**Candidate Name**:- SHARANYA K

**Superset ID**:-6419887

**WEEK – 1 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

**(DESIGN PATTERN AND PRINCIPLES)**

**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New Java Project:**

* Create a new Java project named SingletonPatternExample.

1. **Define a Singleton Class:**

* Create a class named Logger that has a private static instance of itself.
* Ensure the constructor of Logger is private.
* Provide a public static method to get the instance of the Logger class.

1. **Implement the Singleton Pattern:**

* Write code to ensure that the Logger class follows the Singleton design pattern.

1. **Test the Singleton Implementation:**

* Create a test class to verify that only one instance of Logger is created and used across the application.

**Code for the above question:-**

public class Main {

public static void main(String[] args) {

LogService log1 = LogService.getLogger();

log1.writeLog("First message recorded.");

LogService log2 = LogService.getLogger();

log2.writeLog("Second message recorded.");

if (log1 == log2) {

System.out.println("Only one LogService instance is used.");

} else {

System.out.println("Different LogService instances found.");

}

}

}

class LogService {

private static LogService loggerInstance;

private LogService() {

System.out.println("LogService instance initialized.");

}

public static LogService getLogger() {

if (loggerInstance == null) {

loggerInstance = new LogService();

}

return loggerInstance;

}

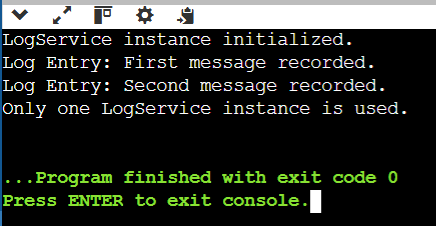
public void writeLog(String msg) {

System.out.println("Log Entry: " + msg);

}

}

**Output Image:-**



**Exercise 2: Implementing the Factory Method Pattern**

**Scenario:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**

* Create a new Java project named FactoryMethodPatternExample.

1. **Define Document Classes:**

* Create interfaces or abstract classes for different document types such as WordDocument, PdfDocument, and ExcelDocument.

1. **Create Concrete Document Classes:**

* Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.

1. **Implement the Factory Method:**

* Create an abstract class DocumentFactory with a method createDocument().
* Create concrete factory classes for each document type that extends DocumentFactory and implements the createDocument() method.

1. **Test the Factory Method Implementation:**

* Create a test class to demonstrate the creation of different document types using the factory method.

**Code for the above question:-**

public class Main {

public static void main(String[] args) {

DocFactory wordCreator = new WordCreator();

DocFactory pdfCreator = new PdfCreator();

DocFactory excelCreator = new ExcelCreator();

Doc wordFile = wordCreator.generateDoc();

Doc pdfFile = pdfCreator.generateDoc();

Doc excelFile = excelCreator.generateDoc();

wordFile.display();

pdfFile.display();

excelFile.display();

}

}

interface Doc {

void display();

}

class WordDoc implements Doc {

public void display() {

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

}

}

class PdfDoc implements Doc {

public void display() {

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

}

}

class ExcelDoc implements Doc {

public void display() {

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

}

}

abstract class DocFactory {

public abstract Doc generateDoc();

}

class WordCreator extends DocFactory {

public Doc generateDoc() {

return new WordDoc();

}

}

class PdfCreator extends DocFactory {

public Doc generateDoc() {

return new PdfDoc();

}

}

class ExcelCreator extends DocFactory {

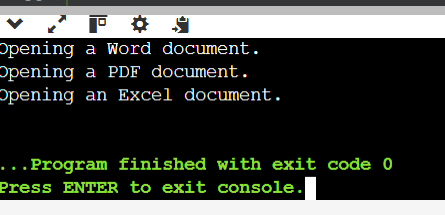
public Doc generateDoc() {

return new ExcelDoc();

}

}

**Output Image:-**



**(ALGORITHM DATA STRUCTURES)**

**Exercise 7: Financial Forecasting**

**Scenario:**

You are developing a financial forecasting tool that predicts future values based on past data.

**Steps:**

1. **Understand Recursive Algorithms:**

* Explain the concept of recursion and how it can simplify certain problems.

1. **Setup:**

* Create a method to calculate the future value using a recursive approach.

1. **Implementation:**

* Implement a recursive algorithm to predict future values based on past growth rates.

1. **Analysis:**

* Discuss the time complexity of your recursive algorithm.
* Explain how to optimize the recursive solution to avoid excessive computation.

**Code for above question:-**

import java.util.Scanner;

public class Main {

public static double predictGrowth(double baseAmount, double annualRate, int yearCount) {

if (yearCount == 0) {

return baseAmount;

}

return predictGrowth(baseAmount, annualRate, yearCount - 1) \* (1 + annualRate);

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Initial investment amount: ");

double startAmount = input.nextDouble();

System.out.print("Expected annual growth rate (e.g., 0.07 for 7%): ");

double growthRate = input.nextDouble();

System.out.print("Forecast for how many years? ");

int duration = input.nextInt();

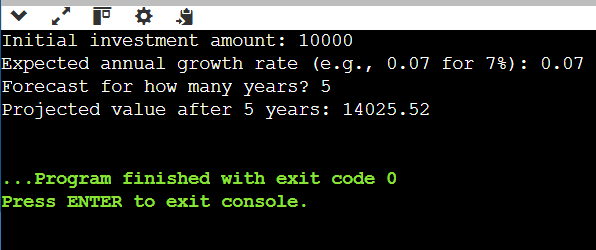
double predictedValue = predictGrowth(startAmount, growthRate, duration);

System.out.printf("Projected value after %d years: %.2f\n", duration, predictedValue);

}

}

**Output Image:-**



**Exercise 2: E-commerce Platform Search Function**

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

Steps:

1. **Understand Asymptotic Notation:**

* Explain Big O notation and how it helps in analyzing algorithms.
* Describe the best, average, and worst-case scenarios for search operations.

1. **Setup:**

* Create a class Product with attributes for searching, such as productId, productName, and category.

1. **Implementation:**

* Implement linear search and binary search algorithms.
* Store products in an array for linear search and a sorted array for binary search.

1. **Analysis:**

* Compare the time complexity of linear and binary search algorithms.
* Discuss which algorithm is more suitable for your platform and why.

**Code for above question:-**

import java.util.\*;

class Item implements Comparable<Item> {

int id;

String name;

String type;

public Item(int id, String name, String type) {

this.id = id;

this.name = name;

this.type = type;

}

public String toString() {

return id + " - " + name + " (" + type + ")";

}

public int compareTo(Item other) {

return Integer.compare(this.id, other.id);

}

}

public class Main {

public static void main(String[] args) {

Item[] inventory = {

new Item(103, "Laptop", "Electronics"),

new Item(101, "Shirt", "Apparel"),

new Item(105, "Headphones", "Electronics"),

new Item(102, "Book", "Education"),

new Item(104, "Shoes", "Footwear")

};

System.out.println("Searching with linear method:");

Item foundLinear = linearFind(inventory, 102);

if (foundLinear != null) {

System.out.println("Match found: " + foundLinear);

} else {

System.out.println("Item not located.");

}

Arrays.sort(inventory);

System.out.println("Searching with binary method:");

Item foundBinary = binaryFind(inventory, 104);

if (foundBinary != null) {

System.out.println("Match found: " + foundBinary);

} else {

System.out.println("Item not located.");

}

}

public static Item linearFind(Item[] list, int id) {

for (Item item : list) {

if (item.id == id) {

return item;

}

}

return null;

}

public static Item binaryFind(Item[] sortedList, int id) {

int start = 0, end = sortedList.length - 1;

while (start <= end) {

int mid = (start + end) / 2;

if (sortedList[mid].id == id) {

return sortedList[mid];

} else if (sortedList[mid].id < id) {

start = mid + 1;

} else {

end = mid - 1;

}

}

return null;

}

}

**Output Image:-**

