## **Week1**

### **Design Patterns and Principles**

**Exercise1: Implementing a Singleton Pattern**

**public class SingletonPatternExample {**

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

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

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

**System.out.println("Are both logger references the same? " + (logger1 == logger2));**

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

**logger1.showLogs();**

**}**

**}**

**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;**

**}**

**private StringBuilder logs = new StringBuilder();**

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

**logs.append("[LOG] ").append(message).append("\n");**

**}**

**public void showLogs() {**

**System.out.println("\n--- Log History ---");**

**System.out.print(logs.toString());**

**}**

**}**

### **OUTPUT:**

**Exercise2: Implementing the Factory Method Pattern**

public class FactoryMethodPatternExample {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document wordDoc = wordFactory.createDocument();

Document pdfDoc = pdfFactory.createDocument();

Document excelDoc = excelFactory.createDocument();

wordDoc.open();

pdfDoc.save();

excelDoc.edit();

System.out.println("\nDocument Types:");

System.out.println("Word doc is: " + wordDoc.getType());

System.out.println("PDF doc is: " + pdfDoc.getType());

System.out.println("Excel doc is: " + excelDoc.getType());

}

}

interface Document {

void open();

void save();

void edit();

String getType();

}

class WordDocument implements Document {

public void open() {

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

}

public void save() {

System.out.println("Saving Word document...");

}

public void edit() {

System.out.println("Editing Word document...");

}

public String getType() {

return "Microsoft Word";

}

}

class PdfDocument implements Document {

public void open() {

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

}

public void save() {

System.out.println("Saving PDF document...");

}

public void edit() {

System.out.println("Editing PDF document...");

}

public String getType() {

return "PDF";

}

}

class ExcelDocument implements Document {

public void open() {

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

}

public void save() {

System.out.println("Saving Excel document...");

}

public void edit() {

System.out.println("Editing Excel spreadsheet...");

}

public String getType() {

return "Microsoft Excel";

}

}

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();

}

}

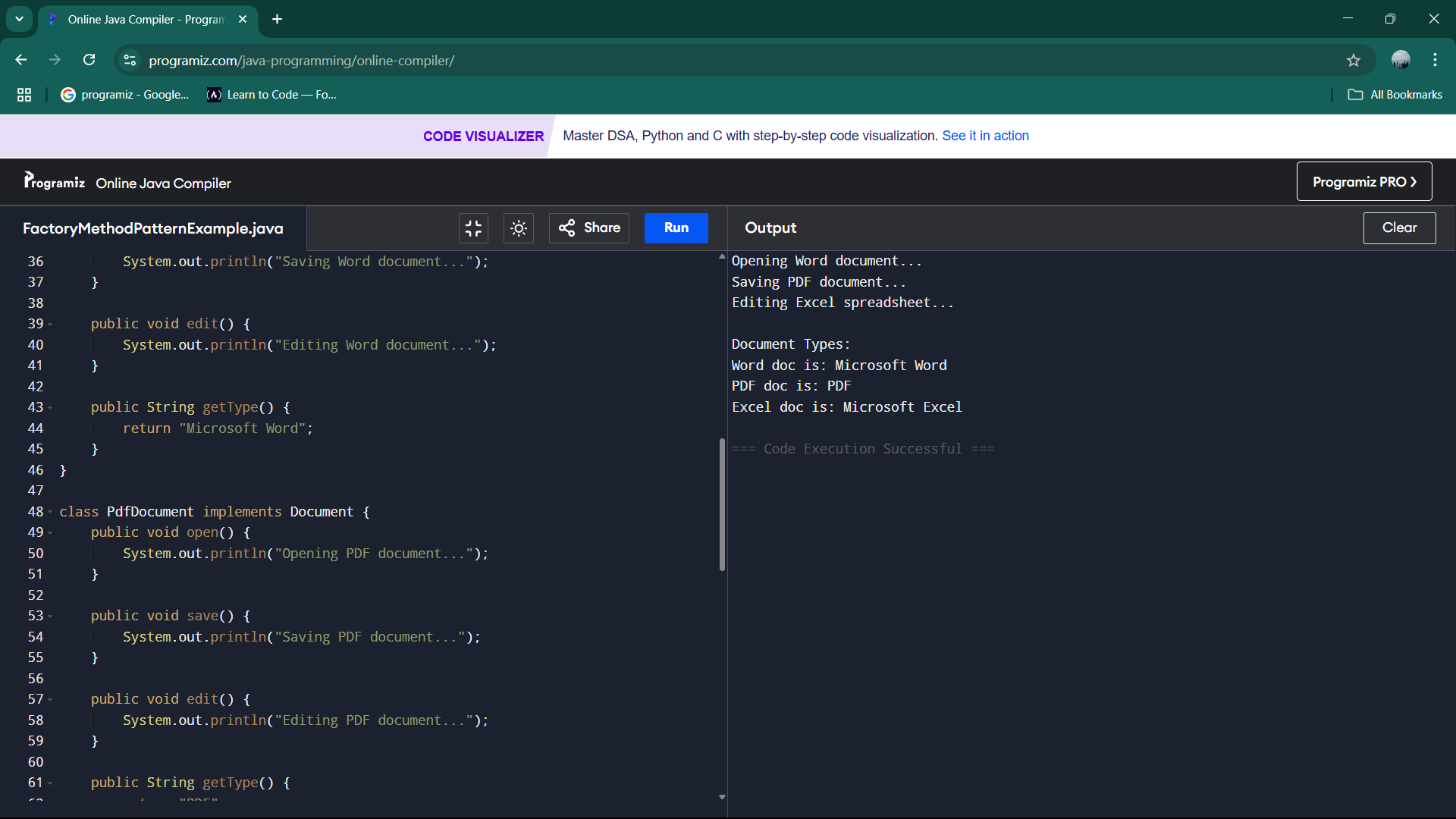
class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**OUTPUT:** 

### **DataStructures And Algorithms**

**Exercise2: E-commerce Platform Search Functions**

import java.util.Arrays;

class Product {

int productId;

String productName;

String category;

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

productId = id;

productName = name;

category = cat;

}

}

public class EcommerceSearch {

public static void main(String[] args) {

Product[] unsortedProducts = {

new Product(101, "Running Shoes", "Footwear"),

new Product(203, "Wireless Mouse", "Electronics"),

new Product(150, "Coffee Mug", "Kitchen"),

new Product(300, "Desk Lamp", "Furniture"),

new Product(275, "Yoga Mat", "Fitness")

};

Product[] sortedProducts = Arrays.copyOf(unsortedProducts, unsortedProducts.length);

Arrays.sort(sortedProducts, (a, b) -> a.productId - b.productId);

int searchId = 300;

System.out.println("Searching for product ID: " + searchId);

System.out.println("\n--- Linear Search ---");

int linearResult = linearSearch(unsortedProducts, searchId);

if(linearResult != -1) {

System.out.println("Found: " + unsortedProducts[linearResult].productName);

} else {

System.out.println("Product not found");

}

// Binary search on sorted array

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

int binaryResult = binarySearch(sortedProducts, searchId);

if(binaryResult != -1) {

System.out.println("Found: " + sortedProducts[binaryResult].productName);

} else {

System.out.println("Product not found");

}

}

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

for(int i = 0; i < products.length; i++) {

System.out.println("Checking product #" + products[i].productId);

if(products[i].productId == targetId) {

return i; // Found at index i

}

}

return -1; // Not found

}

public static int binarySearch(Product[] sortedProducts, int targetId) {

int low = 0;

int high = sortedProducts.length - 1;

while(low <= high) {

int mid = (low + high) / 2;

System.out.println("Checking product #" + sortedProducts[mid].productId);

if(sortedProducts[mid].productId == targetId) {

return mid; // Found at mid index

} else if(sortedProducts[mid].productId < targetId) {

low = mid + 1;

} else {

high = mid - 1;

}

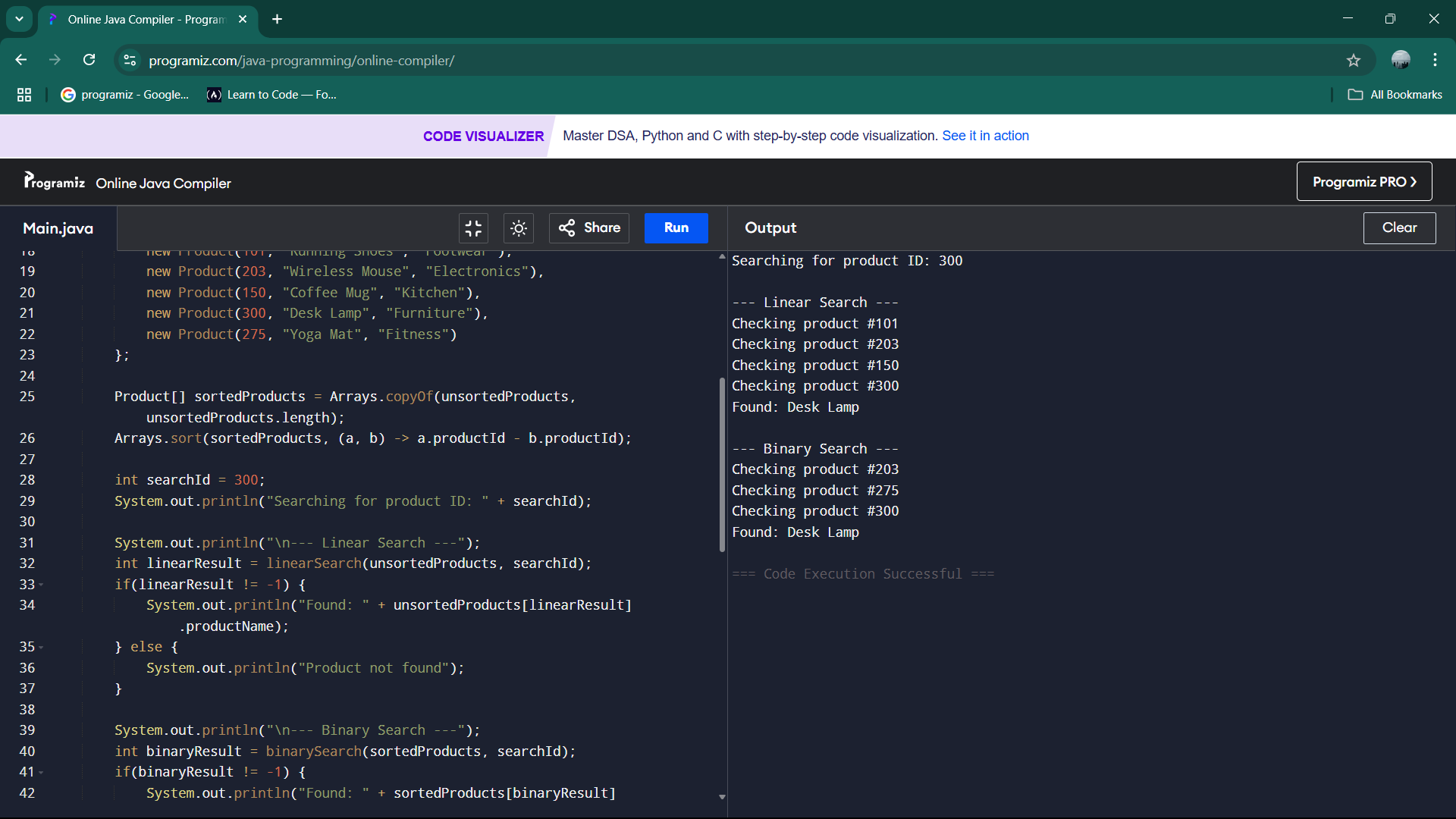
}

return -1;

}

}

**OUTPUT:**



**Exercise7: Financial Forecasting**

**public class FinancialForecast {**

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

**double startAmount = 10000.0;**

**double growthPercent = 5.0;**

**int years = 10;**

**double futureValue = calculateFutureValue(startAmount, growthPercent, years);**

**System.out.println("Starting Amount: $" + startAmount);**

**System.out.println("Growth Rate: " + growthPercent + "% per year");**

**System.out.println("Years: " + years);**

**System.out.printf("Future Value: $%.2f", futureValue);**

**}**

**public static double calculateFutureValue(double amount, double rate, int yearsLeft) {**

**if(yearsLeft == 0) {**

**return amount;**

**}**

**double nextYearAmount = amount \* (1 + rate/100.0);**

**return calculateFutureValue(nextYearAmount, rate, yearsLeft - 1);**

**}**

**}**

**OUTPUT:**

