Classes/Héritage

Définition

Chaque logiciel Java, qu'il s'agisse d'une applet ou d'une application, se compose toujours d'au moins une classe. Toutes les variables, méthodes et lignes d'instructions doivent toujours se trouver dans une classe. Les seules exceptions sont les instructions d'importation de classes, ainsi que celle qui définissent une classe comme appartenant à une bibliothèque de classes.
Une classe dont vous programmez le code source peut se représenter comme un plan de construction de l'objet. Elle se compose de données et de méthodes qui ont un quelconque rapport conceptuel.
Définition du Grand Livre Java 2 chez Micro Application.

On retient ici qu'une classe est un programme Java (ce sont quasiment des synonymes). Par exemple, si MSOffice était écrit en Java, on aurait une classe pour Word, une autre pour Excel et encore une autre pour Outlook, etc. MSOffice serait alors le *package* (paquetage en français) ou le rassemblement de plusieurs classes. Cette vision est naïve car NotePad à lui seul regrouperait plusieurs classes (on pourrait tout faire dans la même classe mais ce serait incompréhensible, imaginez vous retrouvez devant 300 pages de programme à la suite !), mais elle permet de comprendre le concept de classe.

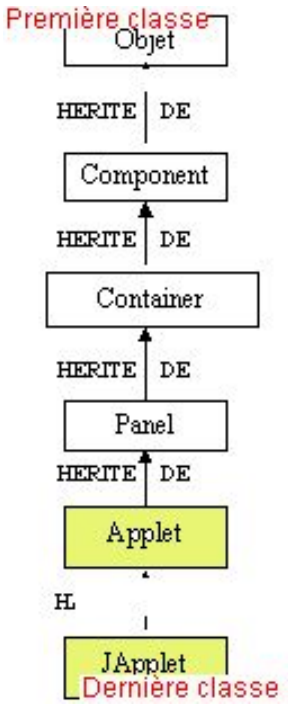
Héritage

Une classe est donc un programme qui définit des objets, variables, méthodes etc. Disons que dans Word 1, il existe une classe (composée de plusieurs méthodes) pour ouvrir un fichier. Dans sa version 2, M… veut améliorer la gestion de l'ouverture des fichiers, tout en gardant les méthodes –qui fonctionnaient parfaitement- de la version 1. La classe de la version 2 doit alors hériter des propriétés de la classe de la version 1. Ceci est comme un héritage génétique : une même base avec certaines choses qui changent.
Si vous avez déjà examiné le code source d'une applet, vous avez eu l'occasion de voir une commande d'héritage. On déclare la classe principale d'une applet ainsi :

- public class NomDeLaClasse extends java.applet.Applet {…}

Votre classe (ici, NomDeLaClasse) *étend* la classe Applet. Ceci signifie qu'elle hérite des propriétés de cette classe. Une de ces propriétés est par exemple la création d'une zone d'affichage graphique (dans votre navigateur). Sans ceci le programme n'es pas une applet. *java.applet* est le chemin d'accès (classpath) qui dit au programme d'aller chercher la classe Applet dans le package *java*, puis dans le répertoire *applet* de celui-ci. Dans la bibliothèque de classes de Java, les classes sont classées (!) dans des répertoires.
La classe NomDeLaClasse et une *sous-classe* de la classe Applet du package *java.applet* qui est elle-même sa *super-classe*.
En Java, une classe ne peut pas hériter de plusieurs classes. Java ne supporte donc pas l'héritage multiple. Java est donc moins flexible que d'autres langages, mais il évite ainsi nombre d'erreurs. Ceci est compensé en partie par les *interfaces*.
Note : sur le graphique JApplet remplace Applet car il s'agit de la classe de Java2 (non supporté par les navigateurs actuels).
Si vous souhaitez interdire l'héritage d'une de vos classes, vous devez la déclarer final.

- final class NonHeritable {…}



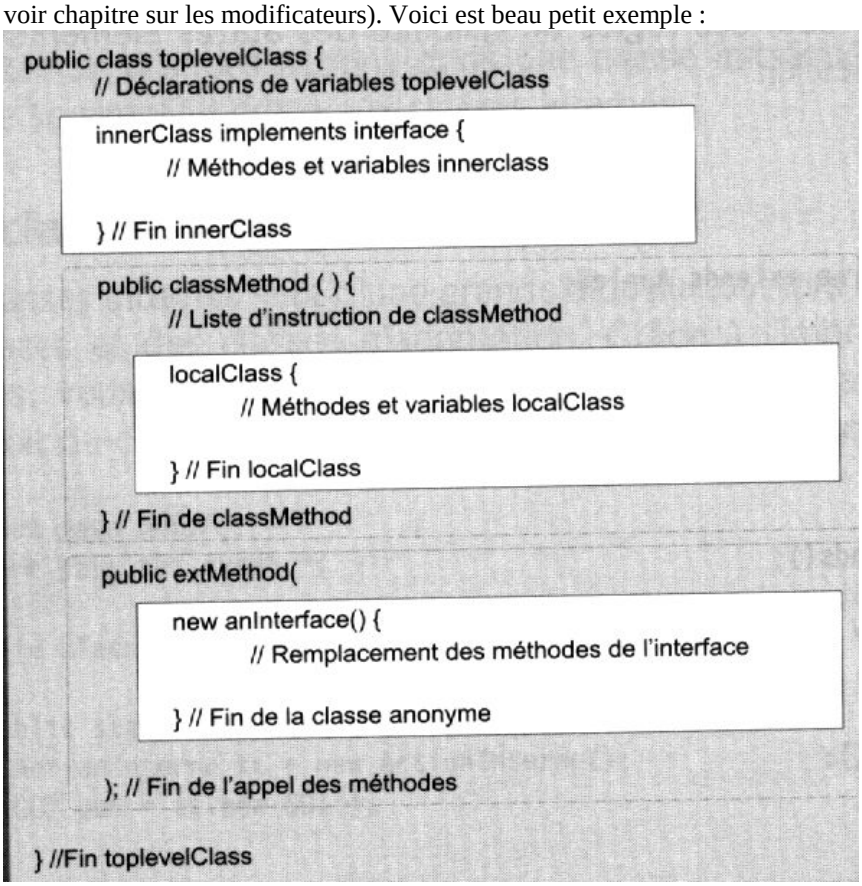
Si vous voulez un schéma d'héritage complet de Java 1.1, téléchargez le PDF (157Ko) à cette adresse :
http://perso.wanadoo.fr/guillaume/Ressources/api_java1.pdf

Création de classes

Les classes sont déclarées avec le mot clé class, le code est renfermé entre deux accolades. On utilise le mot clé extends pour l'héritage.

Classes imbriquées

On peut créer une classe dans une autre classe (saut si elle est déclarée statique avec le mot clé static, voir chapitre sur les modificateurs). Voici est beau petit exemple :



Comme une classe ne peut hériter que d'une seule autre classe, on peut créer des classes imbriquées qui vont résoudre le problème. Les classes imbriquées dans vos programmes devront être gérées avec la plus grande attention car, on se mélange facilement les pinceaux ! sans parler des problèmes de portée de variables…

Ce qui nous amène à aborder un nouveau concept : le *polymorphisme*. Si on appelle une méthode dans une classe, le programme regarde si cette méthode existe (nom + arguments), si elle existe, elle est exécutée. Si cette méthode n'existe pas, le programme la cherche dans la super-classe de la classe qui fait appel à elle. Et on monte ainsi les niveaux jusqu'à ce qu'une méthode adéquate soit trouvée ou que la « remontée » se termine (dans ce cas le compilateur génèrera une erreur). *Lorsqu'une méthode définie dans une sous-classe a le même nom ou les mêmes paramètres qu'une méthode d'une des classe ancêtres* (super-classe), *elle cache la méthode de la classe ancêtre à la sous-classe**. Ceci est très important : une méthode de « bas niveau » cache une méthode de « haut niveau » quand on redéfinit son comportement.

Classes abstraites

*A mesure que l'on remonte dans la hiérarchie des classes, celles-ci deviennent plus générales et souvent plus abstraites. A un certain moment, la classe ancêtre devient tellement générale qu'on la considère surtout comme un moule pour des classes dérivées et non plus comme une véritable classe dotée d'instances**.
Une classe abstraite définit donc une « ligne de conduite » pour ses méthodes. Elle vous met « sur la voie » pour définir plus précisément votre propre classe. Cette classe abstraite ne peut donc pas être instanciée (adaptée en objet). Elle peur contenir des méthodes abstraites mais ce n'est pas une obligation.
Les classes abstraites sont déclarées à l'aide du mot clé abstract.

- public abstract class NomDeLaClasse {…}

Il est surtout important de retenir qu'il est avantageux de placer le plus de fonctionnalités possible dans une super-classe, abstraite ou non.

Importation de classes

On ne va pas s'éterniser là dessus : vous connaissez la commande import qui permet au début du programme de rendre des classes accessibles. Par exemple pour, dans une applet, travailler avec des images, on importera *java.awt.image* avec la commande suivante :

- import java.awt.image.* ;

Les packages JDK les plus utiles

Package	**Description**
Java.applet	Contient la classe Applet et 3 interfaces. Elles servent d'intermédiaires entre une applet et la navigateur.
Java.awt	Interface utilisateur de issue de Java 1. Egalement chargement de graphiques, impression, distribution d'éléments graphiques…
Java.awt.datatransfer	Support du presse-papier (copier/coller…)
Java.awt.event	Gestionnaire d'évènements AWT
Java.awt.image	Gestion des images
Java.beans	Gestion des beans
Java.io	Gestion des entrées/sorties (écriture et lecture de données)
Java.lang	Contient la classe racine Object, types de données avancés…
Java.math	Fonctions mathématiques
Java.net	Client/serveur
Java.rmi	Modules RMI (Remote Method Invocation) pour invoquer des méthodes à distance
Java.security	Fonctions de sécurisation des transmissions
Java.sql	Permet l'accès aux bases de données relationnelles par l'intermédiaire du langage SQL
Java.text	Formatage des chaînes de texte
Java.util	Conversions, structures de données, gestion d'évènements
Javax.swing (le *x* est volontaire)	Tous les composants de l'interface utilisateur SWING de Java 2

issu de *Au cœur de Java 2* par Cay S. Horstmann & Gary Cornell aux éditions Sun (Campus Press).