10.1 Retour sur la programmation orientée objet

# Notion de classe

La classe permet de définir un type d'objet dont les attributs sont partagés par un ensemble d'instances.

Plus particulièrement, une classe définit deux types d'attribut :

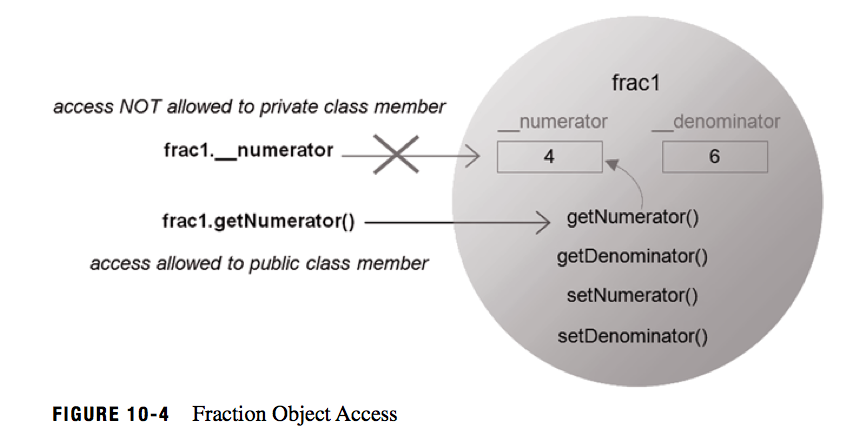
* des variables ;
* des méthodes.

La classe regroupe ces deux types d'attributs en un tout cohérent et fonctionnel. Par exemple, la classe string permet d'engendrer toutes sortes d'instances de chaînes de caractères :

# Notion d’encapsulation

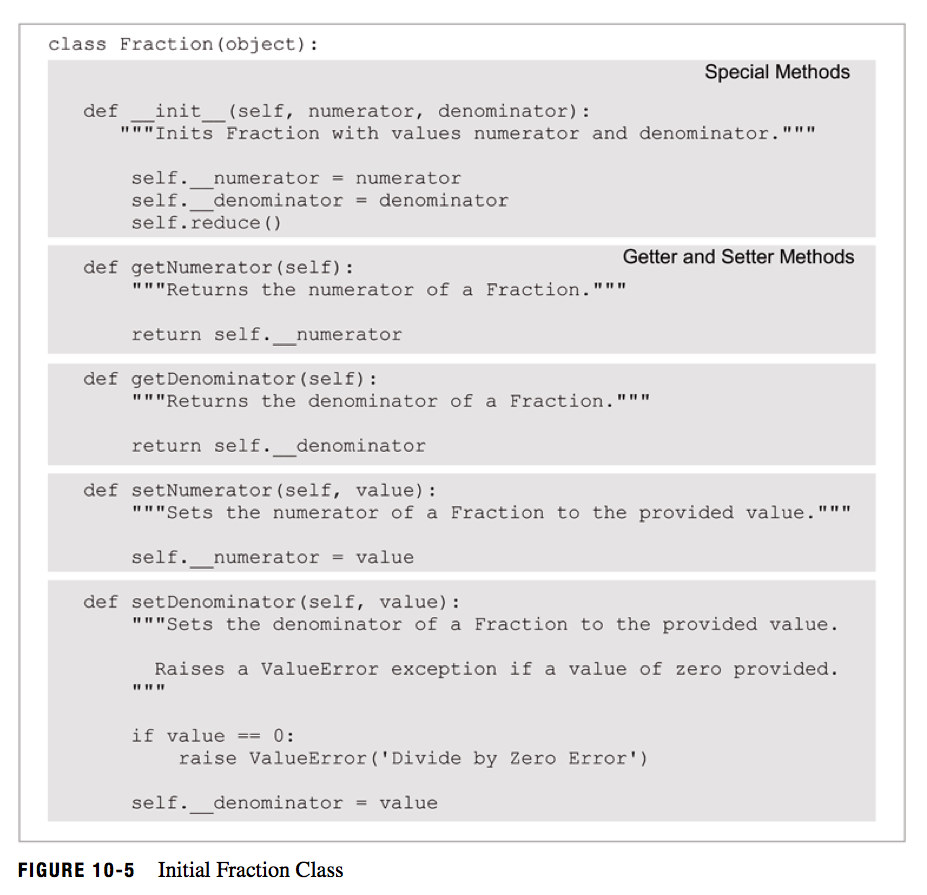
L'encapsulation est le processus qui permet de regrouper les variables et les méthodes qui caractérisent les objets d'une classe. Nous allons illustrer cette notion à l'aide d'une classe de nombres fractionnaires.

L'encapsulation permet de regrouper les variables et les méthodes qui définissent les attributs d'un type d'objet, mais également de protéger certains d'entre eux en les rendant inaccessibles à l'utilisateur. Par exemple, une fraction est constituée d'un numérateur et d'un dénominateur, mais on ne veut pas nécessairement que l'utilisateur de la classe accède directement à ces variables. Souvent, on préférera que l'utilisateur emploie une méthode afin récupérer ou modifier la valeur de ces variables membres de la classe :

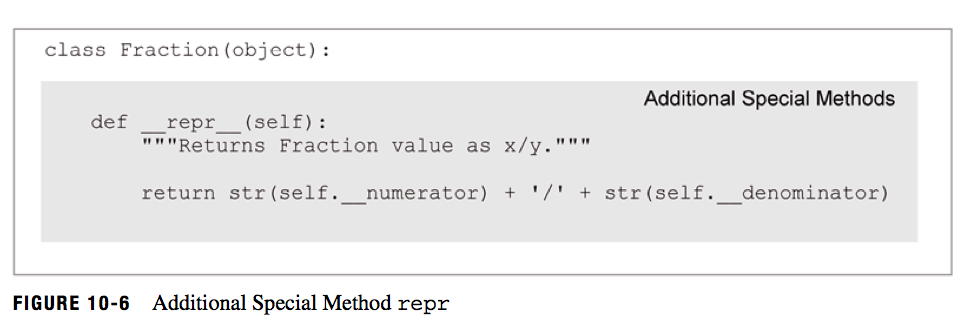


Cette approche évite d'exposer à l'utilisateur les détails d'implantation de la classe.

En python, pour protéger un membre d'une classe, on doit mettre deux caractères de soulignement (\_\_) devant son nom. Alors, Python refusera à l'utilisateur l'accès à cet attribut. Cependant, l'accès sera permis à l'intérieur des méthodes de la classe. Voici un début d'implantation de la classe Fraction, illustrant ce mécanisme de protection :



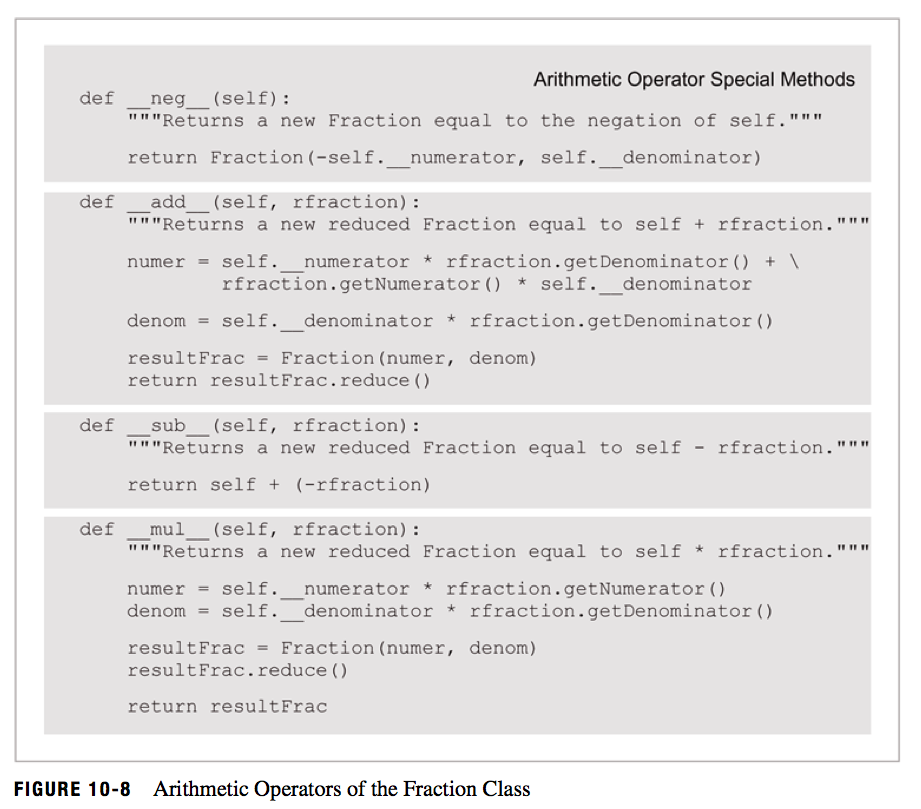
Les méthodes qui servent à accéder aux données de la classe s'appellent des accesseurs (en anglais des « getters » et « setters »). Dans une certaine mesure, ces méthodes d'interface protègent l'utilisateur des changements d'implantation qui pourraient éventuellement survenir dans la classe.



Le rôle de la méthode \_\_repr\_\_ est similaire à celui de la méthode \_\_str\_\_, mais avec une nuance importante. Les deux méthodes doivent produire une représentation de l'objet sous la forme d'une chaîne de caractères, mais dans le premier cas, celle-ci se doit d'être détaillée afin que Python puisse reconstruire l'objet sans équivoque. Dans le second cas, l'objectif est simplement d'être lisible pour un humain. Souvent, les deux méthodes produisent le même résultat. Dans ce cas, c'est \_\_repr\_\_ qui fait le travail et \_\_str\_\_ fait simplement appel à \_\_repr\_\_. Parfois, elles produisent des résultats différents.

Le rôle de la méthode \_\_repr\_\_ est similaire à celui de la méthode \_\_str\_\_, mais avec une nuance importante. Les deux méthodes doivent produire une représentation de l'objet sous la forme d'une chaîne de caractères, mais dans le premier cas, celle-ci se doit d'être détaillée afin que Python puisse reconstruire l'objet sans équivoque. Dans le second cas, l'objectif est simplement d'être lisible pour un humain. Souvent, les deux méthodes produisent le même résultat. Dans ce cas, c'est \_\_repr\_\_ qui fait le travail et \_\_str\_\_ fait simplement appel à \_\_repr\_\_. Parfois, elles produisent des résultats différents.

On peut définir tous les opérateurs qui s'appliquent à nos classes d'objets. Par exemple, les opérateurs arithmétiques :

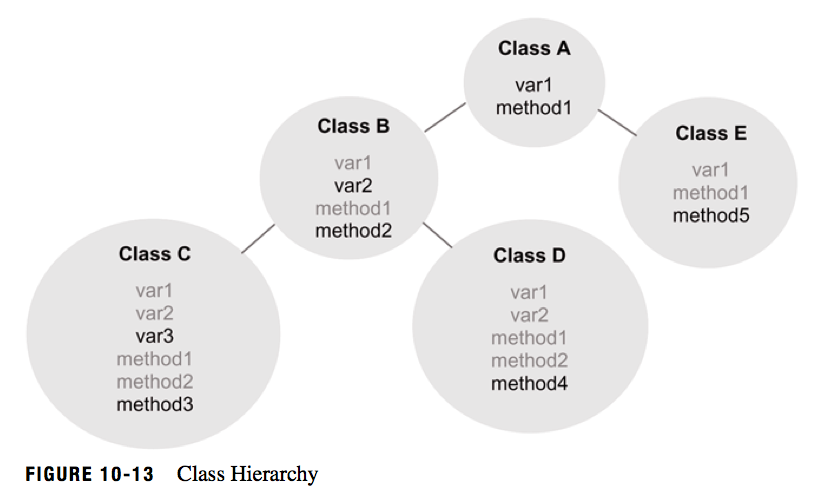


où les opérateurs de comparaison :



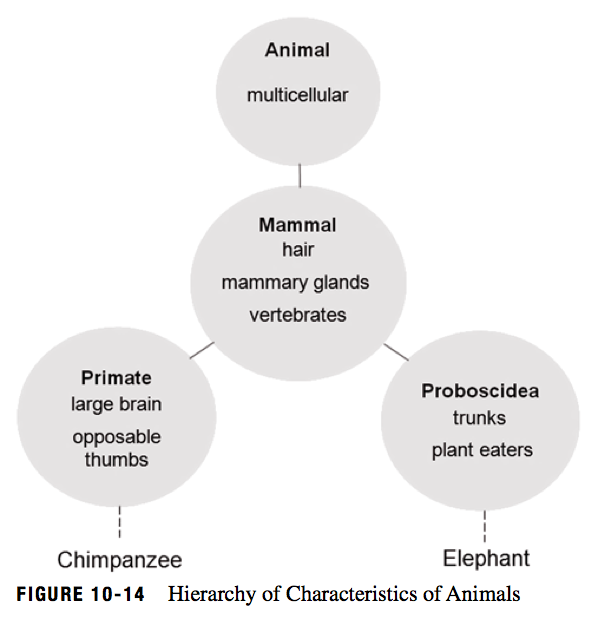
# Notion d’héritage

En orienté objet, l'héritage est l'habilité d'une classe à pouvoir hériter des membres d'une autre classe. Une classe qui hérite est qualifiée de classe dérivée (en anglais « subclass ») et la classe dont elle hérite de classe de base (en anglais « superclass&thinp;»). Dans l'exemple ci-dessous, les classes B et E héritent de A, et C et D héritent de B :



Dans cet exemple, A est une classe de base pour toutes les autres classes et celles-ci hériteront par conséquent de tous ses attributs. De même pour C et D qui hériterons de tous les attributs de B (en plus de ceux de A).

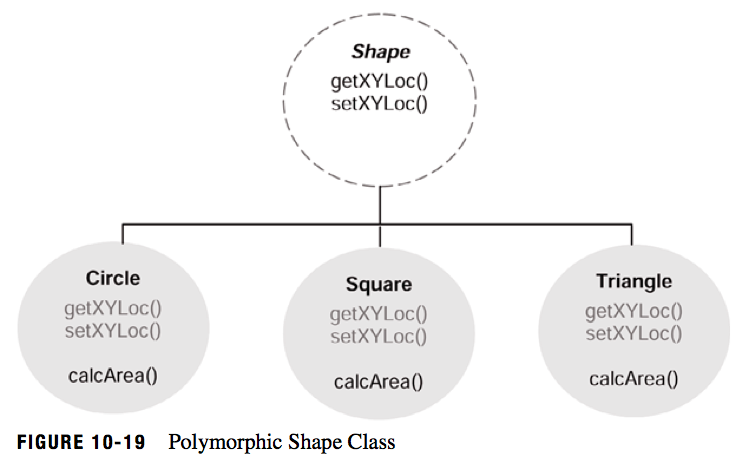
L'héritage est une relation de type « est un cas particulier de » :



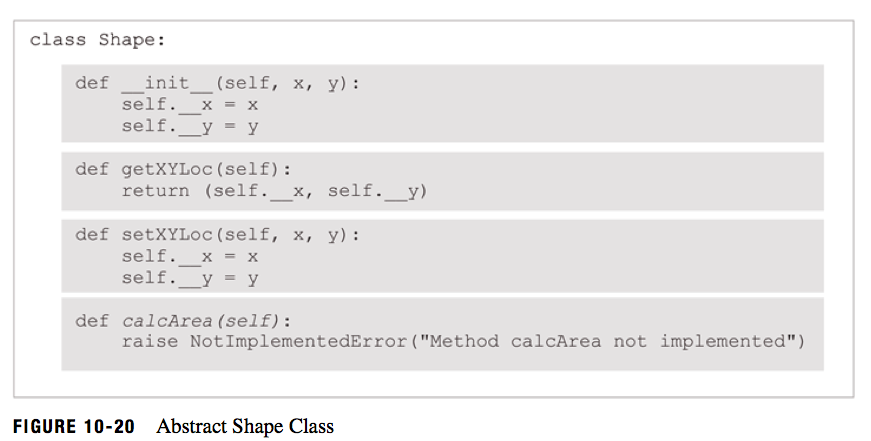
Par exemple, un mammifère est un cas particulier d'un animal, car tous les mammifères possèdent tous les attributs de l'animal, en plus d'attributs qui leur sont propres. De même, les primates possèdent tous les attributs des mammifères, en plus des attributs qui leur sont propres. On dit que les primates ont hérité des attributs des mammifères.

# Notion de polymorphisme

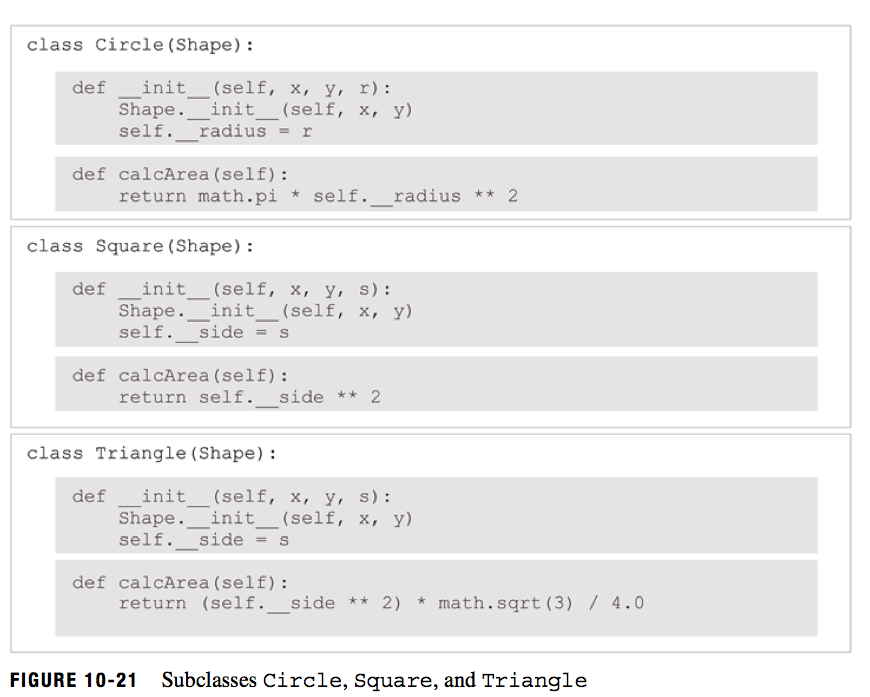
Le polymorphisme est une des forces de la programmation orientée objet. Cela lui permet de traiter de la même manière des objets de différents types, qui ne se comportent pas nécessairement de la même façon. Considérons par exemple le cas d'une classe générale de formes qui possède un emplacement, et trois classes représentant des formes géométriques spécifiques :



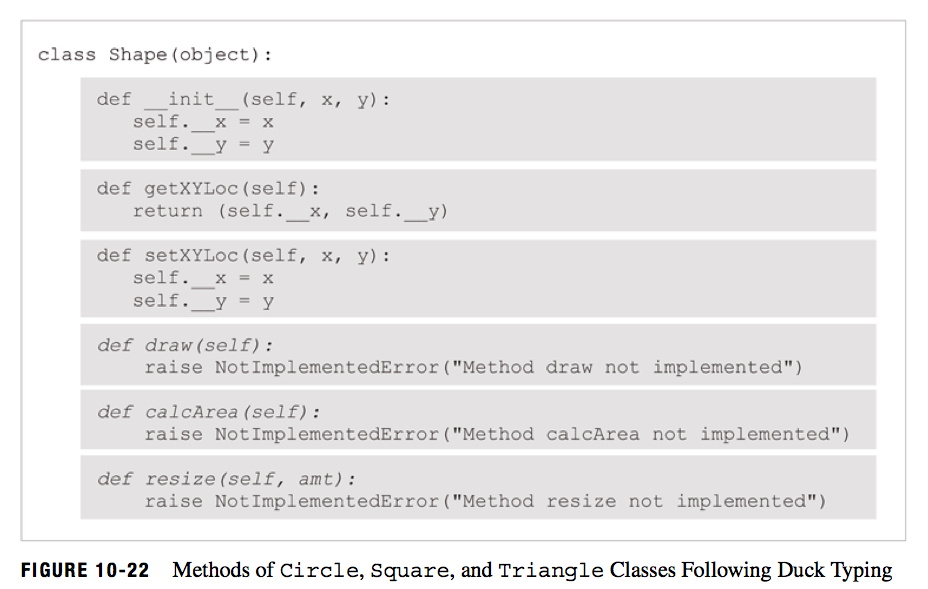
Chaque classe de forme spécifique peut posséder la même méthode calcArea afin de calculer sa surface. La fonctionnalité est la même, mais l'implantation est différente. La classe de base définit la méthode, mais en soulevant une exception indiquant justement que la méthode n'est pas implantée. En effet, la méthode ne peut être implantée dans le cas général, puisque le calcul de surface dépend de la forme choisie :



Ce genre de classe générique est qualifiée de classe abstraite, car on ne peut pas manipuler d'objet de cette classe, puisque des détails d'implantation manquent. Par contre, on peut dériver une classe abstraite afin de produire une multitude de classes spécifiques qui partagent les attributs de leur ancêtre, tout en possédant certains attributs qui leur sont propres. Dans le cas de notre exemple, chacune des classes possède un constructeur spécifique, ainsi qu'une méthode spécifique pour le calcul de sa surface :



Ces classes sont polymorphiques, car on peut manipuler leurs objets comme s'ils appartenaient à une seule classe générique. Ce qu'on fait pour le calcul de la surface, on peut tout aussi bien le faire pour le dessin de la forme dans une fenêtre graphique, par exemple une fenêtre turtle, ou encore pour le redimensionnement de la forme :



En anglais, on parle de duck typing (traduction litérale: « typage de canard ») dans le sens que si ça ressemble à un canard et que ça fait coin-coin comme un canard, alors c'est sans doute un canard. Le polymorphisme permet de traiter toutes les formes géométriques comme si elles étaient toutes semblables, alors qu'elles sont en fait particulières. L'alternative est de tester chacun des cas particuliers à chaque action que l'on veut prendre :



Plus le nombre de cas particuliers augmente, plus le polymorphisme devient avantageux.