**Design Patterns and Principles**

1. **Implementing the Singleton Pattern**

**Logger.java**

package example;

public 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);

}

}

**Main.java**

package example;

public class Main {

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("Same instance – Singleton works!");

} else {

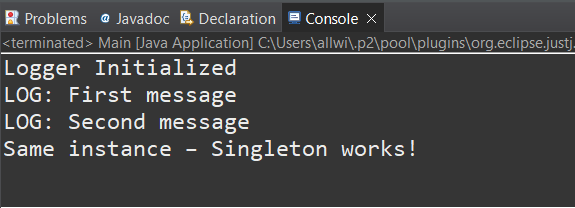
System.out.println("Different instances – Singleton failed!");

}

}

}

**Output**



1. **Implementing the Factory Method Pattern**

**Document.java**

package documentfactory;

public interface Document {

void open();

}

**DocumentFactory.java**

package documentfactory;

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**ExcelDocument.java**

package documentfactory;

public class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

**ExcelDocumentFactory.java**

package documentfactory;

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

**PdfDocument.java**

package documentfactory;

public class PdfDocument implements Document {

@Override

public void open() {

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

}

}

**PdfDocumentFactory.java**

package documentfactory;

public class WordDocument implements Document {

@Override

public void open() {

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

}}

**WordDocument.java**

package documentfactory;

public class WordDocument implements Document {

@Override

public void open() {

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

}}

**WordDocumentFactory.java**

package documentfactory;

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

**Main.java**

package documentfactory;

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();

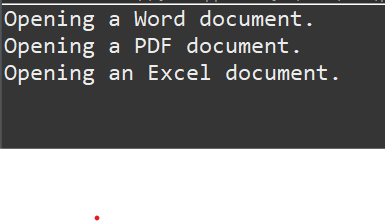
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output**



**Algorithms\_Data Structures**

1. **E-commerce Platform Search Function**

**Product.java**

package ecommerce;

public class Product {

int productId;

String productName;

String category;

public 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 + "]";

}

}

**SearchDemo.java**

package ecommerce;

import java.util.Arrays;

import java.util.Comparator;

public class SearchDemo {

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

for (Product product : products) {

if (product.productName.equalsIgnoreCase(targetName)) {

return product;

}

}

return null;

}

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

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

while (left <= right) {

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

int cmp = products[mid].productName.compareToIgnoreCase(targetName);

if (cmp == 0)

return products[mid];

else if (cmp < 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, "Mobile", "Electronics"),

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

new Product(105, "Bag", "Fashion")

};

Product foundLinear = linearSearch(products, "Watch");

System.out.println("Linear Search Result: " + (foundLinear != null ? foundLinear : "Not Found"));

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

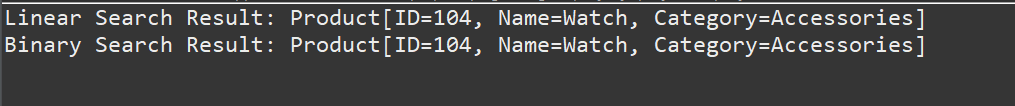
Product foundBinary = binarySearch(products, "Watch");

System.out.println("Binary Search Result: " + (foundBinary != null ? foundBinary : "Not Found"));

}

}

**Output**



1. **Financial Forecasting**

**FinancialForecast.java**

package forecast;

public class FinancialForecast {

public static double forecast(double currentValue, double rate, int years) {

if (years == 0) {

return currentValue;

}

return forecast(currentValue, rate, years - 1) \* (1 + rate);

}

public static void main(String[] args) {

double currentValue = 10000;

double growthRate = 0.10;

int forecastYears = 5;

double futureValue = forecast(currentValue, growthRate, forecastYears);

System.out.printf("Predicted value after %d years: ₹%.2f\n", forecastYears, futureValue);

}

}

**Output**

