**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:**
   1. Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   1. Create a class named Logger that has a private static instance of itself.
   2. Ensure the constructor of Logger is private.
   3. Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   1. Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   1. Create a test class to verify that only one instance of Logger is created and used across the application.

**Solution Code**:

class logger{

private static logger instance;

private logger() {

System.out.println("Private constructor instatiated!!");

}

public static logger getlogger() {

if(instance==null) {

instance=new logger();

}

return instance;

}

public void log(String msg) {

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

}

}

public class Main {

public static void main(String[] args) {

// TODO Auto-generated method stub

logger obj1=logger.getlogger();

logger obj2=logger.getlogger();

obj1.log("Object 1 is executed");

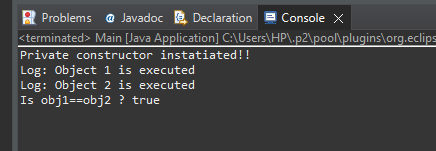
obj2.log("Object 2 is executed");

System.out.println("Is obj1==obj2 ? "+(obj1==obj2));

}

}

**Output**:



**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:**
   1. Create a new Java project named **FactoryMethodPatternExample**.
2. **Define Document Classes:**
   1. Create interfaces or abstract classes for different document types such as **WordDocument**, **PdfDocument**, and **ExcelDocument**.
3. **Create Concrete Document Classes:**
   1. Implement concrete classes for each document type that implements or extends the above interfaces or abstract classes.
4. **Implement the Factory Method:**
   1. Create an abstract class **DocumentFactory** with a method **createDocument()**.
   2. Create concrete factory classes for each document type that extends DocumentFactory and implements the **createDocument()** method.
5. **Test the Factory Method Implementation:**
   1. Create a test class to demonstrate the creation of different document types using the factory method.

**Solution**:

public interface document {

void open();

}

class WordDoc implements document{

@Override

public void open() {

// TODO Auto-generated method stub

System.out.println("Word document is open");

}

}

class PdfDoc implements document{

@Override

public void open() {

// TODO Auto-generated method stub

System.out.println("PDF document is open");

}

}

class ExcelDoc implements document{

@Override

public void open() {

// TODO Auto-generated method stub

System.out.println("Excel document is open");

}

}

abstract class DocumentFactory{

abstract document createDocument();

}

class WordFactory extends DocumentFactory{

@Override

document createDocument() {

// TODO Auto-generated method stub

return new WordDoc();

}

}

class PdfFactory extends DocumentFactory{

@Override

document createDocument() {

// TODO Auto-generated method stub

return new PdfDoc();

}

}

class ExcelFactory extends DocumentFactory{

@Override

document createDocument() {

// TODO Auto-generated method stub

return new ExcelDoc();

}

}

public class Main {

public static void main(String[] args) {

// TODO Auto-generated method stub

DocumentFactory obj1=new WordFactory();

document doc1=obj1.createDocument();

doc1.open();

DocumentFactory obj2=new PdfFactory();

document doc2=obj2.createDocument();

doc2.open();

DocumentFactory obj3=new ExcelFactory();

document doc3=obj3.createDocument();

doc3.open();

}

}

**Output**:

