**WEEK 1**

**DESIGN PATTERN AND PRINCIPLES/DATA STRUCTURES AND ALGORITHMS**

**DESIGN PATTERN AND PRINCIPLES**

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Code:**

public class Main {

static class Logger {

private static Logger instance;

private Logger() {

System.out.println("Logger Initialized");

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("[LOG]: " + message);

}

}

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("Application started.");

logger2.log("This should use the same logger instance.");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (Singleton Verified).");

} else {

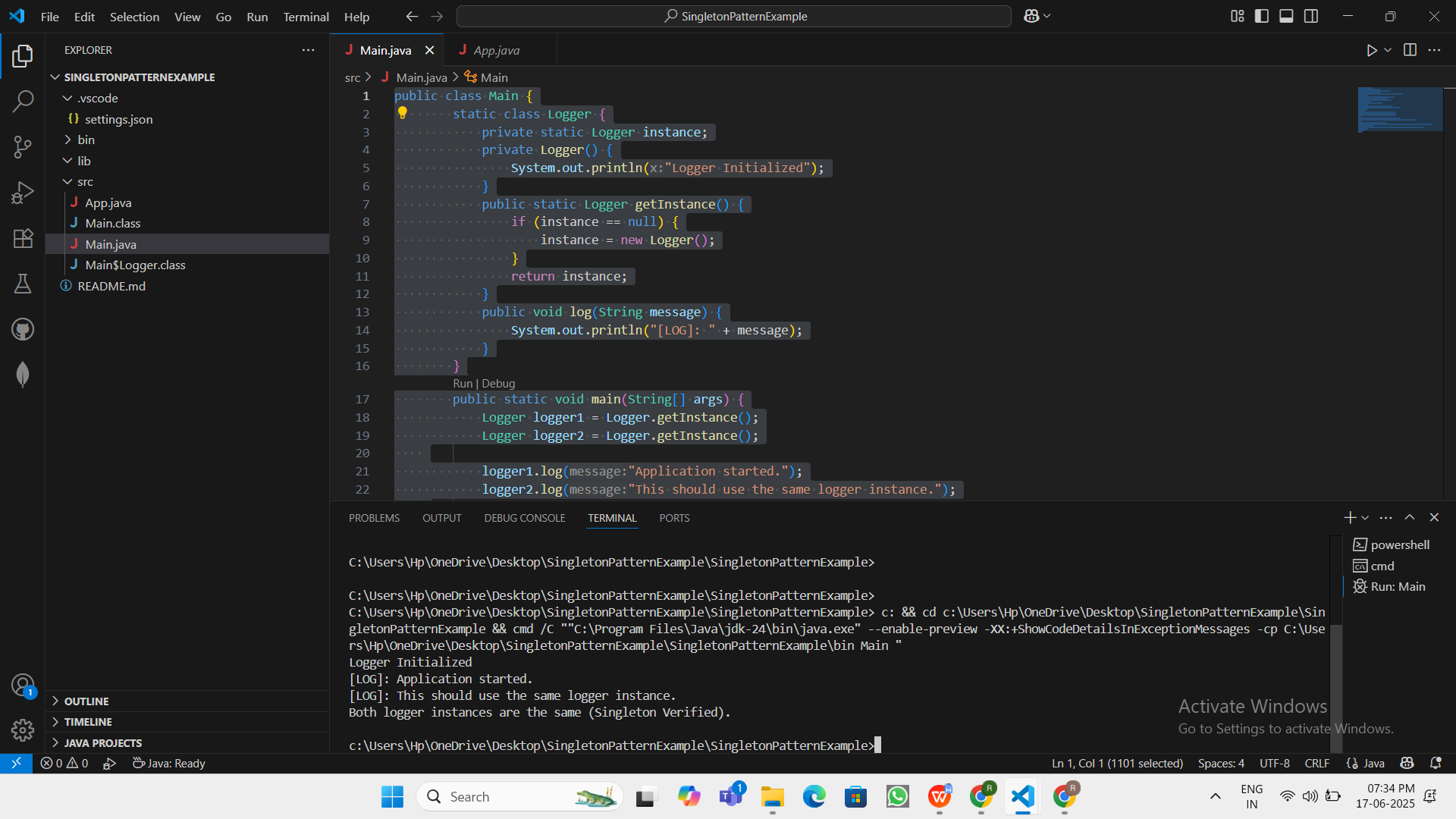
System.out.println("Different instances exist (Singleton Broken).");

}

}

}

**OUTPUT:**



Logger Initialized

[LOG]: Application started.

[LOG]: This should use the same logger instance.

Both logger instances are the same (Singleton Verified).

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

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**Code:**

public class Main {

interface Document {

void open();

}

static class WordDocument implements Document {

public void open() {

System.out.println("Opening Word Document...");

}

}

static class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF Document...");

}

}

static class ExcelDocument implements Document {

public void open() {

System.out.println("Opening Excel Document...");

}

}

abstract static class DocumentFactory {

public abstract Document createDocument();

}

static class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

static class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

static class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelFactory();

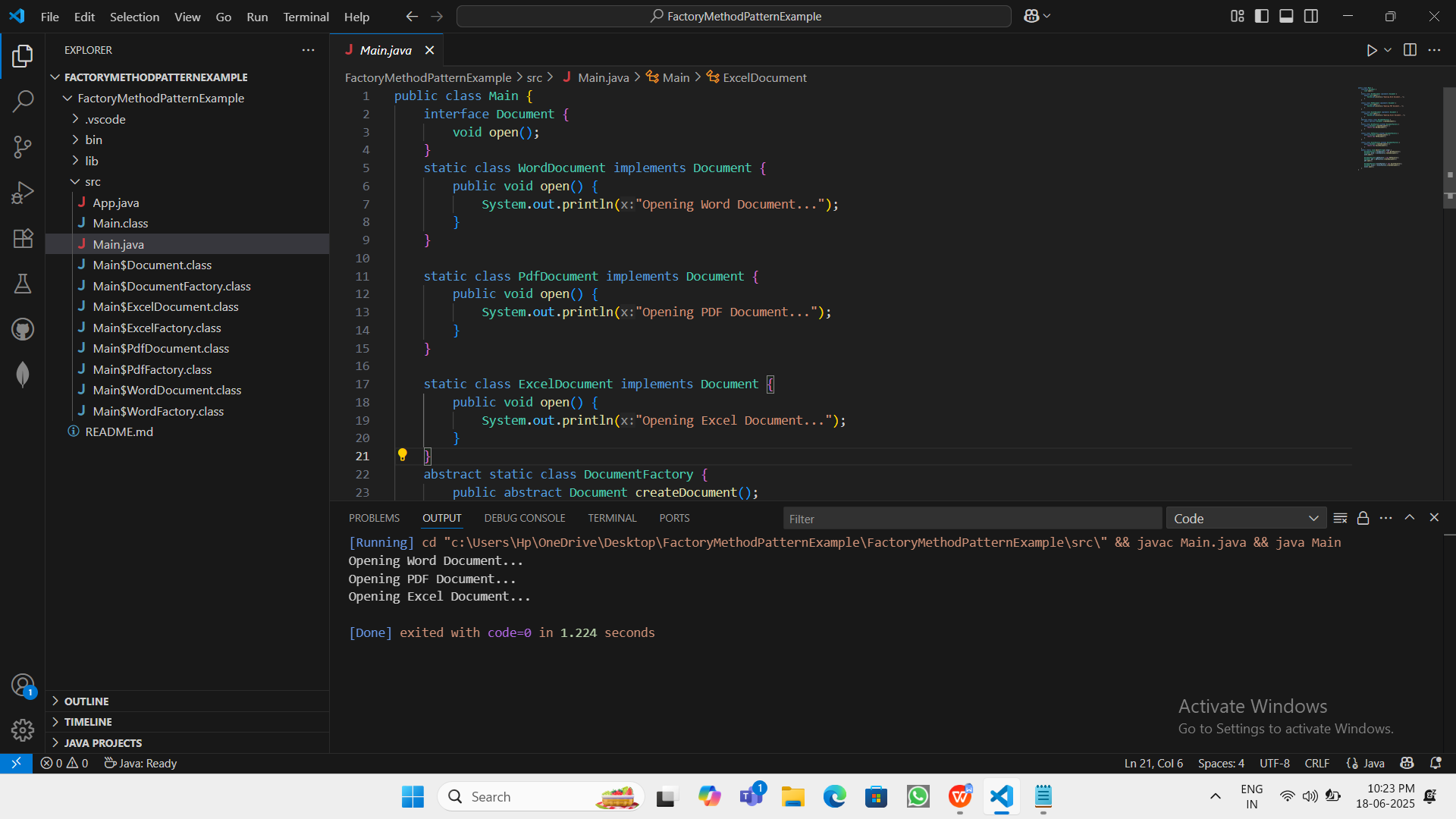
Document excel = excelFactory.createDocument();

excel.open();

}

}

**OUTPUT:**



Opening Word Document...

Opening PDF Document...

Opening Excel Document...

**DATA STRUCTURES AND ALGORITHMS**

**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.

**Code:**

import java.util.Arrays;

import java.util.Comparator;

public class Main {

static class Product {

int productId;

String productName;

String category;

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

this.productId = id;

this.productName = name;

this.category = category;

}

@Override

public String toString() {

return productId + " - " + productName + " (" + category + ")";

}

}

public static Product linearSearch(Product[] products, String name) {

for (Product p : products) {

if (p.productName.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

public static Product binarySearch(Product[] products, String name) {

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

int compare = products[mid].productName.compareToIgnoreCase(name);

if (compare == 0) {

return products[mid];

} else if (compare < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(102, "Shoes", "Fashion"),

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

new Product(104, "Watch", "Accessories"),

new Product(105, "Backpack", "Bags")

};

String searchTarget = "Phone";

System.out.println("🔍 Linear Search:");

Product foundLinear = linearSearch(products, searchTarget);

System.out.println(foundLinear != null ? "Found: " + foundLinear : "Product not found");

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

System.out.println("\n🔍 Binary Search (after sorting):");

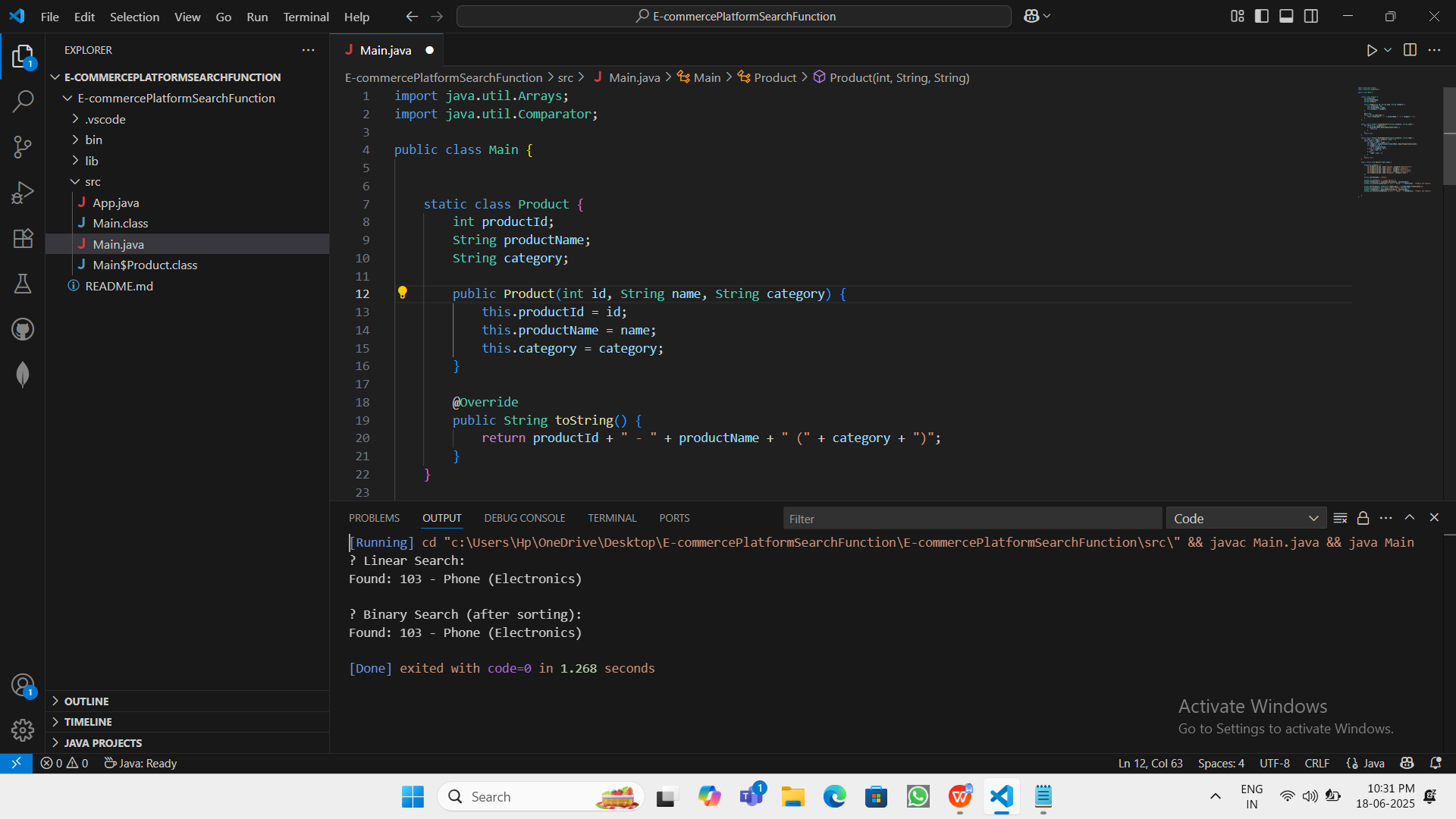
Product foundBinary = binarySearch(products, searchTarget);

System.out.println(foundBinary != null ? "Found: " + foundBinary : "Product not found");

}

}

**OUTPUT:**



? Linear Search:

Found: 103 - Phone (Electronics)

? Binary Search (after sorting):

Found: 103 - Phone (Electronics)

**Exercise 7: Financial Forecasting**

**Scenario:**

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

**Code:**

public class Main {

public static double forecast(double value, double growthRate, int years) {

if (years == 0) {

return value;

}

return forecast(value \* (1 + growthRate), growthRate, years - 1);

}

public static double forecastMemo(double value, double growthRate, int years, double[] memo) {

if (years == 0) {

return value;

}

if (memo[years] != 0) {

return memo[years];

}

memo[years] = forecastMemo(value \* (1 + growthRate), growthRate, years - 1, memo);

return memo[years];

}

public static void main(String[] args) {

double initialValue = 1000;

double growthRate = 0.10;

int years = 5;

double result = forecast(initialValue, growthRate, years);

System.out.println("Future Value (recursive): ₹" + result);

double[] memo = new double[years + 1];

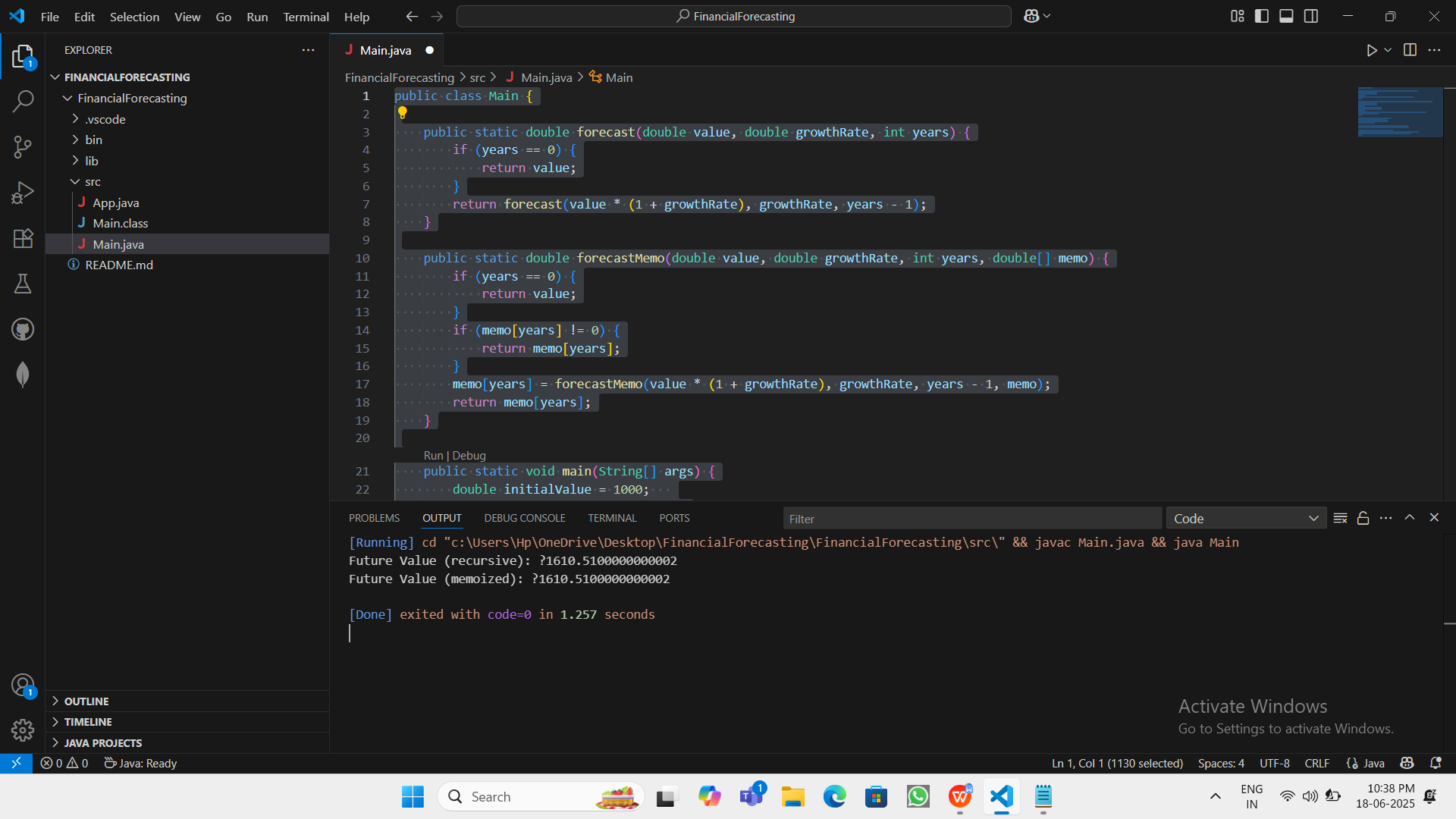
double resultMemo = forecastMemo(initialValue, growthRate, years, memo);

System.out.println("Future Value (memoized): ₹" + resultMemo);

}

}

**OUTPUT:**



Future Value (recursive): ?1610.5100000000002

Future Value (memoized): ?1610.5100000000002