WEEK 1

# **Exercise 1: Implementing the Singleton Pattern**

Logger.java  
  
*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);

    }

}

Loggertest.java

*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 logger instances are the same (Singleton works)");

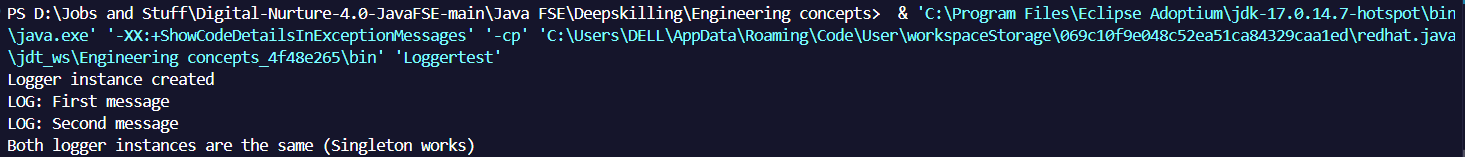
        } else {

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

        }

    }

}



# Exercise 2: Implementing the Factory Method Pattern

public class FactoryTest {

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();

}

}

// Interface

interface Document {

void open();

}

// Concrete Documents

class WordDocument implements Document {

public void open() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

}

// Abstract Factory

abstract class DocumentFactory {

public abstract Document createDocument();

}

// Concrete Factories

class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

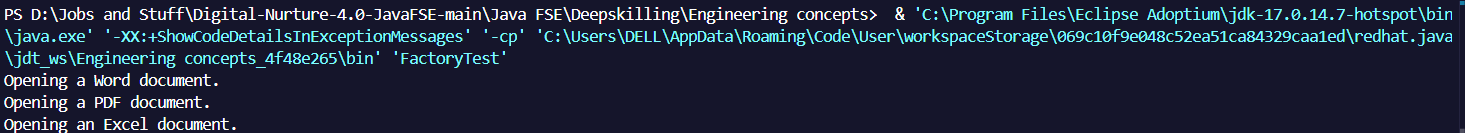
}

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

# Exercise 3: E-commerce Platform Search Function

import java.util.Arrays;

import java.util.Comparator;

// Product class

class Product {

int productId;

String productName;

String category;

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

this.productId = id;

this.productName = name;

this.category = category;

}

public String toString() {

return "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

// Search utilities

class SearchUtils {

// Linear search

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

for (Product p : products) {

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

return p;

}

}

return null;

}

// Binary search

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

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0) return products[mid];

else if (cmp < 0) low = mid + 1;

else high = mid - 1;

}

return null;

}

// Sort helper for binary search

public static void sortByName(Product[] products) {

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

}

}

// Test class

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Shoes", "Fashion"),

new Product(103, "Watch", "Accessories"),

new Product(104, "Phone", "Electronics"),

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

};

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

Product linearResult = SearchUtils.linearSearch(products, "Watch");

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

// Must sort before binary search

SearchUtils.sortByName(products);

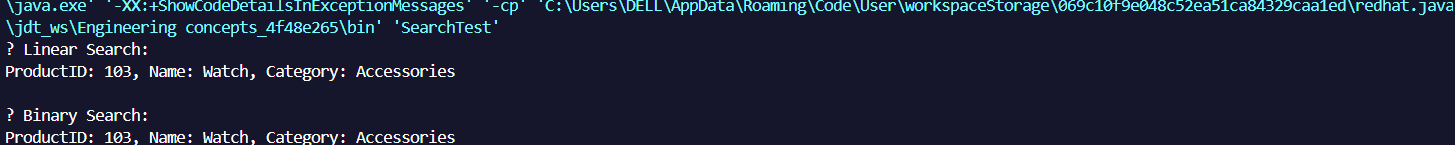
System.out.println("\n Binary Search:");

Product binaryResult = SearchUtils.binarySearch(products, "Watch");

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

}

}



# Exercise 4: Financial Forecasting

public class FinancialForecastMemo {

static double[] memo;

// Recursive method with memoization

public static double forecast(double initial, double rate, int year) {

if (year == 0) return initial;

if (memo[year] != 0) return memo[year];

memo[year] = forecast(initial, rate, year - 1) \* (1 + rate);

return memo[year];

}

public static void main(String[] args) {

double initialValue = 1000.0;

double growthRate = 0.1; // 10%

int forecastYear = 10;

memo = new double[forecastYear + 1];

double result = forecast(initialValue, growthRate, forecastYear);

System.out.printf("🚀 Forecasted value after %d years: ₹%.2f\n", forecastYear, result);

}

}  
  
