1. **Eager initialization** :

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

**private** **static** Singleton instance = **new** Singleton(); *//eagerness*

**private** **Singleton**() {

*//init*

}

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

**return** instance;

}

}

1. **Lazy initialization** : (Consider this if init is **resource heavy** and if it is required in **only some application flows**)

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

**private** **static** Singleton instance;

**private** **Singleton**() {

*//init*

}

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

**if**(instance == **null**) { *//laziness*

instance = **new** Singleton();

}

**return** instance;

}

}

1. **Thread safe Lazy initialization** : (In the lazy init construct above, if there are two or more threads accessing the getInstance method, it might lead to the **creation of spurious multiple instances** of the singleton!)

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

**private** **static** Singleton instance;

**private** **Singleton**() {

*//init*

}

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

*//thread safety*

**if**(instance == **null**) {

instance = **new** Singleton();

}

**return** instance;

}

}

1. **Double checked locking** : (Note that in the above approach, the race condition can be reached only **once** in the entire application lifecycle, ie when **instance is null**! What we are doing is introducing a LOT of overhead wth the *synchronized* keyword. So we need a way to ensure that the locking protection only applies **once**)

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

**private** **volatile** **static** Singleton instance;

**private** **Singleton**() {

*//init*

}

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

**if**(instance == **null**) {

**synchronized**(**this**) {

*//only lock the FIRST time*

**if**(instance == **null**) {

*//The "double" check*

instance = **new** Singleton();

}

}

}

**return** instance;

}

}

1. **Using an enum!** : (The simplest way to define a singleton! And guess what! Enums are **lazily initialized** by the JVM, i.e. they are instantiated the first time they are accessed! The **creation of an enum is thread safe** too, the **JVM** ensures that! :D)

**public** **enum** Singleton {

INSTANCE;

}