Removing Singletons

Adrian Kosmaczewski

2021-07-16

Problem: your code has a big badass manager class with a singleton interface, and you would like to have more flexible, testable code.

The code samples in this article are in Swift, but the concepts translate to pretty much any modern language.

```
class Manager {
    static let singleton = Manager()
}
```

Recipe

Create a protocol called `ManagerProtocol` (Objective-C, Swift) or an interface `ManagerInterface` (Kotlin, Java, C#), or an abstract class with pure virtual methods `IManager` (C++) to contain the signatures of the public properties of your manager.

```
protocol ManagerProtocol {}
```

Make the singleton return a `ManagerProtocol`, instead of a `Manager`.Make your class comply to the protocol; that is, implement the interface.

```
class Manager: ManagerProtocol {
    static let singleton: ManagerProtocol = Manager()
}
```

Now your code will most probably not compile; add the signatures of the public methods and properties that you client code requires to the protocol / interface until everything compiles again.

In the classes that use the singleton, create a public property of type `ManagerProtocol` and initialize it to the value of the singleton at initialization or construction time: in other words, turn this

```
class Client {
    func method() {
        Manager.singleton.doThat()
    }
```

```
}
```

into this

```
class Client {
    var manager: ManagerProtocol = Manager.singleton

    func method() {
        self.manager.doThat()
    }
}
```

Replace every instance of the singleton call with a call to the ivar.

Result

Now your components are decoupled; you can replace the Manager class at runtime with another object, and this is useful for testing your code. It can also be useful at runtime, to switch the behaviour of your code from one implementation to another if needed.

Another side effect is that you have a nice interface / protocol that describes your "manager" as an abstract entity, and this is a powerful tool for documenting the code and establishing relationships among things.