Patron: Singleton

http://www-public.it-sudparis.eu/~gibson/Teaching/DesignPatterns/DesignPatterns-Singleton.ppt
http://www-public.it-sudparis.eu/~gibson/Teaching/DesignPatterns/DesignPatterns-Singleton.pdf

Patron: Singleton

- 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.

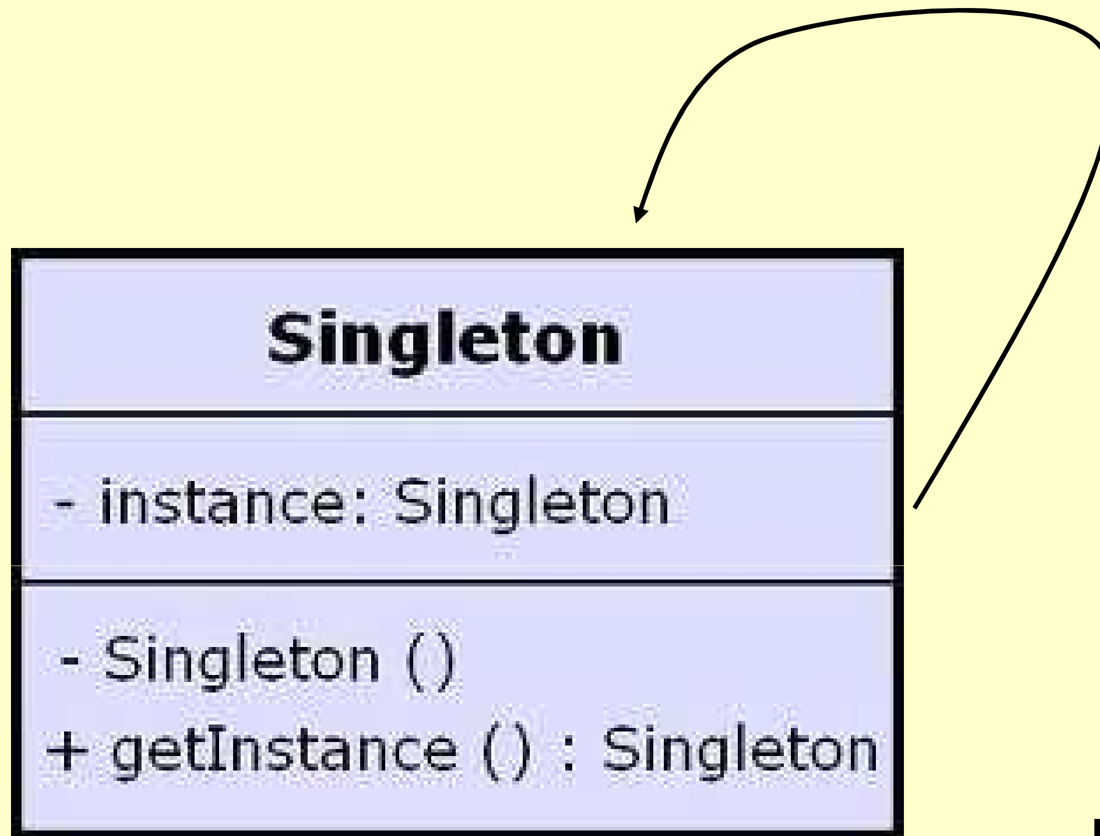
- Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire

- 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.

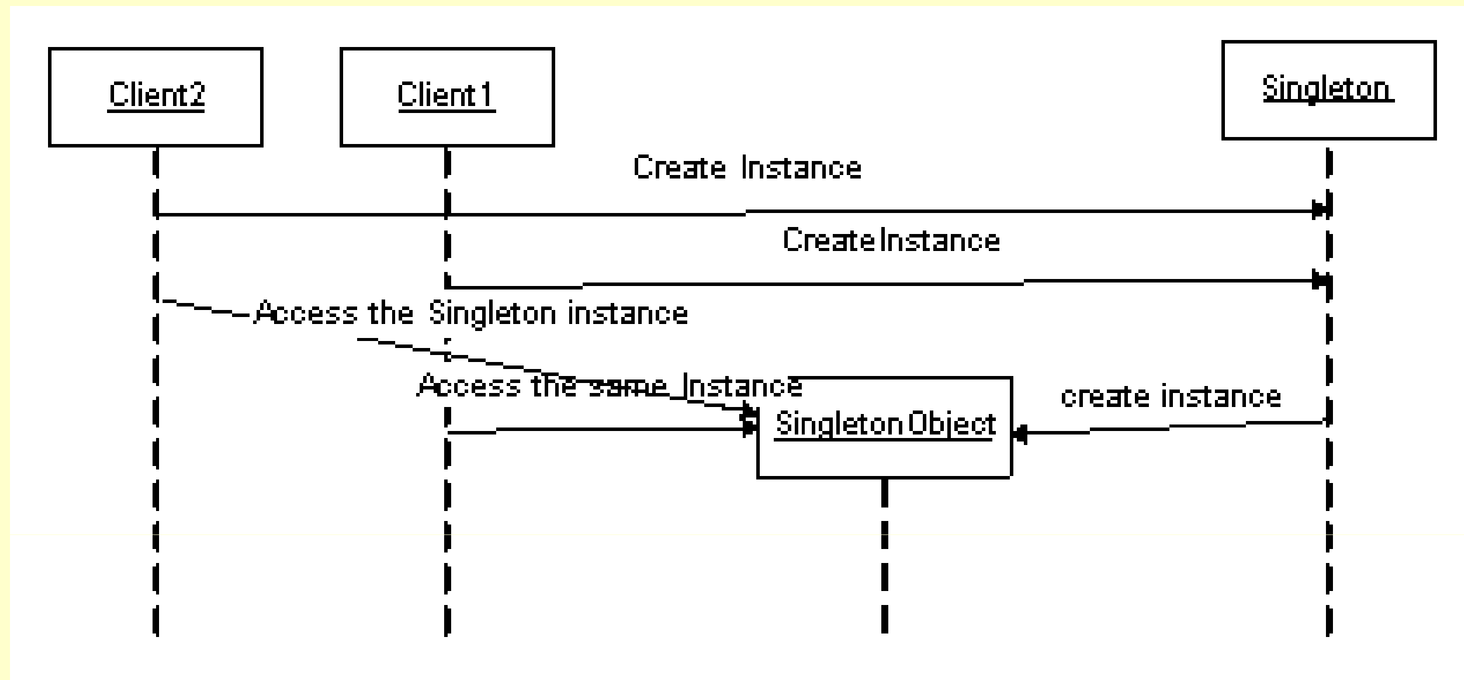
- Le singleton doit être implémenté avec précaution dans les applications multi-thread. Si deux threads exécutent *en même temps* la méthode de création alors que l'objet unique n'existe pas encore, il faut absolument s'assurer qu'un seul créera l'objet, et que l'autre obtiendra une référence vers ce 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

The Singleton Design Pattern



UML sequence diagram

The Singleton Design Pattern

Problem: Examine the 4 Singleton Java implementations in the folder Patterns (http://www-public.int-evry.fr/~gibson/Teaching/DesignPatterns/SourceSingleton.zip)

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

Patron: Singleton (en Java, version1)

```java
package p_singleton;

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;}

}
```

Patron: Singleton (en Java, version1) - Test

```
/**
 * @author J Paul Gibson
 * @version 1
 * Test class for {@link Singleton1}<br>
 *
 * <ul>
 * <li> Construct 2 singleton1 objects using the instance method</li>
 * <li> Set the data value of the first and print info. concerning both
 *  objects to the screen</li>
 * <li> Check that the references and values for both objects are the
 * same</li>
 * </ul>
 * <br>
 * EXPECTED (TYPICAL) OUTPUT
 *
<pre>
First singleton1: Singleton.Singleton1@19821f
First singleton1 data value =  34
Second singleton1: Singleton.Singleton1@19821f
Second singleton1 data value =  34
</pre>
 */
public class TestSingleton1 {
```

Patron: Singleton (en Java, version1) - Test

```java
public static void main(String args[]) {
   // Construct 2 singleton1 objects using the instance method
Singleton1 s1a = Singleton1.instance();
Singleton1 s1b = Singleton1.instance();
// Set the data value.
s1a.setData(34);
System.out.println("First singleton1: " + s1a);
System.out.println("First singleton1 data value =  " +s1a.getData());
System.out.println("Second singleton1: " + s1b);
System.out.println("Second singleton1 data value =  " +s1b.getData());
}
}
```

TP - TO DO:

Compile and execute this class to check that the test is correct

Patron: Singleton (en Java, version2)

```
package p_singleton;

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;}

}
```

Patron: Singleton (en Java, version2) test

```
/**
 * @author J Paul Gibson
 * @version 2
 * Test class for {@link Singleton2}<br>
 *
 * <ul>
 * <li> Construct 2 singleton2 objects using the instance method</li>
 * <li> Set the data value of the first and print info. concerning both objects
to the screen</li>
 * <li> Check that the references and values for both objects are the same</li>
 * </ul>
 * <br>
 * EXPECTED (TYPICAL) OUTPUT
 *
<pre>
First singleton2: Singleton.Singleton2@19821f
First singleton2 data value =  34
Second singleton2: Singleton.Singleton2@19821f
Second singleton2 data value =  34
</pre>
 */
```

Patron: Singleton (en Java, version2) test

```
public class TestSingleton2 {

public static void main(String args[]) {
   // Construct 2 singleton1 objects using the instance method
Singleton2 s2a = Singleton2.uniqueinstance;
Singleton2 s2b = Singleton2.uniqueinstance;
// Set the data value.
s2a.setData(34);
System.out.println("First singleton2: " + s2a);
System.out.println("First singleton2 data value =  " +s2a.getData());
System.out.println("Second singleton2: " + s2b);
System.out.println("Second singleton2 data value =  " +s2b.getData());
}
}
```

TP - TO DO:

Compile and execute this class to check that the test is correct

Patron: Singleton (en Java, version3)

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

TP - TO DO:

How does this 3rd version differ from version1 and version2?

Implement and test this 3rd version (in class TestSingleton3)

Patron: Singleton (en Java, version4)

```
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;}

}
```

Patron: Singleton with Subclassing - TP

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

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 - TP

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

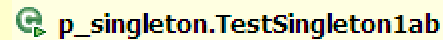
For example, with Singleton1:

```
Singleton1a extends Singleton1
Singleton1b extends Singleton1

// TEST
// Singleton1a s1a = new …?
// Singleton1b s1b = new …?
// Write values of s1a and s2a to screen
```

TO DO: Spend some time thinking about alternative designs.Choose one - try to implement and test it in Java.   Will the same design work with Singleton2 or Singleton3 or Singleton4?

Patron: Singleton with Subclassing - TP



Version:

1 Test class for Singleton1

- Construct 2 singleton1a objects using the instance method
- Check correctly initialised
- Set the data value of the first and print info. concerning both objects to the screen
- Check that the references and values for both objects are the same
- Construct 2 singleton1b objects using the instance method
- Check correctly initialised
- Set the data value of the last and print info. concerning all 4 objects to the screen
- Check that the references and values for all objects are the same

EXPECTED OUTPUT

```
Construct 2 singleton1a objects using the instance method
First singleton1a: p_singleton.Singleton1a@addbf1, data value = 0
Second singleton1a: p_singleton.Singleton1a@addbf1, data value = 0
Setting data value to 34
First singleton1a: p_singleton.Singleton1a@addbf1, data value = 34
Second singleton1a: p_singleton.Singleton1a@addbf1, data value = 34
Construct 2 singleton1b objects using the instance method
First singleton1b: p_singleton.Singleton1a@addbf1, data value = 34
Second singleton1b: p_singleton.Singleton1a@addbf1, data value = 34
Setting data value to 56
First singleton1b: p_singleton.Singleton1a@addbf1, data value = 56
Second singleton1b: p_singleton.Singleton1a@addbf1, data value = 56
First singleton1a: p_singleton.Singleton1a@addbf1, data value = 56
Second singleton1a: p_singleton.Singleton1a@addbf1, data value = 56
```

Author:

J Paul Gibson