Patron: Singleton

http://www-public.tem-tsp.eu/~gibson/Teaching/CSC4522/CSC4522-DesignPatterns-**Singleton.pdf**

Patron: Singleton

https://fr.wikipedia.org/wiki/Singleton_(patron_de_conceptio

•Restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement). Il est utilisé lorsque l'on a besoin d'exactement un objet pour coordonner des opérations dans un système.

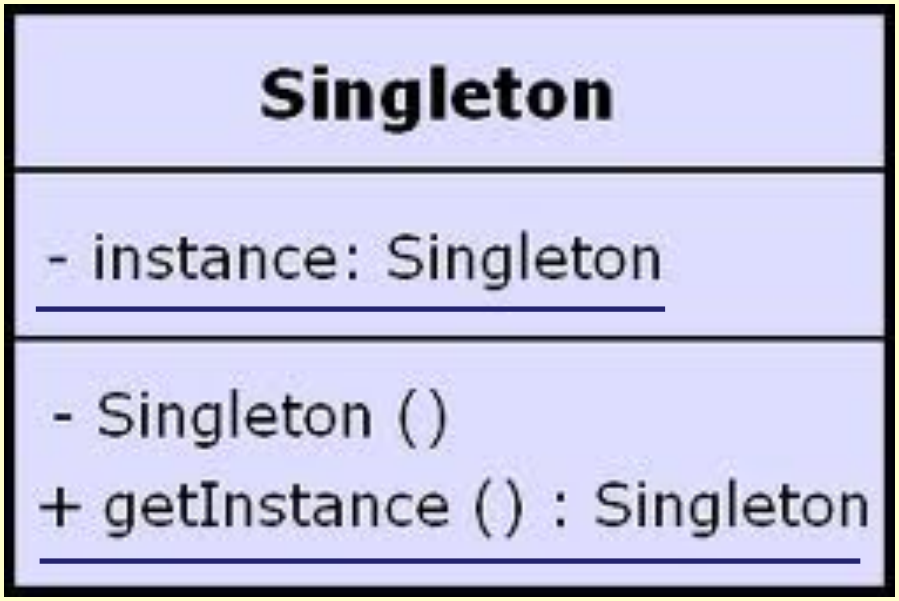
•Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire

•On implémente le singleton en écrivant une classe contenant une méthode qui crée une instance uniquement s'il n'en existe pas encore. Sinon elle renvoie une référence vers l'objet qui existe déjà.

• Dans beaucoup de langages de type objet, il faudra veiller à ce que le constructeur de la classe soit *privé* ou bien *protégé*, afin de s'assurer que la classe ne puisse être instanciée autrement que par la méthode de création contrôlée.

•Le singleton doit être implémenté avec précaution dans les applications multi-thread. Si deux threads exécutent *en même temps* la méthode de création alors que l'objet unique n'existe pas encore, il faut absolument s'assurer qu'un seul créera l'objet, et que l'autre obtiendra une référence vers ce nouvel objet. La solution classique à ce problème consiste à utiliser l'exclusion mutuelle pour indiquer que l'objet est en cours d'instanciation.

The Singleton Design Pattern



UML class diagram

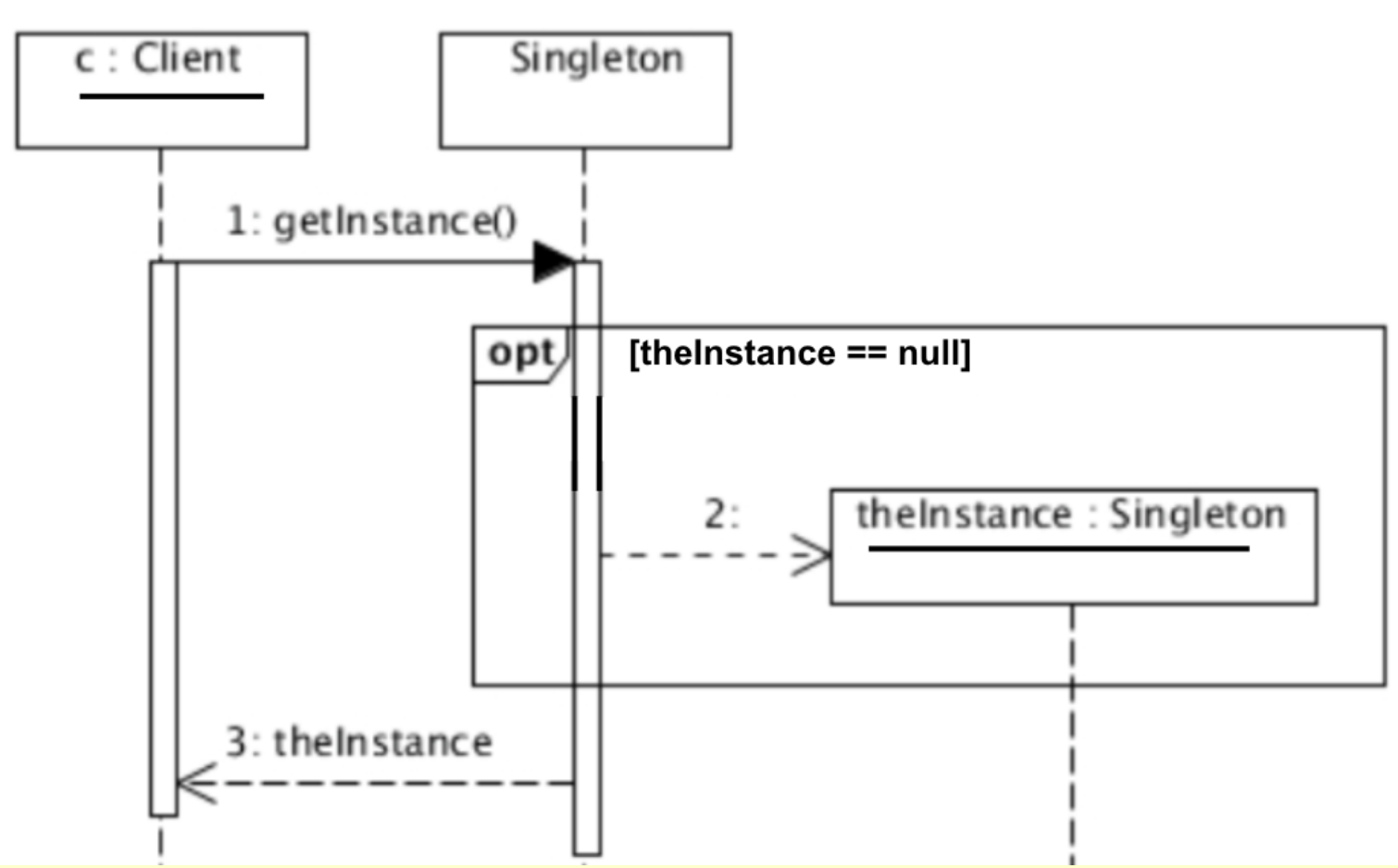
See also:

https://fr.wikibooks.org/wiki/Patrons_de_conception/Singleton

http://sourcemaking.com/design_patterns/singleton

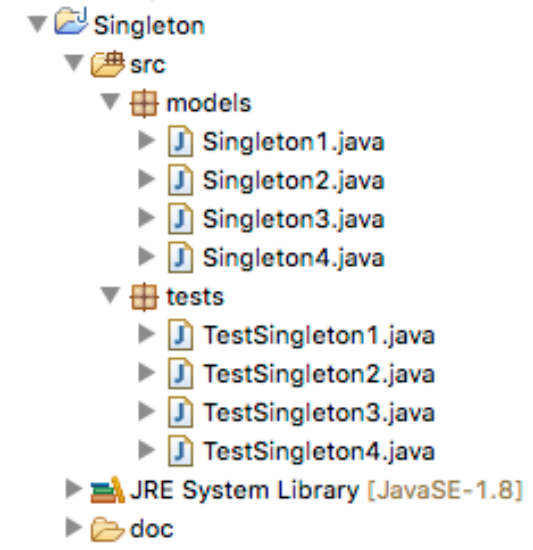
The Singleton Design Pattern

UML sequence diagram



Adapted from: https://www.vainolo.com/2012/04/29/singleton-design-pattern-sequence-diagram/

The Singleton Design Pattern



Problem: Examine the Singleton Java implementations (versions 1, 2, 3 and 4) in the project:

http://www-public.tem-tsp.eu/~gibson/Teaching/CSC4522/SourceCode/Singleton.zip

(Import existing project into Eclipse)

Question: what are the differences between the implementations, and which best corresponds to our requirements/design?

Patron: Singleton (en Java, **version1**)

```
package models;

public class Singleton1 {

protected static Singleton1 uniqueInstance = null;

private int data;

    public synchronized static Singleton1 instance() {
    if(uniqueInstance == null)
       uniqueInstance = new Singleton1();
    return uniqueInstance;
    }

    protected Singleton1() {data=0;}

    public int getData(){return data;}

    public void setData(int d){data =d;}

}
```

QUESTIONS?

Patron: Singleton (en Java, **version1**) - Test

tests.TestSingleton1

Test class for `Singleton1`

- Construct two `Singleton1` instances/objects using the `Singleton1.instance()` method
- Set the `data` value of the first instance and print information concerning both objects to the screen
- Check - on the screen output - that the references and values for both objects are the same

EXPECTED (TYPICAL) OUTPUT

```
First singleton1: models.Singleton1@70dea4e
First singleton1 data value =  34
Second singleton1: models.Singleton1@70dea4e
Second singleton1 data value =  34

 CHECK - on the previous output - that the references and values for both objects are the same
```

Author:
J Paul Gibson

TODO: Read, execute and understand the test code

Patron: Singleton (en Java, **version2**)

```
package models;

public class Singleton2 {

public static final Singleton2 uniqueinstance = new
Singleton2();

private int data;

private Singleton2() {data=0;}

public int getData(){return data;}

public void setData(int d){data =d;}

}
```

QUESTIONS

Patron: Singleton (en Java, **version2**) test

tests.TestSingleton2

Test class for `Singleton2`

- Construct two `Singleton2` instances/objects using the `Singleton2.uniqueinstance` attribute
- Set the `Singleton2.data` value of the first instance and print information concerning both objects to the screen
- Check - on the screen output - that the references and values for both objects are the same

EXPECTED (TYPICAL) OUTPUT

```
First singleton2: models.Singleton1@70dea4e
First singleton2 data value =  34
Second singleton2: models.Singleton1@70dea4e
Second singleton2 data value =  34

 CHECK - on the previous output - that the references and values for both objects are the sa
```

Author:
J Paul Gibson

TODO: Read, execute and understand the test code

Patron: Singleton (en Java, **version3**)

```
public class Singleton3 {
private int data;
private static final Singleton3 instance = new Singleton3();
private Singleton3() { data =0;
}
public static Singleton3 instance() { return instance;
}
public int getData(){ return data;
}
public void setData(int d){ data = d;
}
}
```

QUESTIONS

Patron: Singleton (en Java, **version4**)

Follows Java specific design proposed by Bill Pugh (see ***Concurrent programming in Java: design principles and patterns)***

```
public class Singleton4 {

private int data;

private Singleton4() { data=0; }

private static class SingletonHolder {
    private static final Singleton4 INSTANCE = new Singleton4();
  }

public static Singleton4 getInstance() {
    return SingletonHolder.INSTANCE;
  }

public int getData(){return data;}

public void setData(int d){data =d;}

}
```

QUESTIONS

Patron: Singleton with Subclassing - Optional Practical Work

What if we want to be able to subclass Singleton and have the single instance be a subclass instance?

For example, suppose View has subclasses TextView and GraphicView. We want to instantiate just one view, either textual or graphic. Then all other instantiations use the same (first) instance.

Client code to create view the first time:

```
// Here we choose to instantiate with a text view
View v1 = TextView.instance();
```

Client code to access the view:

```
// The view instance will be the first view created
// In this case a TextView
View v2  = View.instance();
```

Patron: Singleton with Subclassing - Optional Practical Work

The Singleton Design Pattern is meant to give you control over access to the Singleton class. But subclassing allows other code to access your class without you having direct control

The uniqueness of the class cannot be imposed as a compile-time constraint on the subclass unless you use a private constructor (or declare the class to be final).

If you want to allow subclassing, for example, you might make the constructor protected, but then a subclass could provide a public constructor, allowing anyone to make instances.

QUESTION: which of these 2 *subclassable singleton* designs do you prefer:?

Supposing we have a Singleton class A and a class B that is a sub-class of A:

1). You can have a single instance of A *OR* a single instance of B, but not both.

2). You can have exactly one instance of A *AND* exactly one instance of B.

TO DO: Can you implement and test one of these designs?

QUESTION: How can/should this design/code be extended to multiple subclasses?

Patron: Singleton with Subclassing - Optional Practical Work

QUESTION: How could we do this? Will the previous code for a Singleton class work with subclassing? Try it!

For example, with Singleton1:

```
Singleton1a extends Singleton1
Singleton1b extends Singleton1

// TEST
// Singleton1a s1a = new …?
// Singleton1b s1b = new …?
// Write values of s1a and s1b to screen
```

TO DO:
Spend some time thinking about alternative designs.
Choose one - try to implement and test it in Java.