Le livre de Java premier langage

Anne Tasso

Éditions Eyrolles

ISBN : 2-212-09156-7

2000

1

Stocker une information

En décrivant, au chapitre introductif, « Naissance d'un programme », l'algorithme de confection d'un café chaud non sucré, nous avons constaté que la toute première étape pour construire une marche à suivre consistait à déterminer les objets utiles à la résolution du problème. En effet, pour faire du café, nous devons prendre le café, l'eau, le filtre, etc.

De la même façon, lorsqu'un développeur d'applications conçoit un programme, il doit non pas « prendre », au sens littéral du mot, les données numériques mais **définir** ces données ainsi que les objets nécessaires à la réalisation du programme. Cette définition consiste à nommer ces objets et à décrire leur contenu afin qu'ils puissent être stockés en mémoire.

C'est pourquoi nous étudions dans ce chapitre ce qu'est une variable et comment la définir *(« La notion de variable »)*. Nous examinons ensuite, à la section « L'instruction d'affectation », comment placer une valeur dans une variable, par l'intermédiaire de l'instruction d'affectation. Enfin, nous analysons l'incidence du type des variables sur le résultat d'un calcul arithmétique (*« Les opérateurs arithmétiques »*).

Afin de clarifier les explications, vous trouverez tout au long du chapitre des exemples simples et concis. Ces exemples ne sont pas des programmes complets mais de simples extraits, qui éclairent un point précis du concept abordé. Vous trouverez en fin de chapitre (*« Calculer des statistiques sur des opérations bancaires »*), un programme entier qui aborde et résume toutes les notions rencontrées au fil de ce chapitre.

La notion de variable

Une variable permet la manipulation de données et de valeurs. Elle est caractérisée par les éléments suivants :

- Un **nom**, qui sert à repérer un emplacement en mémoire dans lequel une valeur est placée. Le choix du nom d'une variable est libre. Il existe cependant des contraintes, que nous présentons à la section « Les noms de variables ».

- Un **type**, qui détermine la façon dont est traduite la valeur en code binaire ainsi que la taille de l'emplacement mémoire. Nous examinons ce concept à la section « La notion de type ». Plusieurs types simples sont prédéfinis dans le langage Java, et nous en détaillons les caractéristiques à la section « Les types de base en Java ».

Les noms de variables

Le choix des noms de variables n'est pas limité. Il est toutefois recommandé d'utiliser des noms évocateurs. Par exemple, les noms des variables utilisées dans une application qui gère les codes-barres de produits vendus en magasin sont plus certainement « `article`, `prix`, `codebarre` » que « `xyz1`, `xyz2`, `xyz3` ». Les premiers, en effet, évoquent mieux l'information stockée que les seconds.

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 (_) et les caractères `$`, ¤ et µ.
- Le nombre de lettres composant le nom d'une variable est indéfini. Néanmoins, l'objectif d'un nom de variable étant de renseigner le programmeur sur le contenu de la variable, il n'est pas courant de rencontrer des noms de variables de plus de trente lettres.

Exemples

Nom de variable autorisé	Nom de variable interdit	
`compte`	`pourquoi#pas`	caractère # interdit
`num_2` (" _ " et non pas " - ")	`2001espace`	pas de chiffre en début de variable
`undeux` (et non pas `un deux`)	`-plus`	caractère - interdit
`VALEUR_temporaire`	`@adresse`	caractère @ interdit
`Val$solde`	`ah!ah!`	caractère ! interdit

La notion de type

Un programme doit gérer des informations de nature diverse. Ainsi, les valeurs telles que `123` ou `2.4` sont de type numérique tandis que `Spinoza` est un mot composé de caractères. Si l'être humain sait, d'un simple coup d'œil, faire la distinction entre un nombre et un mot, l'ordinateur n'en est pas capable. Le programmeur doit donc « expliquer » à l'ordinateur la nature de chaque donnée. Cette explication passe par la notion de type.

Le type d'une valeur permet de différencier la nature de l'information stockée dans une variable.

À chaque type sont associés les éléments suivants :

- Un code spécifique permettant la traduction de l'information en binaire et réciproquement.
- Un ensemble d'opérations réalisables en fonction du type de variable utilisé. Par exemple, si la division est une opération cohérente pour deux valeurs numériques, elle ne l'est pas pour deux valeurs de type caractère.
- Un intervalle de valeurs possibles dépendant du codage utilisé. Par définition, à chaque type correspond un même nombre d'octets et, par conséquent, un nombre limité de valeurs différentes.

En effet, un octet, est un regroupement de 8 bits, sachant qu'un bit ne peut être qu'un `0` ou un `1`. Lorsqu'une donnée est codée sur 1 octet, elle peut prendre les valeurs `00000000` (8 zéros), ou encore `111111111` (8 un) et toutes les valeurs intermédiaires entre ces deux extrêmes (par exemple `10101010`, `11110000` ou `10010110`).

En fait, une donnée codée sur 8 bits peut, par le jeu des combinaisons de `0` et de `1`, prendre 2^8 valeurs différentes, soit 256 valeurs possibles comprises entre – 128 et 127. L'intervalle [– 128, 127] est en effet composé de 256 valeurs et possède autant de valeurs positives que négatives.

Pour représenter la valeur numérique 120, un codage sur 1 octet suffit, mais pour représenter la valeur 250, 1 octet ne suffit pas, et il est nécessaire d'utiliser un codage sur 2 octets.

Les types de base en Java

Chaque langage de programmation propose un ensemble de types de base permettant la manipulation de valeurs numériques entières, réelles ou caractères. Ces types sont :

- représentés par un mot-clé prédéfini par le langage.
- dits **simples**, car, à un instant donné, une variable de type simple ne peut contenir qu'une et une seule valeur.

À l'opposé, il existe des types appelés types **structurés** qui permettent le stockage, sous un même nom de variable, de plusieurs valeurs de même type ou non. Il s'agit des tableaux, des classes, des vecteurs ou encore des dictionnaires. Ces types structurés sont en général définis par le programmeur. Nous les étudions en détail dans la troisième partie de cet ouvrage, intitulée « Outils et techniques orientés objet ».

Pour sélectionner un type plutôt qu'un autre, le langage Java définit huit types simples, qui appartiennent, selon ce qu'ils représentent, à l'une ou l'autre des quatre catégories suivantes : logique, texte, entier, réel.

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

Deux types définissent cette catégorie, le type `char` et le type `String`.

Le type `char` permet de représenter les caractères isolés, alors que le type `String` sert à décrire des séquences de caractères. En ce sens, il ne s'agit pas d'un type simple.

> ✔ Voir, au chapitre 7, *« Les classes et les objets »*, la section *« La classe `String`, une approche vers la notion d'objet ».*

Pour décrire une variable de type `char`, l'ordinateur utilise un code sur 2 octets. De cette façon, il lui est possible d'utiliser jusqu'à 2^{16} caractères, soit 65 536 caractères différents. En réalité, Java utilise une table de correspondance, appelée jeu de caractères Unicode. Cette table associe un caractère à une valeur numérique.

Par exemple, dans la table Unicode, le caractère `A` majuscule a pour valeur décimale 65, et le caractère `a` minuscule la valeur décimale 97.

La table Unicode est organisée comme suit :

- Les 31 premiers caractères ne peuvent être affichés (tabulation, saut de ligne, bip sonore, etc.).
- Les caractères compris entre le 32e et le 127e correspondent aux caractères du code ASCII (*American Standard Code for Information Interchange*), qui était jusqu'à présent le code définissant tout caractère. Dans cet intervalle, tous les caractères de base sont définis, c'est-à-dire l'ensemble des lettres de l'alphabet, en minuscules et en majuscules, ainsi que les signes de ponctuation et les symboles mathématiques.
- Les caractères compris entre le 128e et le 256e caractères correspondent à des caractères spéciaux, comme les caractères accentués en minuscules et en majuscules et les caractères semi graphiques. Les codes de ces caractères font partie des extensions qui peuvent différer selon les pays ou les environnements de travail. Ces extensions sont définies à partir du jeu de caractères employé par votre environnement et diffèrent donc d'un type d'ordinateur à un autre.

  > ✔ Pour connaître le code Unicode d'un caractère accentué sur votre ordinateur, reportez-vous à l'exemple de la section *« La boucle `for` »* du chapitre 4, *« Faire des répétitions ».*

- À partir du 257e caractère, il est possible de définir son propre jeu de caractères dans la table Unicode, de façon à représenter, par exemple, des caractères arabes, chinois ou japonais.

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 utilisé 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`

Dans certains cas, il est intéressant de représenter une valeur entière sous forme octale ou hexadécimale comme pour l'affichage des caractères de la table Unicode (*voir, au chapitre 2, « Communiquer une information », la section « Afficher les caractères accentués »*).

Valeur décimale	Valeur octale	Valeur hexadécimale
45	055	0x2d

Pour représenter un nombre sous forme octale, il est nécessaire de placer un zéro au début du nombre. Pour la représentation sous forme hexadécimale, les caractères `0x` doivent être placés en début de valeur.

Dans le langage Java, tous les types de la catégorie entier ont un signe (+ ou -).

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 utilisé pour coder l'information. Il en résulte une précision plus ou moins grande suivant le type utilisé.

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`

Exemple

- La valeur 2.15F représente un simple flottant (type `float`).
- La valeur 1.35E22 représente aussi un flottant de grande taille.
- La valeur 6.76F est une valeur de type `float` de taille simple.
- La valeur 463.4E+234D correspond à un flottant de double précision (type `double`).

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 que l'on utilise une variable `float`, la lettre `f` (ou `F`) est indispensable, sous peine d'erreur de compilation.

Comment choisir un type de variable plutôt qu'un autre ?

Sachant qu'une variable de type `int` (codée sur 4 octets) peut prendre toutes les valeurs de l'intervalle [–2147483648/2147483647] et donc prendre, en particulier, toutes les valeurs comprises entre –32768 et 32767 (type `short`) ou même entre –128 et 127 (type `byte`), posons-nous les questions suivantes :

- Pourquoi ne pas déclarer toutes les variables entières d'un programme en type `long` (le type `long` nous offrant le plus grand choix de valeurs entières) ?
- Pourquoi ne pas déclarer les variables réelles d'un programme en type `double` plutôt qu'en `float` ?

Pour répondre à ces questions, examinons le nombre d'octets utilisés par un programme de gestion de comptes bancaires. Pour simplifier, supposons que le programme garde en mémoire les 10 dernières opérations bancaires et le solde de chaque compte. Imaginons enfin que notre banque gère 50 000 comptes.

Pour stocker les 10 dernières opérations, nous devons déclarer 10 variables plus 1 pour le solde du compte, soit 11 variables. Les valeurs sont des montants en francs et centimes, donc des valeurs réelles.

- Si nous déclarons l'ensemble de ces variables en type `double` (8 octets), le programme utilise alors 50 000 × 11 × 8 octets, soit 44 000 000 octets, soit 4,4 mégaoctets de la mémoire de l'ordinateur.
- Si nous choisissons de prendre des variables de type `float` (ce qui reste cohérent, puisque les montants en francs n'ont pas besoin d'être d'une précision extrême), notre programme n'utilise plus que 2,2 mégaoctets, soit deux fois moins que précédemment.

Bien entendu, cet exemple simpliste n'a pour seul objectif que de montrer l'effet du choix du type de variable sur le taux d'occupation de la mémoire de l'ordinateur. Il existe, en réalité, un grand nombre de techniques pour optimiser la gestion de la mémoire de l'ordinateur.

Remarquons cependant que la première démarche pour gérer au mieux la mémoire de l'ordinateur consiste à bien choisir le type de ses variables. Si l'on sait que, par définition, une variable ne dépasse jamais, pour un programme donné, la valeur numérique 120, celle-ci doit être déclarée avec le type `byte`.

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 ;
```

où `type` correspond à l'un des mots-clés à choisir parmi ceux donnés aux sections précédentes (`boolean`, `char`, `String`, `byte`, `short`, `int`, `long`, `float` ou `double`). Si deux variables

de même type sont à déclarer, il n'est pas besoin de répéter le type, une virgule séparant les deux noms suffisant à les distinguer.

Pour expliquer à l'ordinateur que l'instruction de déclaration est terminée pour le type donné, un point virgule (;) est placé obligatoirement à la fin de la ligne d'instruction.

Exemple

```
float           f1, f2 ;        //Déclaration de deux variables de type float,
                                // une virgule sépare les deux noms de variables
long            CodeBar ;       //Déclaration d'une variable de type long
int             test ;          //Déclaration d'une variable de type int
char            choix, tmp ;    //Déclaration de deux variables de type char
boolean         OK ;            //Déclaration d'une variable de type boolean
```

Les instructions de déclaration peuvent être placées indifféremment au début ou en cours de programme. Une fois la variable déclarée, l'interpréteur Java réserve, au cours de l'exécution du programme, un emplacement mémoire correspondant en taille à celle demandée par le type. Il associe ensuite le nom de la variable à l'adresse de l'emplacement mémoire.

À cette étape du programme, remarquons que l'emplacement ainsi défini est vide. Si l'on souhaite afficher son contenu sans y avoir préalablement déposé de valeur, le compilateur émet le message d'erreur suivant : `Variable may not have been initialized`. Cette erreur indique que la variable dont on souhaite afficher le contenu n'a pas été initialisée. Comme l'interpréteur Java ne peut afficher un emplacement mémoire vide, l'exécution du programme n'est pas possible.

L'instruction d'affectation

Une fois la variable déclarée, il est nécessaire de stocker une valeur à l'emplacement mémoire désigné. Pour ce faire, nous utilisons l'instruction d'affectation, qui nous permet d'initialiser ou de modifier, en cours d'exécution du programme, le contenu de l'emplacement mémoire (le contenu d'une variable n'étant, par définition, pas constant).

Rôle et mécanisme de l'affectation

L'affectation est le mécanisme qui permet de placer une valeur dans un emplacement mémoire. Elle a pour forme :

```
Variable = Valeur ;
```

ou encore,

```
Variable = Expression mathématique ;
```

Le signe égal (=) symbolise le fait qu'une valeur est placée dans une variable. Pour éviter toute confusion sur ce signe mathématique bien connu, nous prendrons l'habitude de le traduire par les termes `prend la valeur`.

Examinons les exemples suivants, en supposant que les variables `n` et `p` soient déclarées de type entier :

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

L'instruction d'affectation s'effectue dans l'ordre suivant :

1. Calcule la valeur de l'expression figurant à droite du signe égal ;
2. Range le résultat obtenu dans la variable mentionnée à gauche du signe égal.

D'une manière générale, il est intéressant de remarquer que la variable placée à droite du signe `=` n'est jamais modifiée, alors que celle qui est à gauche l'est toujours. Comme une variable de type simple ne peut stocker qu'une seule valeur à la fois, si la variable située à gauche possède une valeur avant l'affectation, cette valeur est purement et simplement remplacée par la valeur située à droite du signe `=`.

Exemple

```
a = 1 ;
b = a + 3 ;
a = 3 ;
```

Lorsqu'on débute en programmation, une bonne méthode pour comprendre ce que réalise un programme consiste à écrire, pour chaque instruction exécutée, un état de toutes les variables déclarées. Il suffit pour cela de construire un tableau dont chaque colonne représente une variable déclarée dans le programme et chaque ligne une instruction de ce même programme. Soit, pour notre exemple :

instruction	a	b
`a = 1`	1	-
`b = a + 3`	1	4
`a = 3`	3	4

Le tableau est composé des deux colonnes `a` et `b` et des trois lignes associées aux instructions d'affectation du programme. Ce tableau montre que les instructions `a = 1` et `a = 3` font que la valeur initiale de `a (1)` est effacée et écrasée par la valeur `3`.

Déclaration et affectation

Comme nous l'avons vu à la section « Déclarer une variable », la déclaration est utilisée pour réserver un emplacement mémoire. Une fois réservé, l'emplacement reste vide jusqu'à ce qu'une valeur y soit placée par l'intermédiaire de l'affectation.

Il est cependant risqué de déclarer une variable sans lui donner de valeur initiale. En effet, le compilateur Java vérifie strictement si toutes les variables contiennent une valeur ou non. Une erreur de compilation est détectée dès qu'une seule variable ne contient pas de valeur à un moment donné du programme.

Initialiser une variable

Pour éviter toute erreur de compilation, une bonne habitude consiste à initialiser toutes les variables au moment de leur déclaration, en procédant de la façon suivante :

```
float       f1 = 0.0f, f2 = 1.2f ;  // Initialisation de deux float
long        CodeBar = 123456789 ;   // Initialisation d'un long
int         test = 0 ;              // Initialisation d'une variable de type int
boolean     OK = true ;             // Initialisation d'un boolean
```

De cette façon, les variables `f1`, `f2`, `CodeBar` et `OK` sont déclarées. Le compilateur réserve un emplacement mémoire pour chacune d'entre elles. Grâce au signe d'affectation, le compilateur place dans chacun des emplacements mémoire respectifs les valeurs données.

Initialiser une variable de type `char`

Les variables de type `char` s'initialisent d'une façon particulière. Supposons que l'on souhaite déclarer et placer le caractère `n` dans une variable `choix` de type `char`. Pour cela, écrivons l'instruction de déclaration et d'initialisation suivante :

```
char choix = n ;
```

Pour le compilateur, cette instruction est problématique, car il considère `n` non pas comme le « caractère `n` » mais comme une variable appelée `n`.

Pour lever cette ambiguïté, nous devons entourer le caractère `n` d'apostrophes, de la façon suivante :

```
char choix = 'n' ;
```

Ainsi, des données telles que `'a'`, `'*'`, `'$'`, `'3'`, `':'` ou `'?'` sont considérées comme des caractères.

Par contre `c = 'ab'` ne peut s'écrire, car `'ab'` n'est pas un caractère mais un mot de deux caractères. Nous devons, dans ce cas, utiliser une variable de type `String`.

✔ Voir, au chapitre 7, *« Les classes et les objets »,* la section *« La classe `String`, une approche vers la notion d'objet ».*

Quelques confusions à éviter

Le symbole de l'affectation est le signe égal (`=`). Ce signe, très largement utilisé dans l'écriture d'équations mathématiques, est source de confusion lorsqu'il est employé à contre-sens.

Pour mieux nous faire comprendre, étudions trois cas :

1. a = a + 1 ;

   Si cette expression est impossible à écrire d'un point de vue mathématique, elle est très largement utilisée dans le langage informatique. Elle signifie :

   – calculer l'expression `a + 1` ;

– ranger le résultat dans `a`.

Ce qui revient à augmenter de 1 la valeur de a.

2. a + 5 = 3 ;

   Cette expression n'a aucun sens d'un point de vue informatique. Il n'est pas possible de placer une valeur à l'intérieur d'une expression mathématique, puisque aucun emplacement mémoire n'est attribué à une expression mathématique.

3. a = b ; et b = a ;

   À l'inverse de l'écriture mathématique, ces deux instructions ne sont pas équivalentes. La première place le contenu de `b` dans `a`, tandis que la seconde place le contenu de `a` dans `b`.

Échanger les valeurs de deux variables

Nous souhaitons échanger les valeurs de deux variables de même type, appelées `a` et `b` ; c'est-à-dire que nous voulons que `a` prenne la valeur de `b` et que `b` prenne celle de `a`. La pratique courante de l'écriture des expressions mathématiques fait que, dans un premier temps, nous écrivions les instructions suivantes :

```
a = b ;
b = a ;
```

Vérifions sur un exemple si l'exécution de ces deux instructions échange les valeurs de `a` et de `b`. Pour cela, supposons que les variables `a` et `b` contiennent initialement respectivement `2` et `8`.

	a	b
`valeur initiale`	2	8
`a = b`	8	8
`b = a`	8	8

Du fait du mécanisme de l'affectation, la première instruction `a = b` détruit la valeur de `a` en plaçant la valeur de `b` dans la case mémoire `a`. Lorsque la seconde instruction `b = a` est réalisée, la valeur placée dans la variable `b` est celle contenue à cet instant dans la variable `a`, c'est-à-dire la valeur de `b`. Il n'y a donc pas échange, car la valeur de `a` a disparu par écrasement lors de l'exécution de la première instruction.

Une solution consiste à utiliser une variable supplémentaire, destinée à contenir temporairement une copie de la valeur de `a`, avant que cette dernière soit écrasée par la valeur de `b`. Pour évoquer le caractère temporaire de la copie, nous appellerons cette nouvelle variable `tmp` (nous aurions pu choisir tout aussi bien `tempo` ou `ttt`). Voici le déroulement des opérations :

```
tmp = a ;
a = b ;
b = tmp ;
```

Vérifions qu'il y a réellement échange, en supposant que nos variables `a` et `b` contiennent initialement respectivement `2` et `8`.

	a	b	tmp
`valeur initiale`	2	8	_
`tmp = a`	2	8	2
`a = b`	8	8	2
`b = tmp`	8	2	2

À la lecture de ce tableau, nous constatons qu'il y a bien échange des valeurs entre `a` et `b`. La valeur de `a` est copiée dans un premier temps dans la variable `tmp`. La valeur de `a` peut dès lors être effacée par celle de `b`. Pour finir, grâce à la variable `tmp` la variable `b` récupère l'ancienne valeur de `a`.

✔ Une autre solution vous est proposée dans la feuille d'exercices placée à la fin du chapitre.

Les opérateurs arithmétiques

Écrire un programme n'est pas uniquement échanger des valeurs, mais 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 = a 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 le 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.

L'ensemble de ces opérateurs est utilisé pour calculer des expressions mathématiques courantes. Le résultat de ces expressions n'est cependant pas toujours celui auquel on s'attend. Trois phénomènes ont une influence non négligeable sur la valeur du résultat d'un calcul. Ce sont :

- La priorité des opérateurs entre eux ;
- Le type d'une expression mathématique ;
- La transformation de types.

La priorité des opérateurs entre eux

Lorsqu'une expression arithmétique est composée de plusieurs opérations, l'ordinateur doit pouvoir déterminer quel est l'ordre des opérations à effectuer. Le calcul de l'expression `a - b / c * d` peut signifier *a priori* :

- calculer la soustraction puis la division et pour finir la multiplication, soit le calcul : `((a - b) / c) * d` ;
- calculer la multiplication puis la division et pour finir la soustraction, c'est-à-dire l'expression : `a - (b / (c * d))`.

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)
```

Remarquons que 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.

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

Ici, toutes les variables déclarées sont de type réel. Par conséquent, l'opération effectuée donne un résultat de type réel. Ce résultat correspond à la valeur généralement attendue de ce type d'opération.

La transformation de types

Les termes d'une opération ne sont pas nécessairement tous du même type. Pour écrire une opération, toutes les combinaisons entre les différentes catégories de types peuvent se présenter.

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

L'ordinateur ne sait calculer une expression mathématique que lorsque toutes les variables de l'expression sont du même type. En effet, les opérateurs arithmétiques ne sont définis que pour des variables de type identique.

Lorsque tel n'est pas le cas, c'est-à-dire si l'expression est mixte, l'ordinateur doit transformer le type de certaines variables pour que tous les membres de l'expression deviennent de même type.

Cette transformation, appelée **conversion d'ajustement de type**, se réalise suivant une hiérarchie bien déterminée, qui permet de ne pas perdre d'information. On dit que le compilateur respecte l'intégralité des données.

La conversion d'un nombre réel en nombre entier, par exemple, ne peut se réaliser qu'en supprimant les nombres situés après la virgule et en ne gardant que la partie entière du nombre. Une telle conversion ne garantit pas l'intégralité des données car il y a perte de données.

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
```

De cette façon, il est toujours possible de convertir un `byte` en `long` ou un `int` en `float`. Par contre, il est impossible de transformer un `float` en `short` sans perte d'information.

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

La troisième instruction montre que le calcul d'une opération dont les termes sont de type `int` et `float` donne pour résultat un `float`. La dernière instruction révèle que, si le résultat d'une opération est de type `float`, il n'est pas possible de le stocker dans une variable de type `int`. En effet, la division d'un entier par un réel est une opération toujours possible à réaliser (le résultat est de type réel), mais l'affectation directe de ce résultat dans une variable entière est impossible du fait que la conversion entraîne une perte d'information.

Une telle instruction provoque à la compilation une erreur dont le message est : `Incompatible type for =. Explicit cast needed to convert float to int.` Cela signifie : « Type incompatible de part et d'autre du signe =. Pour convertir un `float` en `int`, il est nécessaire de le formuler explicitement par l'intermédiaire d'un cast. »

Le cast

La conversion avec perte d'information est autorisée dans certain 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 ;
result = (int) a / x ;
```

	a	x	result
a = 5	5	_	_
x = 2.0f	5	2.0f	_
result = (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.

Calculer des statistiques sur des opérations bancaires

Pour résumer en pratique l'ensemble des notions abordées dans ce chapitre, nous allons écrire un programme, dont le sujet se rapporte au thème du projet énoncé à la fin du chapitre introductif, « Naissance d'un programme ».

Cahier des charges

L'objectif de ce programme est d'établir des statistiques sur l'utilisation des différents modes de paiement effectués sur un compte bancaire. Nous supposons que les moyens techniques pour débiter un compte sont au nombre de trois : la Carte Bleue, le chéquier et le virement. Pour évaluer le taux d'utilisation de ces trois moyens de paiement, nous devons calculer le pourcentage d'utilisation de chaque technique par rapport aux autres. Par exemple, pour connaître le pourcentage d'utilisation de la Carte Bleue, nous utilisons le calcul suivant :

Nombre de paiements par Carte Bleue / Nombre total de paiements * 100

Liste des opérations

Partant du principe de décomposition d'un problème en sous-tâches plus simples à réaliser, distinguons, pour résoudre la question, les quatre actions suivantes :

1. Déterminer le nombre de débits par Carte Bleue, chèque et virement. Comme il s'agit du premier programme concernant ce thème, nous n'avons pas encore saisi de valeur, ni de ligne comptable. C'est pourquoi nous demandons à l'utilisateur de communiquer au programme ces trois informations, par l'intermédiaire du clavier.
2. Calculer le nombre total de paiements effectués.
3. Calculer le pourcentage d'utilisation de la Carte Bleue, du chéquier et du virement.
4. Afficher l'ensemble des résultats.

Dans un premier temps, nous traiterons séparément chacun de ces points afin de les analyser entièrement. Pour finir, nous écrirons le programme dans son intégralité, en regroupant chacun des points étudiés.

1. Il s'agit d'écrire les instructions qui permettent à l'utilisateur de communiquer des informations à l'ordinateur à l'aide du clavier. Nous avons vu, au chapitre introductif, un exemple de saisie d'une valeur au clavier (*voir « Calcul de la circonférence d'un cercle »*). Cette opération se réalise en deux temps : d'abord l'affichage à l'écran d'un message informant l'utilisateur d'une demande de saisie de valeur, puis la saisie effective de l'information. Pour notre problème, ces deux points se traduisent de la façon suivante :

```
System.out.print(" Nombre de paiement par Carte Bleue ") ;
nbCB = Lire.i() ;
System.out.print(" Nombre de cheques émis ") ;
nbCheque = Lire.i() ;
System.out.print(" Nombre de virements automatiques ") ;
nbVirement = Lire.i() ;
```

Chaque appel de la fonction `System.out.print()` affiche à l'écran le message placé entre `guillemets`. Trois messages sont affichés, chacun indiquant respectivement à quel mode de paiement est associée la valeur saisie par l'utilisateur.

Les valeurs à saisir correspondent aux nombres de débits dans chaque mode de paiement. Ces valeurs sont de type entier. La fonction `Lire.i()` donne l'ordre à l'ordinateur d'attendre la saisie d'une valeur entière. La saisie est effective lorsque l'utilisateur valide sa réponse en appuyant sur la touche « `Entrée` » du clavier. Trois valeurs sont à saisir, et il est nécessaire d'appeler trois fois la fonction `Lire.i()`.

> ✔ Pour plus d'informations sur la fonction `Lire.i()`, voir le chapitre 2, *« Communiquer une information »*.

Une fois saisie, chaque valeur doit être stockée dans un emplacement mémoire distinct. Ces emplacements mémoire correspondent aux trois variables `nbCB`, `nbCheque` et `nbVirement` et sont déclarés en début de programme grâce à l'instruction :

```
int nbCB = 0, nbCheque = 0, nbVirement = 0 ;
```

2. Pour calculer le nombre total de paiements effectués, il suffit de faire la somme de toutes les opérations de débit pour tous les types de paiement, soit l'instruction :

```
nbDebit = nbCB + nbCheque + nbVirement ;
```

La variable `nbDebit` permet la mémorisation du nombre total d'opérations effectuées, quel que soit le mode de paiement. Elle doit être déclarée en même temps que les autres variables du même type :

```
int nbCB = 0, nbCheque = 0, nbVirement = 0, nbDebit = 0 ;
```

3. Pour calculer le pourcentage d'utilisation de la Carte Bleue, du chéquier et du virement, nous allons d'abord étudier le mode Carte Bleue puis appliquer cette analyse aux autres modes de paiement. Rappelons que la formule du calcul de pourcentage pour la Carte Bleue est :

Nombre de paiements par Carte Bleue / Nombre total de paiements * 100

Soit, en utilisant les variables déclarées au point 1 : `nbCB / nbDebit * 100`.

Examinons sur un exemple numérique le résultat d'un tel calcul. Supposons pour cela que nous ayons effectué 10 retraits Carte Bleue sur un total de 40 retraits. Nous obtenons le calcul suivant : `10 / 40 * 100`. Soit `0 * 100`, c'est-à-dire `0`. La division est la première opération exécutée parce qu'elle est du même groupe que la multiplication et qu'elle apparaît en premier dans l'opération. De surcroît, les valeurs étant de type entier, la division a pour résultat un nombre entier. Ici `10/40` a pour résultat `0`.

Pour corriger cette erreur de calcul, l'idée est de réaliser une division sur des valeurs réelles et non sur des entiers. Pour cela, nous utilisons le mécanisme du cast, qui, placé devant la variable `nbCB`, transforme cette dernière en variable de type réel et permet la division en réel. Pour stocker le résultat de cette opération, nous déclarons une variable de type `float`, nommée `prctCB`.

L'instruction :

```
prctCB = (float) nbCB / nbDebit * 100 ;
```

permet de trouver un résultat cohérent. Vérifions cela sur un exemple numérique. Supposons que nous ayons effectué 10 débits par Carte Bleue sur un total de 20 retraits. Grâce au cast, la valeur `10` correspondant à `nbCB` est transformée en `10.0`. La division par 20 a donc un résultat réel égal à 0.5. Le taux d'utilisation de la Carte Bleue est donc de `0.5 * 100`, soit 50 %.

Pour établir le pourcentage relatif aux modes chéquier et virement, il suffit d'appliquer le même calcul, en utilisant des variables appropriées aux deux autres moyens de paiement. En nommant `prctCh` et `prctVi` les variables associées aux modes de paiement par chèque et par virement automatique, le taux d'utilisation pour chacun de ces modes s'écrit :

```
prctCh = (float) nbCheque / nbDebit * 100 ;
prctVi = (float) nbVirement / nbDebit * 100 ;
```

4. L'affichage des résultats s'effectue par l'intermédiaire de la fonction `System.out.print()`. Les valeurs calculées sont commentées de la façon suivante :

```
System.out.println(" Vous avez emis " + nbDebit + " ordres de debit ") ;
System.out.println(" dont " + prctCB + " % par Carte Bleue ") ;
System.out.println("      " + prctCh + " % par cheque ") ;
System.out.println("      " + prctVi + " % par virement ") ;
```

Le programme final s'écrit en regroupant l'ensemble des instructions définies précédemment et en les insérant dans une classe à l'intérieur de la fonction `main()`.

Le code source complet

```
public class Statistique
{
  public static void main (String [] arg)
  {
   int nbCB = 0, nbCheque = 0, nbVirement = 0, nbDebit = 0 ;
   float prctCB, prctCh, prctVi ;
   System.out.print(" Nombre de paiements par Carte Bleue : ") ;
   nbCB = Lire.i() ;
   System.out.print(" Nombre de cheques emis : ") ;
   nbCheque = Lire.i() ;
   System.out.print(" Nombre de virements automatiques : ") ;
   nbVirement = Lire.i() ;

   nbDebit = nbCB + nbCheque + nbVirement;

   prctCB = (float) nbCB / nbDebit * 100 ;
   prctCh = (float) nbCheque / nbDebit * 100 ;
   prctVi = (float) nbVirement / nbDebit * 100 ;

  System.out.println("Vous avez emis " + nbDebit + " ordres de debit") ;
  System.out.println("dont " + prctCB + " % par Carte Bleue") ;
  System.out.println("     " + prctCh + " % par cheque") ;
  System.out.println("     " + prctVi + " % par virement") ;
  }
}
```

Résultat de l'exécution

À l'exécution de ce programme, nous avons à l'écran l'affichage suivant (les caractères grisés sont des valeurs choisies par l'utilisateur) :

```
Nombre de paiements par Carte Bleue : 5
Nombre de cheques emis : 10
Nombre de virements automatiques : 5
Vous avez emis 20 ordres de debit
dont 25.0 % par Carte Bleue
     50.0 % par cheque
     25.0 % par virement
```

Résumé

Une **variable** est caractérisée par un **nom** et un **type**. Le nom sert à repérer un emplacement mémoire. Le type détermine la taille de cet emplacement, ainsi que la manière dont l'information est codée, les opérations autorisées et l'intervalle des valeurs représentables.

Il existe plusieurs types simples, dont les plus utilisés sont les suivants :

- `int`. Présente les entiers variant, pour le langage Java, entre – 2 147 483 648 et 2 147 483 647.
- `double`. Décrit de manière approchée les nombres réels dont la valeur absolue est grande. Les variables de type `double` se notent soit sous forme décimale (67.7, –9.2, 0.48 ou .22), soit sous forme exponentielle 3.14E4, .325707e2, –45.567E-5.
- `char`. Désigne les caractères. Les valeurs de type caractère se notent en plaçant entre apostrophes le caractère lui-même.

L'instruction d'**affectation** permet de placer une valeur dans une variable. Elle est de la forme : `variable = expression;`.

Elle calcule d'abord la valeur de l'expression mentionnée à droite du signe `=`, puis elle l'affecte à la variable placée à gauche du signe.

Il est conseillé d'attribuer une valeur initiale à une variable au moment de sa déclaration. Par exemple `int i = 6;` ou `char c = 'n';`.

Pour calculer des expressions mathématiques, il existe cinq **opérateurs** arithmétiques : `+ - * / %`.

Ces opérateurs sont utilisés respectivement pour l'addition, la soustraction, la multiplication, la division et le modulo (reste de la division entière). Les expressions arithmétiques sont calculées à partir des règles suivantes :

- Entier `+ - * / %` entier donne un entier.
- Réel `+ - * /` réel donne un réel.
- Les opérations mixtes du type :

```
entier + - * / réel ou réel + - * / entier
```

donnent un résultat dans la mesure où la valeur résultante n'est pas dénaturée par la conversion des types. Les conversions sont effectuées automatiquement dans le sens suivant :

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

Un `int` peut donc être transformé en un `double`. L'inverse n'est possible que lorsque le mode de conversion est explicitement décrit dans l'expression, comme dans `n = (int) x`, où `n` est de type `int` et `x` de type `double`.

L'information ainsi transformée est tronquée pour être codée sur moins d'octets.

- Il existe des règles de **priorit**é entre les opérateurs. Pour cela, deux groupes d'opérateurs sont définis.

Groupe 1	Groupe 2
+ -	* / %

Dans un même groupe, l'opération se fait dans l'ordre d'apparition des opérateurs.

Le second groupe a priorité sur le premier.

Les parenthèses permettent la modification des priorités.

Exercices

Repérer les instructions de déclaration, observer la syntaxe d'une instruction

1.1 Observez ce qui suit, et indiquez ce qui est ou n'est pas une déclaration et ce qui est ou n'est pas valide :

```
a. int i, j, valeur ;
b. limite - j = 1024 ;
c. val = valeur / 16 ;
d. char char ;
e. j + 1 ;
f. int X ;
g. float A ;
h. A = X / 2 ;
i. X = A / 2 ;
j. X = X / 2 ;
```

Comprendre le mécanisme de l'affectation

1.2 Quelles sont les valeurs des variables `A`, `B`, `C` après l'exécution de chacun des extraits de programme suivants :

a.

```
float A = 3.5f ;
float B = 1.5f ;
float C ;
C = A + B ;
B = A + C ;
A = B ;
```

b.

```
double A = 0.1 ;
double B = 1.1 ;
double C, D ;
B = A ;
C = B ;
D = C ;
A = D ;
```

1.3 Quelles sont les valeurs des variables `a`, `b` et `c`, `valeur`, `x`, `y` et `z`, après l'exécution de chacune des instructions suivantes :

a.	b.	c.

`int a = 5, b ;`
`b = a + 4 ;`
`a = a + 1 ;`
`b = a - 4 ;`
	`int valeur = 2 ;`
	`valeur = valeur + 1 ;`
	`valeur = valeur * 2 ;`
	`valeur = valeur % 5 ;`
		`int x = 2, y = 10, z ;`
		`z = x + y ;`
		`x = 5 ;`
		`z = z - x ;`

Comprendre le mécanisme d'échange de valeurs

1.4 Dans chacun des cas, quelles sont les valeurs des variables `a` et `b` après l'exécution de chacune des instructions suivantes :

1.	2.

`int a = 5`
`int b = 7`
`a = b`
`b = a`
	`int a = 5`
	`int b = 7`
	`b = a`
	`a = b`

1.5 Laquelle des options suivantes permet d'échanger les valeurs des deux variables `a` et `b` ?

```
a = b ; b = a ;
t = a ; a = b ; b = t ;
t = a ; b = a ; t = b ;
```

1.6 Soit trois variables `a`, `b` et `c` (entières). Écrivez les instructions permutant les valeurs, de sorte que la valeur de `a` passe dans `b`, celle de `b` dans `c` et celle de `c` dans `a`. N'utilisez qu'une (et une seule) variable entière supplémentaire, nommée `tmp`.

1.7 Quel est l'effet des instructions suivantes sur les variables `a` et `b` (pour vous aider, initialisez `a` à 2 et `b` à 5) :

```
a = a + b ;
b = a - b ;
a = a - b ;
```

Calculer des expressions mixtes

1.8 Donnez les valeurs des expressions suivantes, sachant que `i` et `j` sont de type `int` et `x` et `y` de type `double` (`x = 2.0`, `y = 3.0`) :

```
a.      i = 100 / 6 ;
b.      j = 100 % 6 ;
c.      i = 5 % 8
d.      (3 * i - 2 * j) / (2 * x - y)
e.      2 * ((i / 5) + (4 * (j - 3)) % (i + j - 2))
f.      (i - 3 * j) / (x + 2 * y) / (i - j)
```

1.9 Donnez le type et la valeur des expressions suivantes, sachant que `n`, `p`, `r`, `s` et `t` sont de type `int` (`n = 10`, `p = 7`, `r = 8`, `s = 7`, `t = 21`) et que `x` est de type `float` (`x = 2.0f`) :

a.	b.
`x + n % p`	`r + t / s`
`x + n / p`	`( r + t ) / s`
`(x + n) / p`	`r + t % s`
`5. * n`	`(r + t) % s`
`(n + 1) / n`	`r + s / r + s`
`(n + 1.0) / n`	`(r + s) / ( r + s)`
`r + s / t`	`r + s % t`

Comprendre le mécanisme du cast

1.10 Soit les déclarations suivantes :

```
int valeur = 7, chiffre = 2, i1, i2 ;
float f1, f2 ;
```

Quelles sont les valeurs attribuées à `i1`, `i2`, `f1` et `f2` après le calcul de :

```
i1 = valeur / chiffre ;
i2 = chiffre / valeur ;
f1 = (float) (valeur / chiffre) ;
f2 = (float) (valeur / chiffre) + 0.5f ;
i1 = (int) f1 ;
i2 = (int) f2 ;
f1 = (float) valeur / (float) chiffre ;
f2 = (float) valeur / (float) chiffre + 0.5f ;
i1 = (int) f1 ;
i2 = (int) f2 ;
```

Le projet « Gestion d'un compte bancaire »

Déterminer les variables nécessaires au programme

Le programme de gestion d'un compte bancaire ne peut s'écrire et s'exécuter sans aucune variable. Pour pouvoir définir toutes les variables nécessaires à la bonne marche du programme, nous devons examiner attentivement le cahier des charges décrit au chapitre introductif, « Naissance d'un programme ».

La section « Les objets manipulés » nous donne une première idée des variables à déclarer. Toutes les données relatives au compte bancaire y sont décrites.

Un compte bancaire est défini par un ensemble de données :

- un numéro de compte ;
- un type de compte (courant, épargne, joint, etc.) ;

- des lignes comptables possédant chacune une valeur, une date, un thème et un moyen de paiement.

Ces données peuvent être représentées de la façon suivante :

Données	Exemple	Type de l'objet
Numéro du compte	4010.205.530	Suite de caractères
Type du compte	Courant	Suite de caractères
Valeur	–1520.30	Numérique
Date	04 03 1978	Date
Thème	Loyer	Suite de caractères
Moyen de paiement	CB	Suite de caractères

Compte tenu de ces informations, donnez un nom et un type Java pour chaque donnée définie ci-dessus.

Remarquons que le type qui représente les suites de caractères (`String`) n'a pas encore été étudié, ni toutes ses fonctionnalités. Il est possible de transformer pour l'instant les données `Type du compte`, `Thème` et `Moyen de paiement` en caractères simples. Par exemple, le caractère `C` caractérise le type du compte `Courant`, le caractère `J` le compte `Joint` et le caractère `E` le compte `Epargne`.

De la même façon, la donnée `Numéro du compte` peut être transformée dans un premier temps en type `long`.