

Classes et Objets

CO 3

L' "approche objet" consiste à décomposer le problème à résoudre en modules, ayant une structure interne spécifique et liés entre eux par des relations telles que: "est une sorte de", "est composé de", "possède", "utilise".... Les modules modélisent des entités appartenant au domaine du problème.

Une classe est un terme logiciel traduisant la structure du module. C'est donc une unité de décomposition logicielle: notion syntaxique.

Le programme, lors de son exécution, manipule des entités créées à partir de la description donnée par les classes: objets, instances de classes.

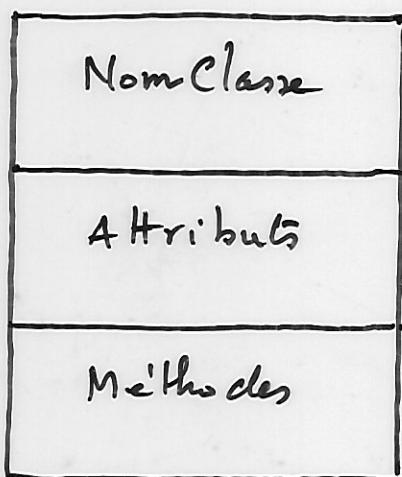
Une classe est donc un type, puisque c'est une description statique d'objets créés lors de l'exécution du programme; c'est une notion rémanente puisque chaque type influence directement l'exécution d'un système logiciel en définissant la forme des objets que le système créera ou manipulera lors de l'exécution.

Le texte d'une classe décrit les propriétés et le comportement des objets d'un certain type. Il le fait en décrivant les propriétés et le comportement d'une instance typique: instance courante.

Une classe possède des caractéristiques (membres) représentées par:

- de l'espace (mémoire): variables, attributs, membres données.
- du temps (calcul): méthodes, routines, fonctions membres.

Notation UML



Java :

→ Déclaration d'une class:

class NomClass {

// Attributs

// Méthodes

}

Ex:

Cercle

rayon: double

surface(): double

class Cercle {

double rayon;

double surface();

return 3.14 * rayon * rayon;

}

}

→ Déclaration d'une instance:

NomClass identificateur;

// Crée une variable pouvant contenir une référence.

à un objet de type "NomClass". (ne crée pas d'objet)

→ Création d'une instance:

identificateur = new NomClass();

On peut rassembler déclaration et création:

NomClass identificateur = new NomClass();

Ex:

Cercle c = new Cercle();

// déclare une instance d'un cercle de rayon 0.

c: Cercle

→ Manipulations:

```

public class Test {
    public static void main (String[] args) {
        Cercl e = new Cercl e();
        c. rayon = 10;
        System.out.println (c. surface());
    }
}

```

Règles:

Le mécanisme de base du calcul orienté objet, est l'appel de caractéristique; qui est de la forme:

ce. attribut

ou ce. méthode (args)

où : ce est la cible de l'appel, qui peut être : - une classe, si la caractéristique est déclarée "static": caractéristique de classe

ou - une instance de classe nommée.

De plus la caractéristique doit être accessible à la classe contenant l'appel (voir plus loin).

Rg:

A l'intérieur d'une classe, un appel de caractéristique de cette classe, n'a pas besoin d'être qualifié: la cible est l'instance courante (qui est désignée par la référence "this" dont l'utilisation est parfois utile pour lever des ambiguïtés)

→ L'encapsulation : premier principe de la programmation objet.

Une classe définit l'état et l'environnement (attributs et méthodes) des objets qu'elle démarre.

L'état d'un objet ne doit être modifié que sous contrôle. Il faut donc pouvoir "cacher" les attributs à son utilisateur de la classe : on les déclare "private".

Règles :

- Une caractéristique déclarée "private" n'est accessible qu'à l'intérieur de la classe elle-même.
- Une caractéristique déclarée "public" est accessible à toute classe
- Généralement les classes sont groupées en paquetages suivant les fonctionnalités. La déclaration d'une classe est généralement précédée de l'indication du paquetage auquel elle appartient :

package xx;

class x { --. . . }

Une caractéristique sans qualificatif de visibilité ou de visibilité paquetage

(visibilité par défaut) : accessible à toute classe du même paquetage.

Rq : Pour permettre l'accès contrôlé en lecture/écriture aux attributs d'une classe on définit des méthodes publiques (get / set) dits accesseurs.

Req . La visibilité d'une classe peut elle même être restreinte au paquetage auquel elle appartient (cas par défaut) ou non restreinte (déclarée "public").

- Un fichier ".java" ne peut contenir qu'une seule classe publique. (le fichier a le même nom que la classe publique).

Ex:

Un fichier "complexe.java"

package co;

public class Complexe {

 private double reel;

 private double ima;

 public double getReel() { return reel; }

 public double getIma() { return ima; }

 public void setReel (double reel)

 { this.reel = reel; }

 public void setIma (double ima)

 { this.ima = ima; }

:

}

→ Constructeurs:

Pour créer un objet on utilise l'opérateur `new` qui s'applique à un constructeur de la classe,

Ex) `Cercle c = new Cercle();`

`"Cercle()"` est une méthode particulière de la classe "Cercle" dite constructeur, qui serv à initialiser l'objet créé.

Un constructeur est une méthode spéciale qui :

- porte le même nom que la classe,
- n'a pas de type de retour (il ne s'agit pas de "void")
- est invoquée à chaque création d'objet par l'intermédiaire de l'opérateur `"new"`.

- Une classe peut définir plusieurs constructeurs (en respectant les règles de surcharge).
- Si une classe ne définit pas explicitement un constructeur il y a toujours un constructeur synthétisé (c'est un constructeur sans argument)
- Du moment que l'on définit un constructeur, le constructeur synthétisé n'est plus disponible.
- Il est généralement nécessaire d'avoir un constructeur taux argument dans une classe (constructeur par défaut)
- Un constructeur est en général une méthode publique.

Ex:

```
public class Complex{
```

```
public Complexe (double real, double imag) {
```

```
    this.real = real; this.imag = imag;
```

{

```
public Complexe (double real) {
```

```
{ this.real = real; imag = 0.0; }
```

```
public Complexe () {
```

```
real = 0.0; imag = 0.0; }
```

{

On peut créer des instances telles que,

Complexe zero = new Complexe(); // (0.0, 0.0)

Complexe c1 = new Complexe(1.5); // (1.5, 0)

Complexe c2 = new Complexe(1.5, 2.5); // (1.5, 2.5)

→ autre utilisation du mot réservé "this"

Pour écrire la réécriture du code (comme dans l'exemple précédent), le mot "this" servira à appeler un constructeur à l'intérieur d'un autre (à condition que cet appel soit la première instruction du constructeur appelant)

Ex: les 2^e et 3^e constructeur peuvent s'écrire:

- . public Complexe (double real) {

- this (real, 0.0); }

- . public Complexe () { this (0.0, 0.0); } }

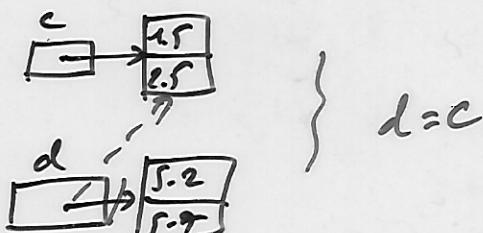
→ Initialisation :

- Les attributs d'une classe sont initialisés par défaut : numérique (0), booléen (false), caractère ('0000'), référence (null).
- L'initialisation à des valeurs données par l'utilisateur se fait dans les constructeurs ou des blocs d'initialisation.
Rq : les variables locales à une méthode ne sont pas initialisées par défaut.

→ Affectations d'objets :

- Dans une instruction telle que `int i = 5;` la variable `i` contient la valeur de l'entier (5). Il en est de même de tout type primaire : une affectation telle que `c = j;` place la valeur de `j` dans `c`.
- Pour les objets, ^{dans} une déclaration telle que :
`Complexe c = new Complexe(1.5, 2.5);`
`c` désigne une référence sur un objet.
 Soit `Complexe d = new Complexe(5.2, 5.7);`

l'affectation `d = c`, concerne des références et non les objets référencés.



`d = c` l'objet qui est repéré par `d` n'est plus accessible (devient une "miette" (garbage)) et sera rendu au système lors d'une

→ Objets comme paramètres de méthodes:

Le passage de paramètre en Java se fait par valeur.

En ce qui concerne les arguments qui sont des objets, la "valeur passée" est celle de la référence (et non une copie de l'objet).

Rg: Toute méthode (non "static") possède un argument caché désignant l'objet appartenant (this).

→ Masquage d'attribut:

Une variable locale ou un paramètre du même nom qu'un attribut, masque ce dernier dans le portée de la variable locale ou du paramètre.

On peut toutefois utiliser le mot réservé "this" pour accéder à l'attribut. - (Voir Ex: Ci-dessus).

→ Types primaires comme classes:

- À chaque type primaire correspond une "enveloppe" (wrapper) qui est une classe:

Boolean, Character, Byte, Short, Integer,
Long, Float, Double, Void

- Il ya conversion implicite d'un type primaire vers son wrapper (boxing) et inversement (unboxing) lorsque le contexte l'exige.

Ex: Integer val = 3; (boxing)

int x = val; (unboxing)

Rg: "val" est une référence:

→ Caractéristique d'instance / caractéristique de classe

Un attribut dans une classe peut être :

- relatif à une instance particulière de la classe : change avec l'instance (Ex: rayon de cercle), on l'appelle dans ce cas : variable d'instance.
- indépendant de toute instance ; partagé pour toutes les instances de la classe (Ex: un attribut dans la classe 'Cercle' qui compte le nombre de cercles créés), on l'appelle dans ce cas : variable de classe.

Une variable de classe doit être déclaré "static"

Chaque objet a sa propre version des variables d'instance, tandis que les variables de classe sont partagées par tous les instances de la classe.

Les méthodes peuvent être de la même façon, des méthodes d'instance ou des méthodes de classe (déclarées 'static').

Une méthode 'static' ne possède pas d'argument caché.

Les caractéristiques de classe peuvent être appellées sans création d'instance particulière (peuvent être qualifiées par le nom de la classe si nécessaire) :

Ex: Math.PI; -- Math.Sen(2); --

Un constructeur ne peut être "static".

Règle:

Les variables et méthodes de classe peuvent être appellées sans qualification dans toute méthode (de classe ou d'instance),

mais les variables et méthodes d'instance ne peuvent être invoquées sans qualification que dans des méthodes d'instance.