Les concepts de l'orienté objet et les techniques d'héritage

Équipe Pédagogique Réseau Informatique@ UVCI
2018

Juillet 2018
Version 1.0

Table des matières

Objectifs

À la fin de cette leçon, vous serez capable de :

- Définir les objets, classes, spécialisation, composition
- Présenter les principes abstraction et encapsulation
- Définir les différentes formes d'héritage
- Présenter ce qu'est un polymorphisme et un patron

Les concepts de l'orienté objet

Objectifs

- Définir les objets, classes, spécialisation, composition
- Présenter les principes abstraction et encapsulation

Le but de ce cours est d'introduire les fondamentaux de l'approche orientée objet

1. L'approche orientée objet

L'intérêt de l'orientée objet

L'orientée objet est une approche de programmation qui confère au programmeur de bonnes propriétés telle que :

- Une grande clarté conceptuelle,
- La robustesse face aux modifications futures,
- Une meilleure maintenabilité.

Une des particularités de l'approche orientée objet est de regrouper les traitements et les données en une seule et même entité comme le montre la figure 1.

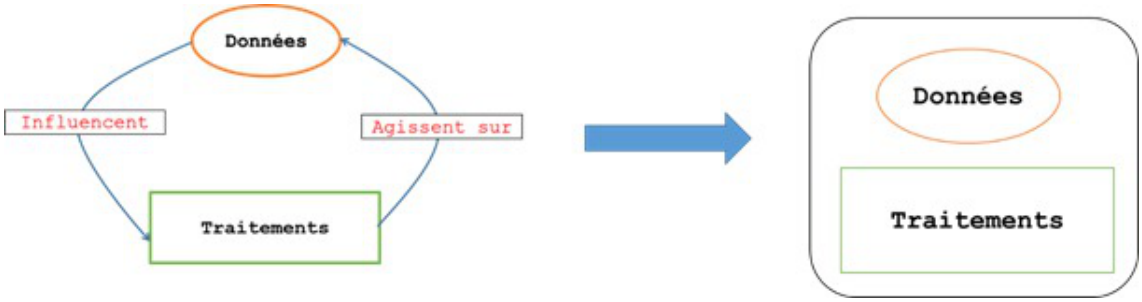

Figure 1 : Modèle orienté objet

Le fait de pouvoir regrouper en une seule et même entité les données caractéristiques d'un objet ainsi que les traitements spécifiques va permettre d'établir un lien explicite entre les différentes entités et donner beaucoup plus de clarté au programme.

Les propriétés fondamentales de l'orienté objet

De façon global, il faut savoir que le Concept Orienté Objet va donner un certain nombre d'outillages permettant davantage de :

- Robustesse par rapport aux changements (amélioration du code) et face aux erreurs de manipulation de données.
- Modularité ;

- Lisibilité à vos programmes ;
- Une meilleure Maintenabilité.

Les concepts de base de l'orienté objet

Un des objectifs principaux de la notion d'objet est de mieux organiser des programmes complexes autour de quatre concepts de base à savoir:

- l'encapsulation
- l'abstraction
- l'héritage et
- le polymorphisme

2. Encapsulation

Principe d'encapsulation

Principe d'encapsulation : regrouper dans le même objet informatique («concept»), les données et les traitements qui lui sont spécifiques :

- attributs : les données incluses dans un objet
- méthodes : les fonctions (= traitements) définies dans un objet

Les objets sont définis par leurs attributs et leurs méthodes.

Les niveaux de perceptions des objets

En plus du regroupement des données et des traitements relatifs à une entité, l'encapsulation permet en effet de définir deux niveaux de perception des objets :

- niveau externe : partie « visible » (par les programmeurs-utilisateurs) :
    - l'interface : prototypes de quelques méthodes bien choisies
- niveau interne : il présente les détails d'implémentation à savoir :
    - les méthodes et attributs accessibles uniquement depuis l'intérieur de l'objet (ou d'objets similaires),
    - la définition de toutes les méthodes de l'objet.

3. Abstraction

Processus d'abstraction

Pour être véritablement intéressant, un objet doit permettre un certain degré d'abstraction.

Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :

- des caractéristiques communes à tous les éléments et
- des mécanismes communs à tous les éléments

Pourquoi abstraire/encapsuler ?

- L'intérêt de regrouper les traitements et les données conceptuellement reliées est de permettre une meilleure visibilité et une meilleure cohérence au programme, d'offrir une plus grande modularité.
- L'intérêt de séparer les niveaux interne et externe est de donner un cadre plus rigoureux à l'utilisation des objets utilisés dans un programme. Les objets ne peuvent être utilisés qu'au travers de leurs interfaces (niveau externe) et donc les éventuelles modifications de la structure

interne restent invisibles à l'extérieur.

Remarque : Règle

les attributs d'un objet ne doivent pas être accessibles depuis l'extérieur, mais uniquement par des méthodes.

4. La Classe

Définition : Une classe

En Orienté Objet :

- le résultat des processus d'encapsulation et d'abstraction s'appelle une classe
- Une classe permet de désigner une catégorie d'objets et définit un type.

une réalisation particulière d'une classe s'appelle une instance et

- Une instance = objet
- Un objet est une variable dont le type est une classe

Une classe est la généralisation de la notion de type défini par l'utilisateur, dans lequel se trouvent associées à la fois des données et des fonctions ou méthodes.

Pour déclarer une classe, on utilise la syntaxe suivant : *class nom_ de_ la_ classe { ... } ;*

Exemple : Exemple de classe

class Rectangle { ... };

Ici Rectangle est le nom de la classe.

Définition : Type dans l'approche orientée objet

la classe définit un nouveau type qui permet de créer différentes variables, comme par exemple: Rectangle rect1, rect2, rect3;

Ici Rectangle est une classe et les éléments rect1, rect2, rect3 ont pour type Rectangle.

Déclaration d'une instance

La déclaration d'une instance d'une classe se fait de façon similaire à la déclaration d'une variable comme en algorithmique :

nom_classe *nom_ instance* ;

Avec notre exemple précédent, *Rectangle* = nom de la classe et *rect1* = le nom d'une instance.

Portée des attributs d'une classe

Les attributs d'une classe constituent des variables directement accessibles dans toutes les méthodes de la classe (i.e. des « variables globales à la classe »). Ces attributs peuvent être des variables publiques ou privés pour la classe.

On parle de « portée de classe ».

Il n'est donc pas nécessaire de les passer comme arguments des méthodes (fonctions ou procédures ou sous algorithmes).

Les méthodes d'une classe

Les méthodes d'une classe sont donc :

- des fonctions qui sont propres à la classe ou

- des procédures (sous algorithmes) qui sont propres à la classe.

Mais qui en plus de ce fait, ont donc accès aux différents attributs de la classe.

Si une méthode a besoin de recevoir des paramètres de l'extérieur de la classe pour pouvoir fonctionner, alors il faudra déclarer les paramètres à la méthode.

Dans la plupart des cas on déclare :

- Privé :
    - Tous les attributs
    - La plupart des méthodes
- Public :
    - Quelques méthodes bien choisies (interface)

Définition externe des méthodes d'une classe

Les méthodes sont des fonctions un peu particulières que l'on met dans une classe y compris avec leur définition mais ceci rend souvent le code peu lisible.

De ce fait, on a pas une vue synthétique de l'ensemble des prototypes des méthodes offertes par une classe.

Pour une meilleure lisibilité du code, mais aussi pour une meilleure modularisation (c'est-à-dire une meilleure séparation entre les différentes parties), il peut être nécessaire de définir des méthodes à l'extérieur de la classe.

Cela permet tout simplement une organisation de l'écriture du programme qui est un tout petit peu différent.

Pour relier la définition d'une méthode à la classe pour laquelle elle est définie, il suffit d'utiliser l'opérateur « :: » de résolution de portée :

La déclaration de la classe contient les prototypes des méthodes

les définitions correspondantes spécifiées à l'extérieur de la déclaration de la classe se font sous la forme :

typeRetour *NomClasse*::*nomFonction*(type1 param1, type2 param2, ...)

{ ...

}

Constructeur d'une classe

Un constructeur est une méthode :

- appelée automatiquement lors de la déclaration d'un objet
- chargée d'effectuer toutes les opérations requises en « début de vie » de l'objet (dont l'initialisation des attributs).

Les constructeurs sont des méthodes presque comme les autres. Les différences sont :

- pas de type de retour
- même nom que la classe
- invoqués automatiquement à chaque fois qu'une instance est créée.

Un constructeur est une fonction membre qui sert à initialiser les données membres.

Remarque : Nombre de constructeurs par classe

Une classe peut donc avoir plusieurs constructeurs, avec de paramètres différents.

Destructeur d'une classe

Si l'initialisation des attributs d'une instance implique la mobilisation de ressources : fichiers, périphériques, portions de mémoire (pointeurs), etc.

il est alors important de libérer ces ressources après usage !

Comme pour l'initialisation, l'invocation explicite de méthodes de libération n'est pas satisfaisante (fastidieuse, source d'erreur, affaiblissement de l'encapsulation).

Le destructeur d'une classe est une méthode sans paramètre

- pas de surcharge possible
- Son nom est celui de la classe, précédé du signe ~(tilda).

Si le destructeur n'est pas défini explicitement par le programmeur, le compilateur du langage de programmation en génère automatiquement une version minimale.

5. Exercice : Exercices

[Solution n°1 p 15]

Exercice : Exercice 1

Quels sont les propriétés fondamentales de l'orienté objet ?

- [ ] la robustesse
- [ ] la modularité
- [ ] la lisibilité des codes
- [ ] une meilleure maintenabilité

Exercice : Exercice 2

Quels sont les concepts de base de l'orienté objet ?

- [ ] encapsulation
- [ ] abstraction
- [ ] héritage
- [ ] polymorphisme

Exercice : Exercice 3

Qu'est ce qui caractérise un objet ?

- [ ] les attributs
- [ ] les méthodes
- [ ] les classes

Exercice : Exercice 4

Une classe est une

- [ ] catégorie d'objet
- [ ] généralisation de la notion de type

Exercice : Exercice 5

Soit la syntaxe suivante : Point point1, point2 ;

point1 et point2 sont des _____ de la classe _____. Point est une _____ et les éléments point1, point2, sont de _____ Point.

Les techniques d'héritage et Polymorphisme

II

Objectifs

- Définir les différentes formes d'héritage ;
- Présenter ce qu'est un polymorphisme et un patron

1. L'héritage

Introduction

Après les notions d'encapsulation et d'abstraction, le troisième aspect essentiel de « Orientée Objet » est la notion d'héritage.

L'héritage représente la relation «est-un».

Il permet de créer des classes plus spécialisées, appelées *sous-classes*, à partir de classes plus générales déjà existantes, appelées *superclasses*.

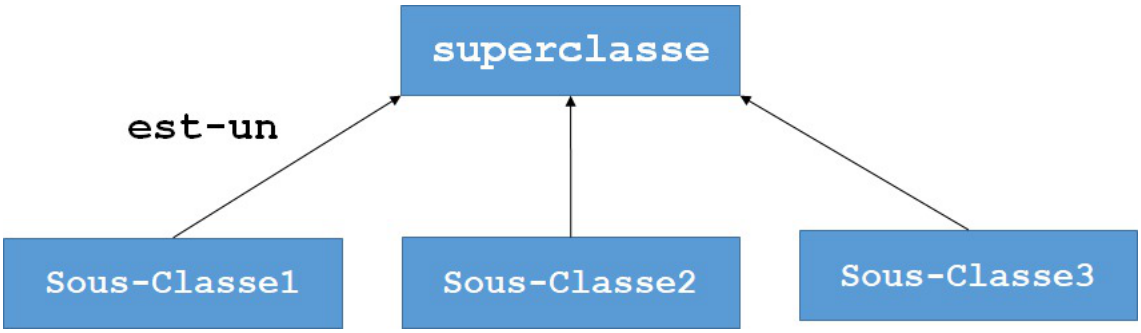

Figure 2 - Superclasse et sous-classe

Principe d'héritage simple

Lorsqu'une *sous-classe C1* est créée à partir d'une *super-classe C*,

- le type est hérité : un C1 est (aussi) un C
- C1 va hériter de l'ensemble :
  - des attributs de C
  - des méthodes de C

(sauf les constructeurs et destructeur)

- Les attributs et méthodes de C vont être disponibles pour C1 sans que l'on ait besoin de les redéfinir explicitement dans C1.

Remarque : sous classe

Par ailleurs :

- des attributs et/ou méthodes supplémentaires peuvent être définis par la sous-classe C1.
- des méthodes héritées de C peuvent être redéfinies dans C1.

Super-classe et sous-classe

Une super-classe :

- est une classe « parente »
- déclare les attributs/méthodes communs
- peut avoir plusieurs sous-classes

Une sous-classe est :

- une classe « enfant »
- étend une (ou plusieurs) super-classe(s)
- hérite des attributs, des méthodes et du type de la super-classe

L'opérateur qui permet de dire qu'une classe hérite d'une super-classe est " *:*". La syntaxe d'héritage est spécifié comme suit :

class NomSousClasse : public *NomSuperClasse*

Remarque : L'héritage simple

Le concept d'héritage constitue l'un des fondements de l'approche Orientée Objet. La figure 1 est un exemple de l'héritage simple.

L'héritage multiple

Dans le cadre de l'héritage multiple, une classe peut *hériter de plusieurs autres classes*, elle offre la possibilité de créer des classes dérivées à partir de plusieurs classes de base.

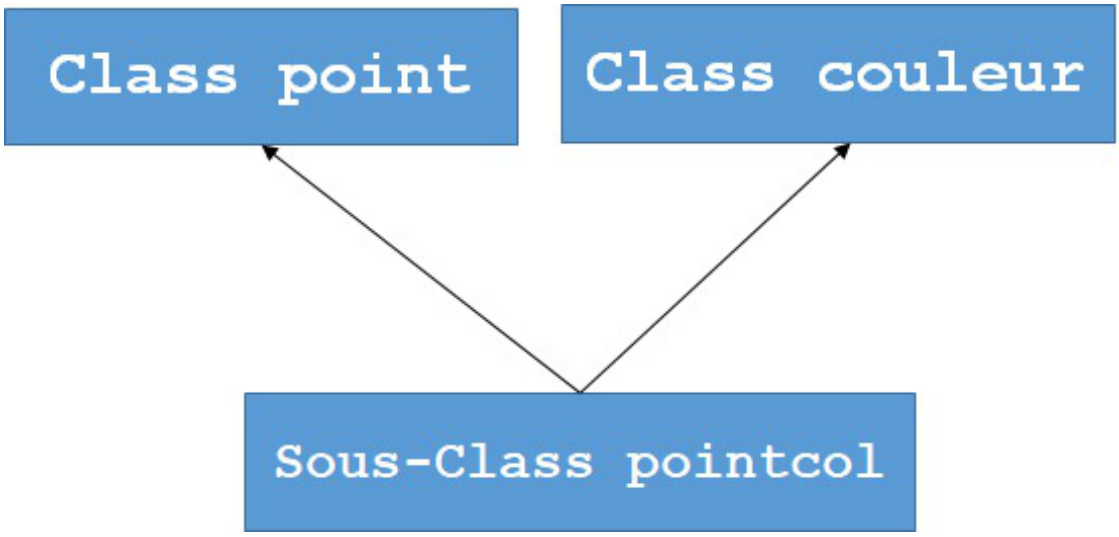

Figure 3 - héritage multiple

2. Le Polymorphisme

Introduction

Le nom de polymorphisme signifie qui peut prendre plusieurs formes. Cette caractéristique est un des concepts essentiels de l'orientée objet. Alors que l'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.

On distingue généralement trois types de polymorphisme :

- Le polymorphisme ad hoc (également surcharge ou en anglais overloading) ;
- Le polymorphisme paramétrique (également généricité ou en anglais template) ;
- Le polymorphisme d'héritage (également redéfinition, spécialisation ou en anglais overriding).

Définition : Polymorphisme Ad hoc

Le polymorphisme ad hoc permet d'avoir des fonctions de même nom, avec des fonctionnalités similaires, dans des classes sans aucun rapport entre elles.

Définition : Polymorphisme paramétrique

Le polymorphisme paramétrique, appelé généricité, représente la possibilité de définir plusieurs fonctions de même nom mais possédant des paramètres différents (en nombre et/ou en type). Le polymorphisme paramétrique rend ainsi possible le choix automatique de la bonne méthode à adopter en fonction du type de donnée passée en paramètre.

Ainsi, on peut par exemple définir plusieurs méthodes homonymes addition() effectuant une somme de valeurs.

- La méthode int addition(int, int) pourra retourner la somme de deux entiers,
- La méthode float addition(float, float) pourra retourner la somme de deux flottants,
- La méthode char addition(char, char) pourra définir au gré de l'auteur la somme de deux caractères.

Définition : Polymorphisme d'héritage

La possibilité de redéfinir une méthode dans des classes héritant d'une classe de base s'appelle la spécialisation. Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque : il s'agit du polymorphisme d'héritage.

Ceci permet de faire abstraction des détails des classes spécialisées d'une famille d'objet, en les masquant par une interface commune (qui est la classe de base).

3. Les patrons

Les patrons de fonctions

Un patron de fonction est en quelque sorte *un modèle* avec lequel le compilateur d'un langage est capable, en fonction des besoins, de générer plusieurs fonctions génériques réelles qui différeront par le type de données qu'elles auront à manipuler.

Un patron condense le code en offrant la possibilité d'écrire une seule fois la définition d'une fonction.

Le mot-clé retenu pour les patrons est template suivi de class qui précise le type. Voici la syntaxe à utiliser :

template <class type>

Les patrons de classe

Les patrons sont aussi applicables sur les classes, on parle alors de patrons de classes.

Les fonctions membres d'un patron de classe sont aussi des patrons de fonctions qui ont un en-tête de patron identique au patron de classe.

De cette façon, nous obtenons des classes génériques qui vont pouvoir, comme les patrons de fonctions, traiter des types de données différents.

Exercice : Exercices

[Solution n°2 p 15]

Exercice : Exercice 1

Quelles sont les différentes formes d'héritage ?

- [ ] l'héritage simple
- [ ] l'héritage multiple
- [ ] l'héritage naturel

Exercice : Exercice 2

Soit la syntaxe suivante : class cercle : public figure

cercle est une classe qui _____ de la classe figure. Dans ce cas, on dit que figure est une _____ et cercle est une _____.

Exercice : Exercice 3

Il existe combien de types de polymorphisme ?

- ( ) 4
- ( ) 3

Solutions des exercices

> Solution n°1

Exercice p. 9

Exercice 1

- [x] la robustesse
- [x] la modularité
- [x] la lisibilité des codes
- [x] une meilleure maintenabilité

Exercice 2

- [x] encapsulation
- [x] abstraction
- [x] héritage
- [x] polymorphisme

Exercice 3

- [x] les attributs
- [x] les méthodes
- [ ] les classes

Exercice 4

- [x] catégorie d'objet
- [x] généralisation de la notion de type

Exercice 5

Soit la syntaxe suivante : Point point1, point2 ;

point1 et point2 sont des `instances` de la classe `Point`. Point est une `classe` et les éléments point1, point2, sont de `type` Point.

> Solution n°2

Exercice p. 14

Exercice 1

- [x] l'héritage simple

☑ l'héritage multiple

☐ l'héritage naturel

Exercice 2

Soit la syntaxe suivante : class cercle : public figure

cercle est une classe qui `hérite` de la classe figure. Dans ce cas, on dit que figure est une `super-classe` et cercle est une `sous-classe`.

Exercice 3

○ 4

◉ 3