INRIA

Singleton

Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

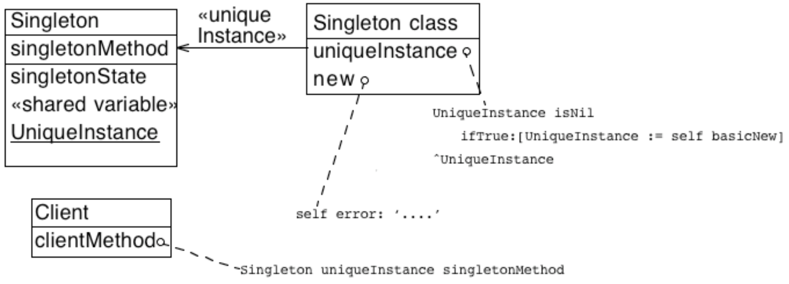
Singleton

Ensure that a class has only one instance, and provide a global point of access to it

The Singleton Pattern

- ***Intent:*** Ensure that a class has only one instance, and provide a global point of access to it
- ***Problem:*** We want a class with a unique instance.
- ***Solution:*** We specialize the #new class method so that if one instance already exists this will be the only one. When the first instance is created, we store and return it as result of #new.

Singleton Possible Structure



```
UniqueInstance isNil
    ifTrue:[UniqueInstance := self basicNew]
^UniqueInstance
```

```
self error: '....'
```

```
^Singleton uniqueInstance singletonMethod
```

The Singleton Pattern

```
|aLan|
aLan := NetworkManager new
aLan == LAN new -> true
aLan uniqueInstance == NetworkManager new -> true

NetWorkManager class
    instanceVariableNames: 'uniqueInstance '

NetworkManager class>>new
    self error: 'should use uniqueInstance'

NetworkManager class>>uniqueInstance
   uniqueInstance isNil
          ifTrue: [ uniqueInstance := self basicNew initialize].
```

The Singleton Pattern

- Providing access to the unique instance is not always necessary.
- It depends on what we want to express. The difference between #new and #uniqueInstance is that #new potentially initializes a new instance, while #uniqueInstance only returns the unique instance (there is no initialization)
- Do we want to communicate that the class has a singleton? ***new***? ***defaultInstance***?

Implementation Issues

- Singletons may be accessed via a global variable (ex: NotificationManager uniqueInstance notifier).

```
SessionModel>>startupWindowSystem
               "Private - Perform OS window system startup"
               Notifier initializeWindowHandles.
               ...
               oldWindows := Notifier windows.
               Notifier initialize.
               ...
               ^oldWindows
```

- Global Variable or Class Method Access
  - Global Variable Access is dangerous: if we reassign Notifier we lose all references to the current window.
  - Class Method Access is better because it provides a single access point. This class is responsible for the singleton instance (creation, initialization,...).

Implementation Issues

Persistent Singleton: only one instance exists and its identity does not change (ex: NotifierManager in Visual Smalltalk)

Transient Singleton: only one instance exists at any time, but that instance changes (ex: SessionModel in Visual Smalltalk, SourceFileManager, Screen in VisualWorks)

Single Active Instance Singleton: a single

Implementation Issues

classVariable or class instance variable

classVariable

- One singleton for a complete hierarchy

Class instance variable

- One singleton per class

Access?

In Smalltalk we cannot prevent a client to send a message (protected in C++). To prevent additional creation we can redefine new/new:

```
Object subclass: #Singleton
        instanceVariableNames: 'uniqueInstance'
        classVariableNames: ''
        poolDictionaries: ''

Singleton class>>new
```

Access using new: not good idea

```
Singleton class>>new
        ^self uniqueInstance
```

The intent (uniqueness) is not clear anymore! New is normally used to return newly created instances. The programmer does not expect this:

```
|screen1 screen2|
screen1 := Screen new.
screen2 := Screen uniqueInstance
```

Favor Instance Behavior

When a class should only have one instance, it could be tempting to define all its behavior at the class level. But this is not good:

- Class behavior represents behavior of classes: "Ordinary objects are used to model the real world. MetaObjects describe these ordinary objects"

- Do not mess up this separation and do not mix domain objects with metaconcerns.

Time and not Scope

Singleton is about ***time*** not ***access***

- ***time***: only one instance is available at the same time
- ***access***: can’t you add an instance to refer to the object?

Singleton for access are as bad as global variables

Often we can avoid singleton by passing/referring to the object instead of favoring a global access point