Design Pattern: Singleton

The singleton pattern is a creational pattern that ensures that a class has only one instance and provides a global point of access to it. This can improve performance as it only need to be instantiated once and has consistent behavior.

How does the singleton pattern work?

This works by hiding the constructor so we can't make an object of it, i.e., `private`. By making the class `final` we can make sure that it cant be extended. We then make a `public static` method that returns the instance of the class. This method checks if the instance is null and if it is it creates a new instance. This is called lazy loading. If we want to make sure that the instance is created as soon as the class is loaded we can use eager loading.

Because you can't make an object of it to access the methods you have to use the static method to access the methods. This means that you can't use polymorphism with a singleton.

Why are singletons controversial?

The singleton pattern is controversial as some people consider it an anti-pattern. As programmers one of the first things we learn is to avoid globals. Using globals is bad because it makes the app harder to control and reason about.

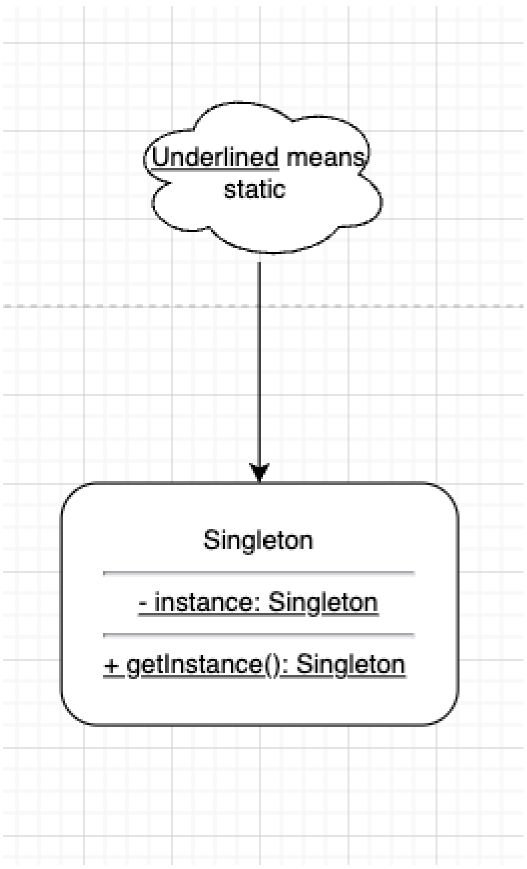
Singleton also makes testing difficult as it is hard to mock. Mocking is when you isolate the code being tested from its dependencies, so that the behavior of the code can be tested in separately. As you can't mock a singleton it makes the things that use it harder to test.

When might you use a singleton?

Database connection. We only want one connection to the database as it is expensive to make a connection (authentication and initialization). With things like databases we want to make sure that we only have one connection as we don't want to have multiple connections to the same database, leading to conflict if you are trying to change multiple field at the same time.

Another example is in a logging system, you typically want to have a single point of access to the log file or log database from anywhere in your application. By using the Singleton pattern, you can ensure that there is only one instance of the logging class throughout your application's runtime.

UML Diagram



Example of a singleton

```
public final class Database{

    // private static Database instance = new Database(); //This be
constructed as soon as the class is loaded to fix this we can use lazy
loading, currently this is called eager loading

    private static Database instance = null;

    if (instance == null){ // we only create a new instance of the
database if it is null.
        instance = new Database(); //This is called lazy loading
    }

    public static Database getInstance(){
        return instance;
    }

    private Database(){} //Hides the constructor
}
```

```
Database database1 = Database.getInstance();
Database database2 = Database.getInstance();
```

```
System.out.println(database1 == database2); //true
```

Eger vs Lazy Loading

There are benefits to both eager and lazy loading. Eager loading is faster as it is already constructed when it is needed (first the program runs) , but takes more resources when the application starts. Lazy loading is slower as it is constructed when it is needed (when the static method is called).

The Singleton Pattern can also be made with enums

```
public enum Database{
    INSTANCE;

    public void connect(){
        System.out.println("Connected");
    }
}
```

Enum is a special class in java that can only have one instance. This is a thread safe way of making a singleton. This is because enums are thread safe and can only be instantiated once. Enums only allow for eger loading.

Enums by default are static and final. This mean that they can't be extended and can only be instantiated once.

```
Database database1 = Database.INSTANCE;
Database database2 = Database.INSTANCE;

System.out.println(database1 == database2); //true
```