**Singleton Design Pattern:** Singleton Pattern is probably the most widely used design pattern. It is a simple pattern, easy to understand and to use. Sometimes it is used in excess and in scenarios where it is not required. In such cases, the disadvantages of using it outweigh the advantages it brings. For this reason, the singleton pattern is sometimes considered an antipattern or pattern singleton.

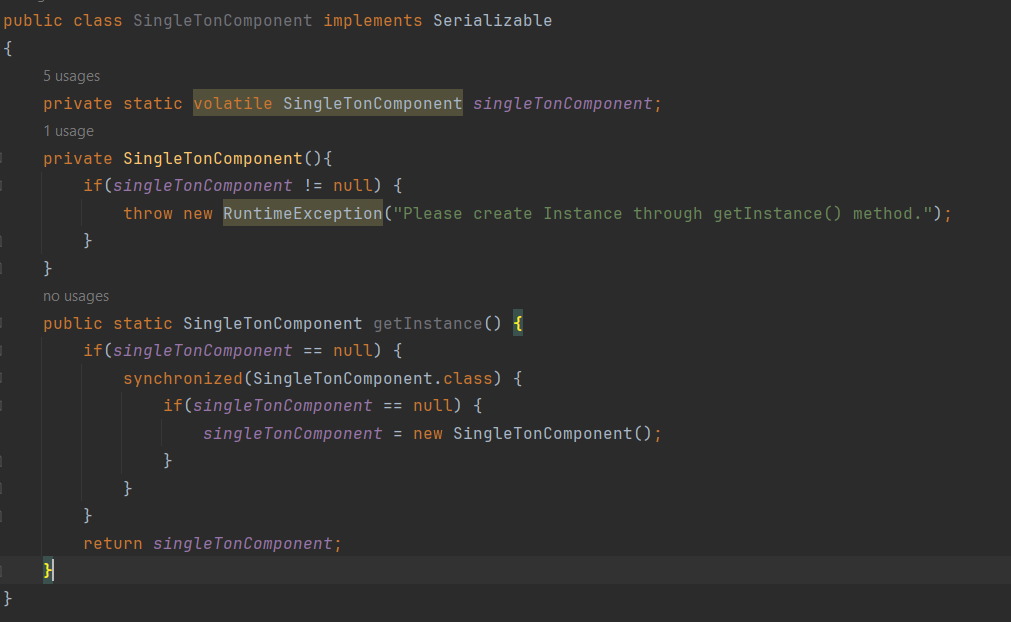
* *The Singleton method or Singleton Design pattern is one of the simplest design patterns. It ensures a class only has one instance, and provides a global point of access to it.*
* When class is having no state, or read only state or sharable state and allowing to create multiple objects for that class will waste the memory, & it will waste CPU time because object create in java costly process.

**When to use Singleton Method Design Pattern?**

Use the Singleton method Design Pattern when:

* There must be exactly one instance of a class and it must be accessible to clients from a well-known access point.
* When the sole instance should be extensible by subclassing and clients should be able to use an extended instance without modifying
* Singleton classes are used for logging, driver objects, caching, and thread pool, database connections.
* To create the perfect singleton design pattern we have to follow below steps.
  1. Create a class with no state or sharable state.
  2. Create private field at class level.
  3. Make the field is volatile it will avoid thread local caching & also if any state change occur it will going to change state in main memory, so that all threads can able to access the changes state.
  4. Create the private constructor it will avoid to access the constructor outside the class.
  5. Create public static method, so that when we call getInstance method it can return the same instance.
  6. Make sure inside getInstance method check whether instance is null, if null create new instance inside synchronized block. Because java is multithreaded programming language due to this multiple threads can invoke same function, which results breaking singleton pattern.

**Lazy Initialization:**



**Eager Initialization:**

* Take care with Java reflection API, Deserialization & Clone method because in all this three cases object going to be created.

