Singleton Pattern

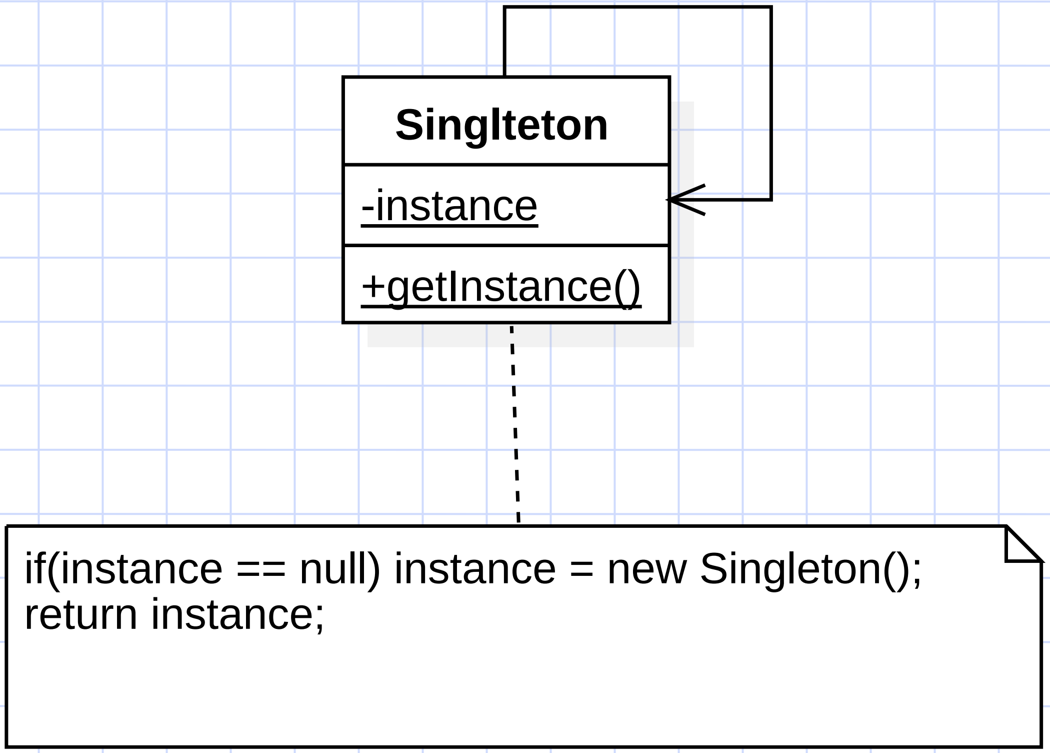
What is Singleton Pattern

- It is highly desirable if we can use some Design Pattern to control the access to that shared resource.
  - A good example is the login process
  - Another example is debugging the shared sources

Singleton Pattern

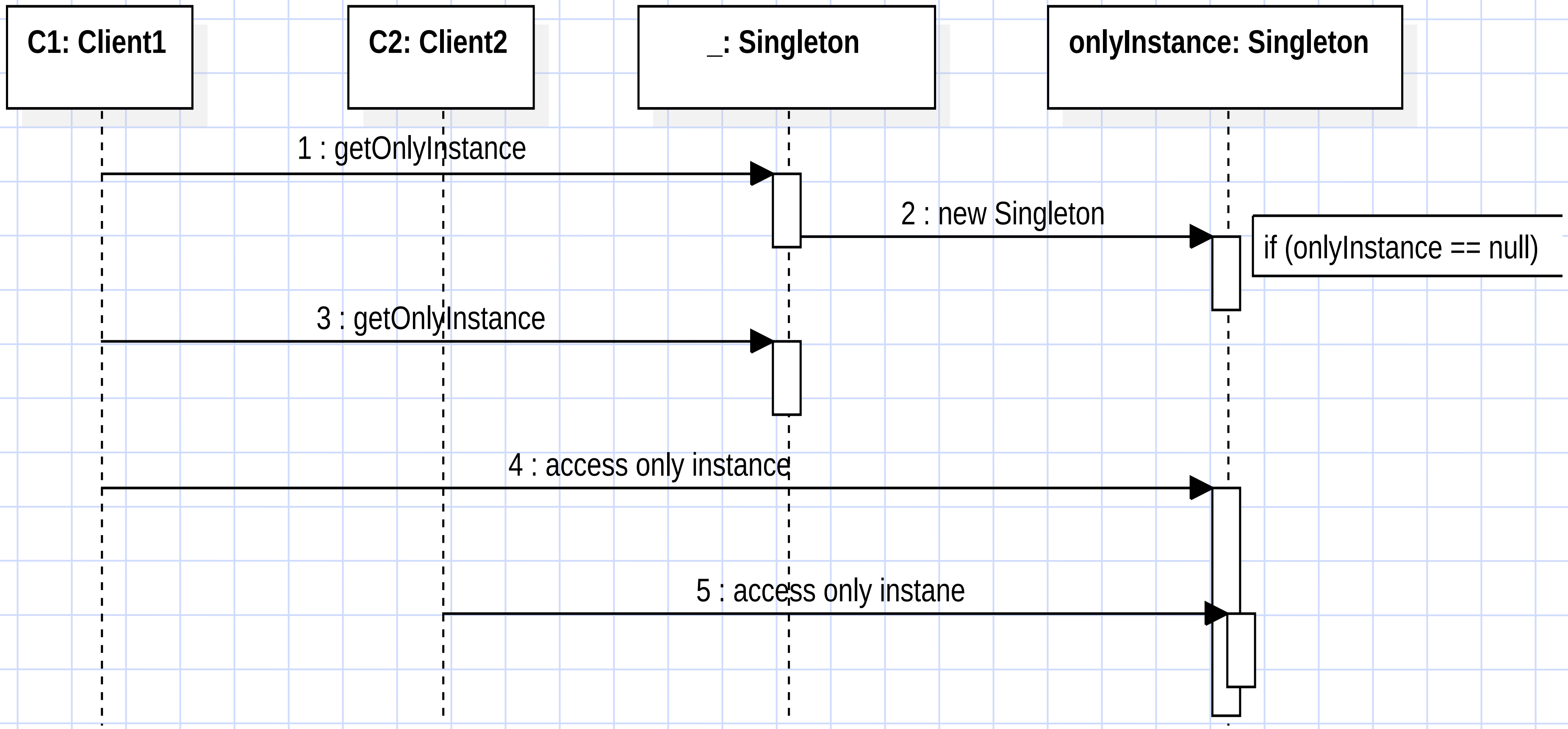
- User shouldn’t be allowed to create instances of the Singleton object.
- Need to have a private class-data-field of the Singleton.
- Need to have a public class-method to have access to private class-data-field

Singleton Pattern



```
if(instance == null) instance = new Singleton();
return instance;
```

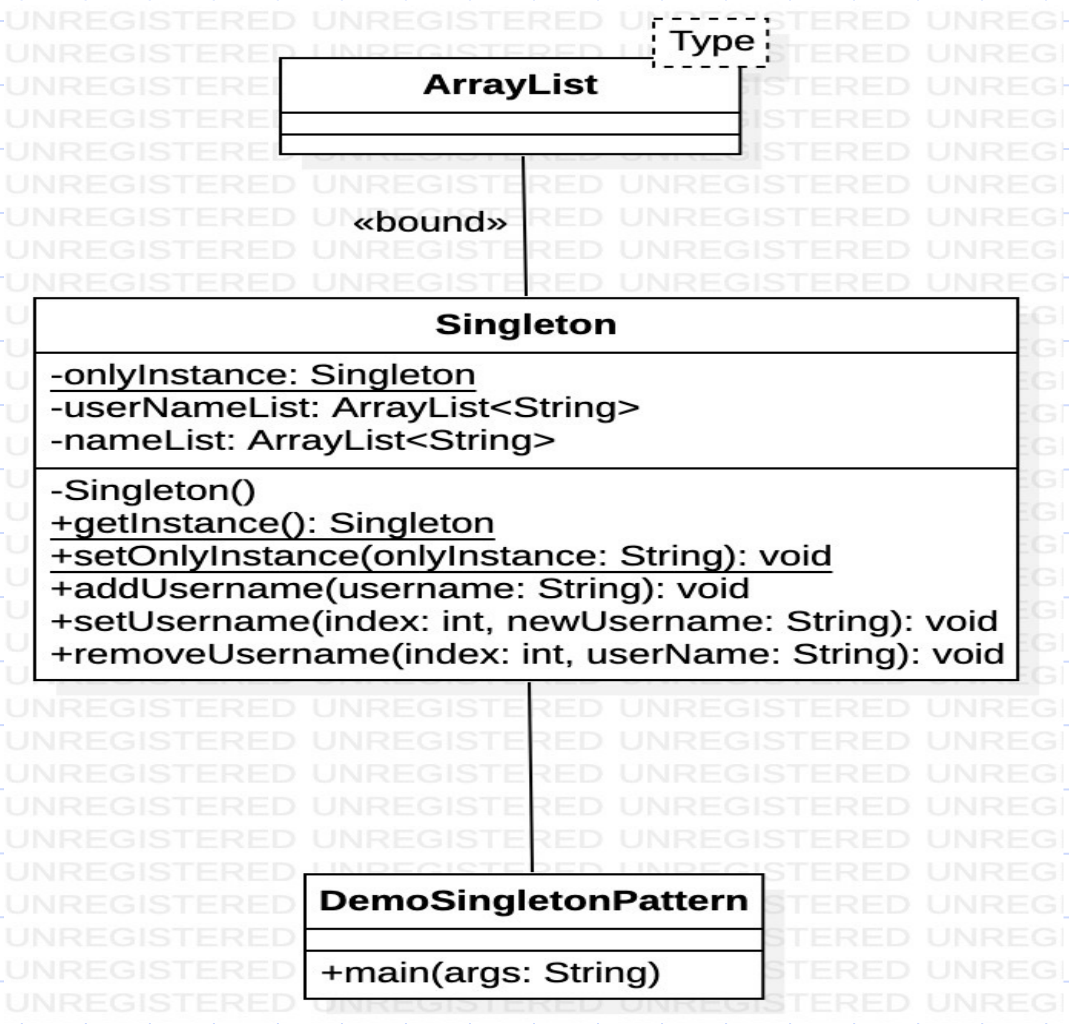
Singleton Pattern Interactions



Singleton Design: Learning by Example

Example

- Let's assume we need to develop an application that needs to keep a list of users and their passwords. We are also concerned to keep only single instance of these lists. Here is the UML model for this case:


Type
ArrayList
«bound»
Singleton
-onlyInstance: Singleton
-userNameList: ArrayList<String>
-nameList: ArrayList<String>
-Singleton()
+getInstance(): Singleton
+setOnlyInstance(onlyInstance: String): void
+addUsername(username: String): void
+setUsername(index: int, newUsername: String): void
+removeUsername(index: int, userName: String): void
DemoSingletonPattern
+main(args: String)

Java Implementation of this model will be discussed during the lectures

Benefits and Drawbacks of Singleton

- Singleton is a famous pattern as its known as simplest to be learned.
- It is useful for using a single copy of the shared resource.
- Drawbacks include:
  - Singleton classes cannot be sub classed.
  - Reduces testing flexibilities:
    - A good design advice is to minimize dependencies between classes. Particularly during unit testing this advice is very helpful.
    - With a singleton pattern this feature will be scarified. Because the object creation part is hidden, we cannot expect the singleton constructor to accept any parameters.