4. Les variables

Une variable a donc un nom et un type. Il existe plusieurs types prédéfinis en Java. Par exemple, le type `int` que l'on a déjà rencontré permet de représenter une valeur sur 4 octets, c'est-à-dire 32 bits. On peut donc représenter des entiers de -2^{31} à 2^{31}-1, c'est-à-dire de −32 768 à 32 767. Voici ces différents types.

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	'\u0000' → '\uffff' (0 à 65535)
byte	Entier très court	1	-128 → +127
short	Entier court	2	-32 768 → +32 767
int	Entier	4	$-2^{31} \approx -2{,}147 \times 10^{9}$ → $+2^{31}-1 \approx 2{,}147 \times 10^{9}$
long	Entier long	8	$-2^{63} \approx -9{,}223 \times 10^{18}$ → $+2^{63}-1 \approx 9{,}223 \times 10^{18}$
float	Nombre réel simple	4	$\pm 2^{-149} \approx 1.4 \times 10^{-45}$ → $\pm 2^{128}-2^{104} \approx 3.4 \times 10^{38}$
double	Nombre réel double	8	$\pm 2^{-1074} \approx 4{,}9 \times 10^{-324}$ → $\pm 2^{1024}-2^{971} \approx 1{,}8 \times 10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Par exemple, un `short`, codé sur 16 bits, permet de représenter des entiers de -2^{15} à 2^{15}-1, c'est-à-dire de −32 767 à 32 768.

Les nombres de type *float* sont codés sur 32 bits dont 23 bits pour la mantisse, 8 bits pour l'exposant et 1 bit pour le signe. Les nombres de type double sont codés sur 64 bits dont 52 bits pour la mantisse, 11 bits pour l'exposant et 1 bit pour le signe. Le plus grand float représentable est donc 1,11111111111111111111111 x 2^{126} $\simeq$ 2 x 2^{126} = 2^{127}. Pour estimer l'ordre de grandeur de ce nombre en base 10, on peut utiliser le fait que $2^{10} \simeq 10^{3}$. Par une règle de trois, on peut donc dire que $2^{128} \simeq 10^{128*3/10} \simeq 10^{38}$. Il ne faut donc pas utiliser un *float* si l'on veut représenter des nombres plus grands que 10^{38}. On utilisera plutôt un *double*.

Il existe également deux types particuliers : char qui permet de représenter un caractère et boolean qui permet de représenter une valeur logique, vrai ou fausse. Nous reviendrons plus tard sur ces deux types.

N'oubliez pas qu'une variable doit être déclarée avant d'être utilisée. On indique son type, puis son nom. Par exemple :

```
float maNoteDeMaths;
```

Donner une valeur à une variable s'appelle une affectation. Elle s'écrit avec le signe =, qui sépare la variable, à gauche, de la valeur qu'on lui donne, à droite. Voici des exemples :

affectation

```
anneeDeNaissance=1998;
age=2016-anneeDeNaissance ;
espace=' ' ;
```

On peut également affecter une valeur dès la déclaration de la variable. Par exemple :

```
int test = 5 ;
```

 Attention, l'affectation n'est pas une égalité. Il est donc possible d'écrire :

```
a=a+1 ;
```

ce qui arrive très souvent quand on veut augmenter de 1 la valeur d'une variable. C'est faux en mathématiques, mais possible en informatique. Encore une fois, ce n'est pas une égalité. Donc cette expression n'a pas de sens ;

```
b+3=7 ;    // erreur à la compilation
```

Les opérateurs arithmétiques utilisables sont notamment : +, -, * , / et %. Les premiers correspondent aux quatre opérateurs classiques, le dernier est le modulo, c'est-à-dire le reste de la division entière.

opérateurs

Le type d'une expression mathématique dépend du type des variables utilisées. Par exemple, a/b est un entier si a et b sont entiers, mais un réel si a ou b sont des réels.

```
int a=7 ;
int b=3 ;
System.out.println(a / b) ;
```

affichera 2 (la division entière de 7 par 3). En revanche,

```
int a=7 ;
float b=3 ;
System.out.println(a / b ) ;
```

affichera 2.33333 puisque l'un des deux est un réel.

Si on veut mettre ce résultat dans une variable entière, cela provoquera une erreur :

```
int a=7;
float b=3;
int res;
res=a / b;    // provoque une erreur à la compilation
```

Java peut en effet transformer un entier en réel, mais par le contraire puisque cela peut amener à une perte d'informations. Il refuse donc d'exécuter le programme.

Si l'on tient absolument à transformer un type avec le risque de perdre des informations, il faut l'indiquer explicitement en précisant, juste avant, le nouveau type de la variable entre parenthèses. Cette opération s'appelle un *cast*. Par exemple, dans le programme suivant, on a transformé un float en int. L'opération est donc une division entière.

cast

```
int a=7;
float b=3;
int res;
res=a / (int)b;    // pas d'erreur
```

Une variable de type char permet de stocker un caractère (et un seul!). Par exemple :

```
char c ;
c='M' ;
```

Il existe un autre type utile qui permet de stocker une chaîne de caractères, c'est-à-dire une suite, éventuellement vide, de caractères : `"BONJOUR"`, `"il pleut"`, `"123"` (à ne pas confondre avec l'entier 123) ou `""` (la chaîne vide). Ce type s'appelle `String`. On peut concaténer deux chaînes avec l'opérateur +. On utilise nextLine() pour lire une variable de type String. Voici un exemple :

```
String nom,chaine = "Bonjour ";
Scanner s=new Scanner(System.in) ;
System.out.println("Quel est votre nom ?") ;
nom = s.nextLine() ;
System.out.println(chaine + nom) ;
```

5. Les tests

Souvent, les instructions ne sont pas toutes exécutées systématiquement. On a parfois besoin dans certaines conditions d'exécuter seulement quelques instructions. Pour cela on utilise une instruction conditionnelle. La syntaxe est la suivante :

if CONDITION {

    INSTRUCTIONS

}

Par exemple :

```
int age, nbPoints ;
Scanner s=new Scanner(System.in) ;
System.out.println("Quel est votre âge ?") ;
age=s.nextInt() ;
if (age < 20) {
       System.out.println("Quel gamin !!!") ;
       nbPoints=100 ;
}
```

Remarquez l'indentation : les instructions sont décalées vers la droite. Lorsqu'il n'y a qu'une instruction dans le if, on peut omettre les accolades.

Une condition est une expression qui doit être soit vraie, soit fausse. Ce n'est donc pas une instruction ! Par exemple, a<10 est une condition et non une instruction. Mais b=6 est une instruction et non une condition. En effet, = est le symbole de l'affectation. Pour signifier l'égalité, on doit utiliser l'opérateur ==. Par exemple :

l'opérateur ==

```
if (age == 18) {
	System.out.println("Vous êtes majeur") ;
}
```

On peut aussi utiliser les opérateurs de comparaison sur les types simples indiqués plus haut : `<`, `<=`, `>`, `>=` et `!=`, ce dernier signifiant différent.

l'opérateur !=

Pour les chaînes de caractères de type String qui, rappelons-le, n'est pas un type simple, on ne peut pas utiliser `==`. Il faut utiliser `equals` pour les comparer :

```
 if (ch1.equals(ch2))…
```

On peut combiner des conditions, avec des opérateurs logiques comme ET, OU et NON. Le OU s'écrit `||`, le ET s'écrit `&&` et le NON s'écrit `!`. On peut donc écrire :

&& et ||

```
if ((age < 18) || (age > 65)) {
	System.out.println("Vous avez droit à une réduction dans le tram") ;
}
```

Nous reviendrons plus tard sur ces opérateurs logiques.

Si on a besoin d'écrire des instructions pour la condition inverse, on peut aussi compléter l'if avec un else. La syntaxe est la suivante :

if CONDITION {

	INSTRUCTIONS

}

else {

	INSTRUCTIONS

}

L'un des deux blocs d'instruction sera forcément exécuté, soit le premier, soit le second, mais pas les deux à la fois ! Par exemple :

```
if (anneeDeNaissance < 2001) {
	System.out.println("Vous êtes né au XXe siècle!") ;
}
else {
	System.out.println("Vous êtes né au XXIe siècle!") ;
}
```

Il existe une version simplifiée lorsque le bloc if (ou le bloc else) ne comporte qu'une seule instruction : les accolades peuvent être omises. L'exemple précédent peut donc s'écrire :

```
if (anneeDeNaissance < 2001)
      System.out.println("Vous êtes né au XXe siècle!") ;
else
      System.out.println("Vous êtes né au XXIe siècle!") ;
```

Attention, dès qu'il y a plus de deux instructions, il faut mettre des accolades, sinon la signification du programme change. Il faut bien penser que Java ne tient pas compte des indentations. Par exemple, si vous ajoutez une instruction au bloc else de l'exemple précédent, sans placer des accolades, vous obtenez ceci :

```
if (anneeDeNaissance < 2001)
      System.out.println("Vous êtes né au XXe siècle!") ;
else
      System.out.println("Vous êtes né au XXIe siècle!") ;
      System.out.println("Et vous êtes très jeune !") ;
```

Ce programme va afficher « Et vous êtes très jeune ! » dans tous les cas ! En effet, pour Java, c'est un programme identique à celui-ci :

```
if (anneeDeNaissance < 2001)
      System.out.println("Vous êtes né au XXe siècle!") ;
else
      System.out.println("Vous êtes né au XXIe siècle!") ;
System.out.println("Et vous êtes très jeune !") ;
```

Il faut donc écrire comme ceci :

```
if (anneeDeNaissance < 2001)
      System.out.println("Vous êtes né au XXe siècle!") ;
else {
      System.out.println("Vous êtes né au XXIe siècle!") ;
      System.out.println("Et vous êtes très jeune !") ;

}
```

 Il n'y a pas de point-virgule à la suite de la condition puisque cela reviendrait à placer un point-virgule au milieu de l'instruction. L'instruction if ne se termine qu'à la dernière instruction du bloc if ou du bloc else. Cette écriture est donc fausse :

```
if (a < 10) ;
      System.out.println("a est inférieur à 10") ;
```

Elle est interprétée par Java comme ceci :

```
if (a < 10) ;
System.out.println("a est inférieur à 10") ;
```

Si a est plus petit que 10, ne rien faire (instruction vide) et afficher le message dans tous les cas...

If imbriqués

Les instructions dans un if ou dans un else peuvent être n'importe quelle instruction java. En particulier, on peut avoir une instruction if dans un if ! Par exemple, si on veut distinguer 3 cas, A, B, C, un simple if-else ne suffira pas puisqu'il ne permet de séparer que 2 cas. On pourra donc écrire :

```
if (condition pour le cas A) {
	cas A
else  {
	if (condition pour le cas B) {
		cas B
	}
	else {
		if (condition pour le cas C) {
			cas C
		}
	}
}
```

Si l'on est absolument certain que l'on ne peut être que dans l'un des 3 cas, le dernier if est inutile et on pourra écrire :

```
if (condition pour le cas A) {
	cas A
}
else  {
	if (condition pour le cas B) {
		cas B
	}
	else {
		cas C
	}
}
```

C'est par exemple le cas de ce programme :

```
if (moyenne < 10) {
	System.out.println("ECHEC") ;
}
else {
	if ((moyenne >= 10) && (moyenne < 12)) {
		System.out.println("PAS DE MENTION") ;
	}
	else {
		if ((moyenne >= 12) && (moyenne < 14)) {
			System.out.println("MENTION AB") ;
		}
		else {
			if (moyenne >= 14) && (moyenne < 16)) {
				System.out.println("MENTION B") ;
			}
			else {
				System.out.println("MENTION TB") ;
			}
		}
	}
}
```

Comme il n'y a qu'une seule instruction if dans chaque bloc else, on peut simplifier un peu le programme en omettant les accolades. On obtient alors ceci (notez l'indentation différente).

```
if (moyenne < 10)
       System.out.println("ECHEC") ;
else if ((moyenne >= 10) && (moyenne < 12))
       System.out.println("PAS DE MENTION") ;
else if ((moyenne >= 12) && (moyenne < 14))
       System.out.println("MENTION AB") ;
else if (moyenne >= 14) && (moyenne < 16))
       System.out.println("MENTION B") ;
else
       System.out.println("MENTION TB") ;
```

La deuxième condition, `((moyenne >= 10) && (moyenne < 12))`, peut être simplifiée parce que, au moment où elle est évaluée, on sait que `moyenne >=10`, sinon la première condition aurait été vraie. On obtient donc :

```
if (moyenne < 10)
       System.out.println("ECHEC") ;
else if (moyenne < 12)
       System.out.println("PAS DE MENTION") ;
else if (moyenne < 14)
       System.out.println("MENTION AB") ;
else if (moyenne < 16)
       System.out.println("MENTION B") ;
else
       System.out.println("MENTION TB") ;
```

On obtient ainsi un schéma plus simple d'imbrications de if :

```
if CONDITION 1 {
        instructions
}
else if CONDITION 2 {
        instructions
}
else if CONDITION 3 {
....

else {
        instructions
}
```

Voici par exemple un programme qui indique le nombre de jours que possède un mois donné d'une année non bissextile. Il y a trois cas possibles : 28 jours, 30 jours et 31 jours. Mais comme il est possible que l'utilisateur se trompe et donne un numéro de mois en dehors de l'intervalle [1,12], on décrit les 3 conditions et on ajoute un else supplémentaire pour traiter ces autres valeurs. Remarquez le moyen de traiter le cas d'un numéro de mois erroné et le test supplémentaire à la fin.

```
import java.util.Scanner;
public class NbJoursDuMois {
    public static void main(String[] args) {
        int mois,nbJours;
        Scanner s = new Scanner(System.in);
        System.out.println("Affiche le nombre de jours d'un mois donné");
        System.out.println("Quel est le numero du mois ?");
        mois=s.nextInt();
        if (mois==2)
            nbJours=28;
        else if (mois== 1||mois==3||mois==5||mois==7||mois==8||mois==10||mois ==12)
            nbJours=31;
        else if (mois== 4||mois == 6||mois ==9||mois ==11)
            nbJours=30;
        else {
            System.out.println("Vous devez entrer un numéro entre 1 et 12");
            nbJours = 0;
        }
        if (nbJours != 0)
            System.out.println("Le mois "+mois+" a "+nbJours+" jours.");
    }
}
```

Le type booléen

Le type `boolean` permet de stocker une valeur logique, vraie ou fausse. Une variable booléenne ne peut donc prendre que deux valeurs (`true` ou `false`). On peut donc écrire :

boolean

```
boolean b ;
b=(x<3) ;
```

ce qui est équivalent à :

```
boolean b ;
if (x<3)
   b=true;
else
   b=false;
```

Comme les variables booléennes représentent des valeurs logiques, on peut les utiliser dans les tests :

```
if (b) {...
```

ou

```
if (b && (c<23)) {...
```

Quelques éléments de logique propositionnelle

Pour pouvoir écrire des conditions complexes, il faut maîtriser la combinaison des opérateurs logiques. Cette partie a pour but de vous y entraîner.

La logique propositionnelle est la partie de la logique qui traite des propositions, c'est-à-dire des assertions qui ne peuvent être que vraies ou fausses. On l'oppose à la

logique des prédicats dans lequel on a des assertions avec variables.

Comme chaque proposition n'a que deux valeurs de vérité possibles (V/F, 0/1, T/⊥, ...), un ensemble de n propositions ne donnera lieu qu'à 2^n situations différentes. Par exemple, avec les trois propositions « il pleut », « c'est un dimanche », « mon ordinateur est en panne », on aura les 8 situations suivantes :

1. il pleut, c'est un dimanche et mon ordinateur est en panne
2. il pleut, c'est un dimanche et mon ordinateur n'est pas en panne
3. il pleut, ce n'est pas un dimanche et mon ordinateur est en panne
4. il pleut, ce n'est pas un dimanche et mon ordinateur n'est pas en panne
5. il ne pleut pas, c'est un dimanche et mon ordinateur est en panne
6. il ne pleut pas, c'est un dimanche et mon ordinateur n'est pas en panne
7. il ne pleut pas, ce n'est pas un dimanche et mon ordinateur est en panne
8. il ne pleut pas, ce n'est pas un dimanche et mon ordinateur n'est pas en panne

table de vérité

On peut chercher parmi ces situations celles qui vérifient une combinaison de propositions avec des connecteurs particuliers. Par exemple, celles correspondant à « il pleut ou ce n'est pas un dimanche » sont 1,2,3,4,7,8. On peut pour cela écrire une table de vérité qui donnera une valeur de vérité à cette combinaison de propositions.

p (il pleut)	*d (dimanche)*	*e (en panne)*	*p ou (non d)*
V	V	V	V
V	V	F	V
V	F	V	V
V	F	F	V
F	V	V	F
F	V	F	F
F	F	V	V
F	F	F	V

Les connecteurs usuels sont le **et** (∧, appelé aussi conjonction), le **ou** (∨, appelé aussi disjonction), le **non** (¬, c'est la négation). On utilise aussi parfois l'implication (⇒). Voici leurs tables de vérité :

p	**q**	**p ∧ q**	**p∨q**	**¬p**	**p ⇒ q**
V	V	V	V	F	V
V	F	F	V	F	F
F	V	F	V	V	V
F	F	F	F	V	V

p ⇒ q est équivalent à ¬p∨q, ce qui n'est pas forcément intuitif!
p ⇒ q ≡ ¬q ⇒ ¬p (appelée **contraposition**)

Exercice. Écrire les tables de vérité des opérateurs suivants et indiquer leur signification
$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$
$p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$
$p|q \equiv \neg p \wedge \neg q$

Le ou exclusif (XOR) est :

p	q	$p \oplus q$
V	V	F
V	F	V
F	V	V
F	F	F

L'équivalence est $p \Leftrightarrow q$:

p	q	$p \Leftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

Une **tautologie** est une proposition toujours vraie, quelle que soit les valeurs de vérité des propositions qui la constituent. Par exemple $p \Rightarrow p$ ou encore $p \vee \neg p$.

Une **contradiction** est le contraire, une proposition toujours fausse. Par exemple : $p \wedge \neg p$ ou encore $\neg p \wedge \neg(p \Rightarrow q)$.

Les lois de Morgan permettent de passer d'une disjonction à une conjonction et réciproquement.

lois de Morgan

$\neg(p \wedge q) \equiv (\neg p \vee \neg q)$
$\neg(p \vee q) \equiv (\neg p \wedge \neg q)$

Ces deux lignes de code Java sont donc équivalentes :

```
while (!(i>N || t[i]==0))
while (i<N && t[i]!=0)
```

C'est très utile de savoir passer d'une écriture à l'autre.

Prenons un autre exemple. Une année est bissextile si elle est multiple de 4, les années séculaires n'étant pas bissextiles, sauf si elle sont multiples de 400. Ceci s'écrit donc en logique :

$$(mult4 \wedge \neg mult100) \vee mult400$$

non bissextile s'écrit donc :

$$\neg((mult4 \wedge \neg mult100) \vee mult400) \equiv (\neg mult4 \vee mult100) \wedge \neg mult400$$

Deux autres règles sont utiles pour modifier des formules :

$(p \wedge q) \vee r \equiv (p \vee r) \wedge (q \vee r)$
$(p \vee q) \wedge r \equiv (p \wedge r) \vee (q \wedge r)$

Exercice. Vérifier que ces formules sont des tautologies :

$p \Rightarrow (q \Rightarrow p)$
$(\neg p \Rightarrow p) \Rightarrow p$
$((p \Rightarrow q) \wedge p) \Rightarrow q$
$(p \Rightarrow q) \vee (r \Rightarrow p)$
$(p \Rightarrow q) \Rightarrow ((q \Rightarrow r) \Rightarrow (p \Rightarrow r))$