Table des matières

1 Notations

- Pour créer un objet <o> de la classe <c> d'attributs $< x_1 >$,...,$< x_n >$, on écrit :

$$< o > \leftarrow < c > (x_1 =< arg_1 >, ..., x_n =< arg_n >)$$

- Pour accéder à la variable d'instance <v> de l'objet <o> : <o>.<v>
- Envoi du message m(arg_1,...,arg_n) à <o>, on note : <o>.m(arg_1,...,arg_n). Lors de cet envoi, les variables d'instance utilisées dans le corps de m réferent dynamiquement aux valeurs distinctes de <o>.

2 Java

2.1 Généralités

- Définir une classe : class <nom> (extends <nom_2>){<a_1>,...,<a_n>}
- Penser à faire des constructeurs (méthodes d'initialisation des objets, du même nom que sa classe)
- Création d'un objet : new <classe>(arg_1,...,arg_n)
- Compilation : javac <nom_fichier.java> : création d'un fichier nom_fichier.class
  java <nom_fichier> ; lance la méthode main()

2.2 Public - privé

- Une variable static a une valeur unique commune à toutes les instances.
- Une méthode static permet d'être appelée indépendemment des instances de sa classe. Par exemple : System.out.println()

- Classe String : permet de créer des chaînes de caractères constantes.
  Classe StringBuffer : créer des chaînes évolutives.
- Ces classes contiennent, entre autre, les méthodes "length()" et "charAt(int)"

- Spécifier un paquetage : Package <NomPaquetage>
- NomPaquetage peut être vu comme un préfixe précédent tous les symboles du fichier.
- Inclure les classes d'un paquetage : import. Exemple :
  import java.applet.Applet //importe la classe Applet
  import java.applet.* //importe toutes les classes

- Différents niveaux de protection : public, protected, private.
- "public classe <NomClasse> {...}" permet à la classe d'être importé d'un autre paquetage. Sinon, elle n'est visible que dans le même paquetage.
- protected : accessible uniquement dans la classe et ses sous classes et les autres clases du même paquetage
- private : uniquement dans sa propre classe.
- Si aucune déclaration, les membres ne sont accessibles qu'aux membres du paquetage.
- Si le membre appartient à l'interface avec les utilisateurs : public.

2.3 Héritage

- Permet la construction et l'extension des applications
- Permet la réutilisation des programmes plus facilement
- Dans le cas d'un héritage multiple, on prend la classe la plus spécifique (ie sous-classe) en premier.
- Existence d'une classe mère : Object

2.4 Polymorphisme

- Méthodes de même noms dans différentes classes. Servent pour des comportements analogues (affichier, décrire...), mais diffèrent selon les classes.
- L'ativation d'un message dépend donc du receveur et des liens d'héritage
- Les variables d'instance et les méthodes de la classe sont prioritaires à celles des surclasses.

2.5 Interfaces

- Méthodes abstraites : elles ne sont définies que dans les sous-classes
- Classe abstraite : qui a une méthode abstraite. Elle ne peut être instanciée.
- abstract class ClasseAbstraite{
  abstract void methodeAbstraite() ; }
- Interface : classe dont toutes les méthodes sont abstraites
- pour remédier à l'héritage simple : utilisation de implements, avec classe abstraite.
- class NomClasse implements ClasseAbstraite

2.6 Collections

- Collection : groupe d'objets, ordonnés ou non, avec plusieurs occurences ou non.
- 3 sous collections : List, Set, Map, qui héritent de Collection
- Tableau $\neq$ Collection : on fait la déclaration d'un tableau comme en C : int[10]

2.6.1 List

- Deux sous classes importantes : LinkedList et ArrayList
- LinkedList : implémentation des listes chaînées.
  - add(int index, Object o) : ajoute o à l'index spécifié
  - addFirst(Object o) : ajoute o en début de liste
  - addLast(Object o) : de même en fin de liste
  - getFirst() et getLast() : renvoie le premier ou le dernier élément de la liste. Penser au cast, car renvoie toujours un Object
  - removeFirst() et removeLast() : parle d'eux-même
  - contains(Object o) : renvoie un booléen, si présent dans la liste ou non
  - clear() : vide la liste
  - size() : renvoie la taille de la liste
- ArrayList : tableaux se redimensionnant automatiquement
  - add(int index, Object o), clear() et size() restent les mêmes
  - get(int index) : renvoie l'Object à l'index spécifié. Exception si dépassement.
- Pour éviter le cast à chaque fois, on utilise les Generics. A la création de la collection, on précise son type.
- ArrayList<Type> monArrayList = new ArrayList<Type>() ;