**WEEK - 1**

**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

**Logger.java**

public class Logger {

private static Logger loggerObject;

private Logger() {

System.out.println("Logger instance created.");

}

public static Logger fetchLogger() {

if (loggerObject == null)

loggerObject = new Logger();

return loggerObject;

}

public void write(String info) {

System.out.println("INFO: " + info);

}

}

**SingletonPattern.java**

public class SingletonPattern {

public static void main(String[] args) {

Logger logOne = Logger.fetchLogger();

Logger logTwo = Logger.fetchLogger();

logOne.write("App started successfully.");

logTwo.write("User logged in.");

if (logOne == logTwo)

System.out.println("Only one Logger instance exists.");

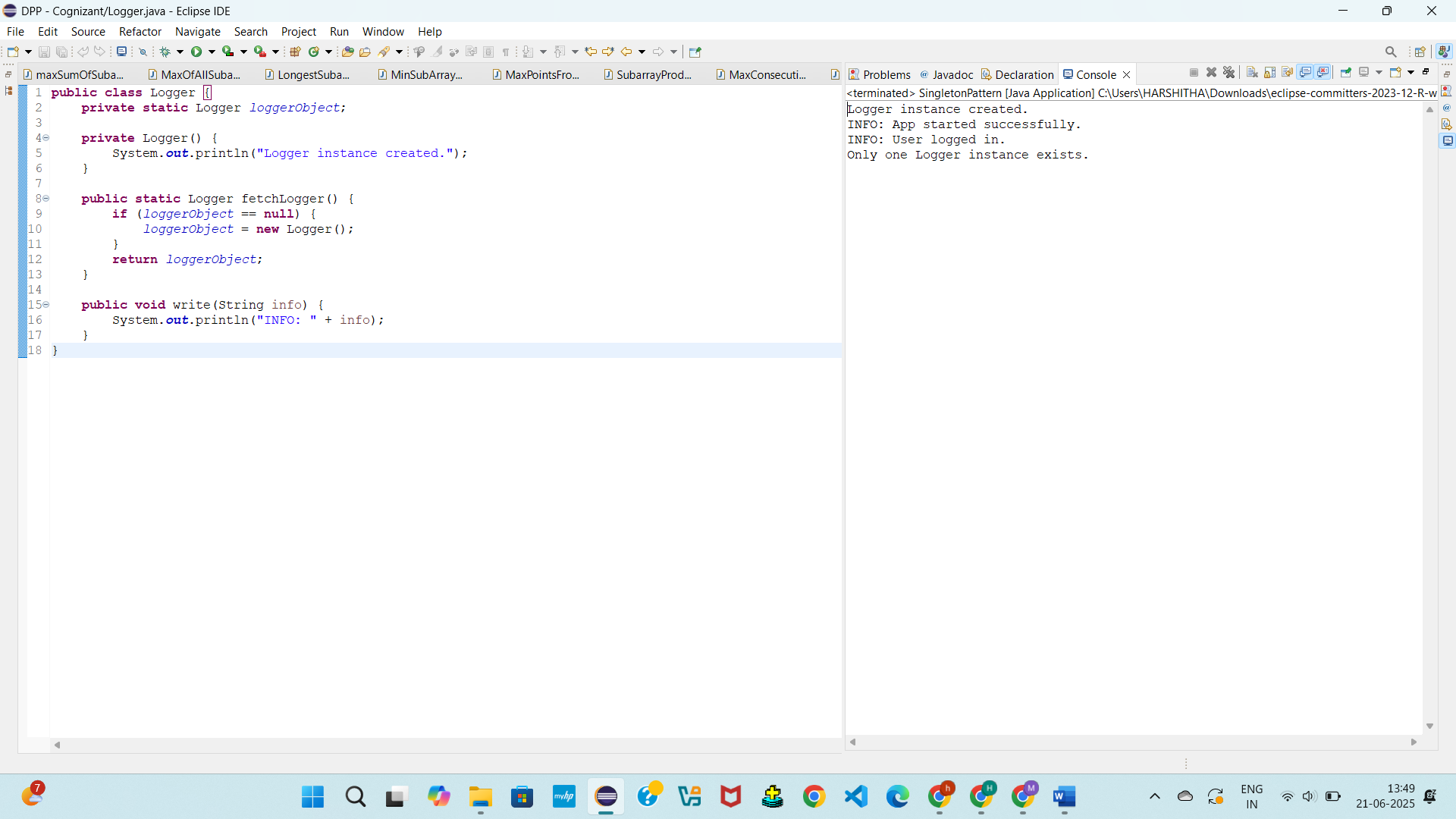
else

System.out.println("Multiple Logger instances found.");

}

}

**OUTPUT:**



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

**Document.java**

public interface Document {

void open();

}

**WordDocument.java**

public class WordDocument implements Document {

public void open() {

System.out.println("Opening a Word document...");

}

}

**PdfDocument.java**

public class PdfDocument implements Document {

public void open() {

System.out.println("Opening a PDF document...");

}

}

**ExcelDocument.java**

public class ExcelDocument implements Document {

public void open() {

System.out.println("Opening an Excel document...");

}

}

**DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**FactoryMethodPattern.java**

public class FactoryMethodPattern {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

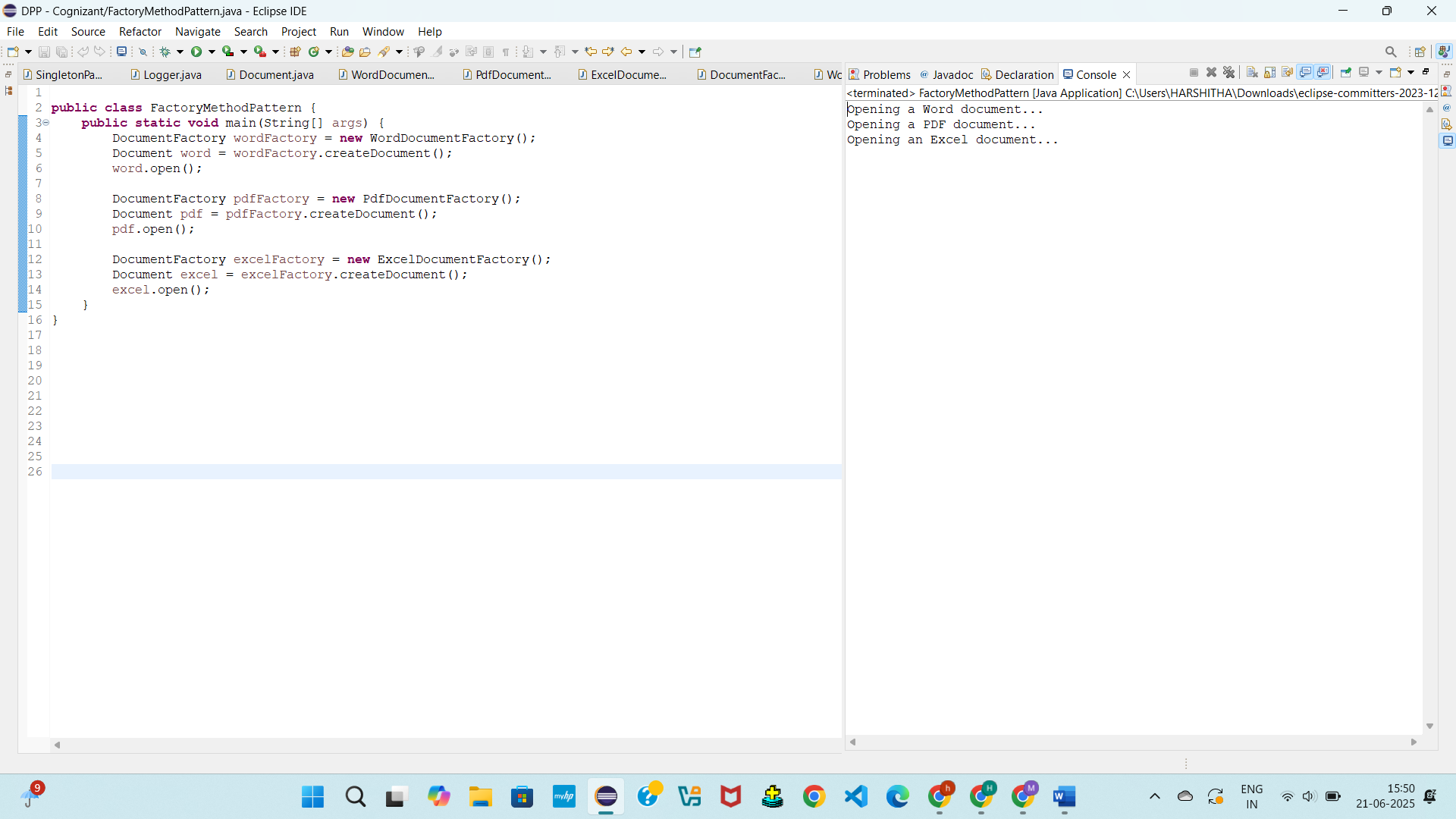
Document excel = excelFactory.createDocument();

excel.open();

}

}

**OUTPUT:**

****

**Data Structures and Algorithms**

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

import java.util.\*;

class Product {

int id;

String name;

String category;

Product(int id, String name, String category) {

this.id = id;

this.name = name;

this.category = category;

}

public String toString() {

return id + ": " + name + " (" + category + ")";

}

}

public class EcommerceSearch {

// Linear Search

public static Product linearSearch(Product[] items, String searchName) {

for (Product p : items) {

if (p.name.equals(searchName))

return p;

}

return null;

}

// Binary Search

public static Product binarySearch(Product[] items, String searchName) {

int low = 0, high = items.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int check = items[mid].name.compareTo(searchName);

if (check == 0) return items[mid];

else if (check < 0) low = mid + 1;

else high = mid - 1;

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Apple", "Food"),

new Product(102, "Shirt", "Clothing"),

new Product(103, "TV", "Electronics"),

new Product(104, "Book", "Stationery"),

new Product(105, "Phone", "Electronics")

};

// Linear Search Example

Product found1 = linearSearch(products, "Book");

if (found1 != null)

System.out.println("Linear Search: " + found1);

else

System.out.println("Linear Search: Product not found");

Arrays.sort(products, Comparator.comparing(p -> p.name.toLowerCase()));

// Binary Search Example

Product found2 = binarySearch(products, "Book");

if (found2 != null)

System.out.println("Binary Search: " + found2);

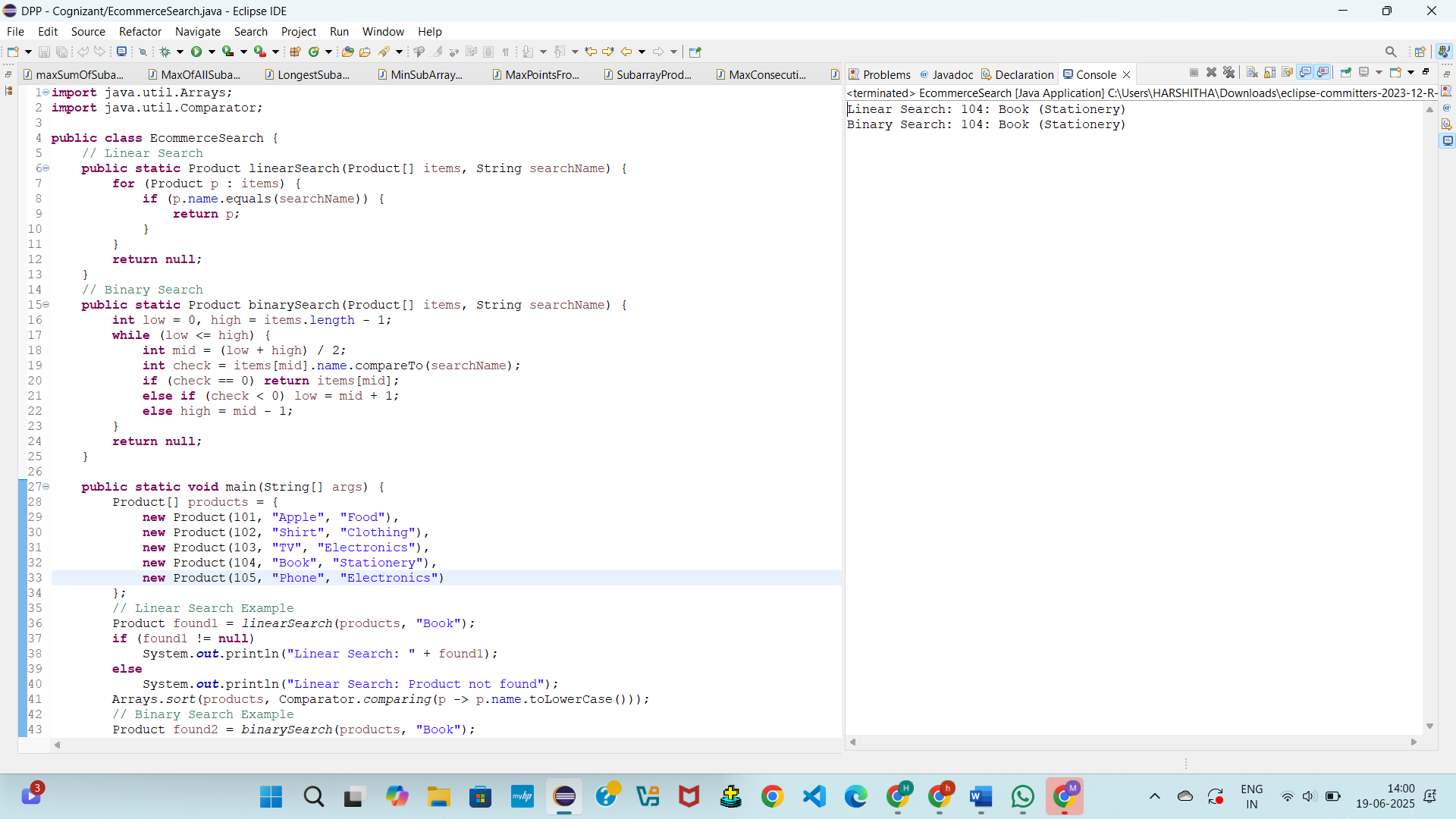
else

System.out.println("Binary Search: Product not found");

}

}

**OUTPUT:**



**Exercise 7: Financial Forecasting**

public class FinancialForecast {

public static int futureValue(int amount, double rate, int years) {

if (years == 0) {

System.out.println("Year 0: Rs." + amount);

return amount;

}

int previousValue = futureValue(amount, rate, years - 1);

int currentValue = (int)(previousValue \* (1 + rate));

System.out.println("Year " + years + ": Rs." + currentValue);

return currentValue;

}

public static void main(String[] args) {

int startingAmount = 1000;

double growthRate = 0.10;

int years = 3;

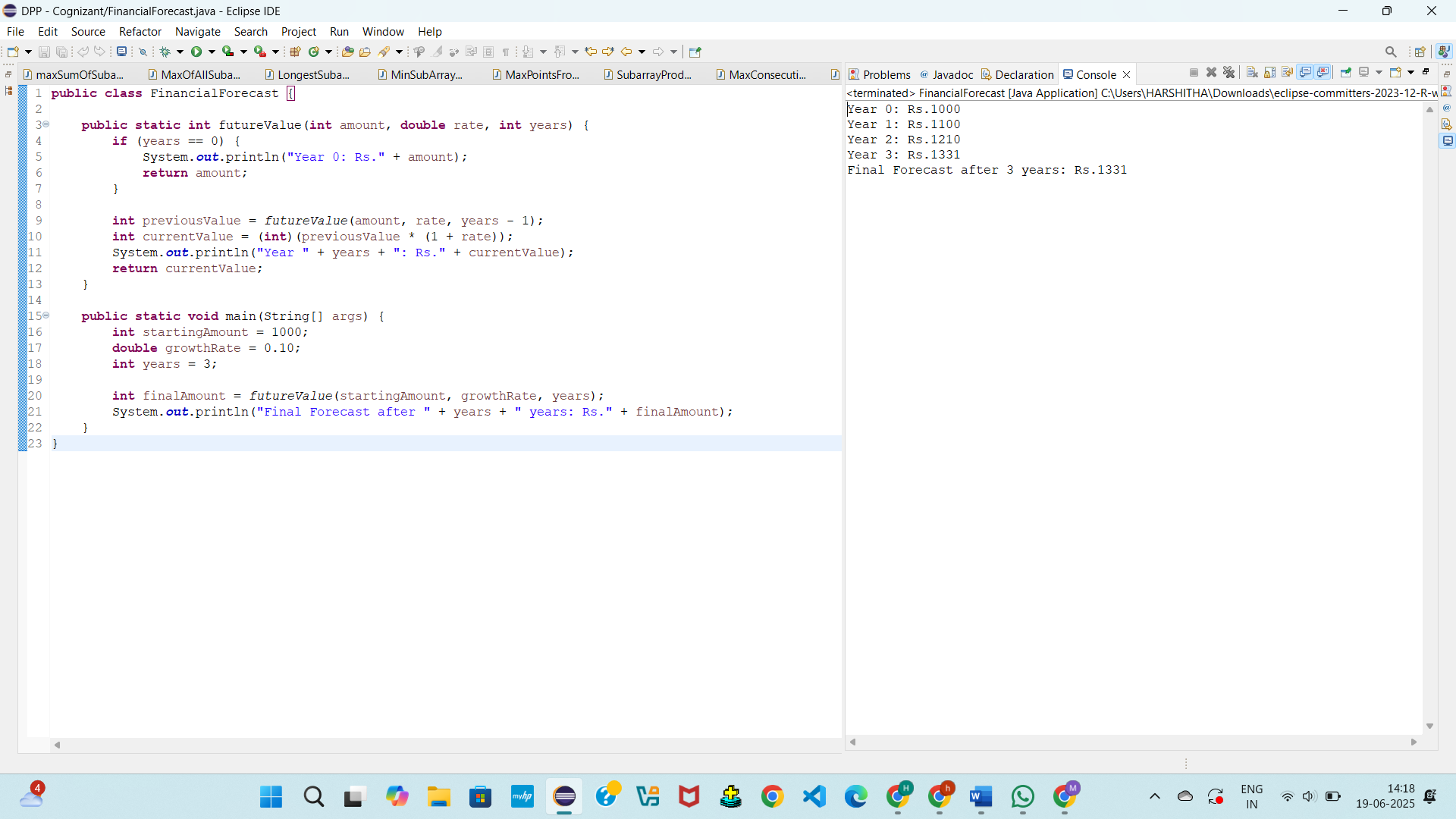
int finalAmount = futureValue(startingAmount, growthRate, years);

System.out.println("Final Forecast after " + years + " years: Rs." + finalAmount);

}

}

**OUTPUT:**

****