variables & opérateurs

Les différents types de variables :

Une déclaration de variable se fait comme ceci :

```
1 <Type de la variable> <Nom de la variable> ;
```

Cette opération se termine toujours par un point-virgule ***";"*** (comme toutes les instructions).

Ensuite, on l'initialise en entrant une valeur.

En Java, nous avons deux types de variables :

des variables de type simple ou *"primitif"* ;

des variables de type complexe ou des *"objets"* .

Ce qu'on appelle des types simples ou types primitifs, en Java, sont des nombres entiers, des nombres réels, des booléens ou encore des caractères.

Les variables de type numérique

Le type ***byte*** (1 octet) peut contenir les entiers entre -128 et +127.

```
byte temperature;
temperature = 64;
```

Le type ***short*** (2 octets) contient les entiers compris entre -32768 et +32767.

```
short vitesseMax;
vitesseMax = 32000;
```

Le type ***int*** (4 octets) va de -2*109 à 2*109 (soit 2 avec 9 zéros derrière) .

```
int temperatureSoleil;
temperatureSoleil = 15600000;
```

Le type long (8 octets)
peut aller de -9×10^{18} à 9×10^{18} .

```
long anneeLumiere;
anneeLumiere = 9460700000000000L;
```

Afin d'informer la JVM que le type utilisé est long, vous DEVEZ ajouter un "L" à la fin de votre nombre, sinon le compilateur essaiera d'allouer ce dernier dans une taille d'espace mémoire de type entier et votre code ne compilera pas si votre nombre est trop grand...

Le type ***float*** (4 octets) est utilisé pour les nombres avec une virgule flottante.

*Pas une virgule, mais un point ! Même si le nombre en question est rond, on écrit " **.0** " derrière celui-ci, le tout suivi de "**f** " .*

```
1 float pi;
2 pi = 3.141592653f;
```

```
1 float nombre;
2 nombre = 2.0f;
```

Le type ***double*** (8 octets) est identique à float, si ce n'est qu'il contient plus de chiffres derrière la virgule et pour qui le suffixe est optionnel.

```
1 double division;
2 division = 0.33333333333333333333333333333333333333333333334d;
```

*Vous pouvez utiliser la lettre "**d**" pour parfaire la déclaration de votre variable. Mais contrairement au float c'est optionnel pour le double, il n'y aura pas d'erreur de compilation*

Le type ***char*** contient un caractère stocké entre apostrophes ' '

```
1  char caractere;
2  caractere = 'A';
```

Le type ***boolean*** , lui, ne peut contenir que deux valeurs:

true (vrai) ou false (faux),

sans guillemets (ces valeurs sont natives dans le langage)

```
1  boolean question;
2  question = true;
```

Le type ***String*** permet de gérer les chaînes de caractères, c'est-à-dire le stockage de texte.

Il s'agit d'une variable d'un type plus complexe que l'on appelle objet. Vous verrez que celle-ci s'utilise un peu différemment des variables précédentes :

```
1  //Première méthode de déclaration
2  String phrase;
3  phrase = "Titi et Grosminet";
4
5  //Deuxième méthode de déclaration
6  String str = new String();
7  str = "Une autre chaîne de caractères";
8
9  //Troisième méthode de déclaration
10 String string = "Une autre chaîne";
11
12 //Quatrième méthode de déclaration
13 String chaine = new String("Et une de plus !");
```

Attention: *String* commence par une majuscule! Et lors de l'initialisation, on utilise des guillemets doubles (" ") .

String n'est pas un type de variable, mais un objet. Notre variable est un objet, on parle aussi d'une instance : ici, une instance de la classe String. Pour faire simple, le fait que cela soit un objet permet de lui associer des méthodes comme equals, valueOf() etc.

Nomenclature

--tous vos noms de classes doivent commencer par une majuscule

--tous vos noms de variables doivent commencer par une minuscule

--si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation

--tout ceci sans accentuation

```
1 int nbre1 = 2, nbre2 = 3, nbre3 = 0;
```

Veillez à bien respecter la casse (majuscules et minuscules), car une déclaration de CHAR à la place de char ou autre chose provoquera une erreur, tout comme une variable de type string à la place de String!

Faites donc bien attention lors de vos déclarations de variables

```
1 public class Toto{}
2 public class Nombre{}
3 public class TotoEtTiti{}
4 String chaine;
5 String chaineDeCaracteres;
6 int nombre;
7 int nombrePlusGrand;
```

Les opérateurs arithmétiques :

Les opérateurs arithmétiques sont ceux que l'on apprend à l'école primaire… ou presque :

+ : permet d'additionner deux variables numériques (mais aussi de concaténer des chaînes de caractères).

- : permet de soustraire deux variables numériques.

***** : permet de multiplier deux variables numériques.

/ : permet de diviser deux variables numériques.

% : permet de renvoyer le reste de la division entière de deux variables de type numérique ; cet opérateur s'appelle le modulo.

Quelques exemples de calcul :

```
1  int nbre1, nbre2, nbre3;  //Déclaration des variables
2
3  nbre1 = 1 + 3;            //nbre1 vaut 4
4  nbre2 = 2 * 6;            //nbre2 vaut 12
5  nbre3 = nbre2 / nbre1;    //nbre3 vaut 3
6  nbre1 = 5 % 2;            //nbre1 vaut 1, car 5 = 2 * 2 + 1
7  nbre2 = 99 % 8;           //nbre2 vaut 3, car 99 = 8 * 12 + 3
8  nbre3 = 6 % 3;            //là, nbre3 vaut 0, car il n'y a pas de reste
```

Autres exemples de manipulations des nombres:

```
1  int nbre1, nbre2, nbre3;    //Déclaration des variables
2  nbre1 = nbre2 = nbre3 = 0;  //Initialisation
3
4  nbre1 = nbre1 + 1;      //nbre1 = lui-même, donc 0 + 1 => nbre1 = 1
5  nbre1 = nbre1 + 1;      //nbre1 = 1 (cf. ci-dessus), maintenant, nbre1 = 1 + 1 = 2
6  nbre2 = nbre1;          //nbre2 = nbre1 = 2
7  nbre2 = nbre2 * 2;      //nbre2 = 2 => nbre2 = 2 * 2 = 4
8  nbre3 = nbre2;          //nbre3 = nbre2 = 4
9  nbre3 = nbre3 / nbre3; //nbre3 = 4 / 4 = 1
10 nbre1 = nbre3;          //nbre1 = nbre3 = 1
11 nbre1 = nbre1 - 1;      //nbre1 = 1 - 1 = 0
```

On peut simplifier l'écriture du code:

```
1 nbre1 = nbre1 + 1;
2 nbre1 += 1;
3 nbre1++;
4 ++nbre1;
```

Les trois premières syntaxes correspondent exactement à la même opération. La troisième sera certainement celle que vous utiliserez le plus, mais elle ne fonctionne que pour augmenter d'une unité la valeur de nbre 1! Si vous voulez augmenter de 2 la valeur d'une variable, utilisez les deux syntaxes précédentes. On appelle cela l'incrémentation. La dernière fait la même chose que la troisième, mais il y a une subtilité. Elle permet de changer la priorité de l'incrémentation par rapport à une condition par exemple.

Pour la soustraction, la syntaxe est identique :

```
1 nbre1 = nbre1 - 1;
2 nbre1 -= 1;
3 nbre1--;
4 --nbre1;
```

Même commentaire que pour l'addition, sauf qu'ici, la troisième syntaxe s'appelle la décrémentation.

Les raccourcis pour la multiplication fonctionnent de la même manière :

```
1  nbre1 = nbre1 * 2;
2  nbre1 *= 2;
3  nbre1 = nbre1 / 2;
4  nbre1 /= 2;
```

Très important : on NE peut faire du traitement arithmétique que sur des variables de même type sous peine de perdre de la précision lors du calcul. On ne s'amuse pas à diviser un int par un float! Ceci est valable pour tous les opérateurs arithmétiques et pour tous les types de variables numériques.

Sachez aussi que vous pouvez tout à fait mettre des opérations dans un affichage, comme ceci :System.out.print("Résultat = " + nbre1/nbre2);
(le **+** joue ici le rôle d'opérateur de concaténation)
/!\ Attention toutefois avec ce genre de notation, car ici le type n'est pas définie et si vous attendiez comme résultats 3,33333333 vous serez déçu de voir que vous obtiendrez 3...

```
1  int resultat = (int)(nbre1 / nbre2);
```

```
1  int resultat = nbre1 / nbre2;
```

Les conversions, ou CAST :

Les variables de type double contiennent plus d'informations que les variables de type int.
Nous allons voir une thématique très importante en Java : La conversion de variables.

D'un type ***int*** en type ***float*** :

```
1 int i = 123;
2 float j = (float)i;
```

D'un type ***int*** en ***double***:

```
1 int i = 123;
2 double j = (double)i;
```

Et inversement :

```
1 double i = 1.23;
2 double j = 2.9999999;
3 int k = (int)i;          //k vaut 1
4 k = (int)j;              //k vaut 2
```

Ce type de conversion s'appelle une *"conversion d'ajustement"*, ou ***cast*** de variable.

Nous pouvons passer directement d'un type int à un type double. L'inverse, cependant, ne se déroulera pas sans une perte de précision. En effet, lorsque nous castons un double en int, la valeur de ce double est tronquée, ce qui signifie que l' int en question ne prendra que la valeur entière du double, quelle que soit celle des décimales.

Pour en revenir à notre problème de tout à l'heure, il est aussi possible de caster le résultat d'une opération mathématique en la mettant entre " ***( )*** " et en la précédant du type de cast souhaité:

```
1 double nbre1 = 10, nbre2 = 3;
2 int resultat = (int)(nbre1 / nbre2);
3 System.out.println("Le résultat est = " + resultat);
```

Voilà qui fonctionne parfaitement. Pour bien faire, vous devriez mettre le résultat de l'opération en type double. Et si on fait l'inverse : si nous déclarons deux entiers et que nous mettons le résultat dans un double? Voici une possibilité :

```
1 int nbre1 = 3, nbre2 = 2;
2 double resultat = nbre1 / nbre2;
3 System.out.println("Le résultat est = " + resultat);
```

Vous aurez 1 comme résultat. Pas de cast ici, car un double peut contenir un int.

En voici une autre :

```
1 int nbre1 = 3, nbre2 = 2;
2 double resultat = (double)(nbre1 / nbre2);
3 System.out.println("Le résultat est = " + resultat);
```

Idem… Afin de comprendre pourquoi, vous devez savoir qu'en Java, comme dans d'autres langages d'ailleurs, il y a la notion de priorité d'opération ; et là, nous en avons un très bon exemple !

Sachez que l'affectation, le calcul, le cast, le test, l'incrémentation… toutes ces choses sont des opérations ! Et Java les fait dans un certain ordre, il y a des priorités.

Dans le cas qui nous intéresse, il y a trois opérations :

un calcul

un cast sur le résultat de l'opération ;

une affectation dans la variable résultat

```
1 int nbre1 = 3, nbre2 = 2;
2 double resultat = (double)(nbre1 / nbre2);
3 System.out.println("Le résultat est = " + resultat);
```

Eh bien, Java exécute notre ligne dans cet ordre !
-Il fait le calcul (ici 3/2),
-il caste le résultat en double,
-puis il l'affecte dans notre variable résultat

Vous vous demandez sûrement pourquoi vous n'avez pas 1.5… C'est simple : lors de la première opération de Java, la JVM voit un cast à effectuer, mais sur un résultat de calcul. La JVM fait ce calcul (division de deux int qui, ici, nous donne 1), puis le cast (toujours 1), et affecte la valeur à la variable (encore et toujours 1).
Donc, pour avoir un résultat correct, il faudrait caster chaque nombre avant de faire l'opération, comme ceci

```java
int nbre1 = 3, nbre2 = 2;
double resultat = (double)(nbre1) / (double)(nbre2);
System.out.println("Le résultat est = " + resultat);
//affiche : Le résultat est = 1.5
```

Maintenant, nous allons transformer l'argument d'un type donné,int par exemple, en String.

```
1 int i = 12;
2 String j = new String();
3 j = j.valueOf(i);
```

"J" est donc une variable de type String contenant la chaîne de caractères 12. Sachez que ceci fonctionne aussi avec les autres types numériques. Voyons maintenant comment faire marche arrière en partant de ce que nous venons de faire.

```
1 int i = 12;
2 String j = new String();
3 j = j.valueOf(i);
4 int k = Integer.valueOf(j).intValue();
```

Maintenant, la variable "k" de type int contient le nombre 12

Il existe des équivalents à intValue() pour les autres types numériques : floatValue(), doubleValue()…

Depuis Java 7 : le formatage des nombres :

Le langage Java est en perpétuelle évolution. Les concepteurs ne cessent d'ajouter de nouvelles fonctionnalités qui simplifient la vie des développeurs. Ainsi dans la version 7 de Java, vous avez la possibilité de formater vos variables de types numériques avec un séparateur, l' underscore "_"

Ce qui peut s'avérer très pratique pour de grands nombres qui peuvent être difficiles à lire. Voici quelques exemples :

```
1 double nombre = 100000000000d; // cast en d
2 //Peut s'écrire ainsi
3 double nombre = 1____000____000___000_000d; // cast en d
4 //Le nombre d'underscore n'a pas d'importance
5
6 //Voici quelques autres exemple d'utilisation
7 int entier = 32_000;
8 double monDouble = 12_34_56_78_89_10d; // cast en d
9 double monDouble2 = 1234_5678_8910d;   // cast en d
```

Les underscore doivent être placés entre deux caractères numériques :

ils ne peuvent donc pas être utilisés en début ou en fin de déclaration ni avant ou après un séparateur de décimal. Ainsi, ces déclarations ne sont pas valides :

```
1 double d = 123_.159;
2 int entier = _123;
3 int entier2 = 123_;
```

Avant Java 7, il était possible de déclarer des expressions numériques en hexadécimal, en utilisant le préfixe 0x :

```
1 int entier = 255; //Peut s'écrire « int entier = 0xFF; »
2 int entier = 20; //Peut s'écrire « int entier = 0x14; »
3 int entier = 5112; //Peut s'écrire « int entier = 0x13_F8; »
```

Depuis java 7, vous avez aussi la possibilité d'utiliser la notation binaire, en utilisant le préfixe 0b :

```
1 int entier = 0b1111_1111; //Est équivalent à : « int entier = 255; »
2 int entier = 0b1000_0000_0000; //Est équivalent à : « int entier = 2048; »
3 int entier = 0b100000000000; //Est équivalent à : « int entier = 2048; »
```

Certains programmes Java travaillent directement sur les bits, il peut donc être plus pratique de les représenter ainsi avant de les manipuler.

Résumé :

--Les variables sont essentielles dans la construction de programmes informatiques.

--On affecte une valeur dans une variable avec l'opérateur égal "**=**" .

--Après avoir affecté une valeur à une variable, l'instruction doit se terminer par un point-virgule "***;***" .

--Vos noms de variables ne doivent contenir ni caractères accentués ni espaces et doivent, dans la mesure du possible, respecter la convention de nommage Java.

--Lorsque vous effectuez des opérations sur des variables, prenez garde à leur type : vous pourriez perdre en précision.

--Vous pouvez ***cast*** un résultat en ajoutant un type devant celui-ci : (int) , (double) , etc.

--Prenez garde aux priorités lorsque vous castez le résultat d'opérations, faute de quoi ce dernier risque d'être incorrect.