Année 2008-2009

esil

Interfaces et classes abstraites

Nicolas Baudru
mél : nicolas.baudru@esil.univmed.fr
page web : nicolas.baudru.perso.esil.univmed.fr

Revenons quelques instants sur la hiérarchie des animaux.

- Nous avons réduit au minimum la duplication de code, en regroupant les comportements communs aux animaux dans une superclasse Animal.
- Nous avons redéfini les méthodes dans une sous-classe quand leur implémentation devait être plus spécifique.
- Nous avons assuré le maximum de souplesse grâce au polymorphisme, en utilisant des arguments et des types de retour de type Animal.

Nous savons que nous pouvons écrire :

```
Loup unLoup = new Loup();
```

Ainsi que :

```
Animal unLoup = new Loup();
```

Mais que peut bien signifier l'instruction suivante :

```
Animal a = new Animal();
```

```
Animal a = new Animal();
```

Quelles sont les valeurs des variables d'instance ?

Nous sommes face au problème suivant : la classe Animal est indispensable pour les besoins de l'héritage, mais cela n'a pas de sens de créer des objets Animal. Il faut donc empêcher l'instanciation de la classe Animal.

Pour cela il suffit de déclarer la classe Animal abstract.

```
abstract public class Animal {
   ...
}
```

Dès lors le compilateur interdira de créer une instance de ce type. Il reste cependant possible d'utiliser des classes abstraites comme superclasse ou avec le polymorphisme.

```
abstract public class Canin extends Animal {
   ...
}

public class TestCanin {
   public void creerCanin() {
      Canin c;    // possible
      c = new Chien();    // posible
      c = new Canin();    // impossible :
       // la classe Canin ne peut être instanciée
      }
   }
```

Une classe abstraite ne sert à rien à moins d'être étendue !
Une classe qui n'est pas abstraite se nomme une classe concrète.

Méthodes abstraites

Il est aussi possible de définir des méthodes abstraites. Comme pour les classes abstraites, une méthode abstraite doit obligatoirement être redéfinie dans les sous-classes plus spécifiques.

Deux points importants :

- Une méthode abstraite n'a pas de corps !
- Une méthode abstraite est toujours contenue dans une classe abstraite.

Une classe abstraite peut contenir à la fois des méthodes abstraites et concrètes.

```
abstract public class Canin extends Animal {

  public abstract void manger() ;  // pas de corps
}
```

Quel est l'interêt d'une méthode abstraite ?

Implémenter les méthodes abstraites

Implémenter une méthode abstraite revient à redéfinir cette méthode.

- Les méthodes abstraites n'ont pas de corps. Elles ne servent qu'à mettre en oeuvre le polymorphisme.
- Une méthode abstraite peut être implémentée dans une sous-classe abstraite.
- La première classe concrète dans votre hiérarchie d'héritage doit implémenter toutes les méthodes abstraites qui ne l'ont pas encore étées.

Exemple : Soit Animal une classe abstraite contenant deux méthodes abstraites manger() et vagabonder(). Soit Canin une classe abstraite étendant (i.e. héritant de) Animal et Chien une classe concrète.

Alors Canin peut très bien implémenter vagabonder(). Et il reste donc à implémenter dans Chien la méthode manger(). Ainsi, dans Chien, toutes les méthodes abstraites ont été implémentées.

Revenons au polymorphisme

Nous voulons pouvoir stocker des chiens et des chats dans un tableau :

```
public class MesAnimaux {
  private Animal[] animaux = new Animal[5];
  private int indiceSuivant = 0;

  public void add(Animal a) {
    if ( indiceSuivant < animaux.length ) {
      animaux[indiceSuivant] = a;
      indiceSuivant++;
    }
  }
}

public TestMesAnimaux {
  public static void main ( String[] args ) {
    MesAnimaux list = new MesAnimaux();
    Chien c = new Chien();
    Chat d = new Chat();
    list.add(c);
    list.add(d);
  }
```

La classe Object est la mère de toutes les classes, i.e. la superclasse universelle. Toutes les classes Java étendent la classe Object.

En fait toute classe qui n'étend pas explicitement une autre classe étend implicitement la classe Object. Donc les deux définitions de classes suivantes sont équivalentes :

```
public class Animal { ... }

public class Animal extends Object { ... }
```

La classe Object contient par exemple les méthodes suivantes (qui sont donc héritées par toutes les autres classes) :

- boolean equals(Object o) : teste si deux objets sont égaux.
- Class getClass() : retourne la classe à partir de laquelle l'objet a été instancié.

La classe Object est-elle abstraite ?

Peut-on redéfinir les méthodes de Object ?

Quelle est l'utilité de la classe Object ?

Pourquoi ne pas utiliser des types Object partout ?

Problème lié au polymorphisme

```
public class MesAnimaux {
  private Animal[] animaux = new Animal[5];
  private int indiceSuivant = 0;

  public void add(Animal a) { ... }
  public Animal get(int i) { return animaux[i]; }
}

public TestMesAnimaux {
  public static void main ( String[] args ) {
    MesAnimaux list = new MesAnimaux();
    Chien c = new Chien();
    list.add(c);
    c = list.get(0); // ne fonctionne pas
  }
}
```

Pourquoi cela ne fonctionne pas ?

Problème lié au polymorphisme

```
Animal a;
Chien c = new Chien();
list.add(c);
c = list.get(0); // ne fonctionne pas
a = list.get(0); // fonctionne
a.aboyer(); // ne marche pas.
a.manger(); // fonctionne
```

Le problème est le suivant : les objets entrent en tant que Chien ou Chat mais sortent en tant qu'Animal. Leur spécificité a été perdue. En fait le compilateur ne peut pas deviner le type de l'objet de sortie. Il considère donc qu'il est du type le plus générique possible.

C'est donc à vous (programmeur) de préciser le type de l'objet. Si vous n'en êtes pas sûr, vous pouvez utiliser l'opérateur instanceOf :

```
if ( a instanceOf Chien ) {
  c = (Chien) a; // on précise que a est un chien
  c.aboyer();
}
```

Des animaux sauvages ou domestiques ...

... au besoin d'un héritage multiple ...

... qui heureusement n'existe pas en Java

Java fournit une solution : les interfaces. Une interface permet d'exploiter les avantages du polymorphisme tout en évitant le problème du "losange de la mort".

Une interface Java est comme une classe 100% abstraite. Comme toutes les méthodes sont abstraites, toute classe qui implémente une interface doit redéfinir toutes les méthodes de cette interface.

Pour définir une interface :

```
public interface Compagnon {
  void etreAmical(); // les méthodes d'une interface sont
  void jouer();      // implicitement public et abstract
}

public class Chien extends Canin implements  Compagnon {
  public void etreAmical() { ... }
  public void jouer() { ... }
  public void vagabonder() { ... }
  public void manger() { ... }
}
```

Avantage des interfaces

Quand vous utilisez une classe comme type polymorphe, les objets auxquels vous attribuez ce type doivent provenir de la même hiérarchie d'héritage. Mais pas de n'importe où : ces objets doivent être des sous-types du type polymorphe.

Avec une interface comme type polymorphe, les objets peuvent venir de n'importe quel point de la hiérarchie d'héritage (et même de hiérarchie différente), la seule condition étant qu'ils appartiennent à une classe qui implémente l'interface.

De plus une classe peut implémenter plusieurs interfaces :

```
public class Chien extends Canin implements
Compagnon, AnimalDeCirque {
   ...
}
```

Rappel : Si la classe est concrète, alors toutes les méthodes abstraites de toutes les superclasses abstraites et de toutes les interfaces doivent avoir été redéfinies.

Classe abstraite, sous-classe et interface : faire le bon choix

- Créer une classe qui n'étend rien quand elle ne réussit pas le test du EST-UN pour tout autre type.
- Créer une sous-classe uniquement si vous voulez obtenir une version plus spécifique d'une classe, redéfinir ou ajouter des comportements.
- Utiliser une classe abstraite quand vous voulez définir un patron pour un groupe de sous-classes et qu'une partie du code est utilisée par toutes les sous-classes.
- Utiliser une interface pour définir un rôle que d'autres classes puissent jouer, indépendamment de leur place dans la structure d'héritage.

Invoquer une méthode de la superclasse

Il suffit d'utiliser le mot clé super :

```
public class Docteur {
  public void examiner() {
    // fait ce que fait habituellement tout bon docteur
  }
}

public class Orthopediste {
  public void examiner() {
    super.examiner();     // méthode du Docteur
    // + examine les articulations ...
  }
}
```

- Quand vous ne voulez pas qu'une classe soit instanciée, utilisez le mot clé abstract.
- Une classe abstraite peut avoir des méthodes abstraites ou non abstraites.
- Une classe qui contient au moins une méthode abstraite doit être abstraite.
- Une méthode abstraite n'a pas de corps. Elle se termine par un " ;".
- Toute méthode abstraite doit être implémentée par la première classe concrète dans la hiérarchie d'héritage.

Ce que nous avons vu ...

- Toute sous-classe Java est une sous-classe directe ou indirecte de la classe Object.
- Comment utiliser le polymorphisme avec les arguments et les types de retour (nécessité du transtypage).
- Java n'autorise pas l'héritage multiple.
- Une interface est comme une classe 100% abstraite. Une classe peut implémenter plusieurs interfaces (mot clé implements). Toutes les méthodes d'une interface doivent être redéfinies.