Chapitre 7

Typage

7.1 Principes généraux

7.1.1 Les types en Java

Il existe en java trois sortes de types que l'on peut utiliser pour déclarer une variable, un paramètre d'une méthode ou la valeur calculée et retournée par une méthode :

1. les types primitifs ou types de base : il s'agit de quelques types prédéfinis comme `int`, `double`, `boolean` et `char`.
2. les types objets. Ils comprennent des types prédéfinis et d'autres peuvent être définis par le programme (on verra cela plus tard). Le premier exemple d'un tel type est `String`. Chacun de ces type est définit par une classe. Le nom du type est le nom de la classe.
3. les types tableaux. Pour chaque type java `T`, il est possible d'utiliser des tableaux dont chaque case contient une valeur de type `T`. Le type de ces tableaux est `T[]`. Le type `T` peut être un type primitif, un type objet ou un type tableau.

Certaines opérations sont disponibles pour les valeurs de tous les types java : ce sont l'affectation, le passage en paramètre à une méthode et le passage comme résultat d'une méthode à l'aide de l'instruction `return`. D'autres opérations sont spécifiques à certains types :

- l'utilisation d'opérateurs est limité à un petit nombre de types prédéfinis. Par exemple les opérateurs arithmétiques pour les nombres, les opérateurs logiques pour les booléens et la concaténation pour les chaînes de caractère.
- l'appel d'une méthode sur un objet est limité aux valeurs de type objet. Nous avons vu un certain nombre de méthodes applicables au type `String`.
- on peut consulter la taille d'un tableau avec `.length`. Cela n'est possible que pour les types tableaux.

7.1.2 Typage

Le typage est un processus de contrôle de cohérence d'un programme réalisé par le langage. Il est en majeure partie réalisé à la compilation. Il comprend diverses vérifications :

- que la valeur stockée dans une variable par une affectation est du même type que celui déclaré pour la variable.
- qu'un opérateur est appliqué sur des valeurs du bon type.

– qu'une méthode est appelée avec le bon nombre de paramètres et que ces paramètres ont le type déclaré dans la définition de la méthode.

Si l'une de ces vérification échoue, le programme est considéré comme incorrect et généralement, il ne se compile pas.

7.2 Conversions de types

Il est parfois possible de convertir une valeur d'un type à un autre type. Cela n'est pas toujours possible. Il existe trois formes de conversions : la conversion *implicite* qui est réalisée automatiquement sans qu'aucune opération de conversion ne soit écrite dans le programme ; la conversion *explicite* qui utilise une opération écrite dans le programme ; la conversion par appel d'une méthode.

7.2.1 Conversions implicites entre types primitifs

Certains types sont convertis automatiquement et sans risque dans un autre. Par exemple, un nombre entier peut être utilisé partout où l'on attend un nombre à virgule. La conversion effectuée consiste à rajouter `.0` au nombre entier. Pareillement, on peut utiliser un caractère partout où un nombre entier est attendu. C'est alors le code du caractère dans la table unicode qui est utilisée comme valeur entière. Voici un exemple de programme qui utilise ces deux conversions.

```
public class ExConv{
    public static void main(String[] args){
        double d;
        int i;
        d = 10;
        d = 10.3 + 10;
        Terminal.ecrireDoubleln(10);
        i = 'a';
        i = 5 + 'a';
        Terminal.ecrireIntln('a');
    }
}
```

Ce programme est correcte. Il se compile et son exécution produit l'affichage suivant :

```
> java ExConv
10.0
97
```

La conversion dans l'autre sens ne marche pas. Si l'on essaye de mettre un nombre à virgule dans une variable entière cela produit une erreur de type détectée à la compilation.

```
public class ExConv2{
    public static void main(String[] args){
        int i;
        i = 10.0;
    }
}
```

```
> javac ExConv2.java
ExConv2.java:4: possible loss of precision
found   : double
required: int
        i = 10.0;
            ^
1 error
```

Un autre exemple de conversion implicite est celle qui convertit n'importe quel type de base vers une chaîne de caractère à lors d'une concaténation de chaînes avec l'opérateur +. Par exemple, si l'on écrit : `"resultat: " + 55`, l'entier 55 est converti en la chaîne `"55"` puis cette chaîne est concaténée à la suite de `"resultat: "`, ce qui donne finalement la chaîne `"resultat: 55"`.

Les conversions implicites autorisées par java sont définies dans le manuel du langage qui explique ce que fait exactement la conversion. En pratique, c'est surtout la conversion des entiers vers les nombres à virgule et la conversion vers les chaînes de caractères dans une concaténation que l'on utilise.

7.2.2 Conversions explicites entre types primitifs

Certaines conversions de types sont possibles à condition de les demander explicitement, au moyen d'une syntaxe consistant à noter entre parenthèse le type vers lequel on veut convertir une valeur. On peut par exemple convertir de cette façon un nombre à virgule en un nombre entier. Cette conversion élimine purement et simplement ce qui suit la virgule. Ce n'est donc pas un arrondi.

```
public class ExConv3{
    public static void main(String[] args){
        int i;
        i = (int) 10.7;
        Terminal.ecrireIntln(i);
    }
}
```

Toutes les conversions ne sont pas possibles. Par exemple, la conversion `(boolean) 5` est invalide car `boolean` n'est pas un type numérique.

```
public class ExConv4{
    public static void main(String[] args){
        boolean b;
        b = (boolean) 5;
    }
}
```

```
> javac ExConv4.java
ExConv4.java:4: inconvertible types
found   : int
required: boolean
        b = (boolean) 5;
                      ^
1 error
```

Les conversions permises sont définies précisément dans le manuel du langage.

7.2.3 Conversions de type par appel de méthode

Certaines conversions ne sont permises ni par conversion implicite, ni par conversion explicite. Prenons l'exemple de la conversion d'une chaîne de caractère vers `int`.

```
public class ExConv6{
    public static void main(String[] args){
        int i;
        String s = "123";
        i = s;
        i = (int) s;
    }
}
```

```
> javac ExConv6.java
ExConv6.java:5: incompatible types
found   : java.lang.String
required: int
        i = s;
            ^
ExConv6.java:6: inconvertible types
found   : java.lang.String
required: int
        i = (int) s;
                  ^
2 errors
```

Cette conversion peut se faire au moyen d'une méthode prédéfinie : `Integer.parseInt`. Cette conversion est susceptible d'échouer à l'exécution si la chaîne de caractère ne contient pas un nombre entier. Il existe une méthode équivalente pour la conversion de `String` en nombre à virgule `double`.

```
public class ExConv5{
    public static void main(String[] args){
        int i;
        String s;
        for (int j=0; j<5; j++){
            Terminal.ecrireString("Entrez␣une␣chaine␣a␣convertir:␣");
            s = Terminal.lireString();
            i = Integer.parseInt(s);
            Terminal.ecrireStringln("conversion␣ok");
        }
    }
}
```

```
> java ExConv5
Entrez une chaine a convertir: 123
conversion ok
Entrez une chaine a convertir: 123.6
Exception in thread "main" java.lang.NumberFormatException: For input string: "
        at java.lang.NumberFormatException.forInputString(NumberFormatExceptior
```

```
        at java.lang.Integer.parseInt(Integer.java:458)
        at java.lang.Integer.parseInt(Integer.java:499)
        at ExConv5.main(ExConv5.java:8)
```

Dans les cas où il n'existe pas de méthode de conversion prédéfinie, il est parfois possible d'écrire une nouvelle méthode qui réalise ce travail.

7.2.4 Types primitifs et classes dédiées

Chaque type primitif est associée à une classe prédéfinie qui contient des méthodes de conversions dans les deux sens vers d'autres types. C'est ainsi que le type `int` est associé à la classe `Integer` qui possède diverses méthodes de conversion :

- `String` vers `int` : `Integer.parseInt`
- `int` vers `String` : `Integer.toString`
- d'autres méthodes vers les nombres à virgules.

Cette classe contient également d'autres choses utiles que nous ne détaillerons pas pour le moment.

type	classe
int	Integer
double	Double
char	Character
boolean	Boolean

La classe associée au type `char` propose un certain nombre de méthodes qui permettent de déterminer la nature d'un caractère passé en paramètre :

- `Character.isDigit` détermine si le caractère est un chiffre (résultat boooléen)
- `Character.isLetter` détermine si le caractère est une lettre
- `Character.isLowerCase` détermine si le caractère est une lettre minuscule
- `Character.isUpperCase` détermine si le caractère est une lettre majuscule

De plus il existe des méthode de conversion :

- `Character.toLowerCase` conversion vers les minuscules. Le résultat est du type `char` tout comme le paramètre.
- `Character.toUpperCase` conversion vers les majuscules.
- `Character.toString` conversion vers une chaîne.

```
public class ConvChar{
    public static void main(String[] args){
        char c;
        for (int i=32;i<300;i++){
            c = (char) i;
            Terminal.ecrireString("la␣caractère␣" + c+ "␣est␣");
            if (Character.isDigit(c)){
                Terminal.ecrireStringln("un␣chiffre");
            }else if(Character.isLowerCase(c)){
                Terminal.ecrireString("une␣miniscule␣et␣la␣majuscule");
                Terminal.ecrireString("␣correspondante␣est␣");
                Terminal.ecrireCharln(Character.toUpperCase(c));
            }else if(Character.isUpperCase(c)){
                Terminal.ecrireString("une␣majuscule␣et␣la␣minuscule␣");
                Terminal.ecrireString("␣correspondante␣est␣");
                Terminal.ecrireCharln(Character.toLowerCase(c));
            }else{
```

```
                Terminal.ecrireStringln("un␣caractère␣de␣ponctuation");
            }
        }
    }
}
```

Le tableau 7.1 donne quelques méthodes de conversion entre types.

7.3 Deux usages des conversions de types

7.3.1 Contrôle du nombre de décimales

Lorsqu'on manipule des nombres à virgule, on veut parfois limiter le nombre de chiffres après la virgule, soit pour le calcul, soit seulement pour l'affichage. C'est particulièrement le cas pour les prix : on ne veut pas de fraction de centime.

Pour avoir au plus deux chiffres après la virgule, on peut multiplier un nombre par 100.0, prendre sa partie entière en le convertissant en int, puis diviser par 100.0.

```
public class TDouble{
    public static void main(String[] args){
        double d = 956.3582737839661;
        double d2= ((int) (d*100.0)/100.0);
        double d3= Math.round(d*100.0)/100.0;
        Terminal.ecrireStringln("d=" + d + "␣d2=" + d2 +
                                "␣d3=" + d3);
    }
}
```

7.3.2 Gérer les erreurs d'entrée au clavier

Lorsqu'on appelle Terminal.lireInt, si ce sont des lettres qui sont entrées au lieu d'un nombre, le programme s'arrête brutalement. Pour éviter cela, on peut d'abord faire Terminal.lireString (qui ne peut pas provoquer d'erreur), puis vérifier que la chaîne ne contient que des chiffres, puis enfin convertir avec Integer.parseInt. La deuxième étape (vérifier que la chaîne ne contient que des chiffres) nécessite l'utilisation d'une méthode des chaînes de caractères que nous verrons plus tard. Nous ne pouvons donc pas encore écrire le code suggéré ici.

méthode	conversion	erreurs	à noter
classe Character			
Character.digit(c,b)	char → int	-1 si c n'est pas un chiffre	b : la base
Character.toString(c)	char → String	pas d'erreur	c : un char
classe Integer			
Integer.parseInt(str)	String → int	Exception si str ne représente pas un entier	str : une chaîne
Integer.toString(i)	int → String	pas d'erreur	i : un entier
Integer.toBinaryString(i)	int → String	pas d'erreur	renvoie la représentation binaire de i
Integer.toString(i,b)	int → String	pas d'erreur	i : un entier, b : la base
classe Boolean			
Boolean.parseBoolean(str)	String → boolean	pas d'erreur	return true si str est `"true"` (ou variante avec des majuscules) et false sinon.
Boolean.toString(b)	boolean → String	pas d'erreur	b : un booléen
classe Double			
Double.parseDouble(str)	String → double	Exception si str ne représente pas un double	str : une chaîne
Double.toString(d)	double → String	pas d'erreur	d : un double
classe String			
String.valueOf(i)	int → String	pas d'erreur	i : un int
String.valueOf(d)	double → String	pas d'erreur	i : un double
String.valueOf(b)	boolean → String	pas d'erreur	b : un booléen
String.valueOf(c)	char → String	pas d'erreur	c : un char
classe Math			
Math.round(d)	double → long	pas d'erreur	calcul d'arrondi. Le résultat est long et non pas int (long : entier codé sur 8 octets au lieu de 4 pour les int).

FIGURE 7.1 – Méthode de conversion de types