**In Singleton double check pattern how to prevent it's breaking from Java Reflection**

<https://stackoverflow.com/questions/20421920/what-are-the-different-ways-we-can-break-a-singleton-pattern-in-java>

Starting with your given code, "Double-Checked Locking" can be brocken at some enviroment,

  It can break if the class is Serializable

  It can break if its 'Clonable`

  You can break by Reflection (I believe)

  it can break if multiple class loaders are loaded the class

**\*How do you solve rule breakers?**

  It is much safer to do eager initialization

  To prevent deserializing to create new object you may override readResolve() method in your class and throw exception

  To prevent cloning, you may overrride clone() and throw CloneNotSupported exception

  To escape for reflective instantion, we can add check in the constructor and throw exception

**Example**

public class Singleton {

    private static final Singleton INSTANCE = new Singleton();

    private Singleton() {

        // Check if we already have an instance

**if (INSTANCE != null) {**

**throw new IllegalStateException("Singleton" +**

**" instance already created.");**

**}**

    }

    public static final Singleton getInstance() {

        return INSTANCE;

    }

    private Object **readResolve**() throws ObjectStreamException         {

            return INSTANCE;

    }

    private Object **writeReplace**() throws ObjectStreamException {

            return INSTANCE;

    }

    public Object **clone**() throws CloneNotSupportedException {

        // return INSTANCE

        throw new CloneNotSupportedException();

    }

}

After all I would suggest to use Enum as the safest way for Singleton (Since java5 the best way to do it is to use an enum)

public static enum SingletonFactory {

    INSTANCE;

    public static SingletonFactory getInstance() {

        return INSTANCE;

    }

}