**WEEK 1**

**DESIGN PATTERNS AND PRINCIPLES**

**Exercise 1: Implementing the Singleton Pattern**

**MY CODE:**

**class Logger {**

**private static Logger instance;**

**private Logger() {**

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

**}**

**public static Logger getInstance() {**

**if (instance == null) {**

**instance = new Logger();**

**}**

**return instance;**

**}**

**public void log(String message) {**

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

**}**

**}**

**public class TestSingleton {**

**public static void main(String[] args) {**

**Logger logger1 = Logger.getInstance();**

**Logger logger2 = Logger.getInstance();**

**logger1.log("First log message.");**

**logger2.log("Second log message.");**

**if (logger1 == logger2) {**

**System.out.println("Both logger1 and logger2 are the same instance.");**

**} else {**

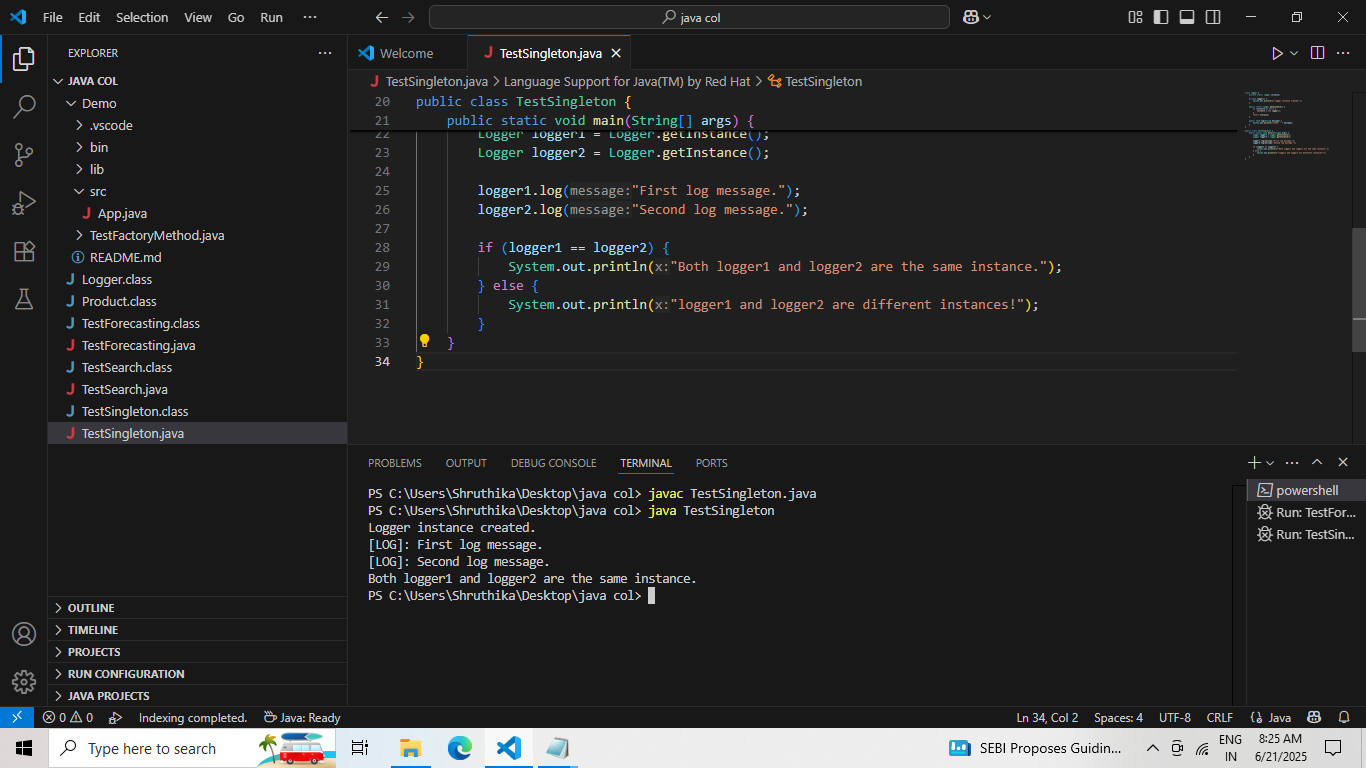
**System.out.println("logger1 and logger2 are different instances!");**

**}**

**}**

**}**

**OUTPUT:**

****

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

**MY CODE:**

**interface Document {**

**void open();**

**}**

**class WordDocument implements Document {**

**public void open() {**

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

**}**

**}**

**class PdfDocument implements Document {**

**public void open() {**

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

**}**

**}**

**class ExcelDocument implements Document {**

**public void open() {**

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

**}**

**}**

**abstract class DocumentFactory {**

**public abstract Document createDocument();**

**}**

**class WordDocumentFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new WordDocument();**

**}**

**}**

**class PdfDocumentFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new PdfDocument();**

**}**

**}**

**class ExcelDocumentFactory extends DocumentFactory {**

**public Document createDocument() {**

**return new ExcelDocument();**

**}**

**}**

**public class TestFactoryMethod {**

**public static void main(String[] args) {**

**DocumentFactory wordFactory = new WordDocumentFactory();**

**DocumentFactory pdfFactory = new PdfDocumentFactory();**

**DocumentFactory excelFactory = new ExcelDocumentFactory();**

**Document doc1 = wordFactory.createDocument();**

**Document doc2 = pdfFactory.createDocument();**

**Document doc3 = excelFactory.createDocument();**

**doc1.open();**

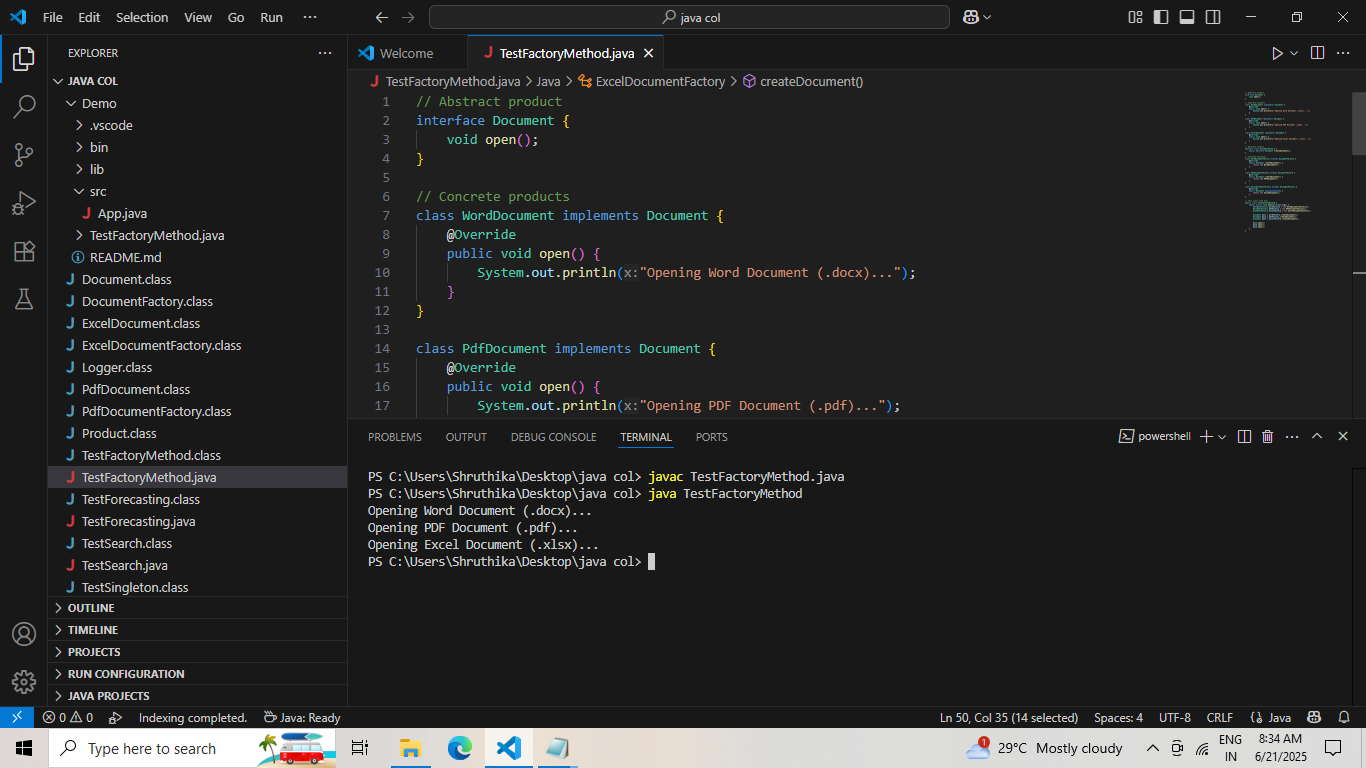
**doc2.open();**

**doc3.open();**

**}**

**}**

**OUTPUT:**

****

**ALGORITHMS AND DATA STRUCTURES**

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

**MY CODE:**

**class Product {**

**int productId;**

**String productName;**

**String category;**

**Product(int productId, String productName, String category) {**

**this.productId = productId;**

**this.productName = productName;**

**this.category = category;**

**}**

**public String toString() {**

**return "Product[ID=" + productId + ", Name=" + productName + ", Category=" + category + "]";**

**}**

**}**

**public class TestSearch {**

**public static Product linearSearch(Product[] products, int id) {**

**for (Product p : products) {**

**if (p.productId == id) return p;**

**}**

**return null;**

**}**

**public static Product binarySearch(Product[] products, int id) {**

**int left = 0;**

**int right = products.length - 1;**

**while (left <= right) {**

**int mid = (left + right) / 2;**

**if (products[mid].productId == id) return products[mid];**

**if (products[mid].productId < id) left = mid + 1;**

**else right = mid - 1;**

**}**

**return null;**

**}**

**public static void main(String[] args) {**

**Product[] products = {**

**new Product(102, "Keyboard", "Electronics"),**

**new Product(305, "Mug", "Home"),**

**new Product(201, "Headphones", "Electronics"),**

**new Product(450, "Chair", "Furniture")**

**};**

**System.out.println("=== Linear Search ===");**

**Product result = linearSearch(products, 201);**

**System.out.println(result != null ? result : "Product not found!");**

**Product[] sortedProducts = {**

**new Product(102, "Keyboard", "Electronics"),**

**new Product(201, "Headphones", "Electronics"),**

**new Product(305, "Mug", "Home"),**

**new Product(450, "Chair", "Furniture")**

**};**

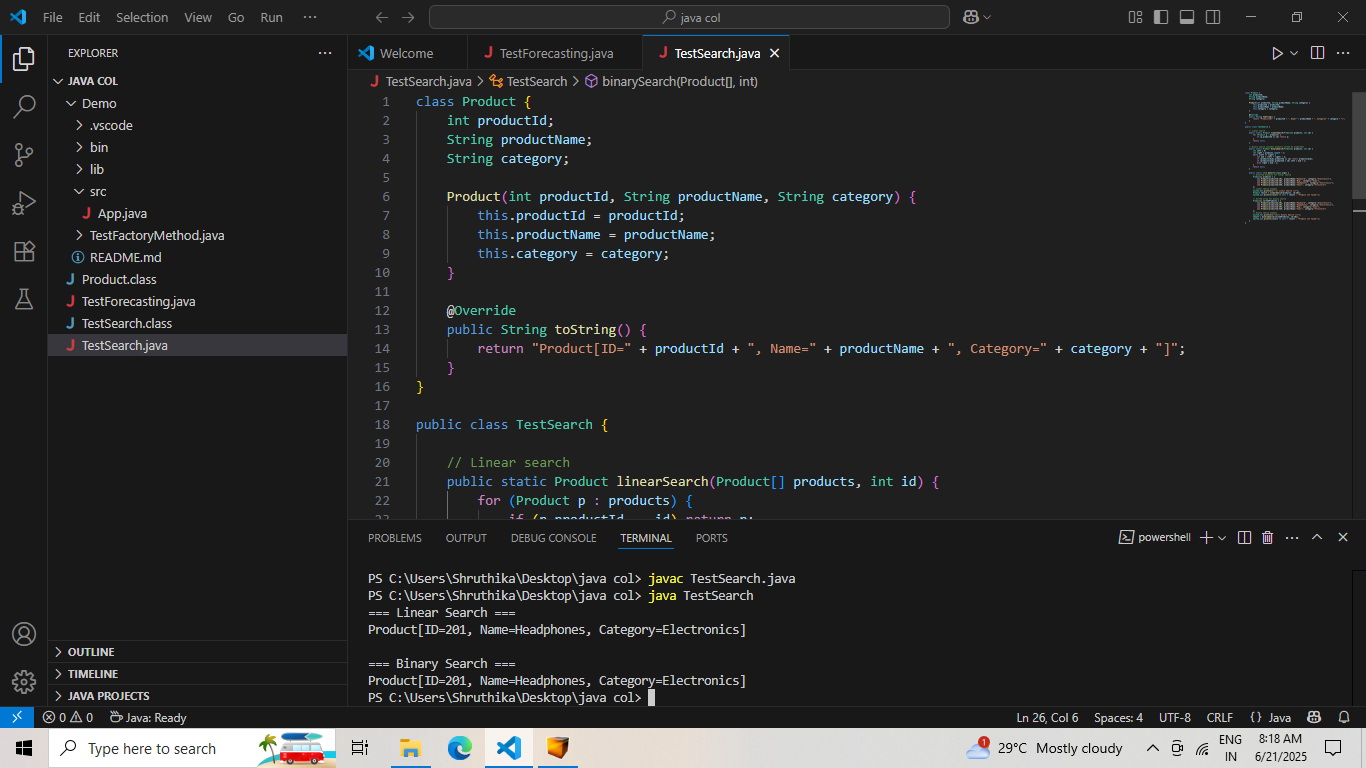
**System.out.println("\n=== Binary Search ===");**

**result = binarySearch(sortedProducts, 201);**

**System.out.println(result != null ? result : "Product not found!");**

**} }**

**OUTPUT:**

****

**Exercise 7: Financial Forecasting**

**MY CODE:**

**public class TestForecasting {**

**// Recursive future value calculator**

**public static double futureValue(double presentValue, double rate, int years) {**

**// Base case: no more years left**

**if (years == 0) {**

**return presentValue;**

**}**

**// Recursive case**

**return futureValue(presentValue, rate, years - 1) \* (1 + rate);**

**}**

**public static void main(String[] args) {**

**double presentValue = 1000.0; // starting value**

**double rate = 0.05; // 5% rate**

**int years = 5;**

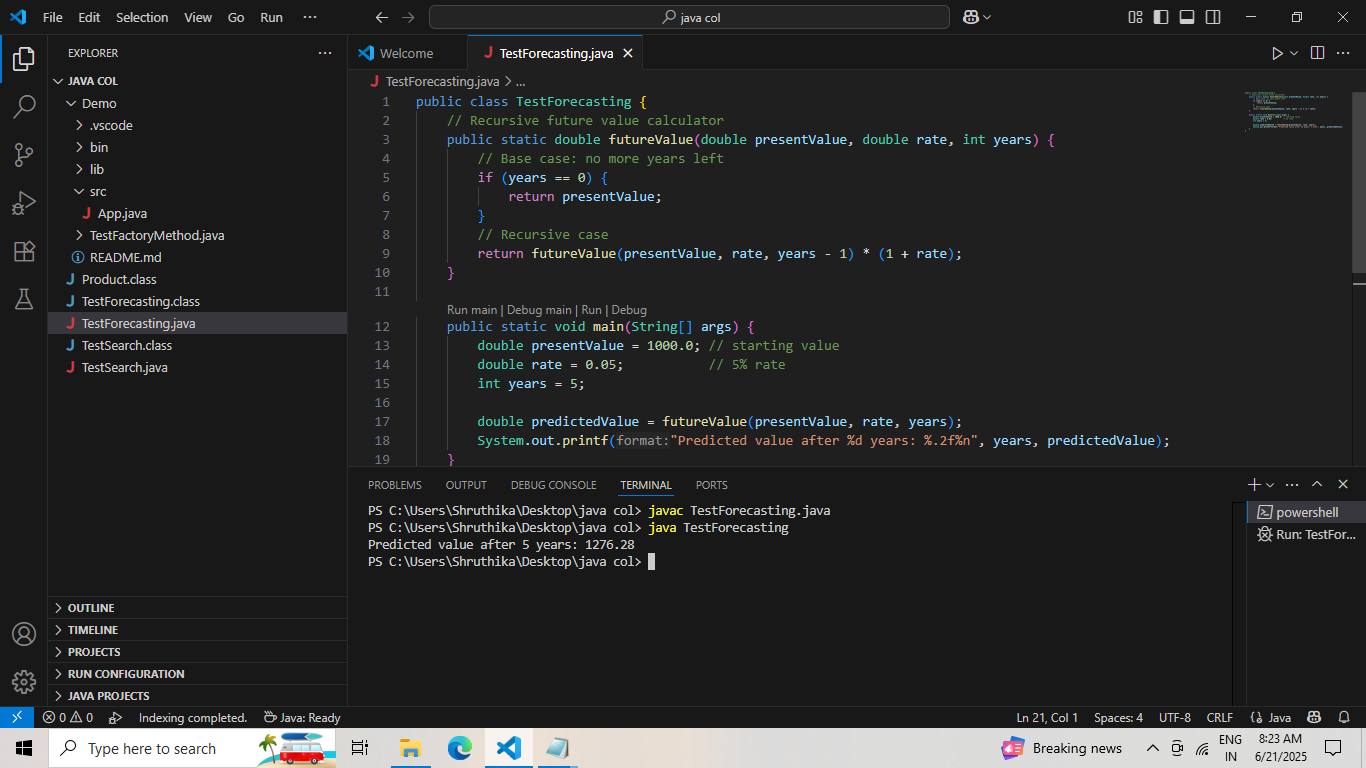
**double predictedValue = futureValue(presentValue, rate, years);**

**System.out.printf("Predicted value after %d years: %.2f%n", years, predictedValue);**

**}**

**}**

**OUTPUT:**

****