Lab 01 – Singleton Implementations in Java

**Part 01**

Thread Safety: for thread-safe implementation, we can use synchronized keyword in the method. But, it reduces the performance due to the cost associated with the synchronized method. So, to avoid this extra overhead every time, double checked locking principle is used. In this approach, the synchronized block is used inside if condition with an additional check to ensure that only one instance of singleton class is created. The following implementation is called lazy instantiation.

**package** cs525.labs.singleton;

//Traditional Implementation of Singleton Pattern

**public** **class** Singleton {

**private** **static** Singleton *uniqueInstance* = **null**;

**private** **int** data = 0;

**public** **static** Singleton getInstance() {

**if** (*uniqueInstance* == **null**) {

**synchronized** (Singleton.**class**) {

**if** (*uniqueInstance* == **null**)

*uniqueInstance* = **new** Singleton();

}

}

**return** *uniqueInstance*;

}

/\*\*

\* The Singleton Constructor. Note that it is private! No client can

\* instantiate a Singleton object directly!

\*/

**private** Singleton() {

}

// Accessors and mutators here!

}

**Part 02**

The below code is thread safe implementation of singleton pattern using Enum. As Java ensures that any Enum value is instantiated only once in a Java program - it will guarantee the single instance. Also, Java Enum values are globally accessible, so is the singleton. This kind of implementation for singleton is eager.

//Enum Implemenation for singleton

**package** cs525.labs.singleton;

**public** **enum** SingletonEnum {

***INSTANCE***;

**public** **static** **void** behavior() {

//Do something!

}

}

//main class

**package** cs525.labs.singleton;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Singleton SingletonLazy = Singleton.*getInstance*();

SingletonEnum singletonEnum = SingletonEnum.***INSTANCE***;

//access method

//or simply, use - SingletonEnum.INSTANCE.behavior();

singletonEnum.*behavior*();

}

}