Programmation Orientée Objet

Définition

Membres

La programmation orientée objet consiste à rapprocher les traitements (fonctions) des données (variables). Cela permet de modéliser des situations de façon plus logique et naturelle. La POO s'articule donc autour de structures appelées **classes** qui possède un état (des attributs) et des comportements (les méthodes). L'ensemble des attributs et méthodes d'une classe sont appelées les membres de la classe.

En Java, on peut définir une classe à l'aide du mot clé `class`. Une classe doit être définie dans un fichier qui porte son nom. Pas convention, les noms des classes sont en PascalCase. Les noms des membres camelCase.

Définition d'une classe :

```
class Cat {


}
```

Pour définir un attribut, on précise d'abord son type, puis son nom :

```
class Cat {
    String name;
}
```

Pour définir une méthode, on précise d'abord son type de retour, puis son nom, puis ses arguments :

```
class Cat {
    String name;

    void pet(int duration) {
        // Corps de la méthode
    }
}
```

Constructeur

Le constructeur est un membre (plus spécifiquement une méthode) particulier de la classe, qui est appelé à son instanciation et qui sert à initialiser l'état de l'objet.

Pour définir un constructeur on définit une méthode qui a le nom de la classe :

```
class Cat {
    String name;

    Cat(String catName){
        name = catName;
    }
}
```

Une classe peut avoir plusieurs constructeurs (avec différents prototypes). Un constructeur peut faire appel à un autre par un appel à `this()`.

```
class Cat {
    String name;

    Cat(){
        this("Felix");
    }

    Cat(String catName){
        name = catName;
    }
}
```

Classes et Instances

Instances

Une classe est un type qui définit de quels membres seront dotés ses instances. Une instance d'une classe (aussi appelée objet), est une variable du type de la classe et possède tous les membres définis par cette dernière, avec des valeurs qui lui sont propres. On peut instancier un objet avec le mot clé `new`, suivi d'un appel au constructeur de la classe :

```
Cat cat = new Cat("Félix");
```

On peut accèder aux membres (attributs ou méthodes) de la classe avec l'opérateur `.` sur la référence de l'objet :

```
System.out.println(cat.name);
cat.pet(10);
```

Membres de classe

On peut aussi déclarer dans une classe des membres (attributs ou méthodes) qui seront communs à toutes les instances d'une classe (une valeur pour toutes les instances) grâce au mot clé `static` :

```
class Cat {
    static numberOfCats = 0;

    Cat(String catName){
        name = catName;
        numberOfCats++;
    }
}
```

Les membres de classe peuvent être utilisés au sein de la classe, ou alors à l'extérieur avec l'opérateur `.` sur le nom de la classe :

```
System.out.println(Cat.numberOfCats);
```

Référence `this`

La référence `this` est accessible dans toute classe dans les contextes non `static`. Elle pointe vers l'instance courante. Elle est utilisée pour lever des ambiguités de nommage :

```
class Cat {
    Cat(String name){
        this.name = name;
    }
}
```

Elle est également utilisée pour des raisons de visibilité afin de permettre de dissocier rapidement une variable locale d'un attribut d'instance.

Références & Garbage Collection

En Java, on dit que tout est référence. Les seuls types valeur sont les types dits primitifs :

- int
- char
- long
- double
- float
- bool
- short
- byte
- void

Tous les autres types (les classes) sont des types références, c'est-à-dire que les variables contiennent les adresses mémoires des objets. Tous les objets sont alloués en mémoire sur le tas, et c'est la JVM qui s'occupe de les désallouer lorsqu'ils ne sont plus référencés via un mécanisme appelé la **Garbage Collection**.

Encapsulation

Packages

Les packages sont les dossiers dans lesquels les classes d'une application sont sémantiquement et logiquement réparties. Les packages servent également d'espaces de noms.

Visibilité

La notion de visibilité permet de restreindre l'accès aux membres d'une classe. Il existe 3 modificateurs d'accès :

- `public` : Accès partout
- `private` : Accès uniquement dans la classe
- `protected` : Accès dans la classe et les sous classes (voir Héritage)

La visibilité par défaut (pas de modification de visibilité) est la visibilité package, et permet l'accès par toutes les classes du package.

Principe d'encapsulation

Le principe d'encapsulation dicte que seul les informations que l'interface publique (l'ensemble des membres publics) d'une classe doit être la plus restreinte possible. C'est pourquoi tous les attributs doivent être privés. Les méthodes sont publiques que si elles ont besoin de l'être.

Pour exposer avec une granularité fine les données, on utilise des méthodes, les accesseurs. Il y a deux types d'accesseurs :

- Les **getters**, pour lire un champs. Convention de nommage : get*NomDuChamps*
- Les **setters**, pour écrire un champs. Convention de nommage : set*NomDuChamps*

Exemple de champs correctement encapsulé :

```
public class Cat {
    private String name;

    public Cat(String name){
        this.name = name;
    }

    public String getName(){
        return this.name;
    }

    public void setName(String name){
        this.name = name;
    }
}
```

Un champs ne doit avoir un getter / setter que si le besoin s'en fait ressentir.

Revision #6
Created 7 November 2020 14:55:07 by Arsène Lapostolet
Updated 18 March 2021 19:39:56 by Arsène Lapostolet