## Strategy Pattern

### Definition

The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance

### Meaning

From the internet we get results similar to ‘A singleton should be used when managing access to a resource which is shared by the entire application’, sometime we get examples of network sockets, database connections (though this isn’t really an issue any more) and resource management. Imagine if we have a huge image, we don’t really want to have multiple instances of the image in memory, instead we can have the image instantiated (stored in memory) once and make sure everyone who needs to use (reference) the image look at the same are of memory. The singleton allows us to do this.

Sometimes this is hard to visualise this type of explanation, and if you’re not familiar it’s difficult to understand the actual concept behind it. A real world example can be found at [Using the Singleton Pattern to Manage Unmanaged Resources - CodeProject](https://www.codeproject.com/Articles/15773/Using-the-Singleton-Pattern-to-Manage-Unmanaged-Re), but I’ve tried to make an example using things you can visualise.

Normally we can have multiple instances of the same class and each instance of the class looks after its own set of data, so we could have 2 identical machines represented as 2 separate instances of the same class. At any time, one machine might have 50 widgets in it, the other might have 100 widgets in it. As we have two instances of the class, one per machine the data is kept separate.

Now, if we have a single bucket of widgets that are used to feed both the machines, we only ever want to have one instance of that bucket class, if either machine takes a widget, the bucket total widgets gets reduced by a value of 1 until its empty.

So if Machine 1 makes an instance of the bucket and declares it to have 50 widgets in it, how does machine 2 get hold of the same bucket without creating its own instance. In our example there are other ways of doing it, creating a single bucket object and passing it into each machine, and having them both work on it, but in more complex systems it gets much more complicated, how does a programmer know if a bucket exists already, how do they ensure they don’t create a second bucket by accident etc.

The singleton pattern fixes this issue, the first time a singleton class is called, its code creates an instance of itself is memory. Normally, if a class is instantiated again, a second copy of itself is created in memory, then a third, fourth etc… With a singleton, the second, third calls do not create new instances, they return back references (pointers) to the first and only instance in memory.

To the programmer, it might look like they have multiple instances but actually they are all looking at the same object in memory.

So Machine 1 creates a variable Machine1Bucket of type Singleton Bucket that creates an object in memory.

Machine 2 uses the same class to creates a variable also of type Singleton Bucket which it calls Machine2Bucket, but instead of the Bucket Singleton class creating a new bucket in memory for machine 2, the variable it returns references (points to) the same memory as bucket 1, so if either bucket changes, they both see the same changes to the data via their own variable.

This way the programmer can ensure both machines are looking at the same bucket.

Singleton classes cannot be instantiated (as instantiation implies creating a new object) or inherited from (as inherited objects can be instantiated). This is done by the use of private constructors, static and sealed classes, which also inform the ide during development if the programmer is doing things wrong. Making the constructor private says to the compiler I have a constructor so don’t make a new one, but because its private, don’t let any one else use it.

There are simple methods for creating singletons, all over the internet, but there are issues if there are multiple threads all calling for the creation of the first singleton. The example in the associated code solves this issue by making use of the Static constructor.

The example I’ve used is from <https://csharpindepth.com/Articles/Singleton> and is by Jon Skeet. There’s some quite advances but interesting text on how it all works.

### Example

See documented code.