**Singleton Design Pattern**

Make the class of the single instance object responsible for creation, initialization, access, and enforcement. Declare the instance as a private static data member. Provide a public static member function that encapsulates all initialization code, and provides access to the instance.

The client calls the accessor function (using the class name and scope resolution operator) whenever a reference to the single instance is required.

Singleton should be considered only if all three of the following criteria are satisfied:

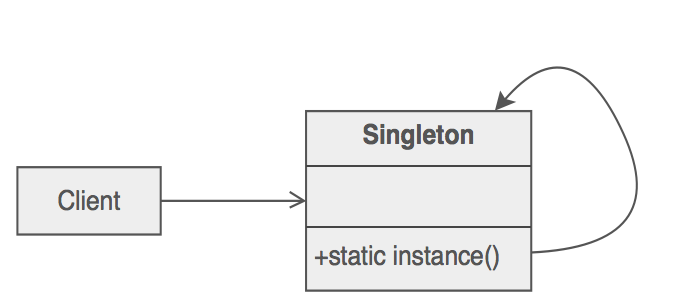
* Ownership of the single instance cannot be reasonably assigned
* Lazy initialization is desirable
* Global access is not otherwise provided for

If ownership of the single instance, when and how initialization occurs, and global access are not issues, Singleton is not sufficiently interesting.

The Singleton pattern can be extended to support access to an application-specific number of instances.

The "static member function accessor" approach will not support subclassing of the Singleton class. If subclassing is desired, refer to the discussion in the book.

**Structure**



Make the class of the single instance responsible for access and "initialization on first use". The single instance is a private static attribute. The accessor function is a public static method.



**Example:**

I needed to use the NSNumberFormatter class to change an integer like “20” into a currency like “$20.00” for a table view I was displaying. Because initializing an NSNumberFormatter can be an expensive operation, especially if I was going to be using the same NSNumberFormatter configuration throughout the app. So, I decided to use a singleton. This project was zipped and attached with this file.