Chapitre XII

Méthodes et variables statiques

1 Exemple: la classe `Math`

Supposons que vous vouliez écrire une méthode `int abs(int x)` dans une classe `Math` qui calcule la valeur absolue d'un nombre `x`. Le comportement de cette méthode ne dépend pas de la valeur des attributs de la classe `Math`. Aussi, pourquoi devrait-on créer un objet de la classe `Math` pour utiliser cette méthode ?

La classe `Math` existe en Java, ainsi que la méthode `abs()`. Effectivement, il n'est pas possible d'instancier cette classe:

```
1 public class TestMath {
2     public static void main(String[] args) {
3         Math objetMath = new Math();
4     }
5 }
```

L'instruction précédente renvoie l'erreur suivante à la compilation:

```
1 > javac TestMath.java
2 TestMath.java:3: error: Math() has private access in Math
3         Math objetMath = new Math();
4                          ^
5 1 error
```

2 Méthodes statiques

Bien que Java soit un langage objet, il existe de rares cas où une instance de classe est inutile.

2.a Déclaration et appel de méthodes statiques

En Java

Le mot clé `static` permet alors à une méthode de s'exécuter sans avoir à instancier la classe qui la contient.

L'appel à une méthode `static` se fait alors en utilisant **le nom de la classe** plutôt que le nom de la référence à un objet.

Par exemple, en écrivant une classe `MaClassMath` qui contient une méthode statique:

```java
public class MaClassMath {
    public static int min (int a, int b) {
        // retourne la plus petite valeur entre a et b
        return ...;
    }
}
```

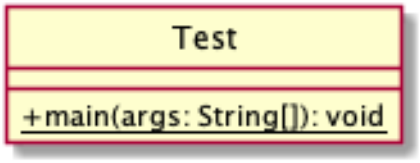
Lorsque l'on souhaite utiliser la méthode `static min`, on l'appelle sur le nom de la classe `MaClassMath`:

```java
public class TestMaClassMath {
      public static void main(String[] args) {
            int x = MaClassMath.min(21, 4);
      }
}
```

En Java, seule les attributs et les méthodes peuvent être *static*. Les classes, elles, ne sont pas *static*[1]. De manière générale, une classe possédant des méthodes statiques n'est pas conçue pour être instanciée. Mais cela ne signifie pas qu'une classe possédant une méthode `static` ne doit jamais être instanciée : si d'autres méthodes de la classes ne sont pas statiques, alors l'instanciation doit être possible.

En UML

En Uml, une méthode statique est déclarée soulignée:



2.b Méthodes statiques et attributs

Considérons le code suivant:

```java
public class Chien {
    private int taille;

    public Chien(int taille) {
        this.taille = taille;
    }

    public static void aboyer() {
        if (taille < 20) {
            System.out.println("kai kai");
        } else {
            System.out.println("Ouf Ouf");
        }
    }
}
```

La compilation de ce code produit l'erreur suivante:

```
> javac Chien.java
Chien.java:5: error: non-static variable taille cannot be referenced from a
    static context
        if (taille < 20) {
            ^
1 error
```

En effet, Comme la méthode `aboyer()` est `static`, elle peut être appelée sans qu'aucune instance n'ait été créée. Or la valeur de l'attribut `taille` ne prend une valeur que lors d'une instanciation.

[1] à l'exception des classes internes *inner class* que nous ne voyons pas dans ce cours et qu'il faut en général éviter)

Note

Une méthode `static` ne peut pas utiliser d'attribut de la classe dans laquelle elle est déclarée.

Note

Pour les même raisons, une méthode `static` ne peut pas utiliser une méthode non-statique (à moins de créer une instance de la classe en question).

2.c Une méthode statique célèbre

Le programme principal Java est appelé par la JVM sans instanciation initiale:

```
1 public static void main(String[] args)
```

Le mot clé `public` indique que la méthode `main` peut être exécutée par la JVM. `static` indique qu'il n'y a pas besoin d'instanciation pour exécuter la méthode `main`. `void` signifie que la méthode de renvoie rien. `main` est le nom de la méthode `args` est le nom du paramètre de la méthode (qui peut être indiqué lorsque l'on lance le programme en ligne commande java NomDeLaClassePrincipale [paramètres optionnels]) `String[]` est le type du paramètre, c'est-à-dire tableau de chaînes de caractères.

3 Variables statiques

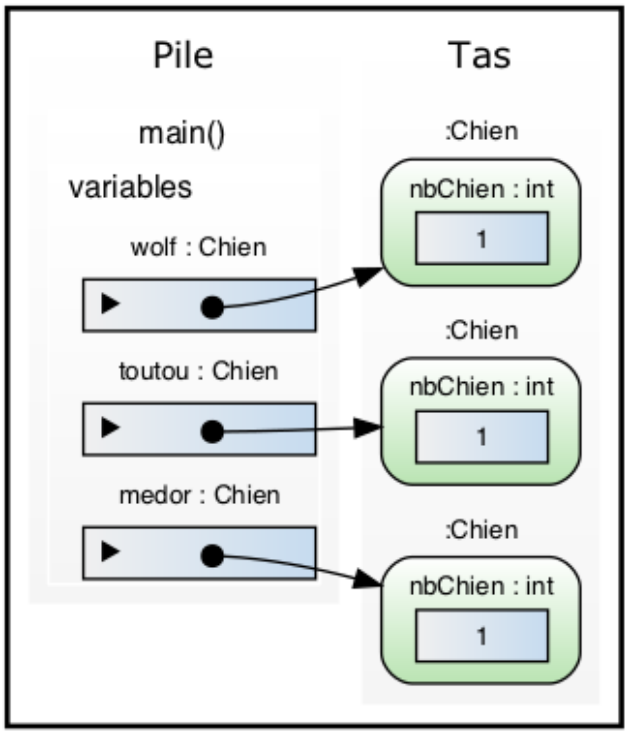
Il existe également des attributs statiques que l'on appelle **variable de classe** par opposition aux variables d'instances (attributs). Une variable de classe est partagée par toutes les instances de la classe.

Considérons dans un premier temps le code suivant:

```
1 public class Chien {
2     private int nbChien;
3
4     public Chien() {
5         nbChien++;
6     }
7 }
```

```
1 public class Test {
2     public static void main(String[] args) {
3         Chien wolf   = new Chien();
4         Chien toutou = new Chien();
5         Chien medor  = new Chien();
6     }
7 }
```

On obtient le diagramme APO suivant:



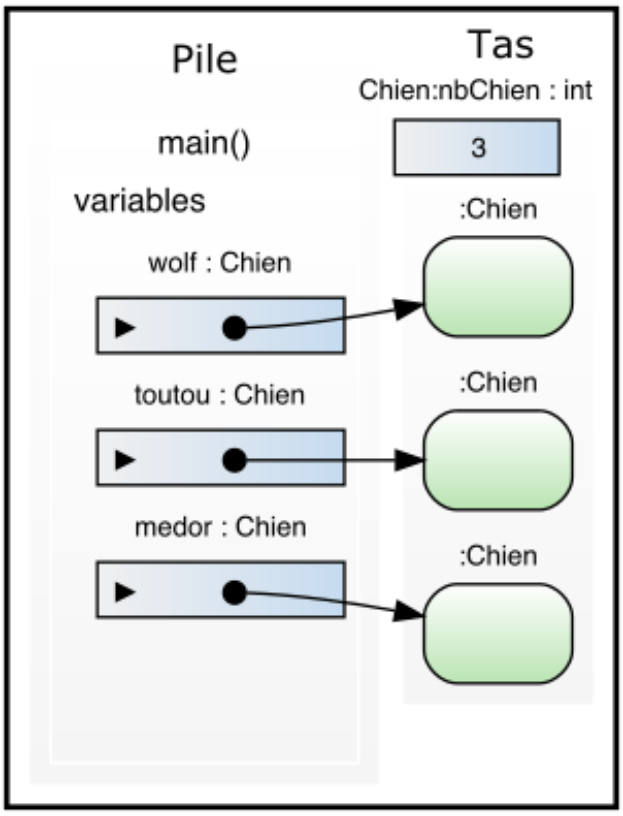
Si l'on souhaite compter le nombre d'instances de `Chien` on peut transformer l'attribut `nbChien` en variable de classe en la déclarant `static`.

On obtient le code suivant:

```
1  public class Chien
2      private static int nbChien;
3
4      public Chien() {
5          nbChien++;
6      }
```

```
1  public class Test {
2      public static void main(String[] args) {
3          Chien wolf   = new Chien();
4          Chien toutou = new Chien();
5          Chien medor  = new Chien();
6      }
7  }
```

On obtient le diagramme APO suivant:



Les variables de classe sont initialisées lorsque la classe est chargée. C'est la JVM qui choisit le moment où la classe va être chargée. En général, le chargement intervient juste avant la création d'un objet de cette classe ou l'invocation d'une méthode statique.

Les règles utilisées pour l'initialisation des variables de classe sont les même que pour les variables d'instances.