ITI 1121. Introduction to Computing II *

Marcel Turcotte
School of Information Technology and Engineering

Version of 14 janvier 2012

Résumé

- Programmation orientée objet
  - Encapsulation

*. These lecture notes are meant to be looked at on a computer screen. Do not print them unless it is necessary.

Programmation orientée objet — OO

- encapsulation
- analyse OO/conception de logiciels
- l'interface d'une classe/d'un objet
- un objet a un état
- composer avec les objets
- classes versus objets
- anatomie d'une classe (implémentation)
- variables : d'instance, de classe, locales et les paramètres (formels et actuels)
- méthodes : de classe ou d'instance
- constructeurs -1-
- polymorphisme *ad hoc* (overloading)
- héritage (réutilisation)
- constructeurs -2-
- polymorphisme : héritage, classes abstraites et interfaces

Programmation Orientée Objet
Conception de logiciels

Pourquoi la programmation orientée objet ?

La conception de logiciels est une **activité abstraite**, en ce sens qu'on ne peut toucher ou voir ses éléments constitutifs.

La programmation orientée objet fournit des outils, classes et objets, qui rendent ce processus **plus palpable** ; on peut dessiner des diagrammes illustrant les objets d'un système ainsi que leurs interactions.

Abstraction

Les abstractions nous permettent d'ignorer les détails d'un concept et de nous concentrer sur certains aspects jugés importants.

On compare souvent une abstraction à une ≪boîte noire≫. On peut l'utiliser sans se préoccuper de son contenu.

Programmation structurée

(≪*Procedural abstraction*≫)

La façon de structurer vos programmes, pour la majeure partie de ITI 1520, était à l'aide de la programmation structurée.

L'abstraction utilisée s'appelle une procédure (ou encore, routine ou fonction pour certains langages, et méthode en Java).

Si une méthode a bien été conçue, elle peut-être utilisée comme une boîte noire ; les détails de son implémentation ne sont pas importants et souvent même non disponibles.

On peut, bien souvent, utiliser une méthode de tri sans en connaître l'algorithme.

⇒ Cette façon de structurer les programmes fonctionne assez bien lorsqu'il y a peu de données et de types.

Abstraction de données

(*Data abstraction*)

Les enregistrements et les structures sont les principales formes de l'abstraction de données.

Ils permettent de regrouper les éléments d'information qui sont logiquement liés les uns aux autres en une seule entité : l'enregistrement.

Par exemple, afin de modéliser un point, une coordonnée.

Abstraction de données

Imaginez concevoir une application traitant des informations liées à des points dans un espace à deux dimensions.

Sans l'abstraction de données, pour comparer deux points, il faut passer en paramètre les coordonnées de chaque point.

```
boolean equal( int x1, int y1, int x2, int y2 ) {
    // ...
}
```

Abstraction de données

Lorsque l'application change (ajout ou retrait de caractéristiques du dossier étudiant), vous devrez mettre à jour toutes méthodes recevant un dossier étudiant.

Sans mentionner que l'énumération de toutes ces variables est très fastidieuse.

La vérification des types est moins puissante, comme le démontre cet exemple : paramètre formels : equal(int x1, int y1, int x2, int y2), paramètres effectifs : equal(x1, x2, y1, y2).

L'abstraction de données est donc essentielle.

Abstraction de données

Par contre, lorsque les données à modéliser sont complexes, et que plusieurs méthodes agissent sur ces données, l'abstraction de données seule est bien souvent inadéquate.

Abstraction de données

Imaginez concevoir un système traitant des transactions bancaires.

Il y aura, dans ce système, à plusieurs endroits, des énoncés semblables à ceux-ci :

```
if ( estCheque( compte ) ) {
   // un certain traitement
} else if ( estEpargne( compte ) ) {
   // traitement
} else {
   // etc.
}
```

Abstraction de données

```
if ( estCheque( compte ) ) {
   // un certain traitement
} else if ( estEpargne( compte ) ) {
   // traitement
} else {
   // etc.
}
```

1. Qu'arrive-t-il lorsqu'un nouveau type de compte doit être ajouté à l'application ? Vous devrez modifier toutes les parties du programme dont l'exécution dépend du type de compte, afin d'insérer un nouveau cas ;
2. Ce qui rend ce processus compliqué c'est le fait que les énoncés liés à un type de compte bancaire sont distribués à travers l'application ;

Abstraction procédurale
+
abstraction de données
=
programmation orientée objet

Le concept central de la programmation orientée objet est l'objet !

Un **objet** c'est
– une abstraction de données (variables), et ;
– une abstraction procédurale (méthodes)

Un logiciel est vu comme une collection d'objets qui interagissent, les uns avec les autres, afin de résoudre un problème commun.

(Java est un langage de programmation orientée objet !)

Un pas en arrière

Lorsque l'on fait de la programmation structurée, ce que vous avez fait en ITI 1520, on se concentre sur les traitements.

En programmation orientée objet, on se contre sur les objets.

Dès le tout début des activités menant à la création d'un logiciel, il faut identifier les objets ainsi que leurs interactions.

On parle alors d'analyse orientée objet.

On peut faire une telle analyse sans connaître les détails de l'implémentation.

Activités du développement d'une application

1. Analyse des besoins
   (définir le problème) ;
2. Design
   (comment diviser le système en sous-systèmes et quelles sont leurs interactions) ;
3. Programmation ;
4. Assurance de qualité ;
5. Gestion du projet ;

⇒ OO contribue à toutes ces activités, mais particulièrement à l'intégration du design et de la programmation.

De gros projets

La programmation orientée objet facilite la conception de gros systèmes, qui sont composés de plusieurs classes développées par plusieurs programmeurs, souvent sur des périodes allant de plusieurs mois à plusieurs années.

Couplage

Nous dirons que deux classes ont un **couplage fort** si l'implémentation d'au moins l'une des deux classes dépend de l'implémentation de l'autre ; c'est-à-dire qu'une classe accède les variables de l'autre.

Plus il y a de dépendances entre les classes plus la mise à jour du système est difficile.

Une application bien conçue est telle que l'interface de chacune des classes est définie de façon claire et précise.

Exemples

Identifier les objets :
- **Commerce-e :** clients, items, inventaire, transactions, . . . ;
- **Jeu d'échec :** pièces, tableau de jeu, usagers ;
- **Manufacture :** lignes d'assemblage, robots, items, pièces, . . . ;

Pour certaines ≪classes≫ d'objets, il n'y a qu'une ≪instance≫ ; c'est le cas du tableau dans l'application du jeu d'échec.

Alors que d'autres classes décrivent les caractéristiques communes d'une collection d'objets.

Chaque objet est **unique** bien que 2 objets peuvent avoir le même état (le contenu des variables d'instances est le ≪même≫).

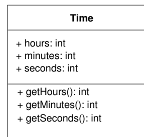
Objet

Un objet a
- des propriétés qui définissent son état ;
- des comportements : ce que l'objet peut faire, ses réponses aux requêtes.

Unified Modeling Language (UML)

UML est un langage graphique standardisé afin de modéliser des systèmes logiciels orientés objet.

Un diagramme de classe est constitué d'une boîte divisée en trois sections : le nom de la classe, les attributs et les méthodes.



Nous introduirons d'autres éléments de notation lorsque nécessaires.

⇒ Simon Benett, Steve McRobb and Ray Farmer (1999) *Object-Oriented Systems Analysis and Design using UML*. McGraw-Hill.

Comment ?

Très bien, un objet regroupe en une même entité des données ainsi que les méthodes qui transforment ces données, mais comment spécifier l'objet.

La classe définit les caractéristiques communes d'une collection d'objets.

Il peut y avoir plusieurs ≪instances≫ (exemples) d'une classe, mais chaque instance n'a qu'une classe.

Les propriétés (l'état) d'un objet sont spécifiées à l'aide de variables d'instances (attributs).

Les comportements de l'objet sont spécifiés à l'aide de méthodes d'instances.

Classe et objet

Est-ce que la distinction entre le concept de classe et d'objet est bien claire ?

Les objets sont des entités crées lors de l'exécution.

Instance et un objet, est-ce le même concept ?

Oui, en effet instance et objet sont deux termes décrivant un seul et même concept.

Le terme instance est préférablement utilisé dans des phrases ayant la forme ≪l'instance de la classe . . .≫, on parle du rôle de l'objet.

Les objets sont des ≪exemples≫ (instances) d'une classe.

Classe et objet

La classe spécifie le contenu des objets.

Dans l'exemple du jeu d'échec, il peut y avoir une classe décrivant les propriétés et comportements qui sont communs à toutes les pièces.

Les propriétés d'une pièce incluent : une couleur, un nom et une position, par exemple.

À l'exécution du programme, plusieurs ≪instances≫ seront créées : le roi noir, la reine blanche, etc.

Un des comportements d'une pièce pourrait être de se déplacer.

Nommer les classes

Utilisez un nom singulier, mettre la première lettre en majuscule, exemple : Pièce.

Les standards rendent les programmes plus faciles à lire.

La déclaration suivante :

```
Compteur compteur;
```

indique clairement l'utilisation de référence **compteur** afin de désigner un objet de la classe **Compteur**.

Autre exemple : **System.out.println()**

Variables d'instance

Qu'est-ce qu'une variable d'instance ?

La classe spécifie les variables d'instance de chaque objet.

Chaque objet réserve un espace mémoire pour chaque variable d'instance.

Les variables d'instance peuvent être d'un type primitif ou référence.

Les variables références sont utilisées afin de créer des associations entre objets : un agenda électronique aura des objets de type événement, un événement a lieu à une certaine heure, etc.

Méthodes d'instance

Qu'est-ce qu'une méthode d'instance ? Quand doit-on définir une méthode d'instance ?

Les méthodes d'instance ont accès aux variables de l'instance.

Compteur

L'exemple le plus simple est sans aucun doute le compteur. Imaginer l'article utilisé en sports afin de cumuler les points.

Celui que j'ai en tête a une fenêtre permettant de lire la valeur courante, un bouton permettant d'incrémenter la valeur du compteur de 1, et un autre afin de remettre la valeur à zéro.

L'état du compteur est la valeur courante représentée par le compteur.

Pourquoi pas ?

Une seule valeur est nécessaire pour modéliser le compteur.

De plus, cette valeur peut facilement être représentée à l'aide d'un type primitif de Java, tel que `int` ou `long`.

Voici un compteur,

```
int counter1;
```

Pourquoi pas ?

Je peux l'initialiser et augmenter sa valeur au besoin,

```
counter1 = 0;
counter1++; // mis pour counter1 = counter1 + 1
```

En voici un autre,

```
int counter2;
```

Et ici tout un tableau de compteurs,

```
int[] counters;
counters = new int[5];
```

⇒ Quel est le problème ?

Pourquoi pas ?

Cette implémentation ne modélise pas très bien le concept du computeur, parce qu'aucun mécanisme ne prévient les usages tels que ceux-ci :

```
counter1 = counter1 + 5;
// ...
counter1 = counter1 - 1;
// ...
```

Pourquoi pas ?

Sur une note plus technique, une telle implémentation rend le partage du compteur difficile.

Qu'entend-t-on par partage ?

```
void foo() {

  // déclare un compteur

  int counter = 0;

  // l'utilise

  // passe la valeur à une autre méthode

  bar(counter);

  // à moins que bar ne retourne la valeur après usage et
  // que foo assigne cette valeur de retour à la variable counter,
  // l'appel à bar n'affectera pas la valeur de counter
  // dans la méthode foo!
}
```

de plus, il se peut que la méthode bar() ait été écrite par quelqu'un d'autre, que vous n'ayez pas accès au code source, qu'il y ait des usages abusifs, la méthode foo() peut oublier de mettre à jour la valeur de counter, etc.

Version OO

```
public class Counter {
    private int value = 0;

    public int getValue() {
      return value;
    }

    public void incr() {
        value++;
    }

    public void reset() {
        value = 0;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Counter counter = new Counter();
        System.out.println("counter.getValue()->"
                               +counter.getValue());
        for (int i=0; i<5; i++) {
            counter.incr();
            System.out.println("counter.getValue()->"
                               +counter.getValue());
        }
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        Counter counter = new Counter();
        System.out.println("counter.getValue()->"
                               +counter.getValue());
        for (int i=0; i<5; i++) {
            counter.incr();
            System.out.println("counter.getValue()->"
                               +counter.getValue());
        }
        counter.value = -9;
    }
}

Test.java:13: value has private access in Counter
        counter.value = -9;
               ^
```