The Singleton Pattern – creating one, and only one, object

Why would we want to do this?

- Queues, caches, registery settings, device drivers, etc., often need a single instantiation of an object
  - For example, there should be a single device driver object for a device
  - We want a single cache for data read from disk so that we don't have duplicate copies of data
- In complicated programs, managing this can be difficult
  - Can use a global variable
  - Global variables are bad – they require code to ensure that it is initialized before any object want to access it. This can be hard in a complicated program
- By using a private constructor for the singleton object, we can avoid the global variable
- The Singleton pattern insures the single object is initialized whenever we first desire to access it

Singleton code (See Example1 code)

```
class Singleton {
private:
  static Singleton* uniqueInstance = nullptr;
  // other instance variables here
  Singleton( ); // could have arguments
public:
  static Singleton* getInstance( );
};

Singleton::Singleton( ) {. . .}
Singleton* Singleton::getInstance( ) {
  if (uniqueInstance == nullptr) {
    uniqueInstance = new Singleton( );
  }
  return uniqueInstance;
}
```

`uniqueinstance` refers to the single instance of a Singleton, if it exists

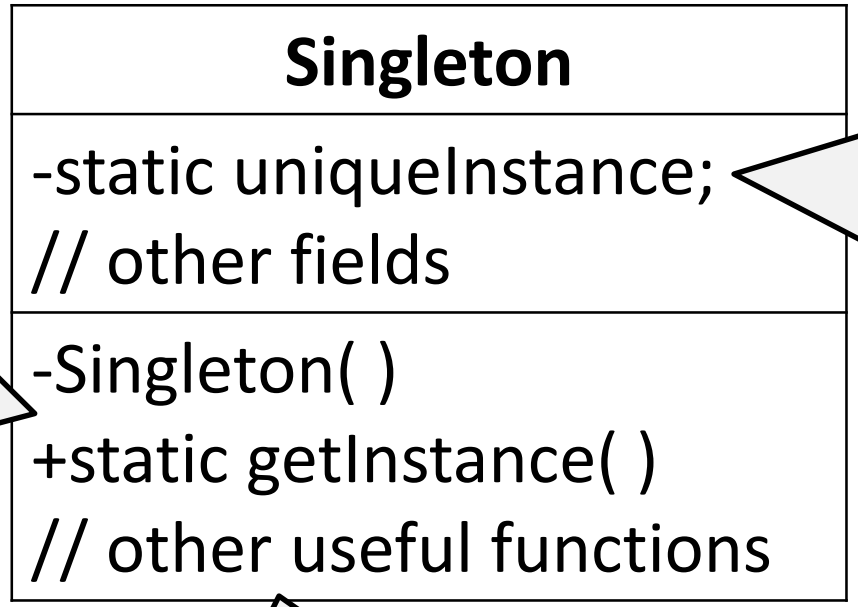
Making the constructor private means that only functions in the class can construct a singleton

`getInstance` allows access to the single object. If it has not been allocated and initialized, getInstance will do that.

Singleton is otherwise a normal class and can have other functions.

The Singleton Pattern

getInstance( ) is a static which allows access from anywhere. It is like a global variable, but supports *lazy* instantiation.



The static `uniqueInstance` variable holds a reference to the one and only singleton object

The class implementing the Singleton pattern both supports the pattern and is a regular class that does non-singleton things, like be a device driver.

Some considerations

- Why is having a Singleton class better than a global variable?
  - Global variables alone don't prevent multiple objects being created
  - Broken windows argument: Global variables tempt developers to pollute the namespace with lots of references to the global variable. Here we can use `Singleton.getInstance( )` instead of referencing the global variable.
  - Broken windows argument 2: once we do sloppy things in our code, like global variables, later programmers will be more likely to do sloppy things.
- Singleton violates the "One class, one responsibility" principle. There's the purpose of the class, and the singleton responsibility. Putting these into a single class simplifies things enough to make this reasonable.
- You can't derive from a Singleton class because its constructor is private.