

## Singleton Pattern defined

Now that you've got the classic implementation of Singleton in your head, it's time to sit back, enjoy a bar of chocolate, and check out the finer points of the Singleton Pattern.

Let's start with the concise definition of the pattern:

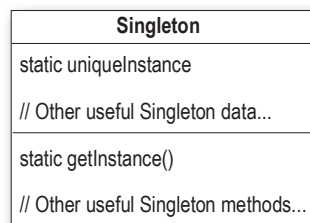
**The Singleton Pattern** ensures a class has only one instance, and provides a global point of access to it.

No big surprises there. But let's break it down a bit more:

- What's really going on here? We're taking a class and letting it manage a single instance of itself. We're also preventing any other class from creating a new instance on its own. To get an instance, you've got to go through the class itself.
- We're also providing a global access point to the instance: whenever you need an instance, just query the class and it will hand you back the single instance. As you've seen, we can implement this so that the Singleton is created in a lazy manner, which is especially important for resource-intensive objects.

Okay, let's check out the class diagram:

The `getInstance()` method is static, which means it's a class method, so you can conveniently access this method from anywhere in your code using `Singleton.getInstance()`. That's just as easy as accessing a global variable, but we get benefits like lazy instantiation from the Singleton.



The `uniqueInstance` class variable holds our one and only instance of Singleton.

A class implementing the Singleton Pattern is more than a Singleton; it is a general-purpose class with its own set of data and methods.