Hands on session

Design Patterns and Principles

**Implementing the Singleton Pattern**

**Definition:**

**Singleton Pattern** is a design pattern that ensures a class has only one instance throughout the entire application and provides a global access point to that instance.

**Steps**

* Create a class Logger
* Make the constructor private
* Create a private static variable of the same class
* Create a public static method

**Code**

public class SingletonPatternExample {

static class Logger {

private static Logger singleInstance;

private Logger() {

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

}

public static Logger getInstance() {

if (singleInstance == null) {

singleInstance = new Logger();

}

return singleInstance;

}

public void log(String message) {

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

}

}

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

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

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

if (logger1 == logger2) {

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

} else {

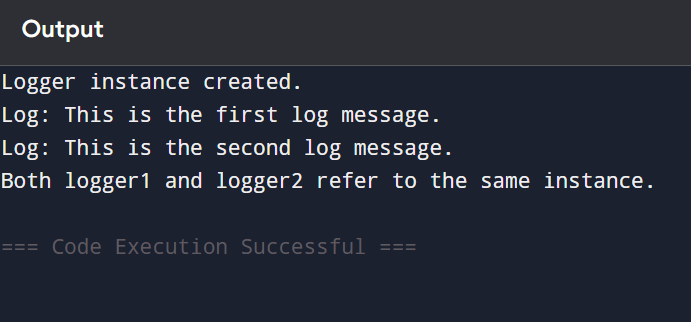
System.out.println("Different instances exist. Singleton failed.");

}

}

}

**Output**



**Summary for this code**

* Only one object of a class is created
* It provides a global access point to that single object

**Implementing the factory method pattern**

The Factory Method Pattern is a creational design pattern that defines an interface for creating objects, but allows subclasses to decide which class to instantiate.It helps in creating objects without specifying their exact class, thus promoting loose coupling and flexibility.

**Steps**

* Create a New Java Project
* Create a Java project named: FactoryMethodPatternExample.
* Define Document Interface or Abstract Class
* Create Concrete Document Classes
* Implement different document types like WordDocument ,PDFDocument,ExcelDocument
* Implement the Factory Method
* Test the Factory Method

**Code**

public class FactoryMethodPatternExample {

interface Document {

void open();

}

static class WordDocument implements Document {

public void open() {

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

}

}

static class PdfDocument implements Document {

public void open() {

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

}

}

static class ExcelDocument implements Document {

public void open() {

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

}

}

static abstract class DocumentFactory {

public abstract Document createDocument();

}

static class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

static class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

static class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

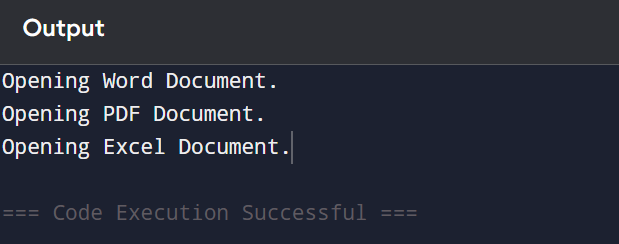
Document excel = excelFactory.createDocument();

excel.open();

}

}

**Output**



**Summary of this code**

* No real files are opened from this code
* Only simulation/output is done via println()
* Output is only text in console