المعهد الوطني للبريد والمواصلات
ⵓⵔⵣⵓⵔ ⵏⵓⵎⵓⵔⵓ ⵏ ⵜⵓⵍⵓⵙⵜ ⴰ ⵓⵎⵙⵓⵍⵓⵖ
Institut National des Postes et Télécommunications

Étude de cas : le design pattern *Singleton*

Travail réalisé par :

- Chrif El Asri Hanane
- El Gamous Khalid

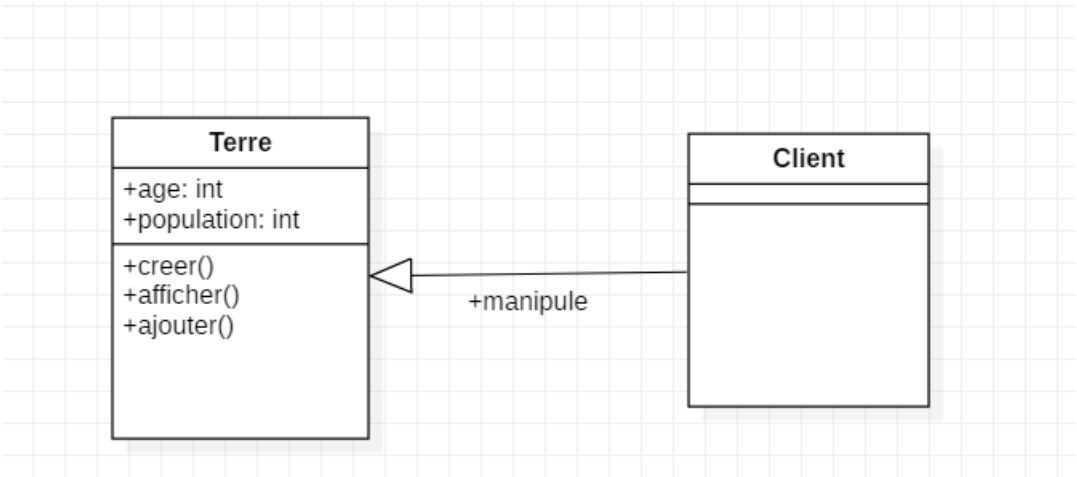
Encadrant:

Prof. Abdeslam EN-NOUAARY

Nous voulons développer une solution orientée objet pour créer et manipuler des objets *Terre.* La problématique c'est qu'on ne doit posséder d'un seul et unique objet Terre (car effectivement il existe une seule planète terre). Du coup, il faut trouver une manière de limiter les objets créés en un seul.

Pour cela on va créer une classe terre et une classe Client qui va créer et manipuler l'objet terre.

Voici le diagramme de classe pour cette solution.



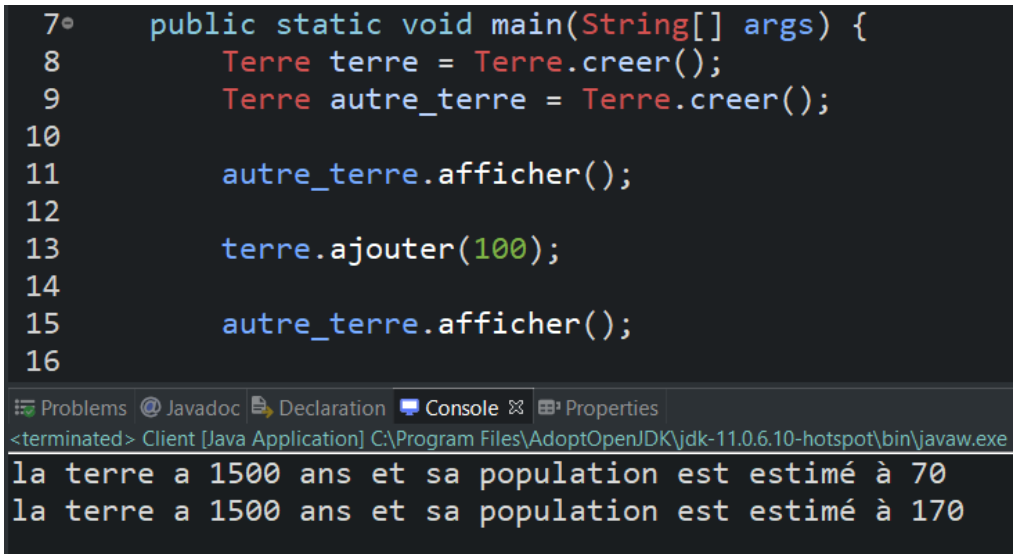
Tout d'abord la classe Terre doit avoir un constructeur privé pour qu'elle ne soit initialisée que depuis la classe Terre elle-même. Ensuite on va ajouter une classe privée TerreHolder à l'intérieur de la classe Terre, Son rôle est d'initialiser la seule et unique instance terre.

La classe Terre possède une méthode créer() qui va retourner l'instance du classe TerreHolder. Ainsi pour créer un objet terre, le client doit appeler la méthode créer(). Ensuite le client peut avoir accès aux différentes méthodes de la classe Terre, notamment afficher() qui nous affiche l'âge et la population, et ajouter(n) qui ajoute le nombre donné en paramètre à la population.

```
    public static void main(String[] args) {
        Terre terre = Terre.creer();

        terre.afficher();

        terre.ajouter(100);

        terre.afficher();
```

```
blems  Javadoc  Declaration  Console  Properties
nated> Client [Java Application] C:\Program Files\AdoptOpenJDK\jdk-11.0.6.10-hotspot\bin\javaw.exe (10 oc
terre a 1500 ans et sa population est estimé à 70
terre a 1500 ans et sa population est estimé à 170
```

Pour s'assurer que notre code est bien un Singleton, on va essayer de créer un autre objet terre, l'afficher et ajouter 100 au premier objet terre et afficher le deuxième objet terre.



On remarque que la population de l'objet autre_terre a été incrémenté de 100, ce qui signifie qu'il n'existe en fait qu'une seule instance de la classe Terre et que les objets terre et autre_terre pointent vers le même emplacement dans la mémoire.

Ce problème se pose dans plusieurs situations. C'est par exemple le cas d'une classe qui implémenterait un pilote pour un périphérique, ou encore un système de journalisation. En effet, instancier deux fois une classe servant de pilote à une imprimante provoquerait une surcharge inutile du système et des comportements incohérents. La solution est identique à ce qu'on a fait là-haut, c'est justement le but du design pattern, on résout un problème qui se pose souvent une fois pour toute.

NB : Vous pouvez trouver le code source dans ce lien

www.github.com/hanane-ca/Singleton.git