Design Patterns - Singleton Pattern

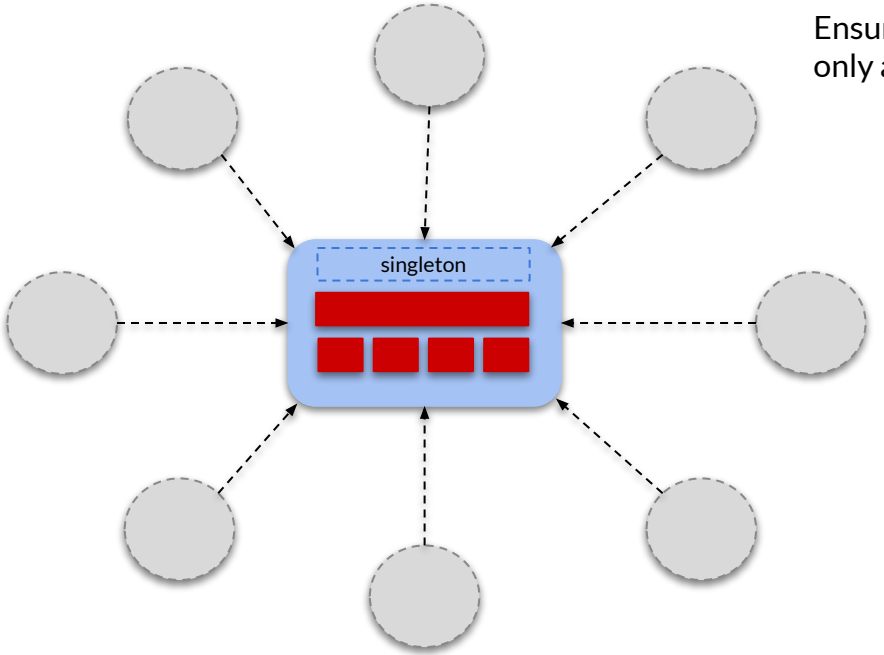
Singleton is a **creational design pattern** that ensures that a class has only one instance and provides an **easy** global access to that instance.

The pattern does not stipulate what to do with a **Singleton**. This is where you can be creative.

The main tenets of this pattern are:

1. Ensure that the class has only a single instance.
2. Provide easy global access to this instance.
3. Control how it is instantiated.
4. Any critical region must be entered serially.

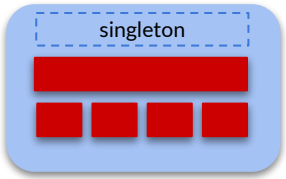
Design Patterns - Singleton Pattern



Ensure that the class has only a **single instance**.

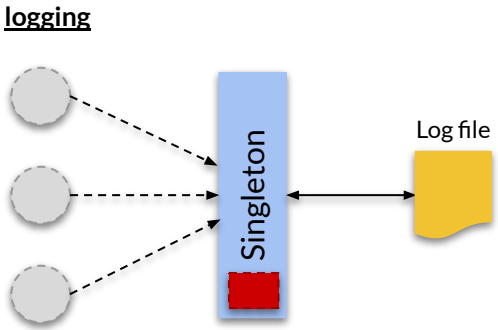
Design Patterns - Singleton Pattern

What can we do inside a Singleton? What is it good for?

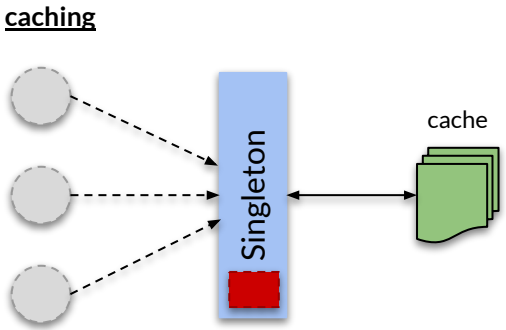


- Loggers
- Caching
- Thread Pools
- Database Connections
- Configuration Access

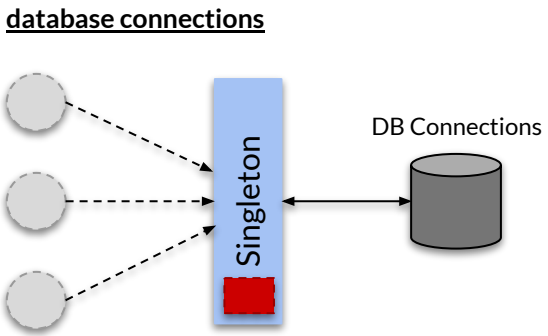
Design Patterns - Singleton Pattern



Design Patterns - Singleton Pattern



Design Patterns - Singleton Pattern



Design Patterns - Singleton Pattern

This pattern is one of the famous **GoF** patterns and its motivation is stated as follows:

> *It's important for some classes to have exactly one instance. Although there can be many printers in a system, there should be only one printer spooler. There should be only one file system and one window manager…*

This pattern is often used with the following GoF patterns:

1. Abstract Factory
2. **Builder**
3. Prototype
4. Facade
5. **State**

Design Patterns - Singleton Pattern

When to use:

1. When you want to control access to a shared resource.

When not to use:

1. Use the Singleton pattern with restraint and do not let it degenerate into just a global access for everything. Global access hides dependencies and might make it harder to read such code, so make sure that you have a good reason to use this pattern.
2. The main question you should ask is: do you violate the SRP (Single Responsibility Principle?) If YES then reconsider using it.

Design Patterns - Singleton Pattern

Design Consideration:

1. When designing a Singleton, consider **lazy construction** which means that the class instance should only be created when it is first needed. In some cases we might consider **eager loading** if, for example, we need the singleton to be always ready and loaded fast.

2. **Thread-Safety** needs to be considered in languages that allow multi-threaded access to ensure that access is properly controlled and locked, so that the state of the singleton (if it has one) is always deterministic.
   a. In Dart and Flutter we do not worry about thread-safety since Dart uses 'isolates' and we simply instantiate the instance in Main isolate.