitive in nature like Fibonacci , factorial etc.*

**Java code:**

Forecasting.java

package ForecastingProblem;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Forecasting {

public static void main(String[] args) {

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

int[] stocks = new int[10];

System.***out***.println("Predicting tomorrows stock values based on past 10 days stock prices: ");

System.***out***.println("Enter the stock prices: ");

for(int i=0;i<10;i++) {

stocks[i]=sc.nextInt();

}

List<String> list = *calculate*(stocks,1,0,0);

System.***out***.println(list);

}

public static List<String> calculate(int[] stocks,int i,int inc,int dec) {

if(i==stocks.length) {

List<String> list = new ArrayList<>();

if(inc>dec) {

list.add("You may see a increase in stock prices for tomorrow");

}else if(inc<dec) {

list.add("You may see a decrease in stock prices for tomorrow");

}else {

list.add("You may neither see a increase or a decrease in stock prices for tomorrow");

}

return list;

}

if(stocks[i]>stocks[i-1]) {

inc++;

return *calculate*(stocks,i+1,inc,dec);

}else if (stocks[i]<stocks[i-1]){

dec++;

return *calculate*(stocks,i+1,inc,dec);

}

return *calculate*(stocks,i+1,inc,dec);

}

} Output:

