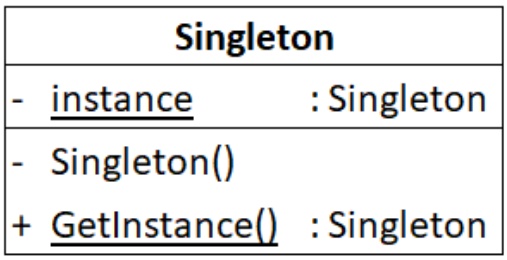
Singleton

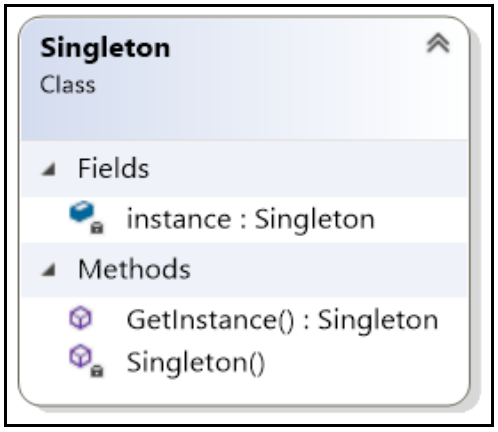
1 Singleton

Das Singleton-Pattern ist eines der einfachsten Designpatterns und zählt zu den Erzeugungsmustern.
Es stellt sicher, dass von einer Klasse **nur eine einzige Instanz** erzeugt werden kann. Diese Instanz wird üblicherweise erst bei der ersten Verwendung tatsächlich erzeugt (**Lazy Loading**).

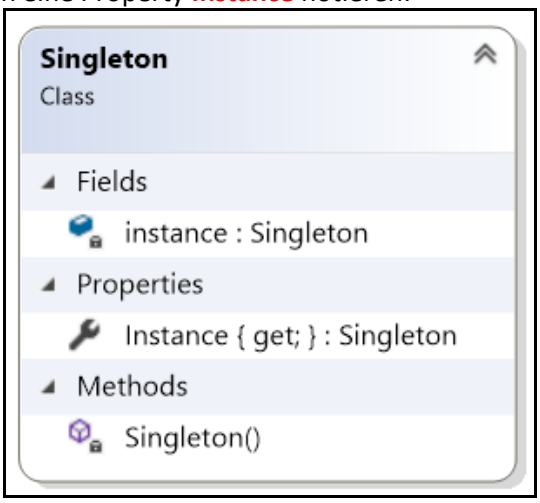
1.1 UML



In C# kann man anstelle von **GetInstance()** auch eine Property **Instance** notieren:



bzw.



Es gibt also nur **private Konstruktoren** – man muss also gegebenenfalls auch den Defaultkonstruktor mit einem privaten Konstruktor ersetzen!

1.2 Erkennungsmerkmale

An folgenden Merkmalen kann man den Einsatz eines Singletons erkennen:

- Man braucht ein bestimmtes Objekt in vielen unterschiedlichen Klassen
- Ein Objekt wird in den Konstruktoren vieler Klassen benötigt
- Die Erstellung eines Objekts kostet viele Ressourcen (ist also „schwergewichtig“), wird aber oft benötigt.
- Ein schwergewichtiges Objekt wird möglicherweise nie verwendet

1.3 Vorgangsweise

Dieses Pattern kann auch nachträglich sehr leicht umgesetzt werden, weil es eine konkrete Vorgangsweise gibt:

1. Statische Instanz definieren (private)
2. Statischen getter (public) für diese Instanz
    a. Keinen setter erzeugen
    b. Beim ersten Zugriff über getter die Instanz erzeugen
3. Konstruktor als private notieren

1.4 Beispiele

In folgenden Beispielen würde sich ein Singleton anbieten:

- Zugriff auf System-Ressourcen

- Datenbankkontext: wenn derselbe Datensatz mit verschiedenen DbContext-Objekten verändert wird, gehen Änderungen verloren
- Bereitstellung eines Verteilmechanismus (Subject in Observer-Pattern)
- Klasse, in der die Applikationskonfiguration gespeichert wird. In WPF also z.B. ein Config-Window.
- Klasse zur Produktion von eindeutigen IDs

1.5 Vor-/Nachteile

1.5.1 Vorteile

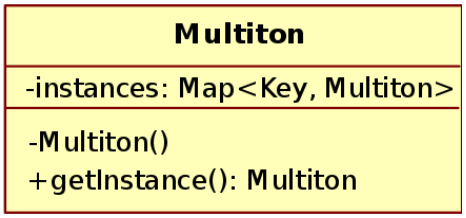
- Man spart sich durch das Singleton die oftmals lästige Parameterübergabe eines bestimmten Objekts an vielen Stellen/Klassen/Methoden seines Projekts.
- Man hat Kontrolle über den Zugriff auf dieses Objekt.
- Lazy Loading: Objekt wird erst erzeugt, wenn es tatsächlich gebraucht wird, also möglicherweise gar nicht.

1.5.2 Nachteile

- Man läuft in Gefahr, dass man zu viel in die Singleton-Klasse verpackt und somit praktisch einfach viele globale Variablen zusammenfasst.
- Das Singleton ist überall in der Applikation verfügbar.
- Man erkennt aus den Schnittstellen einer Klasse nicht mehr, ob das Singleton verwendet wird.
- Testen mit Singletons wird schwieriger, weil sich ein Singleton nicht so einfach durch ein Mock-Objekt ersetzen lässt.
- Der Benutzer hat keine Kontrolle über die Lebensdauer des Singelton, d.h. er kann in der Regel die Ressourcen nicht mehr freigeben.

1.6 Multiton

Gehört nicht zu den ursprünglichen 23 Designpatterns der „Gang of Four“. Es ist praktisch eine minimale Erweiterung der Singleton-Idee.
Manchmal braucht man nicht genau eine Instanz einer Klasse, sondern eine begrenzte Anzahl (aber nicht beliebig viele). Dazu speichert man die einzelnen Singeltons mit einem Key in einer Map:



Als Anwendung kann man sich einen Datenbank-Pool vorstellen, in dem es pro Connectionstring genau ein DbContext-Objekt gibt.