Singleton

Ziel

Von einer Klasse soll höchstens eine Instanz existieren.

Motivation

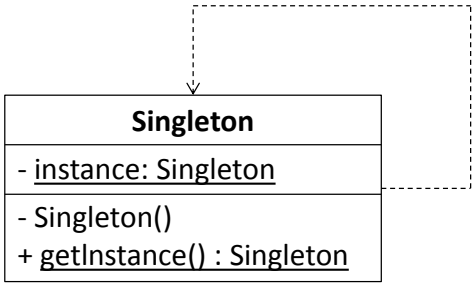
- Cache-Implementierungen
- Objekte welche Registry-Einstellungen oder Präferenzen verwalten
- Thread-Pool
- Klasse, mit welcher MP3 Dateien abgespielt werden. Falls man während dem Abspielen einer Datei eine neue Datei abspielen lässt, so ist das Verhalten unvorhersehbar.
  => Lösung: nur eine Instanz welche das Abspielen koordiniert.
- Treiber (Drucker/Datenbank).
  Bei der Implementierung von Treibern müssen globale Invarianten sichergestellt werden, z.B. dass nicht gleichzeitig auf DB zugegriffen wird.
- Kommunikation über Rechnergrenzen hinweg: Versand von Daten über einen Socket

Erste Lösungsidee: eine Klasse mit nur statischen Variablen und Methoden. Eine Klasse existiert nur einmal pro Class Loader.

Aufgabe: Welche Nachteile hat dieser Ansatz? Beispielsweise: Wenn eine komplexe Initialisierung notwendig ist, wo und wie müsste diese erfolgen?

Schauen wir uns also eine Lösung an, welche ein einziges echtes Objekt verwendet:

Struktur



Implementierung

```
public final class Singleton {
    private static Singleton instance = new Singleton();

    public static Singleton getInstance() {
         return instance;
    }

    private Singleton() {}
}
```

Beispiele:

`java.lang.Runtime` (1 Instanz, repräsentiert das System, auf welchem die JVM läuft)

`java.lang.Class` (n Instanzen, n = Anzahl geladene Klassen)

Aufgaben:

- Der Konstruktor ist `private` deklariert. Welche Probleme treten auf wenn er `protected` deklariert wird?

  Bemerkung:
  Falls Konstruktor `private`, dann kann die Klasse auch als `final` deklariert werden.

- Geben Sie Regeln an, wie `equals` / `hashCode` und `clone` implementiert werden sollen.

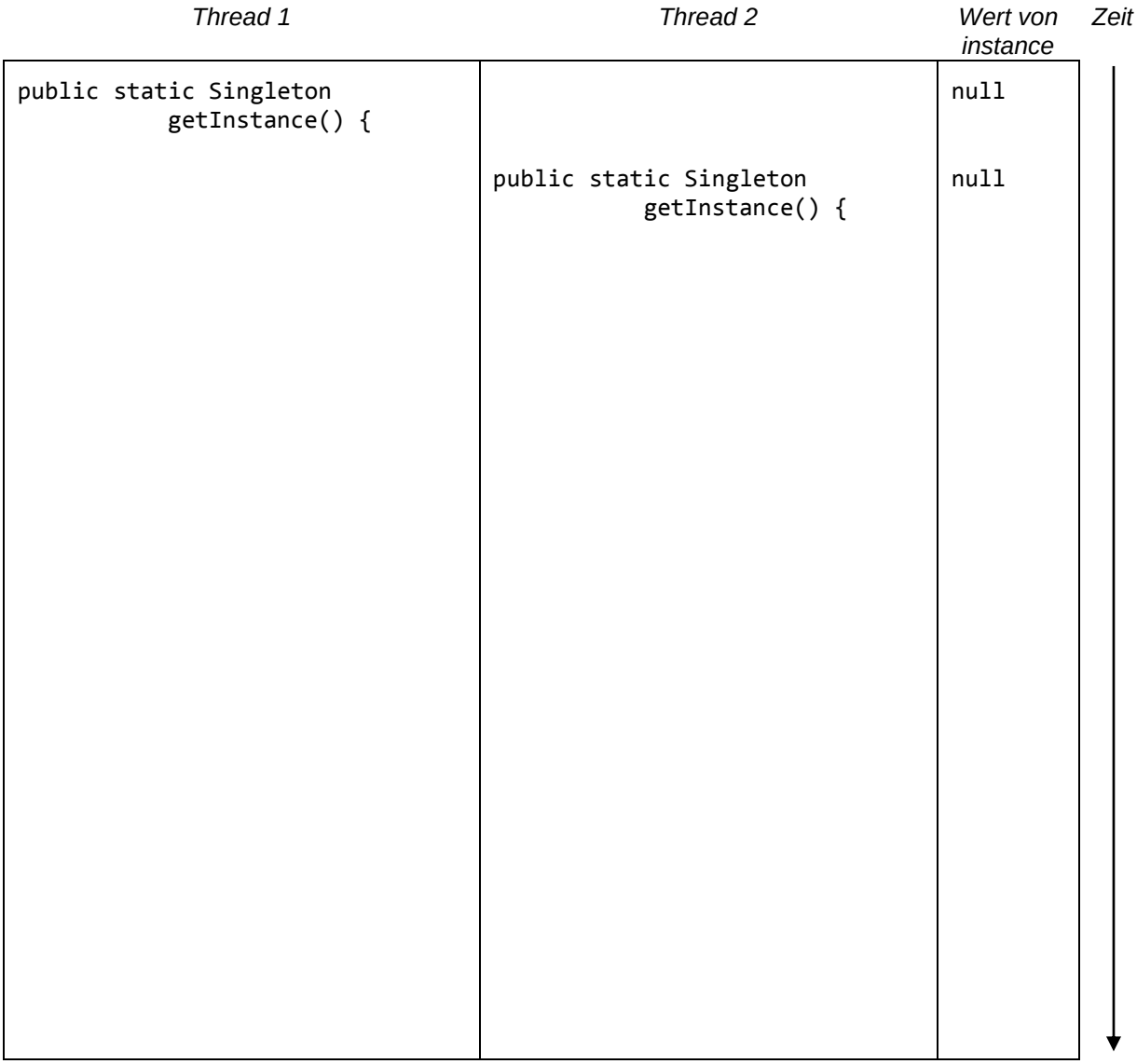
Bemerkung: Falls Objekt serialisierbar: Durch Einlesen können Kopien entstehen!
→ `readResolve`! Siehe dazu z.B. Artikel unter: http://www.javalobby.org/java/forums/t17491.html

Lazy Initialisierung und Synchronisierung

Eine Singleton Implementierung kann wie folgt aussehen:

```
public final class Singleton {
    private static Singleton instance = null;
    public static Singleton getInstance(){
         if(instance == null) instance = new Singleton();
         return instance;
    }
    private Singleton() {}
}
```

Der Zugriff auf die Singleton-Instanz muss synchronisiert sein, sonst können mehrere Objekte erzeugt werden. Erstellen Sie einen Ablauf, bei dem zwei Threads „gleichzeitig“ versuchen, ein Singleton zu erzeugen und dabei so unglücklich agieren, dass am Ende zwei Instanzen entstehen. Betrachten Sie dazu nur die `getInstance()`-Methode



Wie sieht denn eine „thread-safe“ Lösung der Lazy-Initialisierungs-Variante aus?