

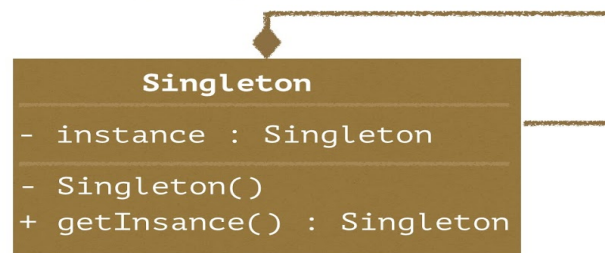
Design Patterns Et Architectures Logicielles

Étude de cas : le design pattern Singleton

LA FILIÈRE : SYSTÈMES UBIQUITAIRES ET DISTRIBUÉS CLOUD ET IoT

Singleton Design Pattern

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



Team :
HAJJAJI Ayyoub
EL AMMARI Souhail

Professor :
Abdeslam EN-NOUAARY

0.1 Explication du problème

Un singleton, c'est un objet construit conformément à sa classe et dont on a la garantie qu'il n'existe qu'une et une seule instance en mémoire à un instant donné. En général, on souhaite que le singleton ne s'initialise pas entièrement, mais seulement à son premier appel, afin d'économiser la mémoire. On appelle cela le mécanisme « lazy ».

Il y a plusieurs exemples dans ce sens citant par exemple : la terre, Africa, le roi du maroc, INPT ...

Dans ce TP on a pris la terre comme exemple puisqu'on peut créer un et un seul objet terre.

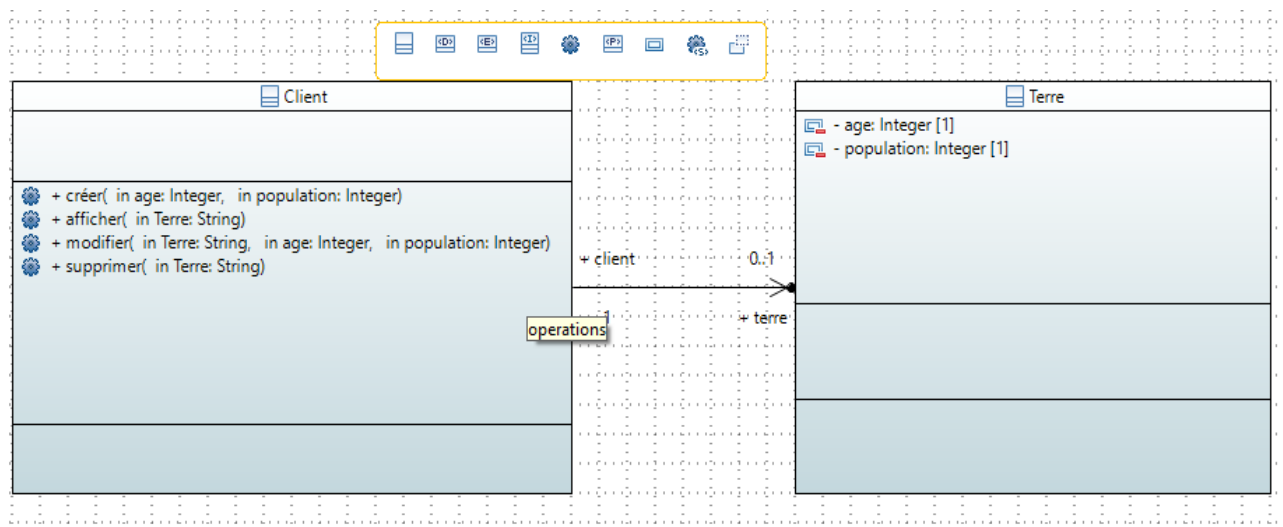


FIGURE 1 – diagramme de classe

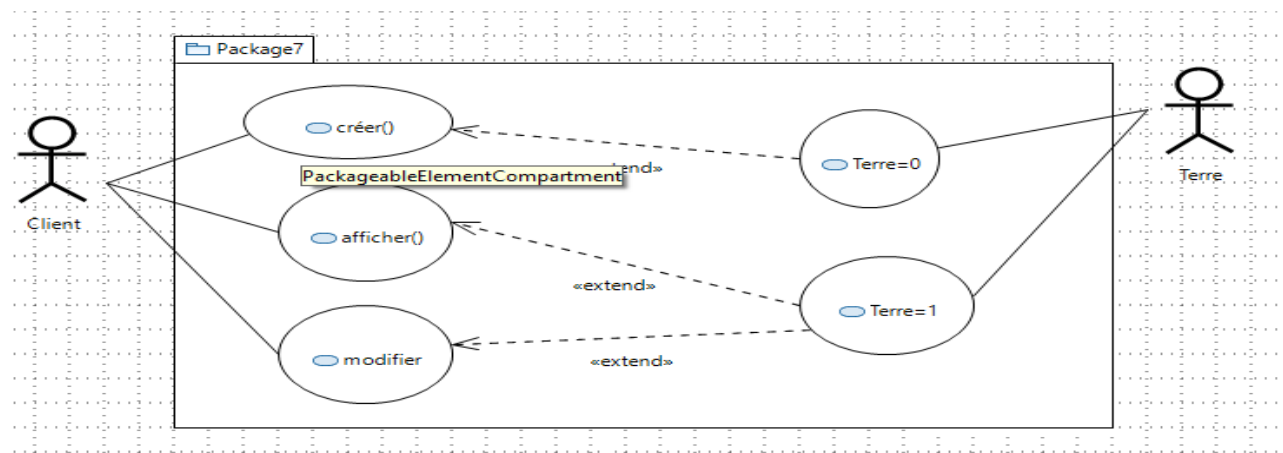


FIGURE 2 – diagramme de cas d'utilisation

0.2 implémentation JAVA

Dans notre implémentation JAVA on a créé une public class terre avec un constructeur privé, et pour créer une instance terre s'elle n'est pas déjà créée on a utilisé une méthode publique : getInstance.

Cette implémentation fonctionne bien dans le cas de l'environnement à un seul thread, mais en ce qui concerne les systèmes multithreads, elle peut causer des problèmes si plusieurs threads sont à l'intérieur de la condition if en même temps.

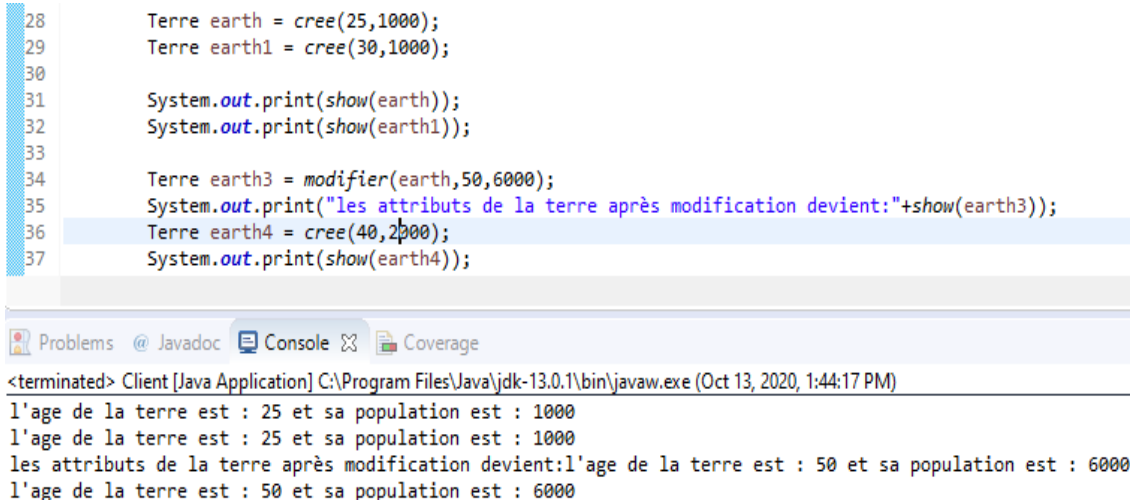
Le moyen le plus simple pour créer une classe singleton Terre consiste à synchroniser la méthode d'accès globale, de sorte qu'un seul thread puisse exécuter cette méthode à la fois.

Dans cette approche on peut déclarer la méthode getInstance de cette manière : *public static synchronized Terre getInstance()*, mais cela réduit les performances en terme du coût associé à la méthode synchronized, donc pour résoudre le problème des systèmes multithreads, on a utilisé un mécanisme de verrou.

dans le programme suivant on a initialiser le verrou à false :

```
private static Terre INSTANCE = null;

public static Terre getInstance(int age, int population)
{
    if (INSTANCE == null) {
        if (verrou == false){
            verrou = true;
            INSTANCE = new Terre(age,population);
        }
        return INSTANCE;
    }
    return INSTANCE;
}
```



The screenshot shows a Java IDE with a code editor and a console window. The code editor contains the following lines:

```
28 Terre earth = cree(25,1000);
29 Terre earth1 = cree(30,1000);
30
31 System.out.print(show(earth));
32 System.out.print(show(earth1));
33
34 Terre earth3 = modifier(earth,50,6000);
35 System.out.print("les attributs de la terre après modification devient:"+show(earth3));
36 Terre earth4 = cree(40,2000);
37 System.out.print(show(earth4));
```

The console window shows the following output:

```
<terminated> Client [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Oct 13, 2020, 1:44:17 PM)
l'age de la terre est : 25 et sa population est : 1000
l'age de la terre est : 25 et sa population est : 1000
les attributs de la terre après modification devient:l'age de la terre est : 50 et sa population est : 6000
l'age de la terre est : 50 et sa population est : 6000
```

FIGURE 3 – Test du programme

Visiter GitHub pour voir le programme complet : <https://github.com/ayy-oub/design-pattern-singlrton>