heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

comem+

Programmation orientée objet

Support de cours

Héritage - les concepts de bases

Nicolas Chabloz

Table des matières

2 Héritage **2**

2.1 Les concepts de base 2

2.2 La spécialisation 2

2.3 La surcharge 3

2.4 Résumé 4

Chapitre 2

Héritage

2.1 Les concepts de base

Il arrive souvent lors de l'étape de la modélisation d'une application (en particulier lors de la recherche des entités composant le problème lors d'une approche orientée objet) de trouver des similtudes en ce qui concerne les données ou les comportements de certaine classe. Bien que ces similitude ne soient pas toujours liées au concept d'héritage que nous allons voir, c'est trés souvent le cas lorsque les classe identifiées sont de même natures. On entend par même nature, des classes dont les comportements et/ou données sont presques identiques, ou sont en grand partie commune. Par exemple, si lors d'une modélisation on s'aperçoit que notre application devra s'occuper de la gestion des clients, des fournisseurs et des collaborateurs, on arrive vite a la conclusion que ces trois classes partageront beaucoup en ce qui concerne leurs attributs. En effet, ces trois classe représentant toute des êtres humains, chacune d'elle possédera au moins des attributs semblables tels q'un nom, un prénom, une date de naissance et peut être aussi des fonctionnalités similaires (par exemple la manière dont les instances de ces classes seront ordonnées).

Afin d'éviter une redondance des attributs, ainsi que des méthodes, il est de bonne pratique d'identifier une classe *mére* aux entités semblables, c'est à dire une classe plus générique dont les autres entitées seraient des spécialisations. Ainsi, dans l'exemple précédemment cité, la classe *mère* serait la classe Personne. Elle regrouperait les similtudes observées des trois classes *dérivées* (les attributs nom, prénom, date de naissance, ... ainsi que les fonctionalités de comparaison). Le diagramme de classe correspondant à l'exemple pourrait ressembler à la figure 2.1

2.2 La spécialisation

Les classes dérivée d'une classe mére sont en faites des spécialisations de celle-ci. En effet, les classes Client, Fournisseur et Collaborateur ont toutes des spécificités que n'a pas la classe Personne. Lors de l'héritage, les classes dérivées héritent des attributs et des méthodes de leur classe mére. Elles possédent donc au minimum tout ce que possède la mère, mais peuvent y rajouter d'autres attributs ou méthodes qui leur sont propres. Par contre on ne peut pas supprimer un attribut ou une méthode, soit on hérite de tout, soit il n'y a pas d'héritage, l'héritage partiel n'existant pas en Java. C'est heureusement rare d'avoir un cas où une classe dérivée a besoin de moins d'attributs que sa classe mère.

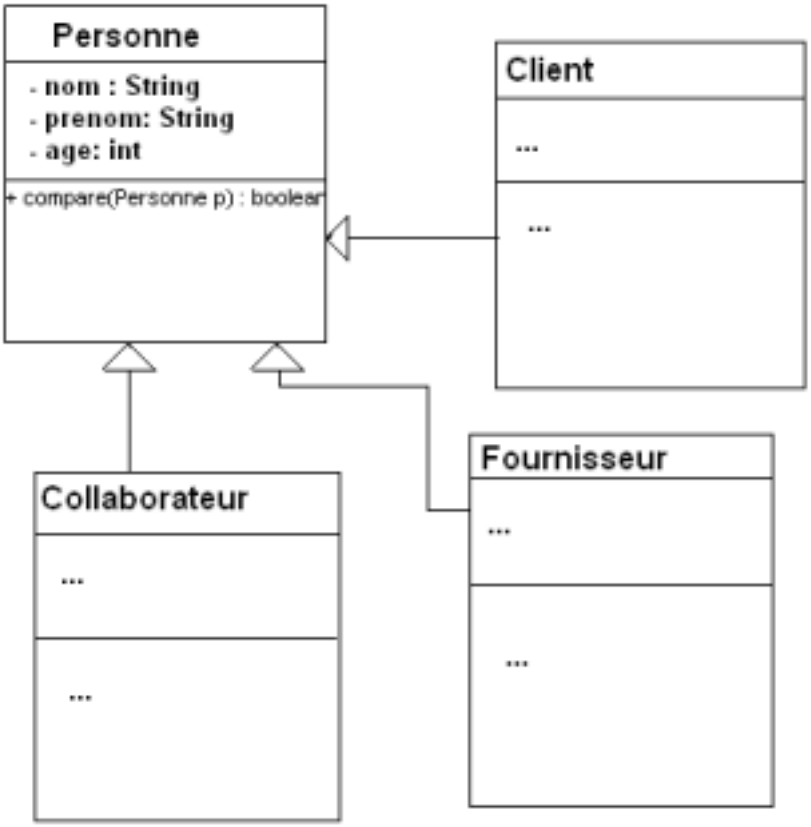

FIGURE 2.1 – Diagramme de classe

Une autre grande force de l'héritage et la mise à jour du code. En effet lorsque l'on opert des changements dans les attributs ou méthodes de la classe mère, ceux-ci sont directement hérités par toutes les classes dérivées.

Il est bien sur possible d'avoir des héritages en cascade, c'est à dire une classe qui hérite d'une autre qui hérite d'une autre et ainsi de suite. Dans notre exemple, on pourrait imaginer une classe CollaborateurTempsPartiel qui hériterait de la classe Collaborateur. Les instances de la classe CollaborateurTempsPartiel auraient alors tous les attributs et méthodes de Collaborateur (et de Personne puisque Collaborateur hérite de Personne).

2.3 La surcharge

Une autre possibilité offerte par l'héritage est *la surcharge* des méthodes (il n'y a pas de surcharge des attributs en Java). La surcharge (ou override en anglais) et le moyen de redéfinir le comportement d'une fonctionnalité existante dans la classe mère. Par exemple, imaginons que la classe Collaborateur ne soit pas ordonée par ordre alphabétique mais plutot par ordre de grandeur des salaire (un attribut spécialisé de Collaborateur inexistant dans la classe mère Personne). Il va donc falloir redéfinir le comportement de la méthode "compare". Pour ce faire, il suffit de ré-écrire la méthode dans la classe dérivée Collaborateur. Bien sûr pour que la surcharge fonctionne bien, il faut que la signature de méthode soit strictement la même que la signature de la méthode définie dans la classe mère.

Dans certain cas de spécialisation, la fonctionnalité reste la même mais est agrémentée d'une partie supplémentaire. Par exemple, la classe CollaborateurTempsPartiel pourrait avoir une méthode "toString" qui afficherait le taux d'activité du collaborateur en plus de tous les attributs déjà affichés par la méthode "toSrting" de sa classe mère. Pour éviter de devoir recoder toutes la fonctionalité dans la classe dérivée il est possible d'appeler une fonctionalité d'une classe mére depuis une classe dérivée. Ainsi, il suffit d'appeler la méthode "toString" de Collaborateur dans la méthode "toString" de CollaborateurTempsPartiel pour éviter de recoder la partie semblable de cette fonctionalité.

2.4 Résumé

Grâce à l'héritage, il est possible d'éviter la redondance entre les classes de même type, de facilité la mise à jour du code et de mieux le structurer. L'héritage permet l'ajout d'attributs ou de fonctionalités dans une classe dérivée, ainsi que la surcharge (totale ou partiel) de fonctionalités existantes. Par contre la suppression d'attribut ou de méthode est impossible. Attention aussi a ne pas voir partout de l'héritage lors de vos modélisation. Par exemple un avion et un oiseau savent tous les deux voler, mais il serait assez particulier de les faires hériter de la même classe (une classe nomée ObjetVolant ? !). Il est plus censé de faire hérité la Classe Avion d'une classe Véhicule et la classe Oiseau d'une classe Animal. Lorsque vous détéctez une simlitude en ce qui concerne les fonctionalités, mais une divergence des type d'entité, il ne faut pas user du principe d'héritage mais plutôt du principe d'interface que nous verrons dans un prochain chapitre.