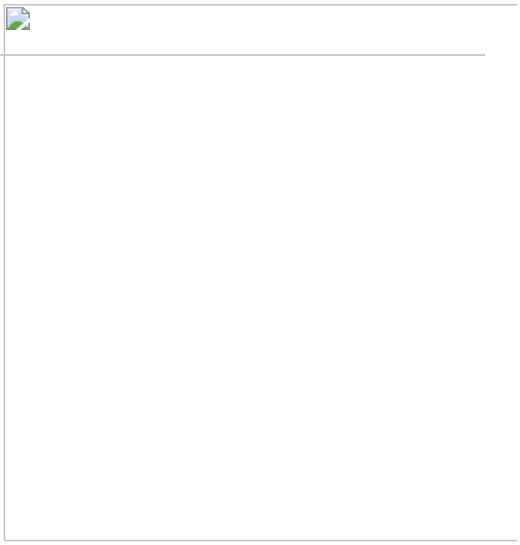
La programmation orientée objet

Par Paul-Arthur DOBETZKY • Publié le 12/10/2015 à 21:03:51 • Noter cet article: ★☆☆☆☆ (5 votes)
Avis favorable du comité de lecture

Paul-Arthur DOBETZKY

Définition de la programmation orientée objet



Deux grands types de programmation existent:
- La programmation procédurale ou structurée comme le C où on définit des fonctions s'appelant mutuellement, chacune faisant un certain nombre d'actions pouvant être divisés en sous actions jusqu'a revenir aux fonctions basiques. Par exemple si l'on veut calculer la vitesse d'un train, le procedural conduira à appeler la fonction *Train_vitesse()* à chaque fois. Le code faisant référence au train est donc dispersé dans l'ensemble du programme.
- La programmation orientée objet (ou OO) où on manipulera uniquement des objets c'est-à-dire des groupes de variables et de méthodes associés à des entités appelées Objets les intégrant naturellement. Dans cette configuration on effectuera directement *Train.vitesse()* . Tout le code touchant à l'objet *Train* se trouve ainsi regroupé ce qui confère un certain avantage au programme ainsi qu'à son programmeur.

Avantages de la programmation orientée objet

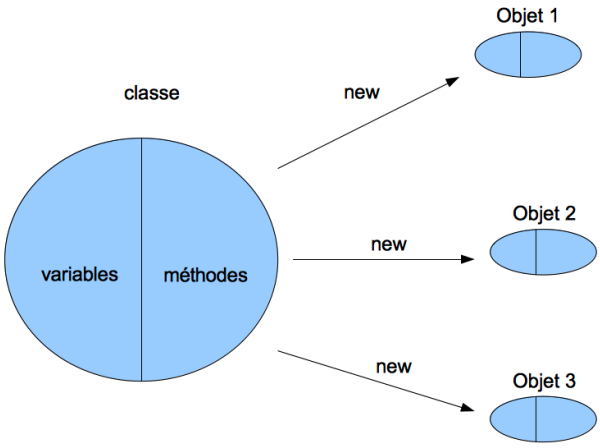
L'orienté objet remplace le procédural dans les grands programmes car il présente de multiples avantages:
-Facilité de compréhension (Permet de regrouper toutes informations sur un *objet* dans le code : si mon programme gère des voitures, l'objet du même nom contiendra la marque de celle-ci, sa vitesse ainsi que sa couleur, etc...)
- L'encapsulation (la programmation orientée objet permet la protection de l'information contenue dans un objet. Les données ne sont pas accessibles directement par l'utilisateur, celui-ci devant passer par les méthodes publiques.).
-La modularité du code (On peut généralement récupérer 80 % du code d'un projet pour le réutiliser sur un projet similaire contrairement à la programmation procédurale. De plus les objets permettent d'éviter la création de code redondant, permettant donc un gain de temps et donc d'argent).

Notions générales

Si l'on veut étudier la programmation orientée objet, on doit d'abord connaitre les notions comprises dans les schémas suivants:

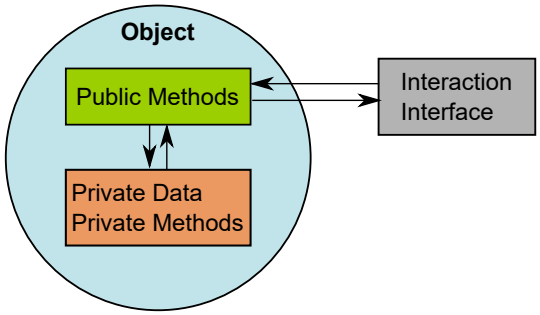
Classes et Objets

En programmation orientée objet un objet est un conteneur d'informations et de mécanisme représentant un sujet manipulé dans le programme. Ce sujet est souvent tangible dans la réalité. Comme son nom l'indique, c'est le concept principal de l' orienté objet.
Si le le conteneur est l'objet, alors la classe est le modèle à partir duquel il reçoit les comportements (méthodes) et caractéristiques (variables). Ceux-ci sont généralement basés sur ceux qu'ils représentent: Un objet de classe *voiture* aura des variables représentant le modèle, la couleur, la vitesse ... et des méthodes pour récupérer ou modifier ces variables.



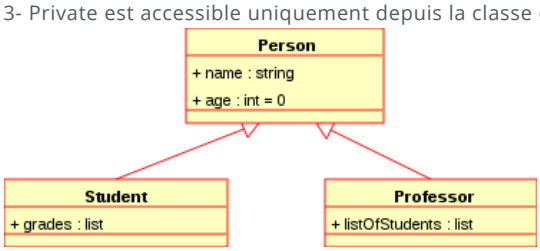
Encapsulation

Comme montré dans le schéma ci-dessous et écrit précédemment l'encapsulation est l'idée de protéger l'information contenue dans un objet et de ne pouvoir récupérer et manipuler ces données qu'à partir de méthodes présentes dans ce même objet. De ce fait l'utilisateur extérieur ne pourra pas modifier les propriétés de l'objet. L'objet est ainsi vu de l'extérieur comme une boîte noire ayant certaines propriétés et ayant un comportement spécifié. Si un programmeur veut ajouter au comportement d'une classe où la modifier il devra le faire par héritage.



Héritage et Polymorphisme

L'héritage est un concept qui permet de réutiliser et d'adapter les objets, celui-ci étant basé sur le principe d'un arbre généalogique: Certaines classes peuvent faire l'objet d'une fragmentation en sous-classes. C'est le cas dans le schéma ci-dessous des classes *Student* et *Professor* qui héritent toutes deux de la classe *Person*, recevant ses méthodes et attributs.
L'héritage peut être partiel : dans ce cas, seule une partie des attributs ou méthode de la classe-mère est repris par la classe-fille. Il peut être simple ou multiple : une même classe peut hériter de plusieurs classes.
Pour qu'une classe fille puisse hériter des propriétés de la classe mère, il faut que les propriétés de la classe mère possèdent des attributs de visibilité compatibles. Il existe dans la plupart des langages trois niveaux de visibilité :
1- Public est accessible directement en dehors de la classe
2- Protected est visible par la classe courante, toutes les classes filles, mais inaccessible en dehors des classes mère/filles.
3- Private est accessible uniquement depuis la classe courante à partir de méthode



Le polymorphisme quand à lui est un concept consistant à donner une interface unique à des attributs pouvant avoir différents types, permettant de n'appeler qu'une seule fonction pour afficher une forme géométrique que ce soit un carré ou un cercle, faisant donc gagner du temps et de la lisibilité au code.