Programmation Orientée Objet

Mohamed Tounsi

Institut Supérieur d'Informatique et de Multimédia Sfax

Novembre 2014

Héritage
En quoi consiste l'héritage ?

- Supposons qu'il existe déjà une classe qui définit un certain nombre de méthodes et qu'on aie besoin d'une classe identique mais pourvue de quelques méthodes supplémentaires.
  - Comment éviter de réécrire la classe de départ ?
  - Regrouper les classes en super-classes en factorisant et spécialisant;
  - La sous-classe hérite des attributs et des méthodes et peut en rajouter de nouveaux.

Le concept de l'héritage spécifie une relation de spécialisation/généralisation.

Héritage
Définitions (1)

L'héritage est un mécanisme permettant le partage et la réutilisation de propriétés entre les objets.

- Lorsque une classe A hérite d'une classe B, alors:
  - A possède toutes les caractéristiques de B et aussi d'autres caractéristiques qui sont spécifiques à A;
  - A est une spécialisation de B (un cas particulier);
  - B est une généralisation de A (cas général);
  - A est appelée classe dérivée (fille);
  - B est appelée classe de base (mère ou super-classe);
  - Tout objet instancié de A est considérée aussi comme un objet de type B;
  - Un objet instancié de B n'est pas forcément un objet de type A.

Héritage
Définitions (2)

- Une classe peut hériter de plusieurs classes: héritage multiple. Exemple: classe Loutre est, à la fois, un mammifère Marin et un mammifère Terrien.
- Une classe de base peut être héritée par plusieurs classes;
- L'héritage minimise l'écriture du code (par réutilisation) et favorise l'extension;
- L'héritage supprime, en grande partie, les redondances dans le code;
- Une classe ne peut hériter d'elle même, évidemment;
- Une classe ne peut hériter d'une classe qui elle même hérite déjà d'elle (car là on se mord la queue).

Héritage

Implémentation de l'héritage en Java

- En Java on utilise le mot clé **extends** qui permet de spécifier la classe dont on hérite.

```
[modificateur acces] class_dérivée extends class_base
```

```
Class Point {
private float abscisse;
private float ordonnee;
public void deplacer (float x, float y)
{abscisse+=x; ordonnee+=y;}
public void afficher()
{System.out.println("abscisse="+abscisse+
"ordonnee="+ordonnee);}}

class Point_coloree extends Point {
private String couleur;
pulic void colorer (String c)
{couleur = c;} }
```

Héritage

Définitions du constructeur de la classe dérivée

- Le constructeur de la classe de base se charge d'initialiser les attributs hérités, il est désigné par super().
- Le mot clé super définit l'instance de l'objet de la classe dont on hérite directement. (super c'est la classe mère)
- Le constructeur de la classe de base est identifié selon les paramètres du super();
- On peut aussi utiliser super pour accéder aux membres définis dans la classe dont on hérite;

Héritage

Définitions du constructeur de la classe dérivée

L'utilisation de super pour invoquer un constructeur de la classe mère doit se faire sous certaines règles et contraintes très importantes:

- On ne peut pas appeler le constructeur de la classe mère en dehors d'un constructeur.
- On ne peut appeler qu'un seul constructeur de la classe mère, et obligatoirement en première instruction du constructeur de la classe fille (afin de respecter la règle d'ordre d'exécution des constructeurs: du plus ancien vers le plus jeune)
- Par défaut, s'il n'y a pas d'appel à un constructeur de la classe mère, tout constructeur appellera obligatoirement le constructeur par défaut de la classe mère, celui qui n'a pas d'argument: super();

Héritage

Définitions du constructeur de la classe dérivée

```
class Point
{ ...
public Point (float x, float y)
{abscisse=x; ordonnee=y;}
... }

class Point Point_colore extends Point
{...
public Point_coloree (float x, float y, String c)
{ super(x,y);
couleur = c;
} ... }
```

Une classe n'hérite pas des constructeurs de la classe qu'elle étend. Il faut nécessairement toujours définir dans une classe tous les constructeurs dont elle a besoin, même si ces derniers ne font que des appels aux constructeurs du mère par super(arguments).

Héritage

Accès aux membres hérités (au sein de la classe dérivée)

- Si les membres (attributs et méthodes) hérités sont déclarés avec le modificateur **private** dans la classe de base, la classe dérivée n'a pas le droit de les manipuler directement malgré qu'ils font parti de sa description.
- Le modificateur **protected**, s'il est utilisé dans la classe de base, autorise la classe dérivée à accéder directement à ses membres hérités (accès réservés à toute les classe dérivées à tous les niveaux, mais aussi aux classes du même package)
- un membre hérité est désigné par super.membre (si l'accès est autorisé)
- En dehors de la définition de la classe dérivée, tous les membres de cette classe sont vus comme des membres non hérités.

Héritage

Accès aux membres hérités (au sein de la classe dérivée)

```
class Point
{ protected float abscisse;
  private float ordonee;
  ...
  public float getordonnee()
  {return ordonee;}
  ...
 }

 class Point_coloree extends Point
 {...
 public String toString()
 {return ("abscisse="+super.abscisse+
 "ordonnee="+super.getordonnee()+"couleur="+couleur);
 }
 ...
 }
```

Héritage
Redéfinition des méthodes héritées

- Une méthode héritée peut être redéfini si sa version initiale n'est pas satisfaisante pour la classe dérivée.
- La redéfinition consiste à conserver l'entête de la méthode et à proposer un code différent.
- Lors de la redéfinition d'une méthode, l'appel de l'ancienne version (celle de la classe de base) est possible par super.nom_méthode(), et ce dans l'endroit du nouveau code que le programmeur juge adéquat.
- Si une méthode héritée est redéfinie, c'est uniquement la nouvelle version qui fait parti de la description de la classe dérivée.

Héritage
Redéfinition des méthodes héritées

```
class Point_coloree extends Point
{ ...
public void afficher ()
{super.afficher();
System.out.println("couleur="+couleur);
}
//redéfinition de la méthode de objet
public String to_String()
{return ("abscisse="+super.abscisse+
"ordonnee="+super.getordonnee()+"couleur="+couleur);
}
}
```

Héritage
Classe et méthode finales

- Java permet d'interdire l'héritage d'une classe en la déclarant avec le modificateur final.
- java permet d'interdire la redéfinition d'une méthode (si elle est héritée) en la déclarant avec le modificateur final.

Héritage

nouveau spectre pour les références d'objets

- Une référence d'une classe de base peut désigner un objet d'une classe dérivée.
- L'opérateur de casting peut être utilisée pour convertir une référence déclarée de type classe de base en une référence de type classe dérivée.

```
public class Point_coloree
{
public static void main (String [] args)
{Point_coloree pc; point p1, p2;
p1=new Point_coloree (1,2, "Bleu");
pc=(Point_coloree) p1;
pc.afficher(); // p1.afficher(); même résultat
p2=pc// p1,p2,pc désignent le même objet
}
}
```

polymorphisme
Les méthodes polymorphes

- Une méthode polymorphe est une méthode déclarée dans une super_classe et redéfinie par une sous_classe;
- Une méthode polymorphe permet de prévoir des opérations similaires sur des objets de natures différentes;
- Les méthodes final ne peuvent être redéfinies et ne sont donc pas polymorphes

polymorphisme
Définition du polymorphisme

- Il s'agit d'invoquer (appeler) des méthodes sans connaître la nature de l'objet correspondant.
- L'appel se fait à partir d'une référence du même type que l'objet correspondant ou de type de sa classe de base (on peut remonter à tous les niveaux de ses classes de bases).
- Un mécanisme dynamique permet de déterminer au moment de l'exécution, laquelle des méthodes à invoquer selon la nature de l'objet référencé (gestion d'un pointeur sur la description de la classe de l'objet).
- Le polymorphisme favorise la propriété d'extension des applications.

polymorphisme
Définition du polymorphisme

```
class Personne
{ ...
public void qui_etes_vous ()
{ System.out.println ("personne");}
..  }

 class Etudiant extends Personne
{ ...
public void qui_etes_vous ()
{ System.out.println ("étudiant");}
.. }

   class Employe extends Personne
{ ...
public void qui_etes_vous ()
{ System.out.println ("employé");}
.. }
```

polymorphisme
Définition du polymorphisme

```
public class population
{public static void main (String [] args)
{ personne pop[] = new personne [3];
pop[0]=new Etudiant();
pop[1]=new Personne();
pop[2]=new Employe();
for (int i=0; i<3; i++)
pop[i].qui_etes_vous();//appel polymorphe
}
}
```