DUT Informatique S3 - Programmation Java

Jean-Claude MARTIN – MARTIN@LIMSI.fr

Cours 4 : Héritage et polymorphisme

Principe de l'héritage

- Besoins : éviter de dupliquer du code (attributs et méthodes) dans différentes classes qui partagent des caractéristiques communes
  - facilite les modifications futures => elles n'ont besoin d'être faites qu'à un seul endroit
  - représentation explicite d'une logique d'héritage des concepts du domaine (relation "est-un")

Syntaxe

- Le mot-clé "extends"

Exemple simple et héritage des attributs

- Pour la gestion d'une bibliothèque on nous demande d'écrire une application traitant des documents de nature diverse : des livres, des revues, des dictionnaires, etc. Les livres, à leur tour, peuvent être des romans ou des manuels.
- Tous les documents ont un numéro d'enregistrement (un entier) et un titre (une chaîne de caractères). Les livres ont, en plus, un auteur (une chaîne) et un nombre de pages (un entier). Les romans ont éventuellement un prix littéraire (un entier conventionnel, parmi : GONCOURT, MEDICIS, INTERALLIE, etc.), tandis que les manuels ont un niveau scolaire (un entier). Les revues ont un mois et une année (des entiers) et les dictionnaires ont une langue (une chaîne de caractères convenue, comme "anglais", "allemand", "espagnol", etc.).
- Tous les objets en question ici (livres, revues, dictionnaires, romans, etc.) doivent pouvoir être manipulées en tant que documents.
- => Schéma des classes
- => ATTENTION : Il faut vérifier la logique de votre schéma notamment qu'il est possible de dire par exemple *un "Roman" est-un "Livre"*
- => Déclaration en Java des classes
- => ATTENTION : ne pas redéclarer les attributs hérités dans les sous-classes !
- => Allocation des objets dans la mémoire : allocation automatique des attributs hérités

```
class Document {
      int no ;
      String titre ;
}

class Livre extends Document {
      String auteur ;
      int nbPages ;
}
```

```
class Roman extends Livre {
      int prixLitteraire ;
}

class Revue extends Document {
      int mois ;
      int annee ;
}
class Test {
            public static void main (String args[]) {
                  Document d = new Document ();
                  d.no = 1 ;
                  Livre l = new Livre ();
                  l.no = 2 ;
                  l.auteur = "Hugo";

                  d.auteur = "Balzac"; => ERREUR DE COMPIL!
            }
      }
```

Vocabulaire :

- classe-mère / super-classe : ……..
- sous-classe / classe dérivée : ……..
- La classe …… étend / hérite de la classe ……

Hiérarchie des classes du JDK

- parcourir l'arborescence des classes dans la javadoc : "tree"
- la classe mère de toutes les classes en Java est : ……;

Héritage des méthodes

- But : réutilisation des fonctionnalités de la classe héritée
- Exemple
  - Forme :
    - attributs : x, y, couleur
    - methode afficher ()
  - Cercle
    - attribut spécifique : …….
  - Main () : création d'une forme et d'un cercle, affichage des attributs

Redéfinition de méthodes dans les sous-classes : POLYMORPHISME

- But de l'héritage : réutilisation des fonctionnalités d'une classe tout en apportant certaines variations, spécifiques de l'objet
- Exemple :
  - Redéfinition de la méthode afficher() dans la classe Cercle
  - Main () : appel de afficher

```
class Forme {
      int x ;
```

```
      int y ;
      int couleur ;

            void afficher () {
                  System.out.println ("x = " + x + " , y =
" + y + ", couleur = " + couleur);
            }
}

class Cercle extends Forme {
      int rayon ;

            void afficher () {
                  System.out.println ("x = " + x + " , y =
" + y + ", couleur = " + couleur + ", rayon = " + rayon);
            }
}

class Test {
                  public static void main (String args[]) {
                        Forme f ;

                        f = new Cercle ();
                        f.x = 40 ; f.y = 15 ;
                        f.afficher ();
                  }
            }
```

- POLYMORPHISME des méthodes :
  - une même méthode (et ses paramètres) peut apparaitre dans différentes classes
  - ce sont différentes formes d'une même méthode
  - la méthode se comporte différement selon l'objet sur lequel elle est appelée
  - le nom et les types et ordre des paramètres doivent être identiques
  - le type retourné peut être différent !
- Résolution d'appel de méthodes

```
Forme f ;
…
f = new Cercle ();
f.afficher ();
```

  - Possibilités
    - Résolution statique : par le compilateur : la methode afficher de la classe Forme est appelée
    - Dynamique : par l'interpréteur : la méthode afficher de la classe Cercle est appelée
  - En java : …….

- C'est la version la plus spécifique de la méthode qui est appelée (la plus basse dans la hiérarchie en commençant à partir de la classe de l'objet)

Ajout d'une méthode agrandir() dans la classe Cercle

- Main () : appel de agrandir sur le cercle, sur la forme : …….

super

- Permet d'accéder explicitement (par exemple en cas d'ambiguitéà) à des attributs ou des méthodes de la classe mère
- Exemple : appel de la méthode afficher de Forme dans la méthode afficher de Cercle

Constructeur d'une classe héritée

- Exemple : class B extends A …
- Chaque sous-classe peut définir son constructeur
- Si le constructeur de B n'appelle pas le constructeur de A cela est fait automatiquement (appel du constructeur sans paramètre de la classe A)

- Appel d'un constructeur de la classe mère avec paramètres :

```
class A {
        A (int i, int j) {…}
…}

class B {
        B () {
                super (xx, yy)
```

Protection des données héritées

```
Class A {
        private x
        protected y
        …
Class B extends A
```

- Est-ce que B peut accéder à l'attribut x : …….
- Est-ce que B peut accéder à l'attribut y : …….
- Est-ce qu'une classe C n'héritant pas de A peut accéder à x : …….
- Est-ce qu'une classe C n'héritant pas de A peut accéder à y : …….
- Est-ce que ces accès peuvent être en lecture / modification : ……

méthodologie pour définir la hiérarchie des classes

- Déterminer les attributs et comportements communs à plusieurs classes
- Définir une classe représentant ces attributs et comportements communs
- Décider si une sous-classe a besoin de comportements spécifiques (redéfinition ou ajout de méthodes)
- ATTENTION : ne pas confondre "est-un" et "a-un"

opérateur instanceof

- Permet de déterminer la classe d'un objet

Tableau d'objet

- Application du polymorphisme à chaque case du tableau
  - Forme formes [] ;
  - t[i].afficher ()
- Si on veut appeler une méthode d'une sous-classe :
  - Tester avec instanceof
  - "caster" l'objet
  - Exemple : …….

Héritage multiple

- Dans certains langages, une classe peut hériter de plusieurs classes
- En Java : est-ce que l'héritage multiple est possible : ……
- Cf cours à venir sur les interfaces