

## **DESIGN PATTERNS AND PRINCIPLE : HANDS-ON ( MANDATORY )**

### **EXERCISE –1 : SINGLETON PATTERN**

- It is to design a system that creates exactly one instance of the logger class and provides a global access point to it .
- Multiple objects could result in duplicate or inconsistent logging.

#### ✓ **CONCEPT :**

##### **1. SINGLETON :**

\* A design pattern that ensures a class has only one instance and provides a global access point to it .

\* Used in : Logging , Configuration setting , Database connection management.

#### ✓ **KEYWORDS :**

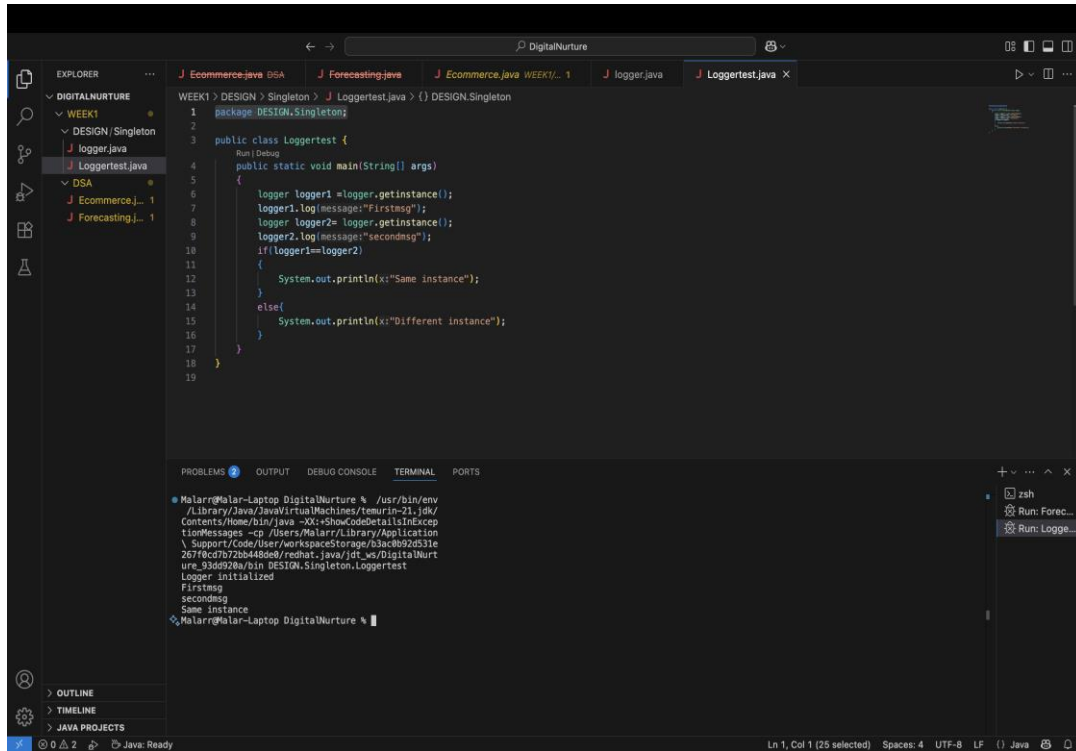
1. *Private* : Restricts access and external object creation.

2. *Instance* : One and only object of the class.

3. *getInstance()* : Method that controls and provides access to the singleton object.

#### ✓ **TIME COMPLEXITY : $O(1)$**

## ✓ OUTPUT :



```
package DESIGN.Singleton;

public class Loggertest {

    public static void main(String[] args) {
        logger logger1 = logger.getInstance();
        logger1.log(message:"Firstmsg");
        logger logger2= logger.getInstance();
        logger2.log(message:"secondmsg");
        if(logger1==logger2)
        {
            System.out.println(x:"Same instance");
        }
        else{
            System.out.println(x:"Different instance");
        }
    }
}
```

Terminal Output:

```
Malar@Malar-Laptop DigitalNurture % ./usr/bin/env
/Library/Java/JavaVirtualMachines/temurin-21.jdk/
Contents/Home/bin/java -XX:+ShowCodeDetailsInExcep
tionMessages -cp /Users/Malar/Library/Application
Support/Code/Java/WorkspacesStorage/B3ac092d531e
267f9cd7b72b4489e9/redhat.java/get_xm/DigitalNur
ture_93dd928a/bin DESIGN.Singleton.Loggertest
Logger Initialized
Firstmsg
secondmsg
Same instance
Malar@Malar-Laptop DigitalNurture %
```

## EXERCISE – 2 : FACTORY METHOD PATTERN

- To create a system that handles multiple type of documents.
- Single creation point that knows how to create the correct document depending on the type.

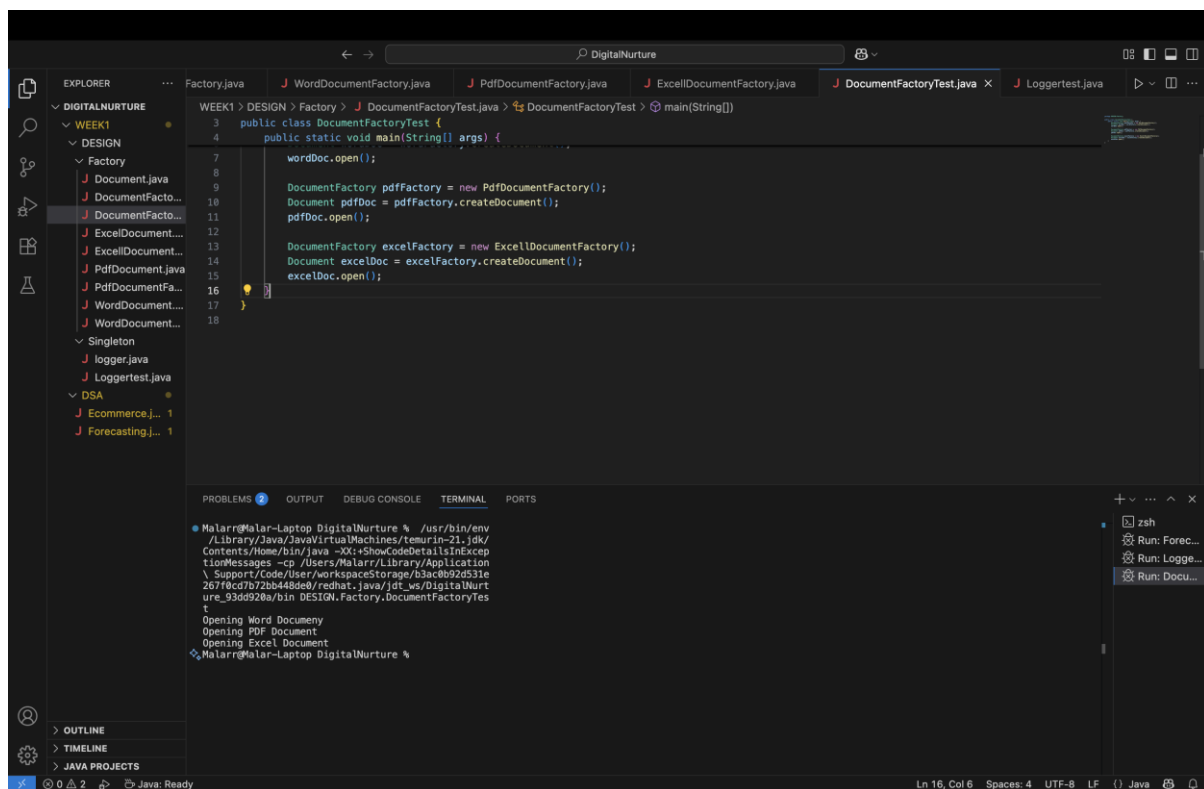
## KEYWORDS :

1. *Interface* : Defines what a class must do (without saying how)
2. *Abstract class* : A base class that cant be instantiated but is extended by other classes.
3. *createDocument()* : Factory method that creates and returns the correct document object.

4. *Implements / extends* : Used to implement interface or extend classes for specific behaviour

✓ **TIME COMPLEXITY** : *createDocument()* ->  $O(1)$

✓ **OUTPUT** :



The screenshot shows an IDE with the following components:

- EXPLORER**: A tree view on the left showing a project named 'DIGITALNURTURE'. It contains folders for 'WEEK1', 'DESIGN', 'Factory', 'Singleton', and 'DSA'. The 'Factory' folder is expanded, showing files like 'Document.java', 'DocumentFactory.java', 'ExcelDocument.java', 'PdfDocument.java', 'WordDocument.java', 'Singleton', 'logger.java', and 'LoggerTest.java'.
- EDITOR**: The main area shows the code for 'DocumentFactoryTest.java'. The code is as follows:

```
1 public class DocumentFactoryTest {
2     public static void main(String[] args) {
3         // ...
4         DocumentFactory pdfFactory = new PdfDocumentFactory();
5         Document pdfDoc = pdfFactory.createDocument();
6         pdfDoc.open();
7
8         DocumentFactory excelFactory = new ExcelDocumentFactory();
9         Document excelDoc = excelFactory.createDocument();
10        excelDoc.open();
11    }
12 }
```
- TERMINAL**: The bottom panel shows the output of the program. It starts with the command to run the program, followed by the output: 'Opening Word Document', 'Opening PDF Document', and 'Opening Excel Document'.