Chapitre 2

Expressions et algorithmes

2.1 Expressions, calculs et instructions

Une **expression** est un morceau de code Java, qui donne une façon de calculer une valeur d'un certain type. L'exécution de cette expression consiste à réaliser ce calcul.

Une **expression** apparaît dans une **instruction** : la valeur calculée sert dans une instruction. Par exemple, dans une affectation, l'expression est située à droite du signe =. Elle est utilisée pour donner une valeur à la variable dont le nom est situé à gauche du =.

Les instructions simples, celles que nous avons utilisées dans notre premier programme, tiennent sur une ligne. Dans ce programme, aucune expression n'est seule sur une ligne : elle est toujours intégrée dans une instruction qui utilise la valeur calculée.

Reprenons ce programme et voyons quelles expressions il contient.

```
1  public class Conversion {
2      public static void main (String[] args) {
3          double euros;
4          double dollars;
5          System.out.println("Somme␣en␣euros?␣");
6          euros = Terminal.lireDouble();
7          dollars = euros *1.118;
8          System.out.println("La␣somme␣en␣dollars:␣");
9          System.out.println(dollars);
10     }
11 }
```

L'exemple le plus typique d'expression est **euros * 1.118**. Il s'agit d'une expression arithmétique qui donne le moyen de calculer un nombre à virgule. A l'exécution, ce calcul sera réalisé en tenant compte de l'état courant de l'exécution et notamment, il utilisera la valeur de la variable `euros` à ce moment là.

On voit que cette expression apparaît dans une instruction d'affectation qui va associer la valeur calculée avec le nom `dollars`.

Il y a d'autres expressions dans le programme, notamment pour l'affectation de la ligne 6. Cette expression est `Terminal.lireDouble()`. Elle est également le moyen de calculer une valeur de type double, c'est-à-dire un nombre à virgule. Ce moyen de calcul consiste à aller chercher la valeur tapée au clavier. Ce type d'expression s'appelle un appel de méthode.

Une autre sorte d'expression apparaît dans les commandes d'affichage : `"Somme en euros? "` est une expression de type chaîne de caractères (String). Cette expression est un moyen de calcul tri-

vial : sa valeur est la chaîne elle-même. Autrement dit, le calcul de `"Somme en euros? "` donne la chaîne de caractères `"Somme en euros? "`.

Finalement, si l'on souligne toutes les expressions du programme, voici ce que cela donne.

```
public class Conversion {
    public static void main (String[] args) {
        double euros;
        double dollars;
        System.out.println("Somme en euros? ");
        euros = Terminal.lireDouble();
        dollars = euros *1.118;
        System.out.println("La somme en dollars: ");
        System.out.println(dollars);
    }
}
```

Avant de passer en revue les différents types d'expressions, nous allons revenir sur les principaux types de Java, ceux que nous allons utiliser dans ce cours, ainsi que sur les opérateurs que l'on peut utiliser avec les valeurs de ces types.

2.1.1 Instructions

Une instruction est un élément de programme qui suffit à changer l'état de la mémoire ou de l'écran de l'ordinateur. Nous en avons vu trois sortes jusqu'ici :

- les déclarations de variables changent la mémoire lors de l'exécution, car une case mémoire est réservée pour la variable déclarée.
- l'affectation d'une valeur à une variable (=) va stocker la valeur dans la mémoire, dans la case réservée à la variable
- les méthodes d'affichage telles que `System.out.println` changent l'état de l'écran.

Chacune de ces sortes d'instruction peut comporter une ou plusieurs expressions : une déclaration de variable pour donner une valeur initiale, une affectation pour donner une valeur à la variable et un appel de méthode pour préciser les paramètres donnés entre parenthèses.

2.2 Types, valeurs et opérateurs

2.2.1 Type double : les nombres à virgule

Le type double est une approximation des nombres réels. Ces nombres sont codés sur 8 octets, 64 bits, ce qui ne permet pas de représenter tous les nombres réels qui sont beaucoup plus nombreux. On sait qu'il y en a une infinité entre deux réels quels qu'il soient.

Les valeurs des double s'écrivent comme des nombres à virgules habituels, à ce détail près qu'on utilise un point au lieu de la virgule, parce que c'est la notation utilisée par les anglo-saxons.

Voici des exemples de nombres à virgule écrits en Java : 1.2, 158.023, -1.5, 3.0

Les opérateurs utilisables sur les nombres à virgule sont les opérateurs arithmétiques notés +,-,*,/. L'étoile sert à noter la multiplication, la barre oblique (slash) sert à noter la division.

Une chose importante à savoir à propos des nombres à virgule est que les calculs effectués sont approximatifs. D'une part, il se peut que le résultat exact d'un calcul ne fasse pas partie des nombres du type double. D'autre part, les algorithmes utilisés dans les calculs sont efficaces mais ne garantissent

pas l'exactitude du résultat. Chaque calcul est vrai à *un chouia près* (les scientifiques préfèrent parler d'*epsilon*, c'est la même idée).

Prenons un exemple concret.

```
public class Chouia{
    public static void main(String[] args){
        System.out.println(0.7*0.4);
    }
}
```

Ce programme a pour but d'afficher le résultat de la multiplication 0.7*0.4. Le résultat exact est 0.28. L'exécution de ce programme affiche :

```
> java Chouia
0.27999999999999997
```

On voit donc que le résultat obtenu n'est pas le résultat exact mais qu'il en est très proche.

2.2.2 Type int : les nombres entiers

Le type **int** représente un intervalle des nombres entiers et contient tous les nombres de cet intervalle. Une valeur est codée sur 4 octets soit 32 bits. Ce type contient tous les nombres compris entre -2147483648 et +2147483647 (-231 et $2^{31} - 1$). Notez que ces nombres ne sont pas très grands (de l'ordre de 2,1 milliards).

Les valeurs du type sont les nombres notés avec la notation habituelle des entiers : 125, 0, -57.

Les opérateurs utilisables sont les opérateurs arithmétiques +, -, *, /. La division de deux entiers donne un résultat entier : c'est une division euclidienne. Par exemple 10/3 vaut 3. Il existe un opérateur noté % qui donne le reste de la division. Par exemple 10%3 vaut 1.

2.2.3 Type char : les caractères

Le type **char** représente les caractères, c'est-à-dire tout ce qu'on peut afficher dans un champ textuel : des lettres, des chiffres, des espaces, des signes de poncutation, des caractères accentués.

Les valeurs du type sont notés entre apostrophes : `'a'`, `'2'`, `'@'`.

En java les caractères sont codés sur 2 octets, soit 16 bits. A chaque caractère est associé un numéro qui est la position de ce caractère dans la table unicode. Par exemple le caractère `'a'` a le numéro 97, l'espace noté `' '` a le numéro 32. Il n'est pas utile de connaître la table par coeur.

Il n'y a pas d'opérateur utile sur les caractères.

2.2.4 Type boolean : les valeurs de vérité

Le type **boolean** représente les valeurs de vérité : vrai noté `true` et faux noté `false` en Java.

Ces valeurs servent à représenter notamment la véracité d'un énoncé. Par exemple : l'utilisateur du programme est un homme. Cet énoncé est soit faux, soit vrai. Dans cet exemple, l'énoncé peut avoir une valeur différente selon l'exécution du programme : le même programme peut être exécuté tant par un homme que par une femme. Mais pour une exécution donnée, c'est soit l'un soit l'autre.

Les opérateurs les plus utiles sont le *et* logique noté &&, le *ou* logique noté || et le *non* logique noté !.

Le type boolean a une grande utilité en programmation, et il désoriente souvent les débutants car on n'a pas l'habitude de l'utiliser dans la vie courante. Nous reviendrons sur son usage dans les prochains chapitres.

2.2.5 Opérateurs de comparaison

En plus des opérateurs donnés pour chaque type et qui calculent des valeurs du type en question, il existe des opérateurs qui permettent de comparer deux valeurs du même type et dont le résultat est une valeur booléenne.

Il existe un opérateur pour tester si deux valeurs sont égales. Il est noté ==. Par exemple, on peut écrire l'expression 1+2==2+1. Le calcul de cette expression donne comme résultat true, car effectivement 1+2 et 2+1 sont égaux. En revanche, l'expression 1+2==100 donnera, au moment du calcul, la valeur false, car 1+2 et 100 ne sont pas égaux.

Généralement, dans un programme, on utilise cela pour tester une valeur qui n'est pas connue au moment de l'écriture, par exemple parce qu'elle dépend d'une entrée du programme. On écrira plutôt des expressions comme euros==0, qui comportent une ou plusieurs variables.

L'opérateur pour tester que deux valeurs sont différentes s'écrit !=.

Les deux opérateurs == et != peuvent s'utiliser avec chacun des types double, int, char et boolean.

Il existe aussi des opérateurs d'ordre qui peuvent s'utiliser sur les valeurs des types double, int, char mais pas sur les boolean. Ce sont les opérateurs inférieur noté <, inférieur ou égal noté <=, supérieur noté > et supérieur ou égal noté >=.

On peut par exemple écrire l'expression x>10. Celle-ci vaut true ou false selon la valeur stockée dans la variable x.

Pour les caractères, l'ordre utilisé est celui de la table unicode.

2.2.6 Type String

Le type String est celui des chaînes de caractères, qui sont des séquences ordonnées de caractères. Chaque élément de la chaîne est un caractère du type char (lettre, chiffre, espace, signe de ponctuation, symbole monétaire, etc).

Les valeurs du type sont notées entre guillemets : `"coucou", "entrer une somme en euros:?", "a"...`

La chaîne qui ne contient aucun caractère se note "" et elle est une chaîne correcte d'usage assez fréquent.

On ne peut utiliser ni les opérateurs d'égalité == et !=, ni les opérateurs d'ordre <, >, <=, >= sur les chaînes de caractères. Nous verrons beaucoup plus tard dans ce cours comment comparer des chaînes de caractères.

Le type String comporte un opérateur utile : la concaténation de chaîne notée + qui permet de coller bout à bout deux chaînes de caractères : "ab"+"cd" donne la chaîne "abcd".

2.3 Les différentes sortes d'expression

En Java, une expression est un moyen de calculer une valeur d'un type donné. Une expression a donc un type, celui du résultat qu'elle calcule.

Il y a quatre sortes d'expressions :

— une valeur est une expression. Le résultat du calcul de cette expression est la valeur elle-même. Par exemple 17 est une expression qui permet de calculer 17. Idem pour true, 'a', "toto".

— une variable est une expression. Le calcul consiste à aller chercher la valeur stockée dans la variable, à lire cette valeur dans la mémoire. Dans notre premier exemple, `euros` est une expression qui désigne une valeur entrée au clavier.

— un opérateur s'applique à une ou plusieurs expressions et constitue une expression. Par exemple + s'applique à deux nombres, par exemple à deux int et produit un int. 12+1 ou x+1 sont donc des expressions. Pour que x+1 soit correct, il faut que la variable x ait le type int. x==1 est une expression et son type est boolean, car c'est le type du résultat du calcul. Ce résultat est la réponse à la question : x est-il égal à 1 ?
— une méthode qui renvoie un résultat est une expression. Par exemple, Terminal.lireInt() est une expression de type int, car l'exécution de cette instruction est le moyen de calculer un int.
Certaines méthodes demandent des paramètres : les paramètres que l'on donne à une méthode entre parenthèse sont des expressions. Prenons l'exemple de la méthode Math.min qui renvoie le plus petit de deux nombres (soit deux entiers, soit deux double). Un appel à la méthode doit comporter deux paramètres entre parenthèse qui sont des expressions. Par exemple Math.min(x+1,17). Ou encore Math.min(x+1, Terminal.lireInt()). Ou encore Math.min(Math.min(x,y),10).

Attention : certaines méthodes ne renvoient pas de résultat et ne sont donc pas des expressions. C'est par exemple la cas de System.out.println et de Terminal.ecrireStringln. Ces méthodes ne peuvent pas apparaître à droite d'une affectation et elles sont généralement seules sur une ligne de programme.

Exemples d'expressions

Pour savoir si une expression est correcte, il faut vérifier que chaque opérateur utilisé a le bon nombre d'opérandes et que chaque appel de méthode a le bon nombre de paramètres. Ensuite, il faut vérifier que les opérandes et paramètres ont les types attendus par l'opérateur ou la méthode.

Voyons des exemples d'expressions correctes.

```
1 + Terminal.lireInt()
```

Voyons d'abord l'appel de méthode : lireInt attend 0 paramètre, il y en a 0, donc l'appel est correct. Voyons ensuite l'opérateur + : il existe plusieurs opérateurs + : celui des int, celui des double et celui des String. Comme 1 est un int, on suppose qu'il s'agit de l'addition des entiers. Il lui faut deux opérandes : il y en a bien deux, 1 et Terminal.lireInt(). Les deux sont de type int, l'expression est correcte et le résultat est de type int.

```
(7<5) && true
```

L'opérateur $<$ permet de comparer deux valeurs du même type. Ici, il y a bien deux opérandes de type int. Cette partie de l'expression est donc correcte et calcule un booléen (en l'occurrence false, puisque 7 n'est pas plus petit que 5). L'opérateur && attend deux opérandes booléens. Il y en a deux et ils sont chacun du bon type : (7<5) et true. Le résultat de l'expression est un booléen.

Voici maintenant quelques exemples d'expressions incorrectes.

```
1 +
```

Il n'y a pas le bon nombre d'opérandes : il en faut deux, il n'y en a qu'un.

```
Math.min(x,y,12)
```

Il n'y a pas le bon nombre de paramètres dans l'appel de méthode.

```
Math.min(x<5,4)
```

Il y a le bon nombre de paramètres, mais le premier n'est pas du bon type : x<5 est une portion d'expression correcte mais de type boolean alors que Math.min veut comparer deux nombres.

2.3.1 Priorité des opérateurs

Lorsqu'il y a plusieurs opérateurs dans une expression sans parenthèses, il y a une priorité qui définit comment se calcule l'expression. Par exemple 10+3*2 sera calculée comme 10+(3*2). On dit que la multiplication est prioritaire par rapport à l'addition.

La priorité des opérateurs est la suivante (du plus prioritaire au moins prioritaire) :

— le signe - unaire qui calcule l'opposé d'un nombre. Par exemple -(x*3).
— la multiplication, la division et le modulo (reste de la division)
— l'addition et la soustraction.

Si on ne se rappelle pas de la priorité des opérateurs, on peut toujours utiliser des parenthèses pour préciser le calcul que l'on souhaite écrire.

2.4 Eléments de conception des programmes

En début de cours, nous allons faire des programmes très simples qui ne nécessitent pas une longue analyse. Nous allons néanmoins prendre l'habitude de procéder à trois choses avant de commencer à écrire les instructions.

1. Exprimer le but du programme
2. Déterminer quelles données sont nécessaires au programme et choisir lesquelles seront des entrées et lesquelles seront dans le programme.
3. Déterminer quel sera le résultat
4. Ecrire le programme en Java

Reprenons point par point les différentes étapes.

2.4.1 Exprimer le but du programme

S'il s'agit d'un exercice, le but du programme est présent dans l'énoncé. Eventuellement, il faut reprendre cet énoncé et le clarifier. S'il y a plusieurs interprétations possibles, il faut en choisir une. Il est donc possible de reformuler l'énoncé pour le simplifier ou le compléter.

S'il n'y a pas d'énoncé préalable, il faut écrire l'équivalent d'un énoncé d'exercice, qui précise ce que doit faire le programme.

2.4.2 Travail sur les données

Il faut identifier quelles sont toutes les données nécessaires pour traiter le problème et pour chaque donnée, déterminer un type Java. Parmi les données, il faut déterminer lesquelles vont changer lors des différentes exécutions du programme. Ce seront les entrées saisies au clavier. Les données qui seront les mêmes pour toutes les exécutions du programme pouront être inscrites dans le programme.

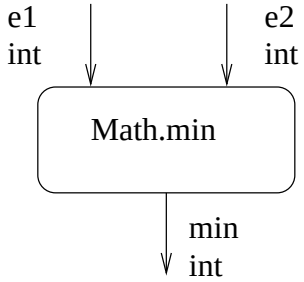
Dans l'exemple de notre premier programme, les données sont la somme en euro et le taux de change. Nous avons choisi dans ce programme de considérer la somme comme une entrée et le taux de change comme une donnée inscrite dans le programme. Ce n'est probablement pas un bon choix puisque le taux de change varie constamment. Le travail sur les données n'a donc pas été très pertinent [1].

1. En fait, ce choix a été fait pour simplifier le premier exemple.

2.4.3 Travail sur le résultat

Il s'agit de préciser combien il y a de valeurs dans le résultat et quels sont leurs types Java.

A l'issue du travail sur les données et le résultat, on peut tracer un petit graphique qui illustre les entrées et sorties du programme avec pour chacune un nom et un type Java.



2.4.4 Ecriture du programme Java

Cette étape devra être raffinée ultérieurement, pour réaliser des programmes complexes. Pour les petits programmes du début d'année, il est possible de commencer à les écrire directement dans l'éditeur ou l'environnement de programmation.

Les deux premières lignes seront un copier-coller des lignes suivantes avec seulement le nom de programme à changer :

```
public class Conversion {
    public static void main (String[] args) {
```

Ensuite viennent les instructions. La structure de nos premiers programme sera :

1. saisir les entrées au clavier
2. calculer le résultat
3. afficher le résultat

Assez rapidement, il pourra y avoir des variantes à cette structure, avec notamment des entrelacement d'entrées et de sorties.