**EXERCISE 1:**

**To define the singleton class**

public 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);

}

}

**Implémentation singleton class**

public class LoggerTest {

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 instances are the same. Singleton works!");

} else {

System.out.println("Instances are different. Singleton failed.");

}

}

}

**EXERCISE 2:**

public class FactoryMethodPatternExample {

interface Document {

void open();

}

static class WordDocument implements Document {

public void open() {

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

}

}

static class PdfDocument implements Document {

public void open() {

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

}

}

static class ExcelDocument implements Document {

public void open() {

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

}

}

static abstract 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 wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**EXERCISE 3:**

import java.util.Arrays;

import java.util.Comparator;

public class EcommerceSearchDemo {

static 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 "[" + productId + "] " + productName + " (" + category + ")";

}

}

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

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

static Product binarySearch(Product[] products, int targetId) {

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

while (left <= right) {

int mid = (left + right) / 2;

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

return products[mid];

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

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product(105, "Smartphone", "Electronics"),

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

new Product(103, "Shoes", "Fashion"),

new Product(102, "Backpack", "Travel"),

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

};

System.out.println("🔎 LINEAR SEARCH (on unsorted array):");

Product resultLinear = linearSearch(products, 103);

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

System.out.println("\n📚 Sorting products by productId for binary search...");

Arrays.sort(products, Comparator.comparingInt(p -> p.productId));

System.out.println("\n🔎 BINARY SEARCH (on sorted array):");

Product resultBinary = binarySearch(products, 103);

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

System.out.println("\n📊 Time Complexity Summary:");

System.out.println("Linear Search: O(n) — good for small or unsorted datasets.");

System.out.println("Binary Search: O(log n) — much faster, but requires sorted data.");

}

}

Binary search is better for performance especially in large-scale applications like an e-commerce platform if you can keep your data sorted. Otherwise, linear search is more flexible for smaller or unsorted datasets.

**EXERCISE 4:**

public class FinancialForecastTool {

static double calculateFutureValueRecursive(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

}

return calculateFutureValueRecursive(presentValue, growthRate, years - 1) \* (1 + growthRate);

}

static double calculateFutureValueIterative(double presentValue, double growthRate, int years) {

double futureValue = presentValue;

for (int i = 1; i <= years; i++) {

futureValue \*= (1 + growthRate);

}

return futureValue;

}

public static void main(String[] args) {

double presentValue = 15000;

double annualGrowthRate = 0.08;

int years = 6;

System.out.println("📈 Financial Forecasting Tool");

System.out.println("Present Value: ₹" + presentValue);

System.out.println("Growth Rate: " + (annualGrowthRate \* 100) + "% per year");

System.out.println("Forecast Period: " + years + " years\n");

double recursiveResult = calculateFutureValueRecursive(presentValue, annualGrowthRate, years);

System.out.printf("🔁 Recursive Prediction: ₹%.2f\n", recursiveResult);

double iterativeResult = calculateFutureValueIterative(presentValue, annualGrowthRate, years);

System.out.printf("⚙️ Iterative Prediction: ₹%.2f\n", iterativeResult);

System.out.println("\n📊 Analysis:");

System.out.println("- Time Complexity (Recursive): O(n)");

System.out.println("- Space Complexity (Recursive): O(n) due to call stack");

System.out.println("- Iterative approach is preferred for large 'n' to avoid stack overflow.");

}

}