

9. Using the same single instance of an object throughout the application

Imagine that you have a requirement to use exactly the same object instance throughout your entire application, regardless of whether the classes that use it can communicate directly with each other or not.

One thing you can do is instantiate it in one place and then pass the instance to every single class. This will work, but it's less than desirable. It means that you will have to write a lot of extra code that will be hard to read and maintain.

Plus, there is nothing that would stop passing a different object reference into a class that uses it, especially if a developer who is currently working on the code doesn't know that the original intention was to keep the reference the same across the board. This will probably introduce some bugs.

Suitable design patterns

Singleton

With this design pattern, the class that you want to have a single instance of will have a static method that will return an instance of this class. Behind the scenes, this method will call a private

constructor the first time you call it, so the class will be instantiated only once. If you call this method again, then the same instance will be returned as was instantiated before.

Otherwise, the class will have no public constructor, so it will not be possible to create a new instance of it by any external code. This will guarantee that no matter where you are calling the static method from, exactly the same instance of the class will be returned.

Why would you want to use Singleton

- Being able to use the same instance of an object in any part of your application without explicitly passing an instantiated object.
- It will prevent you from creating more than one instance of a particular type.

Object Pool

We have already covered Object Pool in [chapter 8](#) as a design pattern primarily intended for managing multiple instances of the same object type without incurring much of performance penalty. However, in order to be accessible throughout the application, the Object Pool also needs to be a singleton.

So, in a nutshell, to make the most of the Object Pool, the class that acts as an Object Pool must also implement the Singleton design pattern.

Why would you use Object Pool as Singleton

- Your entire application shares the same pool of objects.
- You don't have to instantiate more object instances than strictly needed.