Isabelle Le Glaz

Conception Objet - Partie 1

Le développement procédural

En programmation procédurale

Un programme se compose d'une séquence d'opérations sur des structures de données qui mettent en œuvre un algorithme, une procédure, pour résoudre un problème.

Pour la majorité des problèmes, la procédure est suffisamment longue et compliquée pour rendre le programme difficile à écrire et coûteux à maintenir si l'on ne dispose pas de méthodes permettant de gérer sa taille et sa complexité.

Les techniques de programmation procédurale donnent des méthodes pour diviser et structurer les programmes de façon qu'ils soient facile à construire, à comprendre et à maintenir.

Ces techniques concernent essentiellement l'organisation de la séquence d'opérations du programme, et ne tiennent pas compte en général des structures de données sur lesquelles sont effectuées ces opérations.

La décomposition fonctionnelle

La décomposition fonctionnelle est une méthode de division d'un gros programme en fonctions.

Cette méthode de conception aboutit à un ensemble important de fonctions, hiérarchiques, où les fonctions haut-placées délèguent une partie de leur travail à des fonctions inférieures.

Cela s'appelle la programmation descendante, parce qu'on commence par une description de haut niveau du programme , puis on affine peu à peu l'écriture jusqu'à arriver à l'implémentation détaillée de bas niveau.

Limites et inconvénients du développement procédural

Factoriser les traitements n'a malheureusement pas que des avantages. Les fonctions sont devenues interdépendantes : une simple mise à jour du logiciel à un point donné, peut impacter en cascade une multitude d'autres fonctions. On peut minorer cet impact, pour peu qu'on utilise des fonctions plus génériques et des structures de données ouvertes. Mais respecter ces contraintes rend l'écriture du logiciel et sa maintenance plus complexe.

En cas d'évolution majeure du logiciel, le scénario est encore pire. Même si la structure générale du logiciel reste valide, la multiplication des points de maintenance, engendrée par le chaînage des fonctions, rend l'adaptation très laborieuse. Le logiciel doit être retouché dans sa globalité.

Le développement objet

Une solution : La programmation objet

Une application objet met en œuvre un certain nombre d'objets encapsulant données et traitements et communiquant entre eux par l'intermédiaire d'envois de messages.

Ceux-ci sont à la programmation orientée objet ce que sont les appels de sous-programmes à la programmation structurée.

Avantage : on centralise dans les structures de données (les objets) la définition de ces données et le comportement de ces données. Le point de maintenance est devenu unique.

La Programmation Orientée Objet

La programmation orientée objet (POO) ou programmation par objet, a été élaborée par Alan Kay dans les années 1970.

C'est une méthode de développement informatique qui consiste en la définition et l'interaction de briques logicielles appelées objets ;

un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre.

Il possède une structure interne et un comportement, et il sait communiquer avec ses pairs.

Il s'agit donc de représenter ces objets et leurs relations ; la communication entre les objets via leur relation permet de réaliser les fonctionnalités attendues, de résoudre le ou les problèmes.

Vocabulaire de base de la conception objet

Un objet est une entité aux frontières précises qui possède:

- Une identité (un nom).
- Un ensemble d'attributs qui caractérisent son état
- Un ensemble d'opérations (méthodes) qui en définissent le comportement.

Une classe est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.

Un objet est une instance de classe (une occurrence d'un type abstrait).

Les classes

Les classes

Une classe est un ensemble d’objets définis par un ensemble commun d’attributs et de méthodes.

La classe est une structure informatique particulière dans le langage objet. Elle décrit la structure interne des données et elle définit les méthodes qui s'appliqueront aux objets de même famille (même classe) ou type. Elle propose des méthodes de création des objets dont la représentation sera donc celle donnée par la classe génératrice. Les objets sont dits alors instances de la classe.

C'est pourquoi les attributs d'un objet sont aussi appelés variable d'instance et les messages opération d'instance ou encore méthodes d'instance. L'interface de la classe (l'ensemble des opérations visibles) forme les types des objets. Selon le langage de programmation, une classe est soit considérée comme une structure particulière du langage, soit elle même comme un objet (objet non-terminal). Dans ce dernier cas, la classe a besoin elle aussi d'être créée et définie par une classe : ce sont les méta-classes.

Les objets

Les Objets

Un objet est un élément particulier d’une classe, appelé aussi instance. Un objet a une valeur pour chacun des attributs de sa classe. L’ensemble de ses valeurs définit son état.

Un objet peut être défini comme une entité possédant un état interne, sur lequel il est possible d'exercer des actions et possédant des propriétés.

Actions et propriétés correspondent à ce que l'objet donne à voir de lui-même au monde extérieur, c'est à dire à sa partie publique.

L'état correspond à la partie privée de l'objet : lui seul a accès, en général, à la totalité de cet état. Un objet possède donc, à la façon d'un iceberg, une partie publique et une partie privée.

Les objets

Concrètement, un objet est une structure de données valuées et cachées et qui répond à un ensemble de messages. Cette structure de données définit son état tandis que l'ensemble des messages qu'il comprend décrit son comportement :

- Les données ou champs qui décrivent sa structure interne sont appelées ses attributs ;
- L'ensemble des messages forme ce que l'on appelle l'interface de l'objet ; c'est seulement au travers de celui-ci que les objets interagissent entre eux. La réponse à la réception d'un message par un objet est appelée une méthode (méthode de mise en œuvre du message) ; elle décrit comment le message doit être répondu.

Les attributs (ou plus exactement leur représentation informatique) et les méthodes peuvent être cachés et ainsi former une boîte noire. C'est le principe d'encapsulation. Son avantage principal réside dans la capacité à pouvoir modifier la structure interne des objets ou les méthodes associées aux messages sans impacter les utilisateurs des objets.

Les Objets

Les objets ont une identité : deux instances sont différentes même si elles ont exactement les mêmes valeurs pour tous leurs attributs.

La méthode appelée à la création d'un objet est appelée constructeur de l'objet

La méthode appelée à la destruction de l'objet est appelée destructeur.

L’encapsulation

L’Encapsulation

L’encapsulation consiste, pour une classe, à protéger l’accès à ses attributs. Les instances d’autres classes ne peuvent consulter ou modifier ces attributs que via des méthodes (interfaces).

L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.

Pour encapsuler, on utilise des attributs privés (visibles seulement de la même classe), ou protégés (visibles seulement des instances de la classe ou de ses sous-classes), en conjonction avec des méthodes publiques.(accesseurs)

L’encapsulation

- L'encapsulation facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface. Elle rend ainsi l’utilisation de cette classe indépendante de la représentation de ses attributs, voire de l’absence ou de la présence de ces attributs en tant que tels.

- L'encapsulation garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs). Elle permet par exemple d’interdire de modifier un attribut seul alors que sa valeur est liée à un autre attribut de la classe.

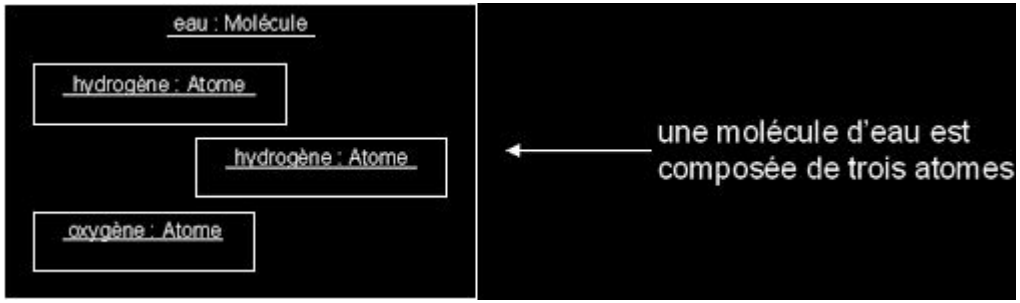
L’agrégation

Agrégation

- Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe.
  Une relation d'agrégation permet donc de définir des objets composés d'autres objets.

- L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.

- Agrégation, exemple :



Héritage

Héritage et polymorphisme

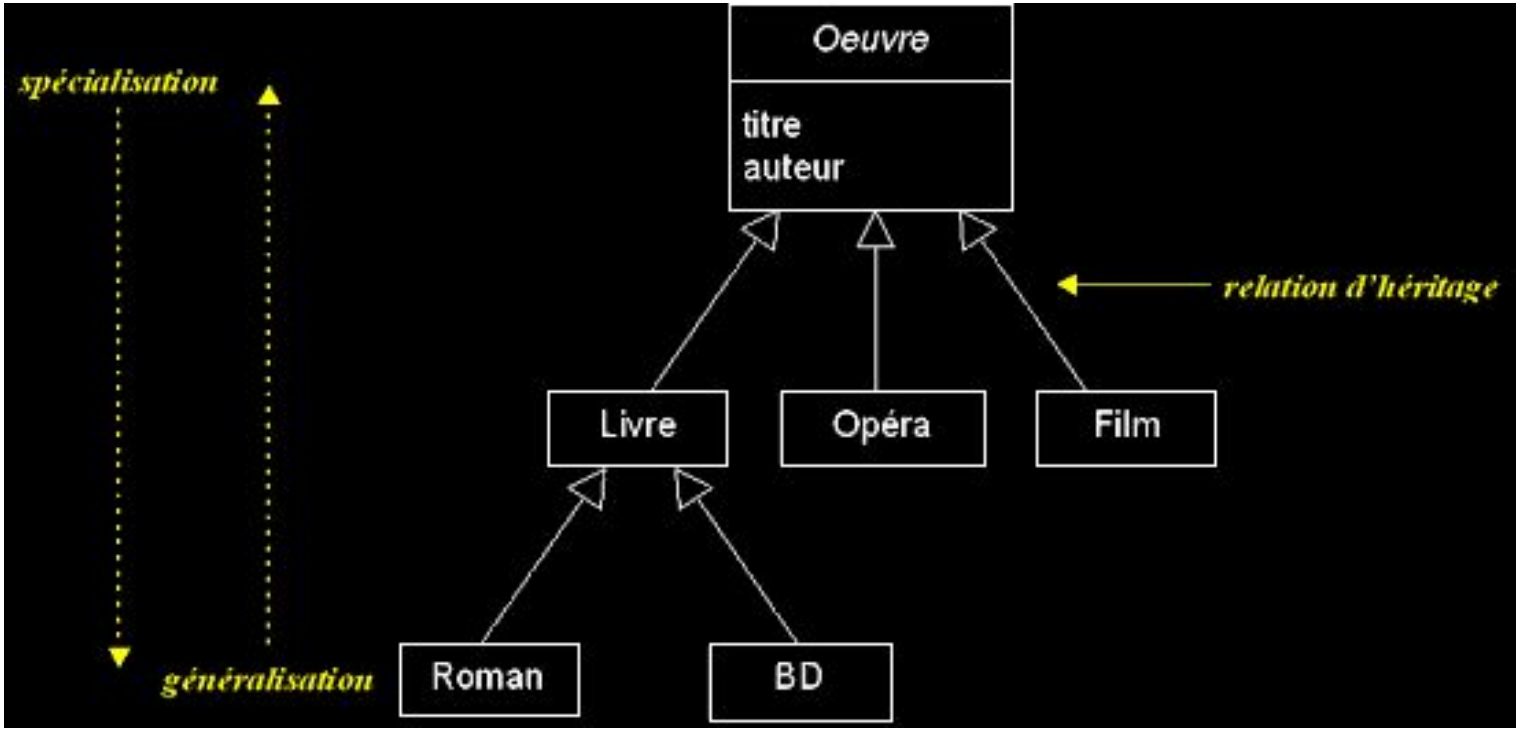
L'héritage est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe.

Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.

Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.

La spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple.

L'héritage évite la duplication et encourage la réutilisation.


Oeuvre
titre
auteur
spécialisation
relation d'héritage
Livre
Opéra
Film
Roman
BD
généralisation

Héritage

- Une classe qui hérite (ou dérive) d'une autre classe a les mêmes attributs et méthodes qu'elle, et peut définir des attributs et méthodes supplémentaires.

- Ses instances peuvent être utilisées comme des instances de la sous-classe ou de la surclasse

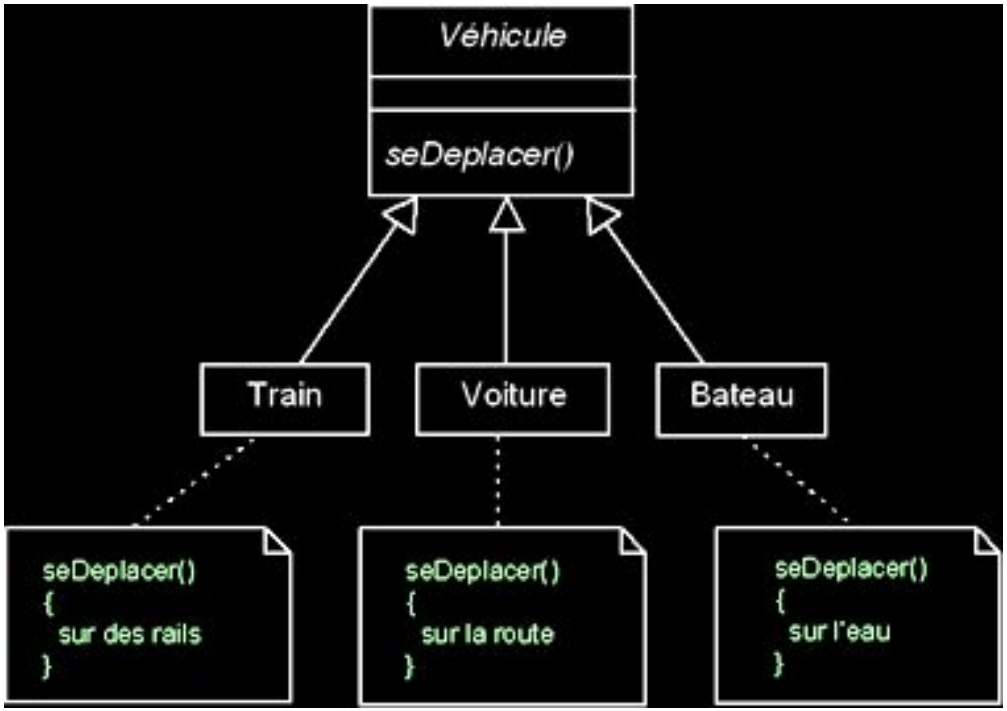
Le polymorphisme

Polymorphisme

Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.

Le polymorphisme augmente la généricité du code.

Une sous-classe peut redéfinir certaines méthodes de la sur-classe. Pour une instance donnée, qu'elle soit utilisée comme instance de la sous-classe ou de la sur-classe, c'est le corps de la méthode correspondant à sa classe réelle qui sera utilisé.


Véhicule
seDeplacer()
Train
Voiture
Bateau
seDeplacer()
{
sur des rails
}
seDeplacer()
{
sur la route
}
seDeplacer()
{
sur l'eau
}

Les objets – Typage et polymorphisme

Dans la programmation par objet, chaque objet est typé. Le type définit la syntaxe (comment l'appeler ?) et la sémantique (qu'est ce qu'il fait ?) des messages auxquels peut répondre un objet. Il correspond donc, à peu de chose près, à l'interface de l'objet

Un objet peut appartenir à plus d'un type. C'est ce que l'on appelle le polymorphisme. Ceci permet d'utiliser des objets de types différents là où est attendue un objet d'un type précis, dès que ceux-ci satisfont le type requis.

Classes abstraites

Méthodes et classes abstraites

Certaines classes sont les sur-classes de nombreuses autres classes, mais n'ont pas d'instance directe.

Cela se traduit généralement par des méthodes abstraites, c'est-à-dire des méthodes déclarées mais non définies. Il ne peut y avoir alors d'instances de la classe (seulement des instances des sous-classes qui définissent ces méthodes abstraites).

Utilisation de l’héritage

- Abstraire : permettre de manipuler des objets de classes différentes comme des instances d’une même sur-classe (en général abstraite)
- Etendre : dériver une classe d’une autre pour y ajouter des caractéristiques
- Factoriser : regrouper du code commun à plusieurs classes.

Interface

Ensemble de déclarations de méthodes sans définition. Une classe implémente une interface si elle définit chacune des méthodes spécifiées dans l'interface.

Une interface donne un point de vue sur le comportement d'un objet, différent du point de vue donné par les sur-classes. On peut manipuler des objets en sachant seulement que leur classe implémente une interface donnée. Une classe peut implémenter plusieurs interfaces.