Diskussionsseiten

1) http://www.theserverside.de/singleton-pattern-in-java/
2) http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html
3) https://funthomas424242.wordpress.com/2012/02/19/cc-singleton-korrekt-implementieren/

Singleton - unsichere Initialisierung

Diese Implementierung hat folgende Besonderheiten:
[1] Theore tisch funkti oniert sie nicht - in der Praxis taugt sie aber als Signleton (siehe dazu folgende Diskussion)
[2] Man darf sich hier bei Zugriff nicht auf ein korrekt initia lis iertes Signleton verlassen, da im konkreten Fall auf Grund des Memory Modelles keine Annahmen darüber zugelassen sind wann die Felder des erzeugten Singleton wirklich alle korrekt befüllt und für jeden Thread konsistent lesbar sind.

```
public class Resource{
  private static Resource singleton;
public static Resource create Ins tan ce(){
      if( sin gle ton ==n ull){ //erster Test
        syn chr oni zed (Re sou rce.cl ass){
            if( sin gle ton ==n ull){ //zweiter Test
                  sin gle ton=new Resour ce();
           } //endif
         } // end sync
     }// endif
  return singleton;
 }
}
```

Details zum Problem unter: http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html

Singleton - korrekte Intialisierung

```
private static volatile Resource resource;
private static volatile Resource singleton;
public static Resource create Ins tan ce(){
  if( res our ce= =null){
    syn chr oni zed (Re sou rce.cl ass){
      if( sin gle ton ==n ull){
              sin gle ton=new Resour ce();
     }
   }
    res our ce= sin gleton;
 }
  return resource;
}
```

Ähnliche Lösungen werden auch hier diskutiert: http://www.angelikalanger.com/Articles/EffectiveJava/41.JMM-DoubleCheck/41.JMM-DoubleCheck.html