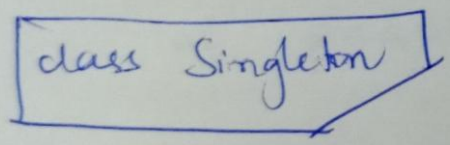
Singleton :-

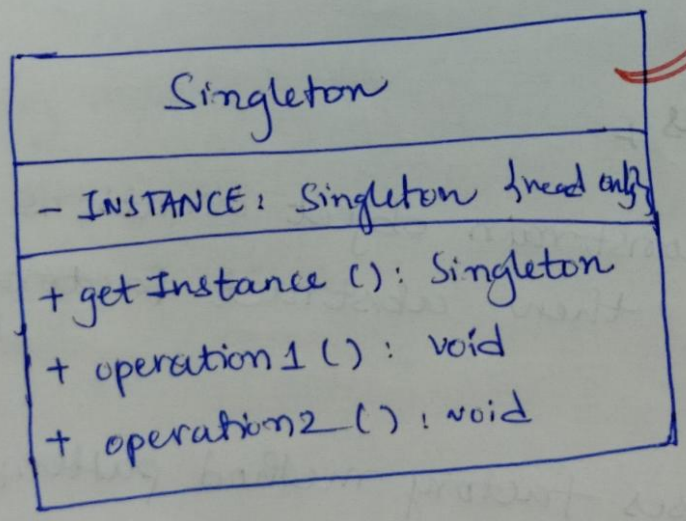
What is Singleton ?

→ A singleton class has only one instance, accessible globally through a single point (via a method / field).

→ Main pblm this pattern solves is to ensure that only a single instance of this class exists.

→ Any state you add in your singleton becomes a part of "global state" of your application





Role: Singleton
- responsible for creating unique instance
- Provides static method to get the instance.

Implement a Singleton :-

- Controlling instance creation
  - class constructor (s) must not be accessible globally
  - subclassing / inheritance must not be allowed.

- Keeping track of instance
  - class itself is a good place to track the instance

- Giving access to the singleton instance
  - A public static method is good choice.
  - Can expose instance as final ~~protect~~ public static field but it won't work for all singleton implementation

⇝ Two options for implementing a singleton

- Early initialisation - Eager Singleton
  - Create singleton as soon as class is loaded.
- Lazy initialisation - Lazy singleton
  - Singleton is created when it is first required.

Implementation Considerations :

- "Early / eager initialisation is the simplest & preferred way. Always try to use this approach first.
- The "classic" singleton pattern implementation uses double check locking and volatile field.
- The lazy initialization holder idiom provides best of both worlds, you don't deal with synchronization issues directly and is easy to implement.

Design Considerations :-

- Singleton creation does not need any parameters. If you find yourself in need of support for constructor arguments, you need a simple factory or factory method pattern instead.
- Make sure that your singletons are not carrying a lot of mutable global state.

Pitfalls :-

- Singleton pattern can deceive you about tree dependencies! Since they are globally accessible its easy to miss dependencies.
- They are hard to unit test. You cannot easily mock the instance that is returned.
- The most common way to implement Singleton in Java is through static variables and they are held per class loader and not per JVM. So they may not be truly Singleton in an OSGi or web application.
- A singleton carrying around a large number of mutal global state is a good indication of an abused Singleton pattern.