Singleton Pattern ensures a class has only one instance and provides a global point of access to it.

***Eagerly Loading Sample:***

public class Singleton {

private static Singleton myInstance = new Singleton();

private Singleton() {}

public static Singleton getInstance() {

return myInstance;

}

}

***Lazy Loading Sample:***

public class Singleton {

private static Singleton myInstance;

private Singleton() {}

public static synchronized Singleton getInstance() {

if(myInstance == null) {

return new Singleton();

}

return myInstance;

}

}

***Double-checked Locking:***

public class Singleton {

private volatile static Singleton myInstance;

private Singleton() {}

public static synchronized Singleton getInstance() {

if(myInstance == null) {

synchronized(Singleton.class) {

if(myInstance == null) {

return new Singleton();

}

}

}

return myInstance;

}

}

Note:

The volatile keyword ensures that multiple threads handle the myInstance variable correctly when it is being initialized to the Singleton instance. Double checked locking don’t work with Java 1.4 or earlier.

* The Singleton Pattern ensures you have atmost one instance of a class in your application.
* The Singleton Pattern also provides a global access point to that instance.
* Java’s implementation of the Singleton Pattern makes use of a private constructor, a static method combined with a static variable.
* Examine your performance and resource constraints and caregully choose an appropriate Singleton implementation for multi threaded application(and we should consider all applications multithreaded)
* Beware of the double-checked locking implementation; it is not thread safe in versions befor Java2 version 5.
* Be careful if you are using multiple class loaders; this could defeat the Singleton implementation and result in multiple instances(Use Eager loading here)
* If you are using a JVM earlier than 1.2, you will need to create a registry of Singletons to defeat the garbage collector.