Vocabulaire de la programmation objet

Alexis Nasr (d'après les slides de Arnaud Labourel)

Objet

Un **objet** :

- peut être **construit**
- est **structuré** : il est constitué d'un ensemble d'**attributs** (données de l'objet)
- possède un **état** : la valeur de ses attributs
- possède une **interface** : les opérations applicables, appelées **méthodes**

Objet et référence

- Chaque objet est identifié par une adresse mémoire appelée **référence**
- C'est cette référence qui permet d'accéder à l'objet

Syntaxe des objets

Construire un objet

```
Classe objet = new Classe(arguments);
```

- Après l'appel au constructeur et l'affectation, la variable contient la référence de l'objet
- Sans appel au constructeur, la variable contient `null` (adresse pointant vers rien)

Accéder à un attribut d'un objet

```
objet.attribut = ...;
```

Appeler une méthode d'un objet

```
objet.méthode(arguments)
```

Attention au `null`

Si une variable ne contient pas de référence à un objet, tout accès à :

- une de ses méthodes
- un de ses attributs

générera une `NullPointerException` (davantage de détails sur les exceptions dans un cours ultérieur).

Régles d'or

- Toujours penser à construire les objets dont on a besoin.
- Éviter si possible d'utiliser `null` dans son code.

Classe

Une **classe** (d'objet) définit des :

- **constructeurs** : des façons de construire/instancier les objets (**instances** de la classe)
- **attributs** (champs, propriétés ou données membres) : la structure des objets de la classe
- **méthodes** : le comportement des objets de la classe

Exemple de syntaxe de définition d'une classe

```
public class NameOfTheClass{
    public final Type1 attribute1 = new Type1(arguments);
    private Type2 attribute2;

    public NameOfTheClass(arguments){
        // constructor code
    }

    public Type2 getAttribute2(){
        return this.attribute2;
    }
}
```

Déclaration d'attributs

Exemples **sans** initialisation

```
private int count;
private final Color backgroundColor;
private final String name;
```

Exemples **avec** initialisation

```
private int count = 0;
private final Color backColor = new Color(0.5,0.2,0);
private final String name = "John Doe";
```

$\Rightarrow$ objets créés avec des valeurs de bases pour les attributs.

Mot-clés

- `private` : accessible que dans la classe
- `public` : accessible partout
- `final` : une seule affectation avant la fin de la construction

Déclaration d'une méthode

```
visibilité + type de retour + nom +
    (type paramètre 1 + nom paramètre 1, ...) + {
  instructions
}
```

Exemples

```
public int getCount() { ... }
public void setCount(int newCount) { ... }
public void launchMissile(GPSCoordinate coord) { ... }
private Vector3D getAcceleration() { ... }
```

Corps d'une méthode

Le corps d'une méthode ou d'une classe est composé :

- d'*instructions*,
- de *structures de contrôle*, permettant de décider quelles instructions exécuter.

Remarques

- Sans structures de contrôle, les instructions sont exécutées une par une, dans le sens de lecture.
- On écrit généralement une instruction ou une structure de contrôle par ligne.
- On utilise l'*indentation* pour mettre en valeur les structures de contrôle.

Déclaration d'une variable

Une *variable* est un espace mémoire nommé pour stocker une valeur *momentanément*.

Portée : une variable (qui n'est pas un attribut) existe :

- depuis le moment où elle est déclarée
- jusqu'à la sortie du bloc { ... } où elle est définie.

Remarques

- chaque variable possède :
  - un type (qui peut correspondre à un type primitif, une classe ou une interface), et ne peut contenir que des valeurs de ce type.
  - une valeur (les variables non-affectées ont une valeur par défaut : `0`, `null`, ...)

```
int count;
Vector2D position;
```

L'affectation

L'*affectation* est l'opération consistant à mettre une valeur dans un espace mémoire définit par une variable (**l-value**) :

L'affectation est soumise aux contraintes de types: la valeur stockée doit être d'un type correspondant à celui déclaré.

```
count = 12;
count = count + 4;
position = new Vector(3, -2.5);
```

la construction

La création d'un nouvel objet se fait avec le mot-clé `new`, qui déclenche :

1. l'allocation d'un espace mémoire pour contenir le nouvel objet,
2. l'initialisation des propriétés de cet objet avec les valeur de bases,
3. l'appel au constructeur de l'objet,
4. le retour de la *référence* (adresse mémoire) de l'objet par l'instruction

```
new Vector(3,-2.5);
new Spaceship();
new ArrayList<Integer>();
```

Le mot-clé `return`

Une méthode peut retourner une valeur lorsqu'elle a terminé son travail, en utilisant le mot-clé `return`.

Lors de l'évaluation de l'instruction `return`, l'expression après le `return` (le résultat) est *d'abord* évaluée (si elle existe), *ensuite* l'exécution de la méthode est interrompue.

Le résultat est retourné au niveau de l'instruction ayant fait l'appel de méthode.

Exemples d'utilisation de `return`

```
return;
return 42;
return new Vector(-5,2.333);
```

Les expressions

Les valeurs sont définies par des *expressions* :

- des littéraux : `0.01`, `-42`, `"toto"`, `...`
- combinaison d'expressions par des opérateurs `+`, `*`, `/`, `-`
- variables et attributs,
- appels de méthode (ayant un type de retour autre que `void`) ou de constructeurs

Les expressions apparaissent :

- en membre droit d'une affectation,
- en paramètre d'un appel de méthode, d'un constructeur,
- dans une condition de boucle ou de test
- après le mot-clé `return`

Section 1

À retenir absolument

Classe

Une **classe** (d'objet) définit des :

- **constructeurs** : des façons de construire/instancier les objets (**instances** de la classe)
- **attributs** (champs, propriétés ou données membres) : la structure des objets de la classe
- **méthodes** : le comportement des objets de la classe

Syntaxe de définition d'une classe

```
public NameOfTheClass{
    public final Type1 attribute1;
    private Type2 attribute2;

    public NameOfTheClass(...){
        // constructor code
    }

    public getAttribute2(){
        return this.attribute2;
    }
}
```