**Candidate Name**:- MEGHANA G K

**Superset ID**:-6424295

**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) {

        Logger loggerOne = Logger.getInstance();

        loggerOne.logMessage("System initialization started.");

        Logger loggerTwo = Logger.getInstance();

        loggerTwo.logMessage("User login event recorded.");

        if (loggerOne == loggerTwo) {

            System.out.println("Both logger references point to the same instance.");

        }

        else

        {

            System.out.println("Different instances exist — Singleton failed.");

        }

    }

}

class Logger {

    private static Logger singleLogger = null;

    private Logger() {

        System.out.println("Logger instance created.");

    }

    public static Logger getInstance() {

        if (singleLogger == null) {

            singleLogger = new Logger();

        }

        return singleLogger;

    }

    public void logMessage(String message) {

        System.out.println("Log: " + message);

    }

}

**Output Image:-**

A screenshot of a computer program

AI-generated content may be incorrect., Picture

**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) {

        DocumentFactory wordFactory = new WordDocumentFactory();

        DocumentFactory pdfFactory = new PdfDocumentFactory();

        DocumentFactory excelFactory = new ExcelDocumentFactory();

        Document word = wordFactory.createDocument();

        Document pdf = pdfFactory.createDocument();

        Document excel = excelFactory.createDocument();

        word.open();

        pdf.open();

        excel.open();

    }

}

interface Document {

    void open();

}

class WordFile implements Document {

    public void open() {

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

    }

}

class PdfFile implements Document {

    public void open() {

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

    }

}

class ExcelFile implements Document {

    public void open() {

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

    }

}

abstract class DocumentFactory {

    public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordFile();

    }

}

class PdfDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfFile();

    }

}

class ExcelDocumentFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelFile();

    }

}

**Output Image:-**

A screenshot of a computer

AI-generated content may be incorrect., Picture

**(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 ForecastApp {

    public static double calculateValue(double principal, double rate, int years) {

        if (years == 0) return principal;

        return calculateValue(principal, rate, years - 1) \* (1 + rate);

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter initial investment: ");

        double amount = scanner.nextDouble();

        System.out.print("Enter annual growth rate (e.g., 0.06 for 6%): ");

        double rate = scanner.nextDouble();

        System.out.print("Enter number of years: ");

        int years = scanner.nextInt();

        double result = calculateValue(amount, rate, years);

        System.out.printf("Value after %d years: %.2f\n", years, result);

    }

}

**Output Image:-**

A screen shot of a computer

AI-generated content may be incorrect., Picture

**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.Arrays;

class Item implements Comparable<Item> {

    int itemId;

    String itemName;

    String itemGroup;

    public Item(int itemId, String itemName, String itemGroup) {

        this.itemId = itemId;

        this.itemName = itemName;

        this.itemGroup = itemGroup;

    }

    @Override

    public String toString() {

        return "ID: " + itemId + ", Name: " + itemName + ", Category: " + itemGroup;

    }

    @Override

    public int compareTo(Item other) {

        return this.itemId - other.itemId;

    }

}

public class Main {

    public static void main(String[] args) {

        // Creating a list of items in inventory

        Item[] inventory = {

            new Item(300, "Smartphone", "Electronics"),

            new Item(120, "Notebook", "Stationery"),

            new Item(450, "Sneakers", "Footwear"),

            new Item(180, "Jacket", "Clothing"),

            new Item(390, "Charger", "Accessories")

        };

        // Performing linear search

        System.out.println("Looking for item with ID 180 (Linear Search):");

        Item resultLinear = findUsingLinearSearch(inventory, 180);

        if (resultLinear != null) {

            System.out.println("Found item: " + resultLinear);

        } else {

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

        }

        // Sorting items for binary search

        Arrays.sort(inventory);

        // Performing binary search

        System.out.println("\nLooking for item with ID 120 (Binary Search):");

        Item resultBinary = findUsingBinarySearch(inventory, 120);

        if (resultBinary != null) {

            System.out.println("Found item: " + resultBinary);

        } else {

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

        }

    }

    // Linear search method

    public static Item findUsingLinearSearch(Item[] items, int targetId) {

        for (Item item : items) {

            if (item.itemId == targetId) {

                return item;

            }

        }

        return null;

    }

    // Binary search method

    public static Item findUsingBinarySearch(Item[] items, int targetId) {

        int start = 0;

        int end = items.length - 1;

        while (start <= end) {

            int middle = (start + end) / 2;

            if (items[middle].itemId == targetId) {

                return items[middle];

            } else if (items[middle].itemId < targetId) {

                start = middle + 1;

            } else {

                end = middle - 1;

            }

        }

        return null;

    }

}

**Output Image:-**

A computer screen with white text

AI-generated content may be incorrect., Picture