**Singleton Design Pattern**

Merve KOC

Inhaltsverzeichnis

1. Einleitung
2. Singleton Pattern Klassen Diagramm
3. Beispiele und Anwendungsgebiete
4. Vorteilen und Nachteilen
5. Multithreading und Singleton
6. Double-Checked Locking
7. Load Balancing (Lastverteilung)

**Einleitung**

* Das Singleton-Pattern wird oft in Softwareprojekten verwendet, um sicherzustellen, dass es nur eine Instanz einer Klasse gibt.
* Das Ziel des Singleton-Patterns ist es, eine kontrollierte, globale Instanz zu haben.

Bei normalen Klassen können Sie beliebig viele neue Instanzen erstellen.

Zum Beispiel:



Im Beispiel wird die Klasse Car zweimal instanziiert, wodurch car1 und car2 zwei verschiedene Objekte darstellen. Sie sind voneinander unabhängig.

Beim Singleton-Design-Pattern ist dies anders: Es wird nur eine einzige Instanz erstellt. Wir verwenden es im gesamten Programm.



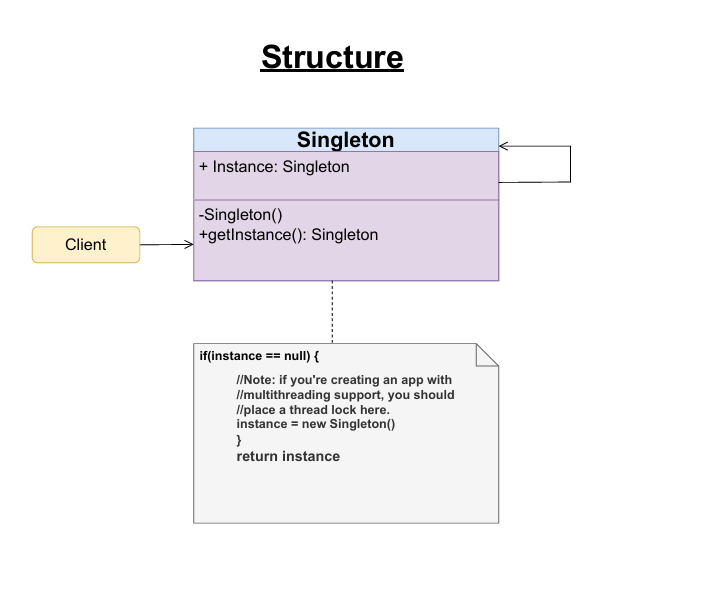
In diesem Code haben wir eine Singleton Klasse.

Der Konstruktor ist privat, das heißt, niemand kann von außen ein neues Objekt davon erstellen.

Nur die Methode getInstance() kann das tun. getInstance() prüft, ob die Instanz bereits existiert. Wenn nicht, wird sie erstellt. Und danach wird immer dieselbe Instanz zurückgegeben, egal wie oft die Methode aufgerufen wird.

Auch wenn wir unterschiedliche Namen wie firstInstance und secondInstance benutzen, gibt getInstance() immer dieselbe Instanz zurück.

**2.Singleton Pattern Klassen Diagramm**



+ (Public): Diesen Public Variable instance ist static, damit sie zur Klasse gehört und nur einmal erstellt wird.

(Private): Diese Private Singleton Methode ist nur innerhalb der Klasse zugänglich.

der Konstruktor ist privat, damit keine anderen Objekte erstellt werden können.

+ (Public): Die Methode getInstance() ist public, damit jeder im Programm darauf zugreifen kann.

**Verbindung zwischen Client und Singleton:**

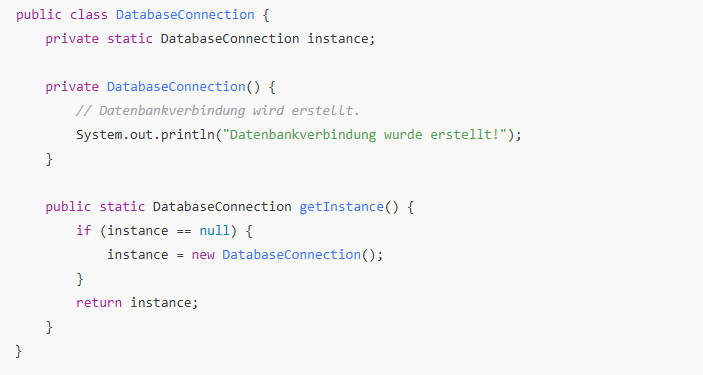
Der Client kommuniziert immer mit derselben Instanz von Singleton, egal wie oft getInstance() aufgerufen wird.

**Voller Pfeil (Client → Singleton):** Zeigt, dass der Client die Singleton-Instanz benutzt. Das bedeutet, der Client greift auf die Methode getInstance() zu.

**Gestrichelter Pfeil:** zeigt, dass getInstance() eine neue Instanz erstellt, wenn noch keine vorhanden ist. Wenn instance schon vorhanden ist, wird diese zurückgegeben

**3.Beispiele und Anwendungsgebiete**

* **Situation1=> DataBase Connection:** Stellen Sie sich vor, eine Anwendung muss sich mit einer Datenbank verbinden, um Daten zu lesen und zu schreiben.
* **Problem:** Wenn jedes Mal eine neue Datenbankverbindung erstellt wird, kann das zu Effizienz- und Leistungsproblemen führen. Jede Verbindung verbraucht zusätzliche Ressourcen.
* **Lösung:** Die Datenbankverbindung wird als Singleton erstellt und von der gesamten Anwendung genutzt. Dadurch kann bei jeder Datenabfrage dieselbe Verbindung genutzt werden, statt jedes Mal eine neue zu erstellen.
* So wird anstelle einer neuen Verbindung jedes Mal dieselbe Verbindung immer wieder verwendet.



Es wird nur ein einziges Mal eine Instanz der Klasse DatabaseConnection erstellt, und diese Instanz wird mit der Methode **getInstance()** zurückgegeben. Dadurch kann im gesamten Programm dieselbe Verbindung genutzt werden, und es wird keine neue Verbindung erstellt.

* **private static**: Es wird nur eine Instanz (instance) erstellt.
* **private Konstruktor**: Verhindert, dass andere Klassen eine neue Instanz erstellen.
* **getInstance()**: Wenn instance null ist, wird eine neue Verbindung erstellt, ansonsten wird die bestehende zurückgegeben.

Das sorgt dafür, dass die Datenbankverbindung kontrolliert und nur einmal verwendet wird.

* **Situation2 => Logging:** Eine Applikation muss Logs speichern, um Aktionen und Fehler zu dokumentieren.
* **Problem:** Wenn mehrere Log-Objekte erstellt werden, können die Logs unordentlich werden und das System kompliziert machen.
* **Lösung:** Wenn die Log-Klasse als Singleton gemacht wird, gibt es nur einen zentralen Punkt für die Log-Aufzeichnung. Alle Aktionen der Anwendung werden in einer einzigen Log-Datei gespeichert, sodass man die Logs leicht verfolgen kann.
* Alle Logs werden an derselben Stelle gesammelt.



Die Klasse **Logger** verwendet das **Singleton-Pattern**, um sicherzustellen, dass nur eine einzige Instanz der Logger-Klasse existiert.

* Es gibt ein privates, statisches Feld instance, das die einzige Instanz speichert.
* Der Konstruktor der Klasse ist privat, sodass keine weiteren Instanzen erstellt werden können.
* Die statische Methode **getInstance()** überprüft, ob die Instanz bereits existiert (ob instance == null ist). Wenn sie noch nicht existiert, wird eine neue Instanz erstellt. Diese Methode sorgt dafür, dass immer die gleiche Instanz zurückgegeben wird.
* Die Methode **log()** wird verwendet, um eine Nachricht auszugeben

**4.Vorteilen und Nachteilen**

**a) Vorteilen:**

* Das Singleton-Pattern bietet eine kontrollierte Art und Weise, globale Zustände zu verwalten.
* Es spart Speicherplatz, da nur eine Instanz erstellt wird.

**b) Nachteilen:**

* Es könnte zu Problemen führen, wenn mehrere Threads gleichzeitig auf die Instanz zugreifen.

Wie können wir es lösen?

-Multithreaded Umgebung

* Wenn viele Benutzer gleichzeitig auf eine einzige Verbindung zugreifen, kann es zu Überlastung und Wartezeiten kommen.

Wie können wir es lösen?

-Load Balancing

**5.Multithreading und Singleton**

* In einer Multithreaded Umgebung kann es passieren, dass mehrere Threads gleichzeitig auf 'getInstance()' zugreifen wollen.
* Zum Beispiel bei einem Webserver: Viele Nutzer senden gleichzeitig Anfragen an den Server, und für jede Anfrage wird ein eigener Thread verwendet.
* Das kann zu mehreren Instanzen führen, was das Singleton-Muster bricht.
* Mit dem 'synchronized' Schlüsselwort stellen wir sicher, dass immer nur ein Thread gleichzeitig die Methode aufrufen kann, und so bleibt die Klasse thread-sicher.

**6.Double-Checked Locking:**

Double-Checked Locking reduziert Wartezeiten, weil nur beim ersten Zugriff ein Lock benutzt wird. Danach ist die Instanz schon da, und alle Threads können schnell darauf zugreifen.

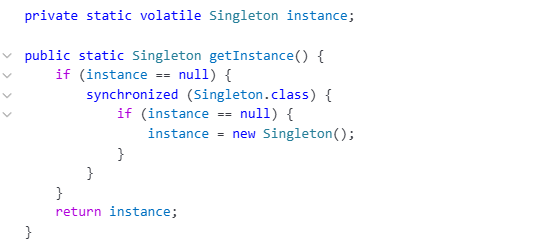
Erste Überprüfung:

Wenn die Instanz schon existiert, wird kein Lock benutzt.

Synchronized Kontrolle:

Wenn null, wird der synchronized-Block benutzt und nochmal geprüft, ob die Instanz wirklich null ist. Nur dann wird eine Instanz erstellt.

Das bedeutet: Beim ersten Zugriff wird die Instanz mit einem Lock erstellt. Bei den nächsten Zugriffen ist die Instanz schon da, und es braucht keinen Lock mehr. So wird die Leistung besser.



**7.Load Balancing (Lastverteilung)**

* Wenn viele Nutzer auf die Datenbank zugreifen, hilft der Load Balancer.
* Der Load Balancer verteilt die Anfragen auf verschiedene Server.
* So wird die Last gleichmäßig verteilt.
* Das verbessert die Leistung und verhindert Überlastung.

**REFERENZENVERZEICHNIS**

1. <https://refactoring.guru/design-patterns/singleton>
2. <https://www.geeksforgeeks.org/singleton-design-pattern/?ref=gcse_outind>
3. <https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm>
4. https://docs.oracle.com/javaee/7/tutorial/ejb-basicexamples002.htm