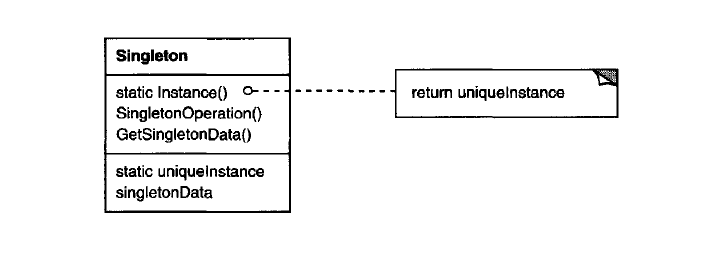
Design patterns

Singleton Pattern:

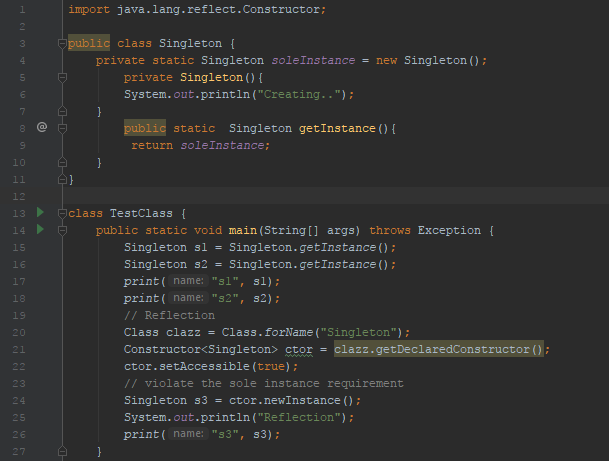
* Ensures that only one instance of a class is created.
* Provide a global point of access to it
* A solution is to make the class itself responsible for keeping track of its sole instance.
  + Hide the constructor of the class(declared private).
  + Define a public static operation that returns the sole instance of the class.

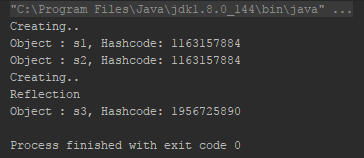


A screenshot of a cell phone

Description generated with very high confidence

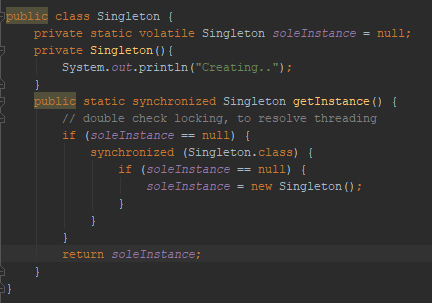
With reflection it can be changed:





* Reflection – how to fix?

Lazily initialize and check of the instance is already created with in the constructor.



Note: synchronized is not required for the method… Also note the volatile keyword.

**Question**: Which classes in JDK uses singleton pattern?  
Answer: java.lang.Runtime : In every Java application there is only one Runtime instance that allows the application to interface with the environment it is running. The getRuntime is equivalent to the getInstance() method of the singleton class.

**Question**: Can the singleton class be subclassed?  
**Answer**: Frankly speaking singleton is just a design pattern and it can be subclassed. However it is worth to understand the logic or requirement behind subclassing a singleton class as the child class might not inherit the singleton pattern objective by extending the Singleton class. However the subclassing can be prevented by using the final keyword in the class declaration.

**Question**: Can there be multiple instance of singleton using cloning?  
**Answer**: That was a good catch! What do we do now? To prevent the another instance to be created of the singleton instance we can throw exception from inside the clone() method.

Façade design pattern:

* Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
* Wrap a complicated subsystem with a simpler interface.

**Advantage of Facade Pattern**

* It shields the clients from the complexities of the sub-system components.
* It promotes loose coupling between subsystems and its clients.

**Usage of Facade Pattern:**

It is used:

* When you want to provide simple interface to a complex sub-system.
* When several dependencies exist between clients and the implementation classes of an abstraction.

