**Week-1 Design Patterns and Principles**

1. Implementing the Singleton:

Code:

public class Main {

static 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 static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("First message");

logger2.log("Second message");

if (logger1 == logger2) {

System.out.println("Both are same. Singleton works");

} else {

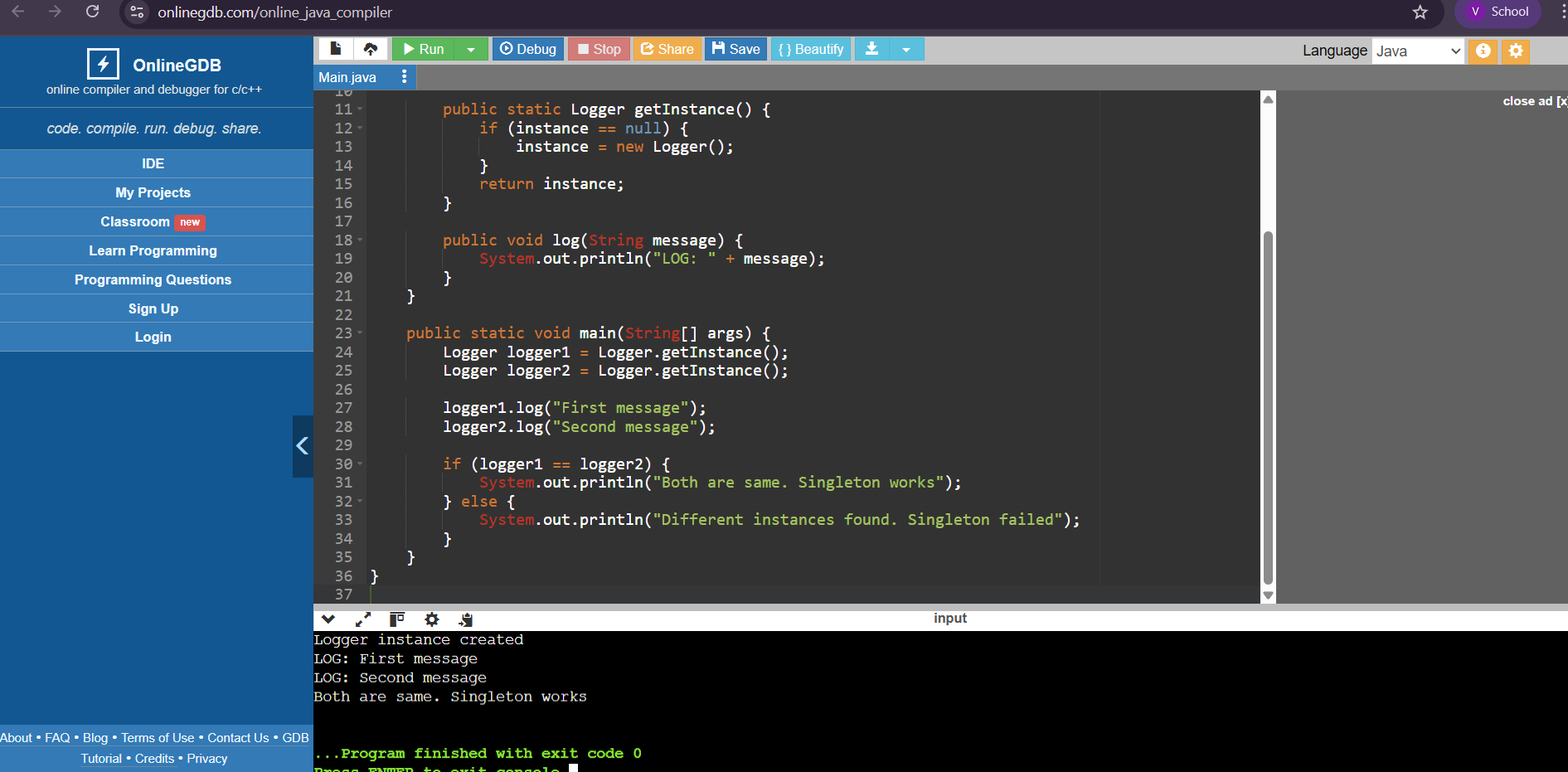
System.out.println("Different instances found. Singleton failed");

}

}

}

OUTPUT:



1. Implementing the Factory Method Pattern :

CODE:

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

class ExcelDocument implements Document {

public void open() {

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

}

}

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 Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

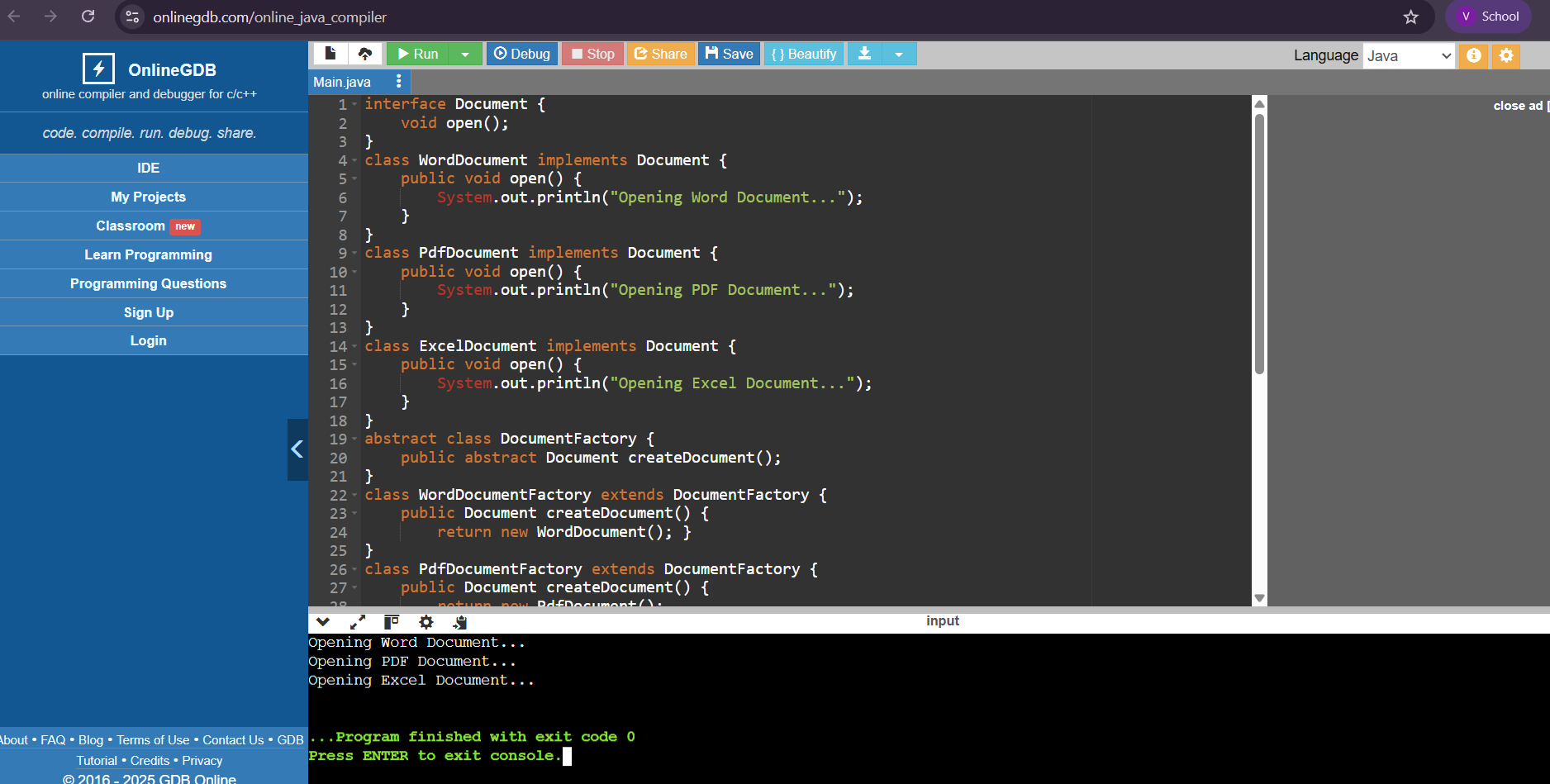
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

OUTPUT:



1. E-Commerce Platform Search Function

CODE:

import java.util.\*;

class Product {

private int id;

private String name;

private String category;

private double price;

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

this.id = id;

this.name = name.toLowerCase();

this.category = category.toLowerCase();

this.price = price;}

public String getName() { return name; }

public String getCategory() { return category; }

public double getPrice() { return price; }

@Override

public String toString() {

return "Product{" + "id=" + id + ", name='" + name + '\'' +

", category='" + category + '\'' + ", price=" + price + '}'; }

}

class ProductRepository {

private List<Product> products = new ArrayList<>();

public ProductRepository() {

products.add(new Product(1, "iPhone 14", "Electronics", 69999));

products.add(new Product(2, "Samsung Galaxy S21", "Electronics", 49999));

products.add(new Product(3, "Nike Running Shoes", "Footwear", 3299));

products.add(new Product(4, "Adidas Sports Shoes", "Footwear", 3899));

products.add(new Product(5, "Sony Headphones", "Electronics", 2599));

products.add(new Product(6, "The Alchemist Book", "Books", 399));

}

public List<Product> getAllProducts() {

return products;

}

}

class SearchEngine {

private ProductRepository repository;

public SearchEngine(ProductRepository repository) {

this.repository = repository;

}

public List<Product> search(String keyword) {

keyword = keyword.toLowerCase();

List<Product> results = new ArrayList<>();

for (Product p : repository.getAllProducts()) {

if (p.getName().contains(keyword) || p.getCategory().contains(keyword)) {

results.add(p);

}

}

return results; }

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

ProductRepository repository = new ProductRepository();

SearchEngine engine = new SearchEngine(repository);

System.out.print("Enter search keyword: ");

String keyword = scanner.nextLine();

List<Product> results = engine.search(keyword);

if (results.isEmpty()) {

System.out.println("No products found for: " + keyword);

} else {

System.out.println("Search Results:");

for (Product p : results) {

System.out.println(p);

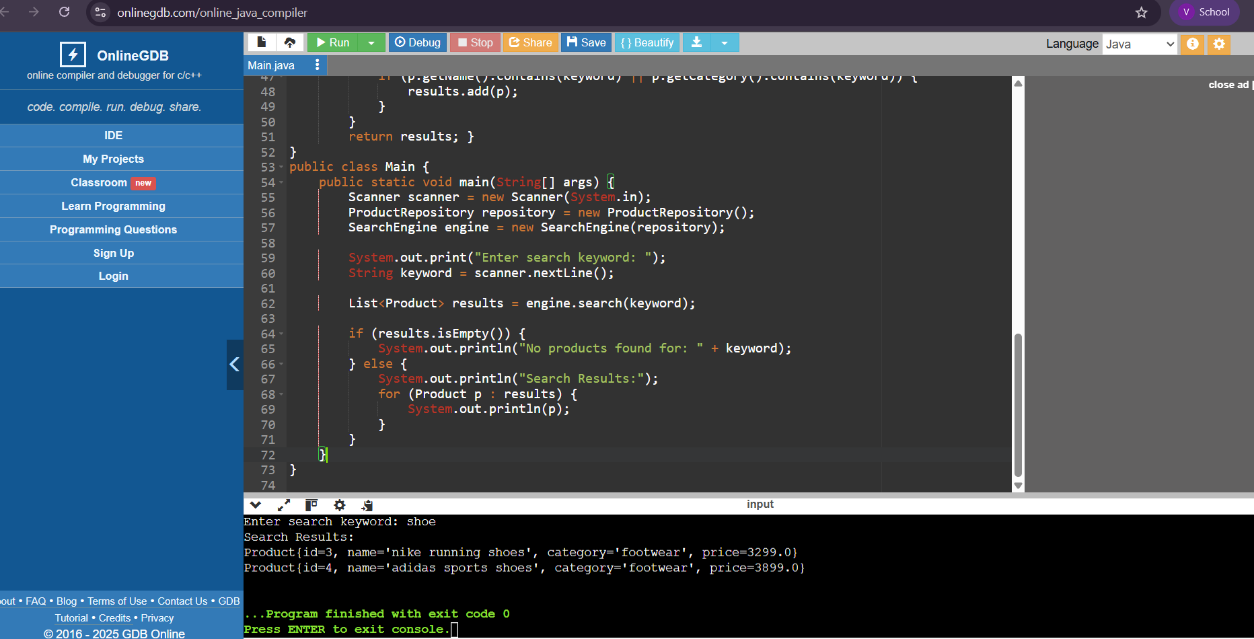
}

}

}

}

OUTPUT:



1. Financial Forecasting

CODE:

import java.util.HashMap;

import java.util.Map;

public class Main {

public static double forecastFutureValue(double currentValue, double growthRate, int years) {

if (years == 0) {

return currentValue;

}

return forecastFutureValue(currentValue \* (1 + growthRate), growthRate, years - 1);

}

public static double forecastMemo(double currentValue, double growthRate, int years, Map<Integer, Double> memo) {

if (years == 0) {

return currentValue;

}

if (memo.containsKey(years)) {

return memo.get(years);

}

double result = forecastMemo(currentValue \* (1 + growthRate), growthRate, years - 1, memo);

memo.put(years, result);

return result;

}

public static void main(String[] args) {

double currentValue = 10000;

double growthRate = 0.07;

int years = 5;

System.out.println("=== Financial Forecasting (Recursive) ===");

double futureValue = forecastFutureValue(currentValue, growthRate, years);

System.out.printf("Future Value (after %d years): ₹%.2f%n", years, futureValue);

System.out.println("\n=== Optimized Version with Memoization ===");

Map<Integer, Double> memo = new HashMap<>();

double optimizedValue = forecastMemo(currentValue, growthRate, years, memo);

System.out.printf("Memoized Future Value (after %d years): ₹%.2f%n", years, optimizedValue);

System.out.println("\n=== Analysis ===");

System.out.println("Memoization avoids redundant calculations in recursive chains.");

}

}

OUTPUT:

