Chapitre 1 : stocker une information

Les noms de variables

Les contraintes suivantes sont à respecter dans l'écriture des noms de variables :

- Le premier caractère d'une variable doit obligatoirement être différent d'un chiffre.
- Aucun espace ne peut figurer dans un nom.
- Les majuscules sont différentes des minuscules, et tout nom de variable possédant une majuscule est différent du même nom écrit en minuscule.
- Les caractères `&`, `~`, `"`, `#`, `'`, `{`, `}`, `(`, `)`, `[`, `]`, `-`, `|`, `` ` ``, `\`, `^`, `@`, `=`, `%`, `*`, `?`, `:`, `/`, `§`, `!`, `<`, `>`, ainsi que `;` et `,` ne peuvent être utilisés dans l'écriture d'un nom de variable.

Tout autre caractère peut être utilisé, y compris les caractères accentués, le caractère de soulignement (_), les caractères grecs (µ) et les symboles monétaires $, _, £, etc.).

Question Parmi les variables suivantes quelles sont celles dont le nom n'est pas autorisé, et pourquoi ?
`Compte, pourquoi#pas, Num_2, -plus, Undeux, 2001espace, @adresse, VALEUR_temporaire, ah!ah!, Val$solde`

Réponse Les noms des variables suivantes ne sont pas autorisés :
`pourquoi#pas, -plus, @adresse, ah!ah!`, car les caractères `#`, `-`, `@` et `!` sont interdits.
`2001espace` car, il n'est pas possible de placer un chiffre en début de variable.
Par contre les noms de variable autorisés sont :
`Compte`, `Num_2` (" _ " et non pas " – "), `Undeux` (et non pas `un deux`), `VALEUR_temporaire`, `Val$solde`.

Les types de base en Java

Catégorie logique

Il s'agit du type `boolean`. Les valeurs logiques ont deux états : `"true"` (vrai) ou `"false"` (faux). Elles ne peuvent prendre aucune autre valeur que ces deux états.

Catégorie caractère

Cette catégorie ne comprend qu'un type de base, le type `char`, qui permet de représenter les caractères isolés.

Remarque Pour décrire une suite de caractères (mots, phrases), on utilise le type `String`, qui n'est pas un type simple, mais un type structuré.

Catégorie entier

Cette catégorie contient quatre types distincts : `byte`, `short`, `int`, `long`. Chacun de ces types autorise la manipulation de valeurs numériques entières, positives ou négatives. Leur différence réside essentiellement dans le nombre d'octets utilisés pour coder le contenu de la variable.

Type	Nombre d'octets	Éventail de valeurs
`byte`	1 octet	De – 128 à 127
`short`	2 octets	De – 32 768 à 32 767
`int`	4 octets	De – 2 147 483 648 à 2 147 483 647
`long`	8 octets	De – 9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

Catégorie réel (flottant)

La catégorie réel permet l'emploi de nombres à virgule, appelés nombres réels ou encore flottants.

Deux types composent cette catégorie, le type `float` et le type `double`. Une expression numérique de cette catégorie peut s'écrire en notation décimale ou exponentielle.

- La notation décimale contient obligatoirement un point symbolisant le caractère « virgule » du chiffre à virgule. Les valeurs `67.3`, `-3.` ou `.64` sont des valeurs réelles utilisant la notation décimale.

- La notation exponentielle utilise la lettre `E` pour déterminer où se trouve la valeur de l'exposant (puissance de 10). Les valeurs 8.76E4 et 6.5E–12 sont des valeurs utilisant la notation exponentielle.

Dans les deux cas, le nombre réel est suivi de la lettre `F` (pour `float`) ou `D` (pour `double`). Les caractères minuscules `f` ou `d` sont également autorisés. La distinction entre `float` et `double` s'effectue sur le nombre d'octets utilisés pour coder l'information. Il en résulte une précision plus ou moins grande suivant le type employé.

Type	Nombre d'octets	Éventail des valeurs
`float`	4 octets	de `1.40239846e-45F` à `3.402823347e38F`
`double`	8 octets	de `4.94065645841246544e-324D` à `1.79769313486231570e308D`

En langage Java, toute valeur numérique réelle est définie par défaut en double précision. Par conséquent, la lettre `d` (ou `D`) placée en fin de valeur n'est pas nécessaire. Par contre, dès qu'on utilise une variable `float`, la lettre `f` (ou `F`) est indispensable, sous peine d'erreur de compilation.

Question Avec quel type de variables les valeurs suivantes peuvent-elles être définies ?

1. `2.15F` et `6.76f`
2. `1.35E22` et `463.4E+234D`

Réponse

1. Les valeurs `2.15F` et `6.76f` sont des nombres à virgule. La lettre F ou f indique à l'interpréteur Java que ces valeurs sont de simple précision. Elles peuvent donc être stockées dans des variables de type `float`.
2. Les valeurs `1.35E22` et `463.4E+234D` sont des nombres à virgule de double précision. La lettre F étant absente, les valeurs sont considérées par l'interpréteur comme étant de type `double`.

Déclarer une variable

La définition d'une variable dans un programme est réalisée par l'intermédiaire de l'instruction de déclaration des variables. Au cours de cette instruction, le programmeur donne le type et le nom de la variable. Pour déclarer une variable, il suffit d'écrire l'instruction selon la syntaxe suivante :

```
type nomdevariable ;
```

ou

```
type nomdevariable1, nomdevariable2 ;
```

Question Quel est le rôle des instructions suivantes ?

```
float   f1, f2 ;
long    CodeBar ;
int     test ;
char     choix, tmp ;
boolean   OK ;
```

Réponse L'instruction :

```
float   f1, f2 ;
```

fait que deux variables de type `float` sont déclarées.

```
long    CodeBar ;
```

permet de déclarer une variable de type `long`.

```
int     test ;
```

permet de déclarer une variable de type `int`.

```
char     choix, tmp ;
```

fait que deux variables de type `char` sont déclarées.

```
boolean   OK ;
```

permet de déclarer une variable de type `boolean`.

L'instruction d'affectation

```
Variable = Valeur ;
```

ou encore,

```
Variable = Expression mathématique ;
```

```
n = 4 ;      // n prend la valeur 4
p = 5*n+1 ; // calcule la valeur de l'expression mathématique soit
             // 5*4+1 et range la valeur obtenue dans la variable
             // représentée par p.
```

Question

Quelles sont les valeurs des variables `prix`, `tva` et `total` après exécution des instructions suivantes :

```
prix = 20 ;
tva = 18.6 ;
total = prix + prix*tva / 100;
```

Réponse

Instruction	prix	tva	total
`prix = 20 ;`	20	-	-
`tva = 18.6 ;`	20	18.6	-
`total = prix + prix * tva / 100 ;`	20	18.6	23.72 (20 + 20 * 18.6 / 100)

Les opérateurs arithmétiques

Écrire un programme ne consiste pas uniquement à échanger des valeurs, c'est aussi calculer des équations mathématiques plus ou moins complexes. Pour exprimer une opération, le langage Java utilise des caractères qui symbolisent les opérateurs arithmétiques.

Symbole	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo

Exemple

Soient `a`, `b`, `c` trois variables de même type.

- L'opération d'addition s'écrit : `a = b + 4.`
- L'opération de soustraction s'écrit : `a = b - 5.`
- L'opération de division s'écrit : `a = b / 2` et non pas $a = \frac{b}{2}$.
- L'opération de multiplication s'écrit : `a = b * 4`
- et non pas `a = 4b` ou `a = 4 x b.`
- L'opération de modulo s'écrit : `a = b % 3.`

Le modulo d'une valeur correspond au reste de la division entière. Ainsi : `5 % 2 = 1.`

Il s'agit de calculer la division en s'arrêtant dès que la valeur du reste devient inférieure au diviseur, de façon à trouver un résultat en nombre entier. L'opérateur `%` n'existe pas pour les réels, pour lesquels la notion de division entière n'existe pas.

Question Les opérations suivantes sont-elles valides ?

```
delta = b2 - 4ac ;
```

$z = \frac{b}{2} + 3\%xa$;

Réponse Aucune des deux opérations n'est valide.

`delta = b2 - 4ac ;` doit s'écrire `delta = b * b - 4 * a * c ;`

$z = \frac{b}{2} + 3\%xa$; doit s'écrire `z = b / 2 + 3 % x * a`

En supposant que *x* corresponde à une variable de type entier et non pas réel. En effet, le modulo est une opération valide uniquement pour les entiers.

La priorité des opérateurs entre eux

Afin d'éviter toute ambiguïté, il existe des règles de priorité entre les opérateurs, règles basées sur la définition de deux groupes d'opérateurs.

Groupe 1	Groupe 2
`+ -`	`* / %`

Les groupes étant ainsi définis, les opérations sont réalisées sachant que :

- Dans un même groupe, l'opération se fait dans l'ordre d'apparition des opérateurs (sens de lecture).
- Le deuxième groupe a priorité sur le premier.

L'expression `a - b / c * d` est calculée de la façon suivante :

Priorité	Opérateur	
Groupe 2	/	Le groupe 2 a priorité sur le groupe 1, et la division apparaît dans le sens de la lecture avant la multiplication.
Groupe 2	*	Le groupe 2 a priorité sur le groupe 1, et la multiplication suit la division.
Groupe 1	-	La soustraction est la dernière opération à exécuter, car elle est du groupe 1.

Cela signifie que l'expression est calculée de la façon suivante :

```
a - (b / c * d)
```

Les parenthèses permettent de modifier les règles de priorité en forçant le calcul préalable de l'expression qui se trouve à l'intérieur des parenthèses. Elles offrent en outre une meilleure lisibilité de l'expression.

Le type d'une expression mathématique

Le résultat d'une expression mathématique peut être déterminé à partir du type de variables (termes) qui composent l'expression.

Terme	Opération	Terme	Résultat
Entier	`+ - * / %`	Entier	Entier
Réel	`+ - * /`	Réel	Réel

De ce fait, pour un même calcul, le résultat diffère selon qu'il est effectué à l'aide de variables de type réel ou de type entier.

Exemple : diviser deux entiers

```
int x = 5 , y = 2, z ;
z = x / y ;
```

	x	y	z
valeur initiale	5	2	-
`z = x / y`	5	2	2

Ici, toutes les variables déclarées sont de type entier. Par conséquent, l'opération effectuée a pour résultat une valeur entière, même si l'opération demandée n'a pas forcément un résultat entier. Soit, pour notre exemple, 2 et non 2.5. Cela ne correspond pas toujours au résultat attendu par le programmeur débutant.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = coefficient * premier / second ;
```

Réponse La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La multiplication `coefficient * premier` donne pour résultat `10`, puis la division de `10` par `second` donne `3`, puisque les deux valeurs sont entières. La variable `résultat` a donc pour valeur `3`.

Question Que vaut la variable `résultat` après exécution des instructions suivantes :

```
int résultat, premier = 5, second = 3, coefficient = 2 ;
résultat = premier / second * coefficient;
```

Réponse La multiplication et la division appartiennent au même groupe, les opérations sont donc réalisées dans le sens de la lecture. La division `premier / second` a pour résultat `1`, puisque les deux valeurs sont entières. Puis la multiplication de `1` par `coefficient` donne `2`. La variable `résultat` a donc pour valeur `2`.

Exemple : diviser deux réels

```
double x = 5 , y = 2, z ;
z = x / y ;
```

	x	y	z
valeur initiale	5	2	-
`z = x / y`	5	2	2.5

La transformation de types

Terme	Opération	Terme	Résultat
`byte`	`+ - * /`	`int`	`int`
`int`	`+ - * /`	`double`	`double`

C'est pourquoi, du fait du codage des données et du nombre d'octets utilisé pour ce codage, le compilateur effectue automatiquement la conversion des données selon l'ordre suivant :

```
byte -> short -> int -> long -> float -> double
```

Exemple

```
int a = 4, result_int ;
float x = 2.0f, result_float ;
result_float = a / x ;
result_int = a / x ;
```

	a	x	result_float	result_int
`a = 4`	4	_	_	_
`x = 2.0f`	4	2.0f	_	_
`result_float = a/x`	4	2.0f	2.0f	_
`result_int = a/x`	4	2.0f	_	Impossible dès la compilation

Le cast

La conversion avec perte d'information est autorisée dans certains cas grâce au mécanisme du *cast*. Il peut être utile de transformer un nombre réel en entier, par exemple pour calculer sa partie entière. Pour cela, le compilateur demande de convertir explicitement les termes de l'opération dans le type souhaité en plaçant devant la variable ou l'opération le type de conversion désiré. Ainsi, pour transformer un `float` en `int`, il suffit de placer le terme `(int)` devant la variable ou l'opération de type `float`.

Exemple

```
int a = 5, result ;
float x = 2.0f ;
result1 = (int) a / x ;
```

	a	x	result1
a = 5	5	_	_
x = 2.0f	5	2.0f	_
result1 = (int) a / x	5	2.0f	2

La dernière instruction montre que la conversion `float` vers `int` est autorisée malgré la perte d'information (le chiffre 5 placé après la virgule disparaît). Cette conversion n'est possible que si elle est précisément indiquée au compilateur.