* **Singleton**

**class** A{

// private static member

**private** **static** A obj;

// Private Constructor

**private** A()

{

}

// public Static factory method

**public** **static** A getInstance()

{

**if** (obj == **null**){

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

{

**if** (obj == **null**)

{

            obj = **new** Singleton();//instance will be created at request time

        }

    }

    }

**return** obj;

 }

**public** **void** doSomething()

{

 //write your code

 }

}

* **What is this pattern?**

Only one instance of the class should be created at any point of time and the same should be used by all other classes.

* **How to Design?**

So, to achieve this we need to design class in such a way that it has

**Private static member** is defined so that it gets memory only once.

**Private constructor**: will be defined so that singleton class will not be instantiated outside

**Public static factory method:** global point of access to the singleton object.

* **Where this pattern can be used? IMP**

It is used in

Database connections

Logging

Caching

Thread pools

* **What is the Advantage of Singleton Design pattern?**

As we are creating only one instance it saves memory.

* **What is the Disadvantage of singleton design pattern? Search for better answer if you can find**

Singleton classes cannot be sub classed.

*They violate Single Responsibility Principle*: because singleton class has to control their own creation and life cycle.

Code is tightly coupled.

* **Best Ways to implement Singleton design pattern**

**https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples**

* + **Early Instantiation or Eger initialization:** In this case singleton object is created at class loading time.

*Disadvantage:* Though it is the easiest method to create a singleton class disadvantage is that instance is created even though client application might not be using it.

* + **Static block initialization:** Static block initialization is same as Early instantiation but in this case singleton object instantiation happens in static block.

Only advantage over early instantiation is that as object is initialized inside static block, we can support exception handling.

* + **Lazy Instantiation:** In this case singleton object will be created when requested for first time.

There are three variants of lazy instantiation

1. Non thread safe
2. Thread safe with factory method synchronization
3. Double checked locking
   * **Bill Pugh Singleton Implementation:**  “Inner static Helper class”

Creating singleton class using an inner static helper class.

*package com.journaldev.singleton;*

*public class BillPughSingleton {*

*private BillPughSingleton(){}*

*private static class SingletonHelper{*

*private static final BillPughSingleton INSTANCE = new BillPughSingleton();*

*}*

*public static BillPughSingleton getInstance(){*

*return SingletonHelper.INSTANCE;*

*}*

*}*

* + **enum singleton.**

Because java ensures enum value is instantiated only once in java program.

**Enum singleton is a guaranteed singleton by JVM.**

public enum SingletonEnum

{

INSTANCE;

public void leaveTheBuilding()

{

...

}

}

* + **Using Reflection to destroy Singleton Pattern**
  + **Serialization and Singleton in distributed systems**

Need to know Double Locking concept in detail

Reference:

[**https://medium.com/@kevalpatel2106/digesting-singleton-design-pattern-in-java-5d434f4f322**](https://medium.com/@kevalpatel2106/digesting-singleton-design-pattern-in-java-5d434f4f322) **...Read this and understands**