Séquence 14

Conversion de type

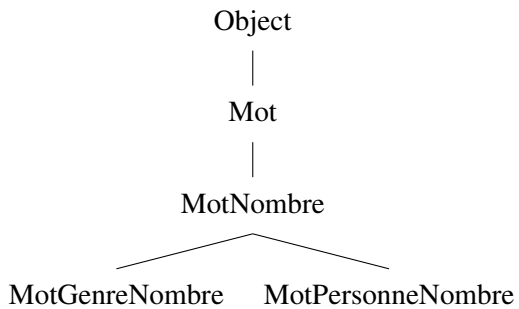
Un objet a plusieurs types qui appartiennent à une branche de l'arbre d'héritage qui part de la racine (type `Object`) et qui va jusqu'à la classe instanciée à la création de l'objet. L'objet peut être stocké dans une variable de n'importe lequel des types de cette branche.

Nous allons voir dans cette séquence de cours comment on peut faire varier le type avec lequel un objet est regardé au cours d'un programme.

1 Changer pour un ascendant

À un moment donné, l'objet est vu avec un certain type T. Si l'on veut changer pour utiliser l'objet avec un type qui est un ascendant de T, il n'y a rien de spécial à faire : il suffit d'affecter l'objet à une variable ou un paramètre du type ascendant et cela ne peut pas provoquer d'erreur car tout objet qui appartient à un type appartient également à tous les ascendant de ce type.

Reprenons l'exemple des mots déjà utilisé pour les questions de typage. Rappelons les relations d'héritage entre classes.



Supposons qu'à un moment donné, un objet soit stocké dans une variable de type `MotNombre`. Cet objet peut être affecté sans problème à une variable d'un type ascendant (`Object` ou `Mot`), ou passé en paramètre à une méthode dont le paramètre a pour type un de ces ascendant, ou encore à un tableau ou un `ArrayList` d'un de ces ascendants.

```
import java.util.ArrayList;
public class ConvertUpper{
    public static void printMot(Mot unmot){
        System.out.println(unmot);
    }
```

```
    public static void main(String[] args){
        MotNombre variable = new MotGenreNombre("blanc","adjectif",
                                                MotGenreNombre.MASCULIN,
                                                MotNombre.SINGULIER);
        Mot varMot;
        Mot[] tabMot = new Mot[5];
        ArrayList<Mot> arli = new ArrayList<Mot>();
        varMot = variable;
        tabMot[0] = variable;
        arli.add(variable);
        printMot(variable);
        System.out.println(varMot);
        System.out.println(tabMot[0]);
        System.out.println(arli.get(0));
    }
}
```

Si un objet appartient à un type, on est sûr qu'il appartient aussi à tous les ascendants de ce type.

2 Changer pour un descendant

Dans l'autre sens, si l'on veut changer pour passer à un descendant du type actuel dans l'arbre d'héritage, on ne peut pas garantir que ce type appartient aux types de l'objet. C'est possible mais pas certain.

Par exemple, si l'on a une variable de type `MotNombre`, cette variable peut contenir un objet d'un des trois types `MotNombre, MotGenreNombre` ou `MotPersonneNombre`. Si l'on demande à convertir le contenu de cette variable en un descendant de la classe `MotNombre`, par exemple, `MotPersonneNombre`, dans deux des trois cas, ce ne sera pas possible parce que cette classe ne fait pas partie des types de l'objet.

Dans ces cas où le changement de type n'est pas toujours possible mais qu'il l'est parfois, il faut faire une demande explicite de conversion de type dans le code. Cela s'écrit en mettant devant la valeur à convertir le nom du type entre parenthèses.

```
1  import java.util.ArrayList;
2  public class ConvertLower{
3      public static void printMot(MotGenreNombre unmot){
4          System.out.println(unmot);
5      }
6      public static void main(String[] args){
7          MotNombre variable = new MotGenreNombre("blanc","adjectif",
8                                                  MotGenreNombre.MASCULIN,
9                                                  MotNombre.SINGULIER);
10         MotGenreNombre varMot;
11         MotGenreNombre[] tabMot = new MotGenreNombre[5];
12         ArrayList<MotGenreNombre> arli = new ArrayList<MotGenreNombre>();
13         varMot = (MotGenreNombre) variable;
14         tabMot[0] = (MotGenreNombre) variable;
15         arli.add((MotGenreNombre) variable);
16         printMot((MotGenreNombre) variable);
17         System.out.println(varMot);
18         System.out.println(tabMot[0]);
```

```
19          System.out.println(arli.get(0));
20      }
21  }
```

Il y a des conversions de type aux lignes 13, 14, 15, 16. Sans ces conversion de types il y aurait des erreurs de compilation à ces quatre lignes.

```
incompatible types: MotNombre cannot be converted to MotGenreNombre
```

La conversion de type explicite s'appelle *cast* en anglais. Elle est toujours permise pour passer d'un type à un de ses descendants.

3 Que fait réellement la conversion de type

La conversion de type ne change rien à l'objet qu'elle affecte. Dans la mémoire de l'ordinateur, dans le tas, l'objet reste le même. Les types de l'objet sont les mêmes et la conversion permet de passer d'un type de l'objet à un autre type que l'objet avait déjà et qui est plus bas dans l'arbre d'héritage.

En fait, cette conversion a deux effets concrets :

— Elle permet de compiler le programme qui sans la conversion aurait une erreur de typage à la compilation.

— Elle insère dans le code qui sera exécuté une vérification de type à l'exécution. Cette vérification s'assure que le type demandé appartient aux types de l'objet considéré. Si ce n'est pas le cas une exception est lancée.

Voyons un exemple.

```
1   import java.util.ArrayList;
2   public class ConvertRuntime{
3       public static void main(String[] args){
4           MotNombre variable = new MotGenreNombre("blanc","adjectif",
5                                                   MotGenreNombre.MASCULIN,
6                                                   MotNombre.SINGULIER);
7           MotGenreNombre varGenre;
8           MotPersonneNombre varPers;
9           varGenre = (MotGenreNombre) variable;
10          System.out.println("Première␣conversion␣ok");
11          varPers = (MotPersonneNombre) variable;
12          System.out.println("Seconde␣conversion␣ok");
13      }
14  }
```

Ce programme comporte deux conversions explicites, lignes 9 et 11. Il y a un seul objet créé qui est instance de `MotGenreNombre` (instanciation lignes 4 à 6). Ses types sont `Object, Mot, MotNombre` et `MotGenreNombre`. Il est donc possible de l'affecter à la variable de type `MotNombre`. La conversion de la ligne 9 est possible parce que le type demandé `MotGenreNombre` est un des types de l'objet. La conversion de la ligne 11 ne peut pas réussir parce que `MotPersonneNombre` ne fait pas partie des types de l'objet.

Néanmoins, la conversion de type est permise par le compilateur parce que `MotPersonneNombre` est un descendant du type de la variable `MotNombre`. Le programme se compile sans erreur.

À l'exécution, la ligne 11 produit une exception.

```
> java ConvertRuntime
Première conversion ok
Exception in thread "main" java.lang.ClassCastException:
class MotGenreNombre cannot be cast to class MotPersonneNombre
        at ConvertRuntime.main(ConvertRuntime.java:11)
```

L'exception lancée dans ce cas est `ClassCastException`.

4 Les types de l'objet sont inconnus à la compilation

Dans le programme `ConvertRuntime`, on peut déterminer facilement le type d'instance de l'objet que l'on convertit. Mais ce n'est généralement pas le cas. Dans la plupart des cas, on ne sait pas avant l'exécution quel est le type d'instance d'un objet manipulé par le programme. Voyons cela sur un petit exemple un peu artificiel.

```
import java.util.ArrayList;
public class ConvertTypeInconnu{
    public static void printMot(Mot unmot){
        System.out.println(unmot);
    }
    public static void main(String[] args){
        MotNombre variable;
        double ausort;
        for (int i=0; i<3; i++){
            ausort = Math.random();
            if (ausort<0.33333){
                variable = new MotGenreNombre("blanc","adjectif",
                                              MotGenreNombre.MASCULIN,
                                              MotNombre.SINGULIER);
            }else if (ausort < 0.6666){
                variable = new MotPersonneNombre("blanchis","verbe",
                                                 MotPersonneNombre.PERS1,
                                                 MotNombre.SINGULIER);
            }else{
                variable = new MotNombre("l'","article",MotNombre.SINGULIER);
            }
            printMot(variable);
        }
    }
}
```

Le type d'instance de l'objet affecté à la variable dépend d'un tirage au sort. Ce tirage au sort (`Math.random()`) étant effectué à l'exécution, on ne peut pas savoir à l'avance quel sera la classe instanciée. On voit également que ce type peut changer d'un tour de boucle à l'autre. Il y a un autre endroit où le type d'instance de l'objet est inconnu, c'est pour le paramètre `unmot` de la méthode `printMot`. Déjà, lors de l'appel en fin de main, on ne connaît pas le type de l'objet passé en paramètre. Mais il faut de plus considérer que cette méthode peut être appelée depuis une autre classe qui peut-être, n'est pas encore écrite quand je compile ce programme. Aucun moyen de savoir donc quel sera le vrai type d'instance de l'objet.

Cela signifie que l'on ne peut pas savoir quels sont les types de l'objet et qu'on en peut pas déterminer à la compilation si une conversion de type va réussir ou lancer l'exception `ClassCastException`.

5 Conversion explicite de types primitifs

Il est parfois possible de convertir une valeur d'un type primitif vers un autre type primitif avec la même syntaxe, c'est-à-dire avec le type désiré entre parenthèses.

Il s'agit d'une opération qu'une autre nature car elle change la valeur convertie. De plus, ces conversions ne peuvent pas échouer à l'exécution.

Toutes les conversions ne sont pas autorisées. Pour savoir lesquelles le sont, il faut lire la spécification du langage ou simplement tester si le compilateur accepte ou non la conversion.

Par exemple `(int) true;`, une conversion pour convertir en entier un booléen provoque une erreur :

```
error: incompatible types: boolean cannot be converted to int
```

Mais la conversion d'un double en entier est permise et permet d'obtenir la partie entière d'un double. Par exemple `System.out.println((int) 1.9);` affiche l'entier 1.