Chapitre 2

Classes et objets

Approche Orientée Objet

Idée de base de A.O.O. repose sur l'**observation** de la façon dont nous procédons dans notre vie de tous les jours.

Nous sommes entourés d'objets que nous manipulons, il nous importe peu de savoir comment ils sont fabriqués.

Dans le domaine du développement d'applications, rien n'existe sauf les DONNÉES.

A.O.O consiste à construire les mécanismes qui:

- structurent ces données;
- les régissent;

=> Afin de mieux les utiliser.

Programmation d'une Application de Gestion de Comptes Bancaires

L'application sert à gérer des dépôts/retraits d'argent.

Le Programmeur ne connaît que les données qui caractérisent les comptes ; noms des clients, numéros de comptes, types de comptes (compte courant, compte épargne etc.), soldes des comptes etc.

La seule chose importante est: le dépôt ou le retrait d'argent, pour ce faire, il faut comprendre la mécanique qui régit le fonctionnement des comptes.

À la disposition du programmeur des objets, les différents types de comptes bancaires où chaque objet sait comment se comporter. Un compte courant par exemple, saurait qu'il n'accorde pas d'intérêts pour l'argent déposé.

Programmation de l'application consiste à transmettre à ses objets un message pour leur dire qu'on désire déposer ou retirer de l'argent, à eux de faire le reste.

Écrire une application orientée objet => transmettre des ordres d'actions à des objets (préexistants et autonomes).

Si cela est réalisable => l'écriture et la maintenance des applications doivent s'en trouver considérablement simplifiées.

Programmation Orientée Objet

Programmation dans laquelle les programmes sont organisés comme des ensembles d'objets coopérant ensemble.

Objet

C'est une entité fermée douée de mémoire et de capacité de traitement.

- agissant sur réception d'un message,

- pouvant fournir un résultat.

Un objet est formé de:

- données => définissant ce qu'il est,

- programmes ou procédures => définissant ce qu'il peut faire.

Un objet est un regroupement dans une entité indépendante de données et de procédures qui manipulent ces données. Ces procédures sont appelées MÉTHODES.

Les données sont séparées du monde extérieur par les méthodes. Ces dernières définissent l'interface de l'objet ~ notion d'**encapsulation**.

Un objet est composé de 2 parties:

- **Partie interface**: opérations qu'on peut faire dessus (partie publique)

- **Partie interne**: données sensibles de l'objet (partie privée)

Les utilisateurs de l'objet ne voient que la partie interface.

- **Retrait**: objet permet le retrait de l'argent,

- **Résultat**: l'argent retiré,

- **Méthodes** (processus) utilisées: aucune idée (peu importe).

Communication avec les objets MESSAGE

Transporte l'information nécessaire aux demandes à satisfaire, c'est le moyen UNIQUE de communication avec les objets (impossible d'accéder directement aux données encapsulées d'un objet).

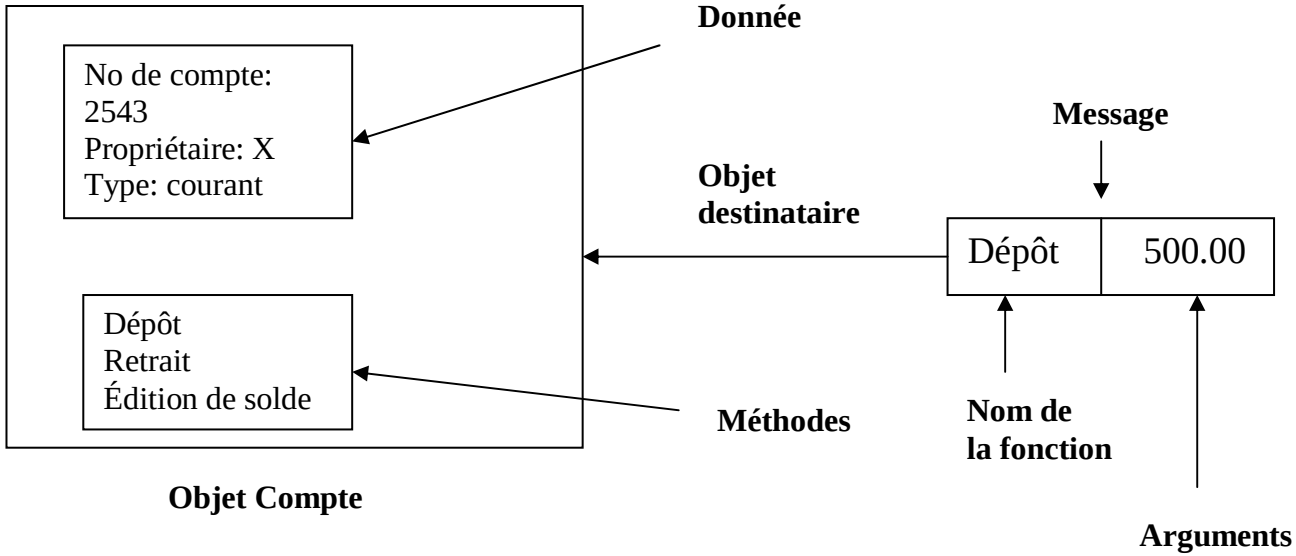
Cette communication doit contenir:

- nom de l'objet destinataire,

- énoncé de la demande (exemple: le nom d'une fonction),

- les arguments nécessaires (pour réaliser la demande)

Envoyer un message à un objet, c'est lui demander d'exécuter une de ses méthodes.



Classes

Soit l'opération mathématique: 2 + 6

Dans cet exemple l'objet: 2

Et le message: +6

Le signe + est le sélecteur ... méthode addition.

L'objet 2 reçoit le message +6, la méthode dont le nom correspond au sélecteur (ici +) est exécutée. Mais le 6 est lui-même un objet

Soit une autre opération mathématique: 6 + 2

Objet: 6

Message: +2

Le signe + est le sélecteur ... méthode addition.

Objets : Nombres entiers, différent que par leur partie DONNÉES.

Créer un nombre infini d'objets? **IMPOSSIBLE**.

Créer un objet générique "MOULE ou PROTOTYPE", contient les méthodes communes à tous les objets "nombres entiers", sa donnée est en réalité une variable susceptible de contenir, au moment de l'appel, la valeur correspondant à ce que nous voulons faire.

Une classe est un modèle décrivant le contenu et le comportement des futurs objets de la classe,

L'ensemble d'objets dont :

- le contenu = les données,

- le comportement = les méthodes.

Instanciation

Fabrication à partir du modèle de la classe, d'un objet particulier, élément de cette classe.

Déclaration

```
public class compte {
    String nom;
    double actif;
    int limite; // limite de crédit
    public void affiche() {
       // etc.
    }
}
```

La déclaration suivante :

```
compte c;
```

Ne réserve pas un espace mémoire pour contenir les éléments de compte (nom, actif et limite). C'est juste un emplacement pour une référence à un objet de type compte. Il faudra donc créer cet emplacement mémoire.

Création

```
compte c = new compte();
```

Création d'un objet de type compte et place sa référence dans c. Ainsi nous pouvons accéder aux membres de la classe, comme par exemple la méthode affiche :

```
c.affiche();
```

Constructeur

C'est une méthode appelée à la création de l'objet et sert à initialiser ses attributs.

Un constructeur permet donc d'automatiser le mécanisme d'initialisation d'un objet.

> En java, le compilateur initialise les membres données de la classe avec les valeurs de défaut suivantes : boolean/false ; char/caractère de code nul ; entier (byte, short, long, int)/0 ; flottant(float, double)/0.f ou 0. ; objet/null

Un constructeur ne retourne rien (ne pas mettre void) ; est public et porte le même nom que la classe.

syntaxe: nom_de_la_classe(arguments); //les arguments ne sont pas obligatoires.

Chaque classe a un constructeur par défaut (il est caché, mais il existe).

```
compte CB = new compte();
```

Cette instruction fait appel à un constructeur par défaut défini par le compilateur comme étant:

```
compte(){ // fait quelque chose ... }
```

Il est appelé si aucun constructeur n'a été défini explicitement dans la classe. Si par contre dans une classe donné, il a été défini un constructeur (comme vu précédemment dans le cas de la classe compte), ce dernier masque le constructeur par défaut.

Ainsi, le constructeur par défaut cesse d'exister et devient donc inaccessible ; de ce fait, l'instruction suivante:

```
compte CB new compte();
```

Elle génère une erreur, car il n'existe aucun constructeur

Un constructeur ne peut pas être appelé directement depuis une autre méthode:

```
compte c = new compte();
c.compte() ;
```

Construction et initialisation des objets

- initialisation par défaut (déjà vu).
- initialisation explicite lors de la déclaration des champs.
- exécution des instructions du corps du constructeur.

```
class A {
   public A(...) {// etc.}
   int n=10;
   int p;
}

A a = new A(...);
```

```
class A {
    public A(...) {// etc.}
    int n=10;
    int p = n+5;
}

A a = new A(...);
```

Attention à l'ordre :

```
int p = n+5;
int n=10;
```

Attribut final et constructeur

- Attribut final => initialisé qu'une seule fois.

```
class A {
   public final int x =10;
}
```

- Champ déclaré final doit être initialisé au plus tard dans le constructeur.

```
class A {
   public A() {
        x = 10;
   }
   public final int x;
}

class A {
   public A(int nn) {
        x = nn;
   }
   public final int x;
}
```

- Ne comptez pas trop sur l'initialisation par défaut pour faire le travail!

```
class A {
   public A() {// on ne touche pas à x.}
   public final int x; // erreur de compilation
}
```

=> de préférence oublier l'existence même de l'initialisation par défaut.

Destructeur

- Appelé automatiquement avant la destruction de l'objet.

- Sert à rendre des ressources aux systèmes prises par l'objet à détruire.

- Opération prise en charge par le garbage collector (GC) (ramasse-miettes).

Méthodes Statiques

- Elles sont partagées de toutes les instances (objets) d'une classe.

- Elles sont accessibles indépendamment de l'existence de l'instance d'une classe.

Accès usuel et recommandé :

```
Classe.methode
Classe : nom de la classe, membre : nom de la méthode.
```

- Attention : Une méthode statique ne peut accéder qu'à des variables statiques.

Membres Statiques

- Ils sont partagés de toutes les instances (objets) d'une classe.

- Ils sont accessibles indépendamment de l'existence de l'instance d'une classe.

Accès usuel et recommandé :

```
Classe.membre
Classe : nom de la classe, membre : nom du membre.
```