Dissecting the classic Singleton Pattern implementation

Watch it!

If you're just flipping through the book, don't blindly type in this code; you'll see it has a few issues later in the chapter.

Let's rename MyClass to Singleton.

We have a static variable to hold our one instance of the class Singleton.

```
public class Singleton {
    private static Singleton uniqueInstance;

    // other useful instance variables here

    private Singleton() {}

    public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }

    // other useful methods here
}
```

Our constructor is declared private; only Singleton can instantiate this class!

The getInstance() method gives us a way to instantiate the class and also to return an instance of it.

Of course, Singleton is a normal class; it has other useful instance variables and methods.

Code Up Close

uniqueInstance holds our ONE instance; remember, it is a static variable.

If uniqueInstance is null, then we haven't created the instance yet...

...and, if it doesn't exist, we instantiate Singleton through its private constructor and assign it to uniqueInstance. Note that if we never need the instance, it never gets created; this is lazy instantiation.

```
if (uniqueInstance == null) {
    uniqueInstance = new Singleton();
}
return uniqueInstance;
```

If uniqueInstance wasn't null, then it was previously created. We just fall through to the return statement.

By the time we hit this code, we have an instance and we return it.