Singleton ID

Student:

Izabela Kuźniar

Teacher:

Andrea Corradini

Course:

Software Design Patterns

# Name and category

Singleton is a creational pattern, because we create an object even though it is supposed to be just one.

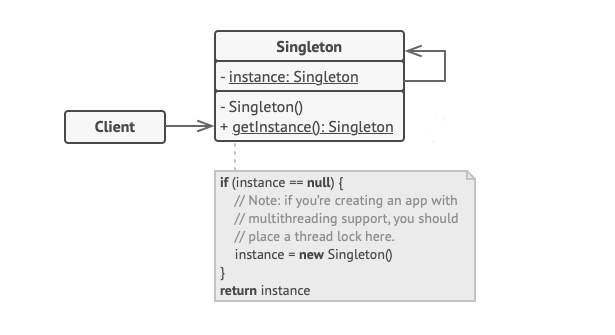
# Intent:

Singleton allows program to create only one instance of itself, it-s purpose is to control object creation.

# Motivation:

Singleton is used when we want to manage the access to a resource shared by the entire application. It would make errors if we would have multiple instances. For example a coffee machine, if it would be getting few requests at the time it could be a disaster.

# Structure as a UML class diagram



# Implementation:

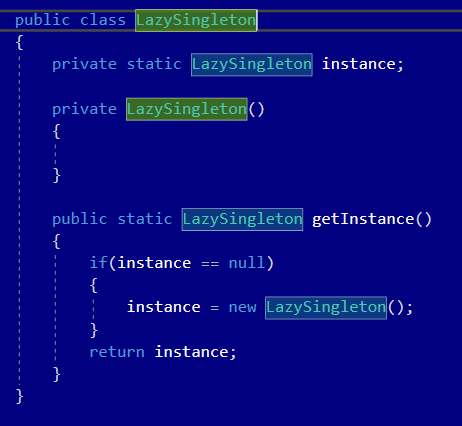
Singleton is defined only when BOTH of those conditions are fulfilled:

1. One instance only can be created
2. There is only one access point

There are different ways to implement a Singleton that could influence security of a system.

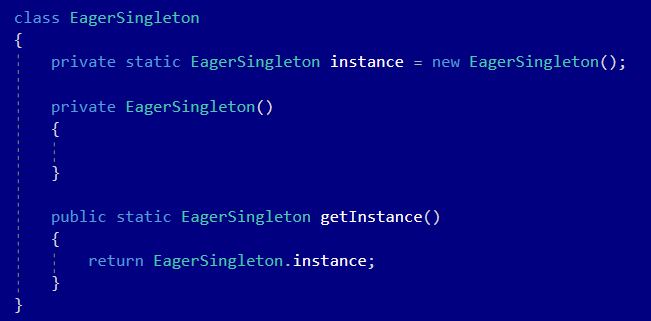
Lazy Singleton (it is waiting until you really need it):

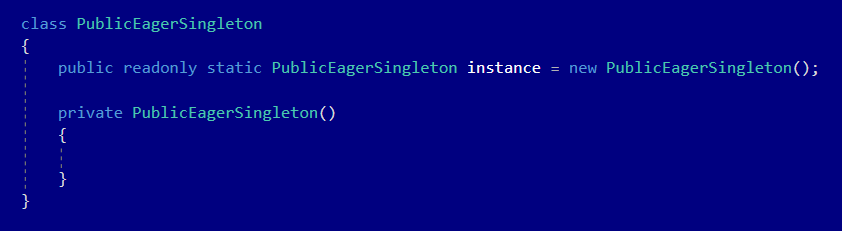
* PRIVATE constructor – this does not allow create new objects in other classes by using “new” operator
* PRIVATE STATIC variable of Singleton instance
* PUBLIC method to get the instance – as the only access point



Eager Singleton (gets initialized as soon as the program runs):

* PRIVATE constructor – this does not allow create new objects in other classes by using “new” operator
* PUBLIC (then no need for PUBLIC method to get Instance)/PRIVATE (still needed PUBLIC method to get instance) READONLY variable of Singleton instance





# Consequences:

Benefits:

* If Lazy Singleton we wait until we need it
* If Eager Singleton – thread safe
* You are sure that the class has only one instance
* You have a global access point

Drawbacks:

* If Lazy Singleton is not thread safe.
* Singleton violates the single responsibility principle: they control their own creation and lifecycle.
* Cause the code to be tightly coupled: makes them difficult to fake during the tests

# Known uses

* Connection to database
* Registry in Windows
* System to log

# Related patterns

1. Façade: can be transformed into a singleton, because one object of the façade is enough mostly.
2. Flyweight could resemble Singleton if we would reduce all shared states.
3. Abstract Factories, Builders and Prototypes can all be implemented as singleton.