Vincent **Granet**
Jean-Pierre **Regourd**

Aide-mémoire
Java

4e édition

DUNOD

Photo de couverture : © iStock.com/4774344sean

978-2-10-072713-1

5 rue Laromiguière, 75005 Paris
www.dunod.com

à Maud et Marianne

Table des matières

Avant-propos VIII

1. Java, les bases 1
   - 1.1 Une première application 1
   - 1.2 Les types élémentaires 6
   - 1.3 Les identificateurs 10
   - 1.4 Les expressions 12
   - 1.5 Les énoncés 14

2. Les objets de base 27
   - 2.1 Les chaînes de caractères 27
   - 2.2 Les conteneurs 31
   - 2.3 Les tableaux 36
   - 2.4 Les énumérations 43

3. Le modèle objet de Java 47
   - 3.1 Un fil rouge 47
   - 3.2 Les concepts 48
   - 3.3 La classe 49
   - 3.4 Les variables et les méthodes 54
   - 3.5 Le constructeur d'instances 61

3.6 Les paquetages 63
3.7 L'accessibilité des éléments 66

4. Les mécanismes de programmation par objets 69
4.1 L'héritage 69
4.2 Les classes abstraites 77
4.3 Les interfaces 78
4.4 Les classes et interfaces internes 85
4.5 Le masquage, la redéfinition et la surcharge 86
4.6 Le polymorphisme 89
4.7 La réflexion 93

5. La généricité 103
5.1 L'utilisation 103
5.2 Les classes génériques 106
5.3 Les interfaces génériques 110
5.4 La généricité et le polymorphisme 111
5.5 Les méthodes génériques 113

6. Le contrôle de l'exécution 115
6.1 Les exceptions 115
6.2 Les processus légers 128

7. Les fonctions anonymes 147
7.1 La déclaration 148
7.2 L'utilisation 152
7.3 La fermeture 157
7.4 Les méthodes par défaut 160

8. Les entrées-sorties 163
8.1 Les différents flots 163
8.2 La construction de flots 164
8.3 Les fichiers 168
8.4 Les fichiers à accès direct 177

8.5 Les sorties formatées 178
8.6 Les fichiers standard 180

9. L'API 185
9.1 Les collections et les tables 185
9.2 Les flots d'éléments 198
9.3 Le réseau 209
9.4 AWT et Swing 222
9.5 Les applets 235

10. JDBC 245
10.1 L'architecture de JDBC 245
10.2 Le contenu de l'API JDBC 247
10.3 Le schéma d'utilisation 256
10.4 Les API autour de JDBC 270

11. L'environnement Java 273
11.1 La distribution officielle 273
11.2 Les outils 277
11.3 Java et la sécurité 290
11.4 Les sites web Java 294

Annexe 295

Bibliographie 297

Index 299

Avant-propos

Cette quatrième édition de l'*Aide-mémoire de Java* intègre les récents enrichissements apportés au langage Java par la version 8. Un nouveau chapitre est consacré aux fonctions anonymes (ou lambda expressions), structures fondamentales de la programmation fonctionnelle, qui font leur entrée à côté du modèle objet, modèle de base du langage. Ce nouveau mécanisme de programmation est largement employé dans l'API, particulièrement pour en simplifier l'usage. La refonte du livre consécutive à cet apport majeur, a aussi été l'occasion d'ajouter une introduction à la réflexion et à son usage en Java. Tous les mécanismes de programmation présents dans Java 8 sont ainsi abordés dans cette nouvelle édition.

Depuis ses origines en 1991, le langage à objets Java connaît un engouement qui ne cesse de croître et une riche communauté de concepteurs et d'utilisateurs s'est développée et consolidée autour de ce langage. Son modèle objet, simple mais néanmoins puissant en fait, aujourd'hui, un des rares langages généralistes à être aussi bien enseigné dans les universités, qu'utilisé dans le monde industriel.

Malgré sa forme réduite, ce livre est une présentation pédagogique et didactique qui couvre les aspects fondamentaux et appliqués du langage. Si l'ensemble des mécanismes de Java est détaillé, une présentation exhaustive des milliers de classes (plus de 4 000) de l'API (*Application Programming Interface*) est impossible. Notre choix, forcément partiel et partial, a été guidé par l'intérêt pratique des classes retenues.

Cet ouvrage a aussi été conçu pour offrir au lecteur une présentation condensée de Java 8. Il s'adresse tout aussi bien aux lecteurs ayant déjà une pratique de la programmation, et désireux d'aborder un nouveau type de langage, qu'aux programmeurs Java ressentant le besoin d'un ouvrage synthétique de la version 8.

La première partie du livre, jusqu'au chapitre 7, expose les éléments et les mécanismes propres au langage. La seconde s'attache à présenter l'environnement de programmation du langage.

Les deux premiers chapitres sont consacrés aux constructions et aux objets de base de Java. Au travers d'un exemple *fil rouge*, les deux suivants décrivent en détail le modèle objet de Java.

La généricité, mécanisme de paramétrage des classes abondamment utilisé dans l'API, est l'objet du chapitre 5. Partie intégrante du langage, les exceptions et les processus légers (*threads*) sont abordés dans le chapitre 6. Enfin, le chapitre 7 présente les fonctions anonymes, grande nouveauté de Java 8.

La seconde partie du livre présente l'API et les outils de développement. Le chapitre 8 donne une présentation détaillée des mécanismes d'entrée-sortie. L'accent a été mis sur la manipulation des fichiers. Le chapitre 9 décrit une sélection de classes de l'API. Tout d'abord, nous y traitons les structures de données, collections, tables et les nouveaux objets *stream*, qui permettent, à l'aide de fonctions anonymes, le traitement efficace de grands flots de données. Puis, nous présentons, les classes pour la communication réseau, les environnements graphiques AWT et Swing, pour achever cette revue de l'API avec les applets. Le chapitre 10 présente une API particulièrement importante, JDBC. Nous y décrivons les éléments essentiels indispensables au maniement des bases de données en nous appuyant sur un exemple *suffisamment complet* pour initier de réels développements dans ce vaste domaine d'application. Le dernier chapitre reste consacré aux outils de développement de la distribution officielle Java SE 8. Il s'achève par un catalogue de sites web consacrés à Java, complété, à la fin de l'ouvrage, par une bibliographie thématique.

Le site de cet ouvrage propose les codes sources de certains exemples du livre. Il est accessible à l'adresse :

`http://go.polytechnice.fr/AideMemoireJava`

Pour terminer, nous souhaitons remercier toute l'équipe Dunod, en particulier Carole Trochu et Jean-Luc Blanc, pour leur aide précieuse, leurs conseils avisés et pour la relecture de cette quatrième édition.

Sophia Antipolis et Tourrette Levens, mai 2015.

1 Java, les bases

Comme l'indique son titre, ce premier chapitre présente les notions de base du langage Java. Après avoir donné, à l'aide d'un exemple, une première description de son modèle objet, nous présenterons les éléments fondamentaux du langage : les types élémentaires, les expressions et les énoncés.

1.1 Une première application

Généralement, les ouvrages d'apprentissage des langages de programmation commencent par la présentation d'une application simple qui donne au lecteur une première impression sur le langage. Nous ne dérogerons pas à cette règle en présentant une première application qui met en évidence plusieurs notions fondamentales du langage à objets Java. Celles-ci sont brièvement décrites dans cette section mais seront, bien sûr, plus amplement détaillées dans les chapitres suivants.

Cette première application, donnée dans la figure 1.1, affiche la date du jour en toutes lettres, comme par exemple *dimanche 28 juin 2015*. Elle nous permettra de mettre en évidence les notions de commentaire, de classe, de méthode et de variable.

Les commentaires

Les commentaires jouent un rôle essentiel dans la compréhension des fonctionnalités d'une application, mais aussi et surtout, dans la vérification de sa validité.

```
/**
 * Cette application écrit la date du jour
 * en toutes lettres (e.g. dimanche 28 juin 2015)
 */
import java.util.Date;
import java.text.SimpleDateFormat;

class DateDuJour {
    public static void main(String[] args) {
        // créer la date du jour
        Date dateDuJour;
        dateDuJour = new Date();
        // créer son modèle de présentation
        SimpleDateFormat fd;
        fd = new SimpleDateFormat("EEEE dd MMMM yyyy");
        // afficher la date du jour selon ce modèle
        System.out.println(fd.format(dateDuJour));
    }
}
```

Figure 1.1 Application DateDuJour

Le langage propose trois formes de commentaires différenciés par leurs parenthéseurs :

1. `/**` *suite quelconque de caractères* `*/`
2. `/*` *suite quelconque de caractères* `*/`
3. `//` *suite quelconque de caractères jusqu'à la fin de la ligne*

La première sert à la documentation des classes et d'autres éléments des applications, et peut être exploitée par le générateur `javadoc`[1]. La documentation produite au format HTML (*HyperText Markup Language*) offre une présentation régulière et lisible à l'aide de n'importe quel navigateur. Les commentaires de documentation peuvent apparaître sur plusieurs lignes et contenir des directives spéciales de mise en pages destinées au générateur `javadoc`.

1. Cf. chapitre 11.

La deuxième et la troisième forme de commentaire sont généralement utilisées pour la description de l'implémentation du code (antécédent, conséquent, invariant, etc.). Notez que dans la troisième forme, le commentaire s'achève à la fin de la ligne, alors que dans la deuxième il peut apparaître sur plusieurs lignes, comme pour les commentaires de documentation.

Les classes

Le langage à objets JAVA est un langage de classes. Cela veut dire que la conception d'une application écrite dans ce langage suit un modèle de structuration autour des objets décrits par des classes qui en précisent les propriétés, variables et méthodes.

Les variables désignent des données qui sont des valeurs de tous types. Les méthodes sont des actions, c'est-à-dire des procédures ou des fonctions. Par défaut, les variables et les méthodes sont liées aux objets, mais si leur déclaration le précise, à l'aide du mot-clé **`static`**, elles sont alors liées à la classe [2].

D'un point de vue statique, celui du compilateur, une application JAVA est une collection de classes qui décrit les objets qui seront manipulés à l'exécution. Notre première application ne comporte qu'une seule classe, nommée `DateDuJour`, écrite dans un fichier qui porte son nom et suffixé par `.java`, c'est-à-dire `DateDuJour.java`. Par convention [3], la lettre initiale de chacun des mots qui forment le nom de la classe est en majuscule.

Toutefois, notre application utilise quatre autres classes, `String`, `System`, `Date` et `SimpleDateFormat`, définies par ailleurs et mises à la disposition du programmeur par l'environnement de programmation JAVA. Ces classes, avec plusieurs centaines d'autres, forment ce que l'on appelle l'API (*Application Programming Interface*) JAVA [4].

---

2. Cf. chapitre 3.

3. Voir l'URL `http://www.oracle.com/technetwork/java/codeconvtoc-136057.html` pour les conventions d'écriture de code JAVA.

4. Cf. chapitre 9.

Les classes unies par une même sémantique sont regroupées dans des *paquetages*[5]. L'API de Java est formée de nombreux paquetages comme par exemple `java.lang`, `java.io` ou encore `java.util`[6]. Sauf pour `java.lang`, le nom d'une classe doit être qualifié par le nom du paquetage auquel elle appartient. Toutefois, la directive d'importation, **`import`**, permet un accès non qualifié aux classes des paquetages. Ainsi dans notre application, la directive d'importation de la classe `Date` appartenant au paquetage `java.util` permet l'utilisation du nom `Date` dans la classe `DateDuJour` sans avoir à le préfixer par `java.util`. Il en va de même pour la classe `SimpleDateFormat` du paquetage `java.text`.

La méthode main

L'exécution d'une application Java commence implicitement par celle d'une méthode statique `main` dont l'en-tête est *toujours* le même. En particulier, le paramètre formel `args` est un tableau de chaînes de caractères qui donne accès aux paramètres du programme lors de son exécution. Même s'il reste inutilisé par la suite (comme dans notre application), ce paramètre *doit* apparaître dans l'en-tête.

Le corps d'une méthode peut contenir des déclarations de variables locales et des instructions. Java est un langage *typé*, ce qui veut dire que les déclarations des noms utilisés dans l'application doivent spécifier les types des objets qu'ils désigneront. En Java, les classes sont assimilées à des types.

La méthode `main` de notre exemple déclare deux variables locales. La première, `dateDuJour`, permet de désigner des objets de la classe `Date`. De tels objets représentent un temps écoulé en millisecondes depuis une date de référence[7]. La seconde variable, `fd`, permet, quant à elle, de désigner des objets de la classe `SimpleDateFormat` qui définit des modèles de présentation des dates.

---

5. Cf. chapitre 3.

6. Le paquetage `java.lang` contient les classes de base du langage, `java.io` des classes pour les entrées-sorties, et `java.util` des classes utilitaires nécessaires à bien des applications.

7. 1er janvier 1970, GMT 00 :00 :00.

Pour créer, on dit *instancier*, les objets que pourront désigner ces deux variables, on utilise l'opérateur **new** qui applique un constructeur. Un constructeur est une sorte de méthode qui porte toujours le nom de la classe dans laquelle il est défini, et dont le rôle est d'initialiser les variables de l'objet au moment de sa création.

Le constructeur `Date()` initialise un objet `Date` à la date du jour, c'est-à-dire le nombre de millisecondes entre la date de référence et le moment de la création de l'objet. L'opérateur d'affectation, le signe =, associe à la variable `dateDuJour` une référence à l'objet créé. La valeur de la variable `dateDuJour` *n'est pas* l'objet créé mais une sorte de pointeur sur cet objet, appelé référence. La constante symbolique **null** est une référence particulière pour signifier l'absence de référence à un objet particulier [8].

De même, la seconde affectation associe à la variable `fd` une référence à un objet `SimpleDateFormat`. Le constructeur utilisé possède comme paramètre une chaîne de caractères (définie par la classe `String`) dont la valeur décrit un modèle de présentation des dates selon la forme : jour (EEEE), numéro du jour (dd), mois (MMMM) et année (yyyy).

Pour accéder aux variables ou aux méthodes d'un objet, on utilise une *notation pointée* qui suit la syntaxe : nom de l'objet, suivi d'un point, suivi du nom de la variable ou de la méthode avec ses paramètres effectifs.

La classe `SimpleDateFormat` possède une méthode `format`. Elle renvoie une chaîne de caractères qui est la représentation d'un objet `Date` passé en paramètre suivant le modèle de présentation des dates de l'objet courant. Appliquée à l'objet désigné par `fd`, cette méthode renvoie la date du jour selon son modèle (`fd.format(dateDuJour)`).

Enfin, l'écriture sur la sortie standard de la chaîne de caractères qui représente la date du jour est faite par l'application de la méthode `println` sur l'objet statique `out` de la classe `System`.

Compiler et exécuter l'application

Pour exécuter une application rédigée en Java, il faut tout d'abord la traduire, à l'aide d'un compilateur, dans une forme intermédiaire appelée

8. C'est l'équivalent des constantes `NULL` et `nil` des langages C et Lisp.

Java Byte Code, placée dans des fichiers suffixés par `.class`. Cette forme est indépendante de l'ordinateur utilisé.

On procède ensuite à son exécution par interprétation avec la *Machine Virtuelle Java*[9] (en anglais, *Java Virtual Machine*, JVM). Dans l'environnement de programmation distribué librement[10] par Oracle, JDK (*Java Development Kit*), le compilateur s'appelle `javac` et l'interprète `java`.

Les commandes suivantes procèdent à la compilation et à l'exécution de l'application contenue dans le fichier `DateDuJour.java`.

```
javac DateDuJour.java
java DateDuJour
```

```
dimanche 28 juin 2015
```

Notez que le suffixe `.class` du fichier `DateDuJour.class` ne doit pas apparaître dans la commande d'exécution de l'interprète Java.

1.2 Les types élémentaires

Un type est un ensemble de valeurs possédant des caractéristiques communes, généralement définies par la nature des opérations qu'on peut leur appliquer, comme celle d'accès à une valeur. En particulier, une valeur d'un type *élémentaire* ne peut être manipulée que dans sa totalité. Cette notion s'oppose à celle de type *structuré*, c'est-à-dire en Java à celle de classe ou de tableau. Le langage propose huit types *prédéfinis élémentaires*, quatre types entiers, deux types réels, un type booléen et un type caractère.

1.2.1 Les types entiers

Le langage Java possède quatre types entiers. Les types **`byte`**, **`short`**, **`int`**, **`long`** sont signés et représentés en complément à deux. Les entiers du type **`byte`** sont représentés sur 8 bits, ceux du type **`short`** sur 16 bits, ceux du type **`int`** sur 32 bits et ceux du type **`long`** sur 64 bits.

9. Cf. chapitre 11.

10. Java est logiciel libre depuis 2006.

Les constantes littérales entières en base 10 sont dénotées par une suite de chiffres compris entre `0` et `9`. Le langage permet également d'exprimer des valeurs en base 8 ou 16, en les faisant précéder, respectivement, par les préfixes `0` et `0x`.

```
3 125 0 0777 3456 234 0xAC12
```

Le tableau 1.1 montre les opérateurs arithmétiques et relationnels qui peuvent être appliqués sur les types entiers.

Opérations	Opérateur	Fonction	Exemple
Arithmétiques	-	opposé	`-45`
	+	addition	`45 + 5`
	-	soustraction	`a - 4`
	*	multiplication	`a * b`
	/	division	`5 / 45`
	%	modulo	`a % 4`
Relationnelles	<	inférieur	`a < b`
	<=	inférieur ou égal	`a <= b`
	==	égal	`a == b`
	!=	différent	`a != b`
	>	supérieur	`a > b`
	>=	supérieur ou égal	`a >= b`

Tableau 1.1 Opérateurs sur les types entiers

1.2.2 Les types réels

Comme pour les entiers, Java définit plusieurs types réels : **`float`** et **`double`**. Les réels du type **`float`** sont en simple précision codés sur 32 bits, ceux du type **`double`** sont en double précision codés sur 64 bits. La représentation des réels suit la norme IEEE 754.

La dénotation d'une constante littérale réelle suit la syntaxe donnée ci-dessous. Les crochets indiquent que l'argument qu'ils entourent est optionnel, la barre verticale indique un choix, et `entier` est un nombre entier en base 10.

```
entier[.][entier][[e|E][±]entier]
[entier][.]entier[[e|E][±]entier]
```

Ainsi, les constantes réelles suivantes sont valides :

```
123.456   34.0   .0   12.   56e34   4E-5 45.67e2   4567
```

Le tableau 1.2 donne les opérateurs arithmétiques et relationnels applicables sur les types réels.

Opérations	Opérateur	Fonction	Exemple
Arithmétiques	-	opposé	`-45.5+5e12`
	+	addition	`45.5+5e12`
	-	soustraction	`a - 4.3`
	*	multiplication	`a * b`
	/	division	`5 / 45`
Relationnelles	<	inférieur	`a < b`
	<=	inférieur ou égal	`a <= b`
	==	égal	`a == b`
	!=	différent	`a != b`
	>	supérieur	`a > b`
	>=	supérieur ou égal	`a >= b`

Tableau 1.2 Opérateurs sur les types réels

1.2.3 Le type booléen

Le type **`boolean`** définit un ensemble à deux valeurs : **`true`** et **`false`**. Les opérateurs applicables sur des opérandes booléens sont donnés par le tableau 1.3.

Les opérateurs de disjonction et de conjonction conditionnelle, `||` et `&&`, sont semblables à ceux du langage C. Le résultat de `p||q` est **`true`** si `p` est vrai quelle que soit la valeur de `q` qui, dans ce cas, n'est pas évaluée. De même, le résultat de `p&&q` est **`false`** si `p` est faux quelle que soit la valeur de `q` qui, dans ce cas, n'est pas évaluée.

Opérateur	Fonction	Exemple
!	négation	`! p`
|	disjonction	`p | q`
^	disjonction exclusive	`p ^ q`
&	conjonction	`p & q`
||	disjonction conditionnelle	`p || q`
&&	conjonction conditionnelle	`p && q`

Tableau 1.3 Opérateurs booléens

1.2.4 Le type caractère

Le type **`char`** utilise le jeu de caractères UNICODE [11]. Les constantes littérales de type caractère sont dénotées entre deux apostrophes. Ainsi, les caractères `'a'` et `'ç'` représentent les lettres alphabétiques a et c cédille. Il est fondamental de comprendre la différence entre les notations `'2'` et `2`. La première représente un caractère et la seconde un entier.

Certains caractères, en particulier ceux non imprimables, possèdent une représentation particulière, préfixée par le caractère `\`. Le tableau 1.4 donne une partie de ces caractères, *i.e.* ceux les plus fréquemment utilisés.

Caractère	Notation
passage à la ligne	`\n`
tabulation	`\t`
retour en arrière	`\r`
saut de page	`\f`
backslash	`\\`
apostrophe	`\'`

Tableau 1.4 Caractères spéciaux

Il existe une relation d'ordre sur le type **`char`**. On peut donc appliquer les opérateurs de relation <, <=, =, !=, > et >= sur des opérandes de type caractère.

11. La version 8 de JAVA utilise la version 6.2.0 de l'UNICODE Standard. Voir le site `www.unicode.org` pour une description complète de ce jeu de caractères.

Les caractères peuvent être dénotés par la valeur hexadécimale de leur ordinal[12], précédée par les caractères `\` et `u`. Par exemple, `'\u0041'` est le caractère d'ordinal 65, c'est-à-dire la lettre A. Cette notation particulière trouve tout son intérêt lorsqu'il s'agit de dénoter des caractères graphiques. Par exemple, les caractères `'\u12cc'` et `'\u1356'` représentent les lettres éthiopiennes ጌ et ፖ, les caractères `'\u2200'` et `'\u2208'` représentent les symboles mathématiques ∀ et ∈, et `'\u20AC'` est le symbole €.

Depuis la version 7 de Java, de nouveaux caractères sont disponibles comme les émoticônes. Ainsi, les caractères `'\u1F60B'` `'\u1F649'` correspondent, respectivement, à 😋 et 🙉.

1.3 Les identificateurs

Un *identificateur* sert à nommer un élément du langage, comme par exemple une variable, une constante symbolique, une méthode, ou encore une classe. Un identificateur est une suite de caractères Unicode. Toutefois, il ne peut débuter par un chiffre ou un caractère réservé du langage (*e.g.* le symbole ?). Afin d'accroître la lisibilité des programmes, les identificateurs doivent posséder une signification qui exprime clairement leur utilisation ultérieure. Il est fortement conseillé d'utiliser les caractères accentués, s'ils sont nécessaires. Il est aussi possible d'utiliser toutes sortes de symboles, mathématiques ou autres.

Java se réserve certains identificateurs, qui ne pourront évidemment pas être choisis par le programmeur. Ces identificateurs sont appelés *mots-clés*. Java possède 48 mots-clés. Ce sont, par exemple, **`class`**, **`if`** ou **`while`**.

```
// cinq identificateurs dont un mot-clé
DateDuJour   agent_007   while   π   toString
```

Si la saisie directe du symbole Unicode π n'est pas possible, il sera toujours permis d'employer sa notation hexadécimale `\u03C0`.

12. Le numéro d'ordre du caractère dans le jeu de caractères.

1.3.1 Les variables

Une variable est un identificateur qui sert à désigner plusieurs valeurs prises dans un type particulier. Nous l'avons vu, JAVA est un langage *typé*, c'est-à-dire qu'il exige en particulier que les variables soient, avant leur utilisation, *déclarées* avec le type des valeurs qu'elles pourront désigner. Dans une déclaration de variable JAVA, le type précède l'identificateur. Notez que l'on déclare plusieurs variables d'un même type en les séparant par des virgules. Les déclarations suivantes sont valides :

```
int âge;
char voyelle, consonne;
double Δ, racRéelle1, racRéelle2;
boolean majeur;
```

Il est important de comprendre que la déclaration d'une variable de type élémentaire a pour effet d'*allouer en mémoire centrale* une zone dont la taille est égale à celle d'une valeur du type de la variable. Cette zone est accessible par le nom de la variable.

Une façon d'associer une valeur à une variable est d'utiliser l'opérateur d'affectation, = (le signe égal[13]). Le type de la valeur affectée à la variable doit être conforme[14] à sa déclaration. Les affectations suivantes sont valides :

```
âge = 10;
voyelle = 'e';
Δ = 13.15;
majeur = âge >= 18;
```

Il est également possible d'initialiser une variable au moment de sa déclaration comme suit :

```
int âge = 21;
char voyelle, consonne = 'z';
// voyelle n'est pas initialisée
```

13. À ne pas confondre avec l'opérateur d'égalité ==.
14. C'est-à-dire de même type ou compatible.

1.3.2 Les constantes

Une déclaration de constante symbolique, avec le mot-clé **`final`**, permet d'établir un lien *définitif* entre un identificateur et une valeur particulière. La valeur d'une constante ne peut donc pas être modifiée. En Java, on déclarera la constante π comme suit :

```
final double π = 3.14159265358979;
```

Notez que dans de nombreux langages de programmation, la déclaration d'une constante n'entraîne aucune allocation mémoire. En Java, déclarer une constante consiste à déclarer une variable (il y a donc allocation de la mémoire) précédée du mot-clé **`final`** qui interdit toute modification de la variable après sa première affectation. Ainsi, la déclaration précédente aurait pu tout aussi bien s'écrire :

```
final double π;
π = 3.14159265358979;
```

1.4 Les expressions

Les *expressions* composent entre eux des opérandes et des opérateurs selon une notation *infixe*. Les opérandes sont par exemple des constantes, des variables, des appels de méthodes ou des expressions. Les opérateurs correspondent à des opérations qui portent sur un, deux ou trois opérandes. La notation infixe place les opérandes de part et d'autre de l'opérateur, comme dans $x + y$.

1.4.1 Évaluation

Le but d'une expression est de calculer, lors de son *évaluation*, un résultat. L'évaluation d'une expression Java produit une valeur unique. Le résultat d'une expression est déterminé par l'ordre d'évaluation des opérandes et des opérateurs qui la composent. En Java, l'ordre habituel d'évaluation est de gauche à droite. Toutefois, cet ordre peut être remis en question lorsque

l'opérateur est associatif à droite (comme l'opérateur d'affectation [15]), par les règles de priorité [16] (les opérateurs les plus prioritaires sont évalués en premier), ou l'utilisation des parenthèses.

Dans le fragement de code suivant, la première expression est simplement évaluée de gauche à droite (addition puis soustraction) ; la seconde est évaluée de droite à gauche (1 est affecté à b, puis b est affecté à a) ; la troisième évalue la somme de a et du produit de b par c. La quatrième expression remet en cause la priorité des opérateurs grâce aux parenthèses (somme, puis produit).

```
a + b - c   // évaluation de gauche à droite
a = b = 1   // évaluation de droite à gauche
a + b * c   // évaluation de la somme de a et (b*c)
(a + b) * c // évaluation de (a + b) d'abord
```

1.4.2 Type d'une expression

Le résultat de l'évaluation d'une expression JAVA est typé. Ainsi, si `p` et `q` sont deux booléens, l'expression `p && q` est une expression booléenne puisqu'elle produit un résultat booléen. Une expression peut très bien être formée à partir d'opérateurs manipulant des objets de types différents. Par exemple, `(x != 0) && (y != 0)` est formée des deux opérandes booléens de l'opérateur de conjonction eux-mêmes composés à partir de deux opérandes numériques reliés par des opérateurs de relation à résultat booléen.

1.4.3 Conversion de type

Les valeurs, mais pas toutes, peuvent être converties d'un type vers un autre. On distingue deux types de conversion. Les conversions *implicites* pour lesquelles l'opérateur décide de la conversion à faire en fonction de

15. En JAVA, une affectation est donc une expression.

16. Les règles de priorité des opérateurs JAVA sont données en annexe à la fin de l'ouvrage.

la nature de l'opérande ; les conversions *explicites*, pour lesquelles le programmeur est responsable de la mise en œuvre de la conversion.

En Java, les conversions de type *implicites* sont relativement nombreuses, en partie dues à l'existence de plusieurs types entiers et réels. Ces conversions, appelées également *promotions*, transforment implicitement un objet de type T en un objet d'un type T', lorsque le contexte l'exige. Par exemple, l'évaluation de l'expression $1 + 4.76$ provoquera la conversion implicite de l'entier 1 en un réel double précision 1.0, suivie d'une addition réelle. Dans `int x = 'a'`, l'ordinal du caractère `'a'` est affecté à la variable entière `x`. Les promotions possibles sont données par le tableau 1.5.

Type	Promotion
`float`	`double`
`long`	`float` ou `double`
`int`	`long`, `float` ou `double`
`short`	`int`, `long`, `float` ou `double`
`char`	`int`, `long`, `float` ou `double`

Tableau 1.5 Promotions de type

Les conversions *explicites*, appelées cast en anglais, sont faites à l'aide d'un opérateur de conversion. Sa dénotation consiste à placer entre parenthèses, devant l'expression à convertir, le type dans lequel elle doit être convertie. Dans l'exemple suivant, le réel 1.6 est converti explicitement en un entier 1 (suppression de la partie décimale).

```
(int) 1.6 // est l'entier 1
```

1.5 Les énoncés

Comme de nombreux langages, Java propose des énoncés simples, composés, conditionnels et itératifs. Ces énoncés sont assez semblables à ceux du langage C, mais comportent toutefois quelques petites différences.

JAVA propose également depuis sa version 1.4, un énoncé pour traiter les assertions.

Dans les descriptions syntaxiques qui suivront, E représentera indifféremment un bloc ou un énoncé simple, conditionnel ou itératif, e une expression et B une expression booléenne.

1.5.1 Énoncés simples

L'énoncé le plus simple est l'*affectation* terminée par un point-virgule. Notez qu'une affectation est une expression, qui fournit donc un résultat. Dans les énoncés simples suivants, le résultat de son évaluation n'est pas utilisé.

```
âge = 26;
consonne = 'z';
Δ = b*b−4*a*c;
présent = true;
```

Dans l'énoncé suivant, l'évaluation de l'affectation de l'entier 5 à `b`, fournit le résultat 5, qui est converti en 5.0 puis affecté à `a`.

```
double a; int b;
a = b = 5;
// a = 5.0 et b = 5
```

JAVA propose aussi des affectations composées de la forme : `a` *op*`= b` équivalentes à `a = a` *op* `b`. L'opérateur *op* peut être pris parmi les suivants : `+ − * / % & ^ | << >> >>>` [17].

```
âge += 5;    // est équivalent à âge = âge + 5;
x += y*3;   // est équivalent à x = x + (y*3);
q |= true;  // est équivalent à q = q | true;
```

Enfin, les opérateurs d'affectation ++ et −− permettent une incrémentation ou une décrémentation de 1.

17. Ces trois derniers opérateurs sont les opérateurs de décalage binaire.

```
âge++;  // ou
++âge;  // est équivalent à âge = âge + 1;
```

```
âge--;  // ou
--âge;  // est équivalent à âge = âge - 1;
```

L'évaluation des opérateurs ++ et -- donne un résultat qui peut être utilisé par ailleurs.

```
// âge = 20
x = âge++;
// âge = 21 et x = 20
```

Le résultat de l'évaluation de `âge++` est la valeur de `âge` avant l'incrémentation, c'est donc 20 qui est affecté à `x` dans l'exemple précédent. Avec l'opérateur de décrémentation, on obtient :

```
// âge = 20
x = âge--;
// âge = 19 et x = 20
```

Les deux opérateurs peuvent aussi être placés avant leur opérande, `++âge` ou `--âge`. Le résultat de l'évaluation est alors la valeur de `âge` après l'incrémentation (ou la décrémentation).

```
// âge = 20
x = ++âge;
// âge = 21 et x = 21

// âge = 20
x = --âge;
// âge = 19 et x = 19
```

1.5.2 Bloc

Un bloc est un groupement d'énoncés et de déclarations placés entre accolades. La suite d'énoncés peut être alors considérée comme formant un énoncé unique.

```
// x est défini
{
  x = x+1;
  double y= Math.sin(1.34);
  // y est défini
  y *= x;
}
// y n'existe plus
```

Les noms déclarés dans un bloc (*i.e.* `y` dans l'exemple précédent) sont locaux et n'ont pas d'existence en dehors du bloc. Il est évidemment possible de définir des blocs emboîtés dans d'autres blocs. Notez aussi que le corps d'une méthode [18] est défini par un bloc.

L'exemple précédent utilise la méthode statique `sin` qui appartient à la classe `java.lang.Math`. Cette classe possède les définitions des constantes `PI` et `E`, ainsi que de nombreuses autres fonctions mathématiques, comme par exemple la méthode `sqrt` pour le calcul d'une racine carrée que nous utiliserons plus loin.

1.5.3 L'énoncé si-alors-sinon

L'énoncé *si-alors-sinon* de JAVA est semblable à celui de la plupart des langages de programmation. Il permet d'exécuter ou non un énoncé selon une condition à valeur booléenne. Sa forme générale est :

```
if (B) E1 else E2
```

Si B est vrai, l'énoncé E_1 est exécuté, sinon c'est l'énoncé E_2. Notez l'absence de mot-clé *alors* et les parenthèses obligatoires autour de l'expression booléenne.

18. Cf. chapitre 2.

L'exemple suivant affecte à une variable `max` le maximum de deux valeurs :

```
if (a>b) max = a; else max = b;
```

Par ailleurs, si la partie *alors* ou la partie *sinon* comportent plusieurs énoncés, il faudra les regrouper dans un bloc.

```
// résolution de l'équation du second degré
// ax²+bx+c=0, avec x≠0
Δ = (b*b)-4*a*c;
if (Δ>=0) { //calcul des racines réelles
   r1 = (-b+Math.sqrt(Δ))/(2*a);
   r2 = (-b-Math.sqrt(Δ))/(2*a);
}
else { //calcul des racines complexes
   r1 = r2 = -b/(2*a);
   i1 = Math.sqrt(-Δ)/(2*a);
   i2 = -i1;
}
```

Enfin, comme dans l'exemple suivant, il est possible d'omettre la partie *sinon* de l'énoncé conditionnel.

```
// x un entier quelconque
if (x<0) x = -x;
// x un entier positif
```

Java propose également un opérateur ternaire qui définit une *expression* conditionnelle. Sa forme générale est :

```
B ? e1 : e2
```

Cette expression a pour valeur le résultat de l'évaluation de l'*expression* e_1 si B est vrai, sinon c'est celui de l'*expression* e_2.

Dans certains cas, le remplacement de l'énoncé *si-alors-sinon* par cette expression permet d'alléger l'écriture de l'application. Pour le calcul du maximum, on préférera écrire :