IFT1025 été 2010

Miklós Csűrös

28 mai 2010

3 Interfaces

3.1 Déclarations et implantations

À l'addition d'être une sous-classe, une classe peut implanter une (ou même plus qu'une) interface.

Une **interface** est une collection de déclaration de méthodes d'objet abstraites et de constantes (variables statiques finales). Du côté syntaxe, on écrit l'interface comme un fichier de classe, en utilisant le mot-clé `interface` au lieu de `class`.

```
public interface UneInterface
{
    /**
     * Déclaration d'une méthode qui
     * doit être implantée.
     * @param x un argument
     * @return une valeur
     */
    public abstract int methodeAImplanter(int x);

    /**
     * Déclaration d'une constante.
     */
    public static final int UNE_CONSTANTE = 2010;
}
```

L'interface ou les interfaces implantées par une classe sont énumerées après le mot-clé `implements`.

```
public class ArrayList
    extends AbstractList
    implements List, RandomAccess, Cloneable, Serializable
{
    ...
}
```

La classe doit implanter chaque méthode déclarée dans l'interface. Dans d'autre mots, l'implantation doit fournir soit une méthode abstraite (implantation par classe abstraite), soit une méthode complètement définie avec la même signature. L'implantation peut utiliser les constantes déclarées comme toutes autres variables statiques.

```
public class UneClasse implements UneInterface
{
    /**
     * Définition d'une méthode de l'interface
     */
   @Override // mot-clé optionnel en Java 5
   public int methodeAImplanter(int x)
   {
       if (x<UNE_CONSTANTE) return UNE_CONSTANTE;
       else return x;
   }
}
```

Règles syntaxiques

- ⋆ une interface est toujours publique (même si le mot-clé `public` est omis) ; un modificateur `private` ou `protected` cause une erreur de compilation
- ⋆ les méthodes et variables déclarées sont toujours publiques (même si le mot-clé `public` est omis) ; un modificateur `private` ou `protected` cause une erreur de compilation
- ⋆ les méthodes sont toujours abstraites (même si le mot-clé `abstract` est omis) ; la définition d'une méthode cause une erreur de compilation
- ⋆ les variables sont toujours statiques et finales (même si `static` ou `final` sont omis)
- ⋆ il est interdit de déclarer une variable sans initialisation
- ⋆ il est interdit de déclarer une méthode statique
- ⋆ il est interdit de déclarer un constructeur

3.2 Polymorphisme

Une interface représente un contrat entre le client et l'implantation. L'interface définit un type qui peut être utilisé comme une classe.

```
    ...
    UneClasse objet = new UneClasse();
    UneInterface I = objet; // conversion automatique
    int c = I.methodeAimplanter(5); // liaison dynamique pour la méthode d'instance
```

Cette mécanisme permet une grande flexibilité dans la production du code générique, comme la méthode `sort` de la classe `java.lang.Arrays` qui trie un tableau selon «l'ordre naturel» des éléments, où l'ordre est défini à l'aide de l'interface `java.lang.Comparable`.

```
import java.util.Arrays; // pour le tri par sort()
class Ecureuil extends Rongeur implements Comparable<Ecureuil>
{
    /**
     * La seule méthode déclarée dans l'interface Comparable
     */
    public int compareTo(Ecureuil autre)
    {
        return this.getSize()-autre.getSize();
    }

    public static void trier(Ecureuil[] colonie)
    {
        Arrays.sort(colonie); // tri utilise la méthode compareTo
    }
}
```

Si l'argument de `Arrays.sort` contient des éléments non-comparables, une exception `ClassCastException` est jetée.

3.3 Héritage et interfaces

Une interface peut être la sous-interface d'une autre ou de plusieurs autres pour déclarer des méthodes et constantes additionnelles. Les déclarations des superinterfaces sont héritées dans la sous-interface.

```
interface List extends Collection, Iterable // "public" implié
{ ...
```

Dans le cas de héritage multiple, les constantes doivent être nommées sans ambiguïté (donc utiliser le contexte `UneInterface.UNE_CONSTANTE` dans le cas d'un nom identiques). Les méthodes avec signatures équivalentes doivent avoir le même type de valeur retournée.