**DEEP NURTURE 4.0----WEEK-1**

**Design Patterns and principles:**

**Excercise1:**

**Logger.java:**

package LoggerApp;

public class Logger {

    private static final Logger instance = new Logger(); // Eager initialization

    private Logger() {

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

    }

    public static Logger getInstance() {

        return instance;

    }

    public void log(String message) {

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

    }

}

**Main.java:**

package LoggerApp;

public class Main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        logger1.log("This is the first log message.");

        Logger logger2 = Logger.getInstance();

        logger2.log("This is the second log message.");

        if (logger1 == logger2) {

            System.out.println("Both logger1 and logger2 refer to the same instance.");

        } else {

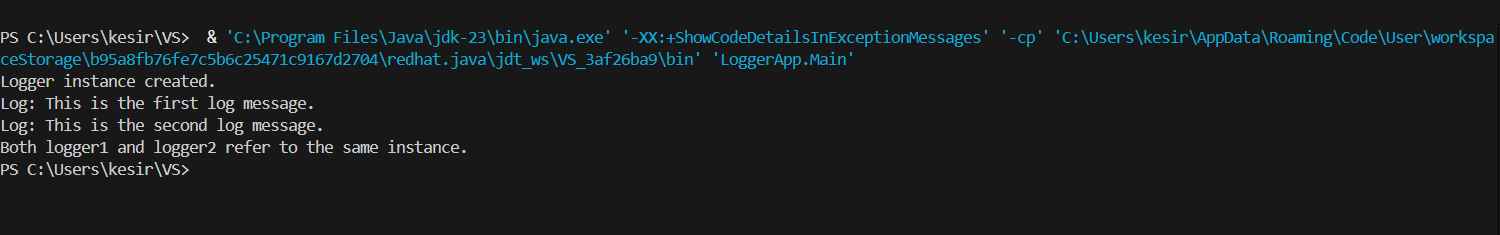
            System.out.println("logger1 and logger2 are different instances.");

        }

    }

}

**OUTPUT:**



**Excercise2:**

**File.java:**

package factory;

public interface File {

void open();

}

**FileFactory.java:**

package factory;

public abstract class FileFactory {

public abstract File generateFile();

}

**WordFile.java:**

package factory;

public class WordFile implements File {

@Override

public void open() {

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

}

}

**WordFileFactory.java:**

package factory;

public class WordFileFactory extends FileFactory {

@Override

public File generateFile() {

return new WordFile();

}

}

**PdfFile.java:**

package factory;

public class PdfFile implements File {

@Override

public void open() {

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

}

}

**PdfFileFactory.java:**

package factory;

public class PdfFileFactory extends FileFactory {

@Override

public File generateFile() {

return new PdfFile();

}

}

**ExcelFile.java:**

package factory;

public class ExcelFile implements File {

@Override

public void open() {

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

}

}

**ExcelFileFactory.java:**

package factory;

public class ExcelFileFactory extends FileFactory {

@Override

public File generateFile() {

return new ExcelFile();

}

}

**Main1.java:**

package FactoryPattern;

public class Main1 {

    public static void main(String[] args) {

        FileFactory wordFactory = new WordFileFactory();

        FileFactory pdfFactory = new PdfFileFactory();

        FileFactory excelFactory = new ExcelFileFactory();

        File word = wordFactory.generateFile();

        File pdf = pdfFactory.generateFile();

        File excel = excelFactory.generateFile();

        word.open();

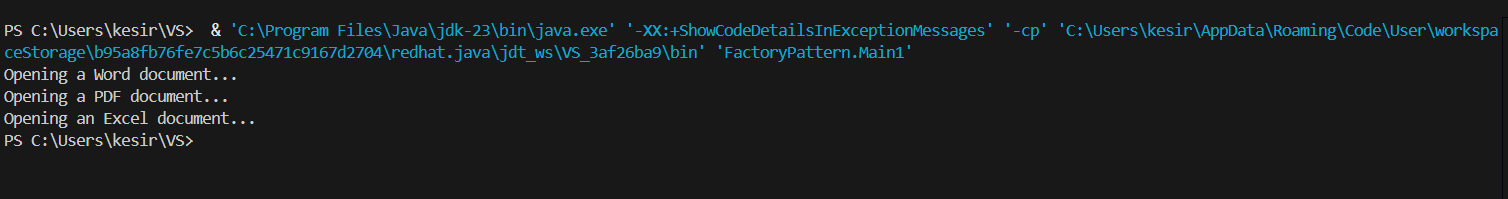
        pdf.open();

        excel.open();

    }

}

**OUTPUT:**



**Algorithm Data Structures:**

**Excercise1:**

**Big O Notation:**

* Big O notation describes the worst-case time complexity of an algorithm, showing how the runtime scales with the size of the input.

| **Complexity** | **Description** |
| --- | --- |
| O(1) **--------------------->** | Constant time |
| O(log n) **--------------------->** | Logarithmic time |
| O(n) **--------------------->** | Linear time |
| O(n log n) **--------------------->** | Log-linear time |
| O(n²) **--------------------->** | Quadratic time |

**Search Scenarios:**

* **Best Case:** Found at first attempt (start of list).
* **Average Case:** Found after scanning half the list**.**
* **Worst Case:** Element not found or found at the end**.**

| **Search Type** | **Best Case** | **Average Case** | **Worst Case** |
| --- | --- | --- | --- |
| Linear Search | O(1) | O(n) | O(n) |
| Binary Search | O(1) | O(log n) | O(log n) |

**Item.java:**

package  Ecommerce;

public class Item {

    int id;

    String name;

    String type;

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

        this.id = id;

        this.name = name;

        this.type = type;

    }

    @Override

    public String toString() {

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

    }

}

**ItemSearch.java:**

package Ecommerce;

import java.util.Arrays;

import java.util.Comparator;

public class ItemSearch {

    public static Item findByNameLinear(Item[] items, String targetName) {

        for (Item item : items) {

            if (item.name.equalsIgnoreCase(targetName)) {

                return item;

            }

        }

        return null;

    }

    public static Item findByNameBinary(Item[] items, String targetName) {

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

        while (left <= right) {

            int mid = (left + right) / 2;

            int cmp = items[mid].name.compareToIgnoreCase(targetName);

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

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

            else right = mid - 1;

        }

        return null;

    }

    public static void main(String[] args) {

        Item[] catalog = {

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

            new Item(102, "Phone", "Electronics"),

            new Item(103, "Shirt", "Clothing"),

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

            new Item(105, "Book", "Stationery")

        };

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

        Item match1 = findByNameLinear(catalog, "Shirt");

        System.out.println(match1 != null ? match1 : "Item not found");

        Arrays.sort(catalog, Comparator.comparing(i -> i.name.toLowerCase()));

        System.out.println("\nBinary Search:");

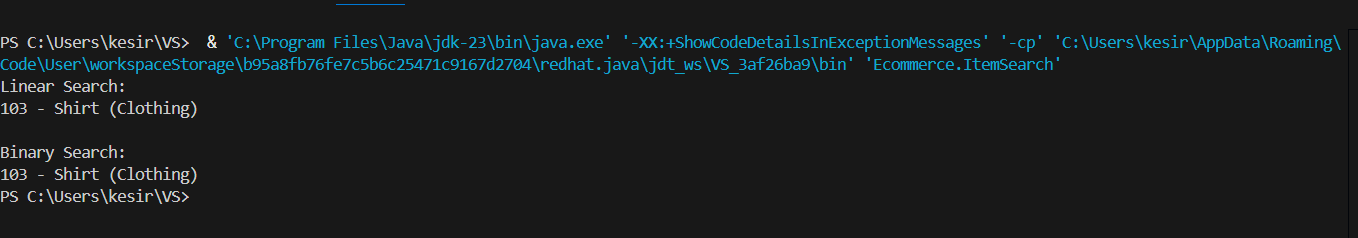
        Item match2 = findByNameBinary(catalog, "Shirt");

        System.out.println(match2 != null ? match2 : "Item not found");

    }

}

**OUTPUT:**



**Excercise2:**

* Recursion is a technique where a method calls itself to solve a smaller instance of the same problem.

**Example:**  
To compute compound future value:

* FutureValue(n)=FutureValue(n−1)×(1+r)
* Base case:FutureValue(0)=initialValue

Benefits:

* Elegant and simplifies code.
* Especially useful in problems that follow a repeated pattern (e.g., financial growth over years).

WealthProjection.java:

package Financial;

public class WealthProjection {

    public static double calculateRecursive(double principal, double rate, int time) {

        if (time == 0) return principal;

        return calculateRecursive(principal, rate, time - 1) \* (1 + rate);

    }

    public static double calculateMemoized(double principal, double rate, int time, Double[] memo) {

        if (time == 0) return principal;

        if (memo[time] != null) return memo[time];

        memo[time] = calculateMemoized(principal, rate, time - 1, memo) \* (1 + rate);

        return memo[time];

    }

    public static void main(String[] args) {

        double startingAmount = 5000;

        double annualRate = 0.10;  // 10%

        int duration = 5;          // 5 years

        System.out.println("Simple Projection:");

        double result1 = calculateRecursive(startingAmount, annualRate, duration);

        System.out.printf("Final amount after %d years: %.2f\n", duration, result1);

        Double[] memo = new Double[duration + 1];

        System.out.println("\nMemoized Projection:");

        double result2 = calculateMemoized(startingAmount, annualRate, duration, memo);

        System.out.printf("Final amount after %d years: %.2f\n", duration, result2);

    }

}

**OUTPUT:**

