Les interfaces en programmation objet

Introduction

Le chapitre sur l'héritage montre que l'on peut dériver les classes en les spécialisant de plus en plus. Ainsi, l'héritage offre une méthode simple pour l'ajout d'attributs ou de fonctionnalités et pour la spécialisation de méthodes grâce à leur réécriture. Les classes liées par l'héritage sont regroupées par type d'entité. Voila un exemple où l'on remarque bien le regroupement par type : la classe Aigle hérite de Rapace qui hérite d'Oiseau, qui hérite d'Animal.

Malgré la force du concept d'héritage, celui-ci ne peut couvrir tous les cas de ressemblance entre les classes. En effet, même si le lien entre Aigle et Oiseau est intuitif, il en est de même entre les classes Oiseau et Avion. Mais la, pas question d'utiliser le principe d'héritage, car un avion n'a rien à voir avec un oiseau. Leur ressemblance est entre leurs aptitudes : un avion sait voler, un oiseau aussi.

Afin de représenter ces liens de fonctionnalités, les langages de programmation objets offrent le concept des interfaces. Une interface est un ensemble de fonctionnalités (une ou plusieurs fonctionnalités) que les classes qui l'implémentent doivent posséder. Pour reprendre l'exemple de l'avion et de l'oiseau, les deux classes posséderont surement une fonctionnalité de vol. Afin de s'assurer que c'est le cas, les deux classes Avion et Oiseau vont implémenter l'interface Vol.

Où utiliser les interfaces ?

Les interfaces sont souvent utilisées pour des fonctionnalités standards, facilement identifiables et implémentées fréquemment. En voilà quelques exemples : les fonctionnalités de tri, de dessin, de collision, de sérialisation (voir chapitre suivant), … La plupart des langages de programmation objets offrent des interfaces déjà existantes que le programmeur peut implémenter afin de bénéficier de meilleurs communications avec d'autres classes existantes. Par exemple, une classe permettant le tri des informations d'une collection d'objets ne pourra fonctionner que si les éléments de la collection sont ordonnables. Le contrôle de la présence d'une interface de comparaison des éléments entre eux offre la certitude que les éléments possèdent bien cette fonctionnalité.

Conclusion

Les interfaces offrent un bon moyen de contrôler les similitudes de fonctionnalités entre les classes, palliant ainsi au problème de l'absence d'héritage multiple dans certain langage de programmation (comme Java par exemple) . Elle est souvent indissociable des fonctionnalités de tri et de sérialisation. Et rien que ces deux points méritent largement que les programmeurs connaissent ce sujet.