Chapitre 2:

Les bases du langage Java

Les commentaires

- /* commentaire sur une ou plusieurs lignes */
  - Identiques à ceux existant dans le langage C

- // commentaire sur une seule ligne
  - Identiques à ceux existant en C++

- /** commentaire d'explication */
  - Les commentaires d'explication se placent généralement juste avant une déclaration (d'attribut ou de méthode)
  - Ils sont récupérés par l'utilitaire javadoc et inclus dans la documentation ainsi générée.

Instructions, blocs, etc.

- Les instructions Java se terminent par un **;**
- Les blocs sont délimités par :

  { pour le début de bloc

  } pour la fin du bloc

  – Un bloc permet de définir un regroupement d'instructions. La définition d'une classe ou d'une méthode se fait dans un bloc.

- Les espaces, tabulations, sauts de ligne sont autorisés.
  - Cela permet de présenter un code plus lisible.

Les identificateurs

- On a besoin de nommer les classes, les variables, les constantes, etc. ; on parle d'identificateur.
- Les identificateurs commencent par une lettre ou _
- Conventions sur les identificateurs :
  - Si plusieurs mots sont accolés, mettre une majuscule à chacun des mots sauf le premier : **« uneVariableEntiere »**
  - La première lettre est majuscule pour les classes et les interfaces : **« MaClasse, UneJolieFenetre »**
  - La première lettre est minuscule pour les méthodes, les attributs et les variables : **« setLongueur, i, uneFenetre »**
  - Les constantes sont entièrement en majuscules : **«LONGUEUR_MAX »**

Les mots réservés

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	false
final	finally	float	for	if
implements	import	long	int	interface
instanceof	native	new	null	package
private	protected	public	return	short
static	super	switch	void	this
throw	throws	transient	true	try
syncronized	volatile	while		

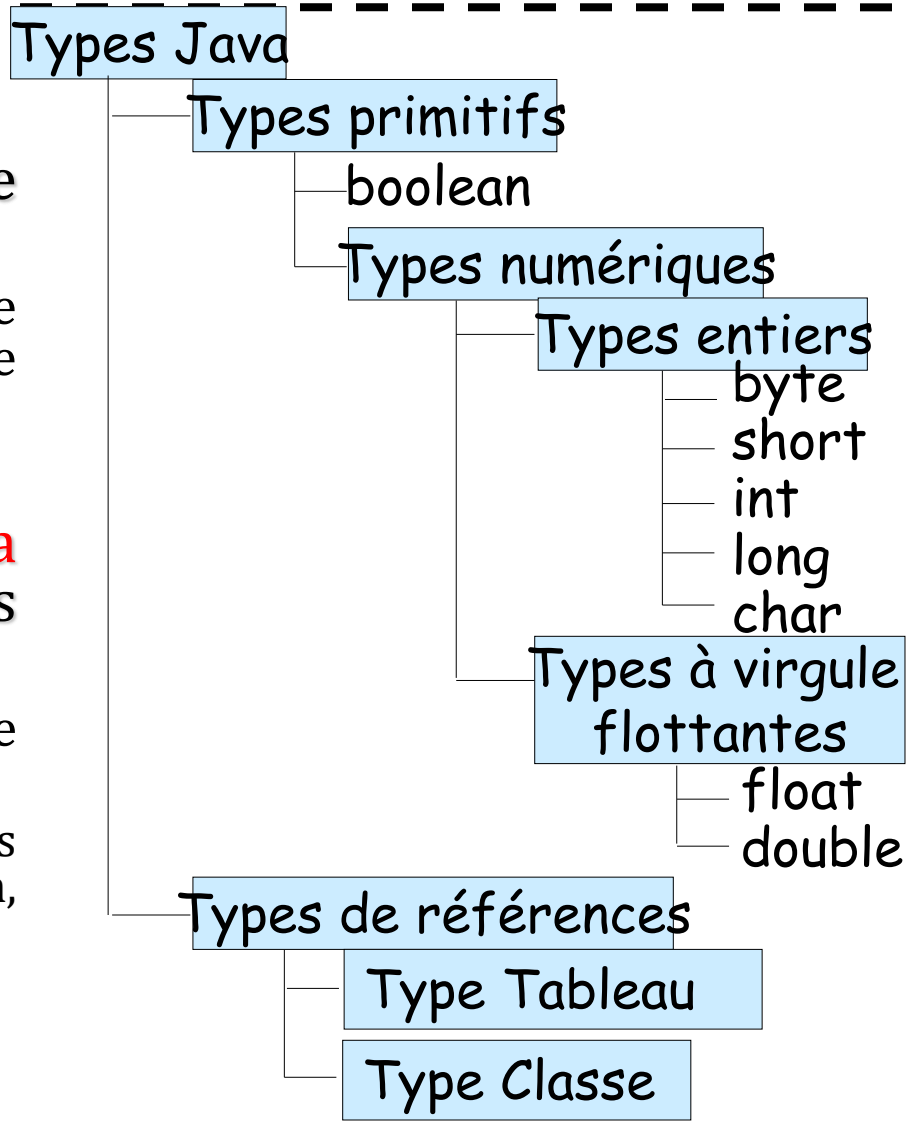
II.1.

Les types et les opérateurs

Les types

- Le langage Java est un langage fortement typé :
  - Chaque variable et chaque expression possède un type bien défini.

- Les types de données de Java sont divisés en deux grands groupes :
  - Les types primitifs (ou de base)
    - Exemple : Les types numériques, le type booléen, ...
  - Les types références



Les types primitifs

- boolean : false ou true
- byte : entier 8 bits
- short : entier 16 bits
- int : entier 32 bits
- long : entier 64 bits
- char : caractère Unicode 16 bits
- float : nombre décimal à virgule flottante 32 bits
- double : nombre décimal à virgule flottante 64 bits

Déclaration et initialisation

- Type identificateur [= constante ou expression];
  - int `int x=12;`
  - short `short x= 32;` (`short x=33000;` // Hors limite)
  - long `long x= 200L;` // Nombre accolé à un L
  - byte `byte x=012;` // Nombre commençant avec un 0
  - double `double x=23.2323;`
  - float `float x= 23.233F;` // Nombre accolé à un F
  - char `char c='a';` `char c='\u0061';` `char c=(char)97;`
  - boolean `boolean b=true;`
- Et éventuellement, un « modificateur d'accès ou de visibilité » : final double PI=3.14159

Les types de référence

- Un objet est créé par l'opérateur new qui appelle son constructeur de classe
  - String str1; // variable non initialisée = variable vide
  - String str2=**null**; // variable initialisée par une référence null
  - String pays = **new String**("France"); // pays est le nom de la variable faisant référence à l'objet String
    - Analogue à **String pays= "France";**
- Un objet ne peut pas exister si une variable de référence ne pointe pas vers lui : les variables str1 et str2 n'existent pas
- **String** n'est pas un type primitif, mais une classe

Les opérateurs

- Les opérateurs dans Java sont regroupés par :
  - type d'opérations :
    - d'affectation
    - numérique,
    - de comparaison,
    - logique,
    - sur les chaînes de caractères,
    - de manipulations binaires.
  - le nombre d'opérandes :
    - unaire,
    - binaire,
    - ternaire.

Les opérateurs unaires

Opérateurs unaires	Action	Exemple
-	**négation**	**i=-j**
++	**incrémentation de 1**	**i=j++ ou i=++j**
--	**décrémentation de 1**	**i=j-- ou i=--j**

- ++ et -- peuvent préfixer ou postfixer la variable.
  - i = j++ : post-incrémentation
    - La valeur en cours de j est affectée à i et ensuite la valeur de j est incrémentée de 1.
  - i = ++ j : pré-incrémentation
    - La valeur en cours de j est incrémentée de 1 et ensuite la valeur de j est affectée à i.

Les opérateurs binaires

Opérateurs binaires	Action	Exemple	Syntaxe équivalent
+	addition	i = j+k;	
+=		i += 2;	i= i + 2
-	soustraction	i = j - k;	
-=		i -= j;	i= i - j
*	multiplication	x=2*y;	
*=		x *=x;	x= x * x
/	division (tronque si les arguments sont entiers)	i =j/k;	
/=		x /= 10;	x= x /10
%	modulo	i = j %k;	
%=		i %=2	i= i %2
>>	décalage vers la droite	i>>k;	$i/2^k$ (si i>0)
<<	décalage vers la gauche	i<<k;	$i*2^k$

Les opérateurs relationnels

- Dans le langage Java, le résultat d'une comparaison est true ou false

Opérateurs relationnels	Action	Exemple
<	plus petit que	x<i;
>	plus grand que	i>100;
<=	plus petit ou égal que	j<=k;
>=	plus grand ou égal que	c>='a';
==	égal à	i==20;
!=	différent de	c!='z';

Les opérateurs logiques

Opérateurs logiques	Action	Exemple
!	négation	!p;
^	OU exclusif	p ^ false
&&	ET	(i<10) && (a>3)
||	OU	(a>0) || (b>0)

Opérateurs ternaires

- Un unique opérateur ternaire.

- Cette expression est une sorte de si-alors-sinon sous forme d'expression :

  - a = (condition e) ? x : y
    - si la condition e est vraie alors a vaut x sinon elle vaut y.

  - Exemple : a = (v==2) ? 1 : 0;
    - Cette expression affecte à la variable a la valeur 1 si v vaut 2, sinon affecte à la variable a la valeur 0.

La conversion de types

- Il y a 2 catégories de conversions possibles :

  - Conversions explicites :
    - celles faites sur une demande explicite par un programmeur.

  - Conversions implicites :
    - celles faites automatiquement par un compilateur **:**
      - lors d'une affectation,
      - lors d'une opération arithmétique,
      - lors d'un passage de paramètres (lors de l'invocation d'une méthode),

La conversion de types

- Conversion explicite :
  - Objectif :
    - changer le type d'une donnée si besoin.
  - Comment ? :
    - Préfixer l'opérande par le type choisi.
    - Encadrer le type choisi par des parenthèses.
  - Exemple :
    - double d = 2.5 ;
    - long l = **(**long**)** d ;
- Conversion implicite lors d'une affectation :
  - Objectif :
    - changer automatiquement le type d'une donnée si besoin.

La conversion de types

- La conversion numérique est faite automatiquement (implicitement) vers le type le plus riche dans une opération arithmétique

```
int i;
i  = 'A';
System.out.print( i ) ; // vaut 65
```

- La conversion peut être faite explicitement vers un type plus pauvre

```
double x = 2.1;
int a;

a= (int)x ;  // vaut 2
int b = a * 1500;
```

- ***Hiérarchie des types : byte < short < int < long < float < double***

La conversion de types

Conversion de	*Conversion vers* boolean	byte	short	char	int	long	float	double
boolean	-	N	N	N	N	N	N	N
byte	N	-	Y	C	Y	Y	Y	Y
short	N	C	-	C	Y	Y	Y	Y
char	N	C	C	-	Y	Y	Y	Y
int	N	C	C	C	-	Y	Y*	Y
long	N	C	C	C	C	-	Y*	Y*
float	N	C	C	C	C	C	-	Y
double	N	C	C	C	C	C	C	-

Y = OUI (YES), N = NON (NO), C = Cast (besoin de conversion explicite)

II.2.

Les structures de contrôle

Conditions, boucles, …

Les structures de contrôle

- Les structures de contrôles permettent d'arrêter l'exécution linéaire des instructions (de bas en haut et de gauche à droite)
- Elles permettent d'exécuter conditionnellement une instruction, ou de réaliser une boucle

Type d'instruction	Mots clés utilisés
Décision	if() else – switch() case
Boucle	for(; ;) – while () – do while()
Traitement d'exceptions	try catch finally – throw
Branchement	label : -- break – continue -- return

if-else

- Instruction conditionnelle :

```
if (condition)
{bloc 1}

if (condition) {bloc 1}
else {bloc 2}

if (condition 1) {bloc 1}
else if (condition 2){bloc 2}
else {bloc N}
```

```
if (x > y) {
    int tmp = x;
    x = y;
    y = tmp;
} else
    x = 0;
                ...
if ((x > s1)&& (x < s2))
  y=3*x+1;
else
  y=0;
```

```
if (i == j ){
    System.out.println (" i
  est égal à j " );
}
else if ( i>j){
      System.out.println ("
  i est supérieur à j" );
}
else {
     System.out.println ("
  i est inférieur à j " );
}
```

switch-case

```
switch nomVariable
{
     case valeur1 : {...
          break;
     }
     ...
     case valeurn : {...
          break;
     }
     default : {...
          break;
     }
} ;
```

```
int i = 1;
switch (i)
{   case 0 :
              System.out.println ("Zero");
               break;
    case 1 :
               System.out.println ("Un");
               break;
    case 2 :
              System.out.println ("Deux");
               break;
    default :
              System.out.println ("Autre");
               break;
}
```

Attention en JAVA :

- nomVariable : QUE de type “intégral” : boolean , char, int, long et short
- break; OBLIGATOIRE !

for

- Boucle for

```
for (expr1; expr2; expr3)
{bloc}
```

fonctionnement :

expr1
if (expr2==true){
bloc
expr3
}

```
float moyenne= 0;
// Initialisation d'un tableau d'entier
int[] tab = { 2, 5, -1, 4, 3 };

for (int i =0; i < tab.length; i++)
          // conversion!
           moyenne+=tab[i];

moyenne /= tab.length;
System.out.println("La moyenne est
      "+moyenne);
// Si moyenne etait un int, la division
// serait entière
```

while et do-while

- Boucles while

```
while(condition){
   Bloc
}

do {
   Bloc
}while(condition);
```

```
// Chercher un élément nul dans un
      tableau
int i = 0;
while ((tab[i] != 0)&& (i<tab.length))
      i++;

System.out.println("Le premier élément
      nul est en "+ i);
                  ...
int somme=1;
int i=borneSuperieure;
do{
      somme+=i
      i--
} while (i>0)
```

break

- Interruption de boucle
  - Interruption non étiquetée : sortie de la boucle la plus haute.

```
while( i <= 100 ) {
   i += 10;
   if( i >= 2 * n + 1 ) {
      break;
   }
}
```

  - Interruption étiquetée : sortie d'une boucle imbriquée.

```
boucle_1:
while( i <= 100 ) {
   i += 10;
boucle_2:
   while( i < j ) {
      i++;
      if( i >= 2 * n + 1 ) {
         break boucle_2;
       }
   }
}
```

continue

- Continuation de boucle = court-circuit de la fin d'une itération

```
int x = 0;
while (x < 10)
{
  x++;
  if (x == 5)
  {
    continue;
  }
  System.out.print(x + " ");
}
```

Produira

```
> 1 2 3 4 6 7 8 9 10
```

Les tableaux

- *Array*
  - Stockage d'éléments tous du même type
  - Structure à part entière
  - Un tableau est un objet référencé
  - Assimilable à une classe
  - Création en trois étapes
    1. déclaration
    2. allocation de mémoire
    3. initialisation des éléments

Les tableaux

- Indiqué par `[]`
  - Deux possibilités

```
type[] nom;
type nom[];
```

```
int[] tableau1;
int tableau2[];
int[][] matrice;  // tableau bidimensionnel
int[] x, y[]; //équivalent à int x[],y[][];
int tab[10];  // ne compile pas
```

Allocation et initialisation

- Alloué dynamiquement
  - à l'aide du mot clé **new**

```
int[] tableau1;  // déclaration
tableau1 = new int[10];  // allocation
int tableau2[]; // déclaration
tableau2 = new int[35]; // allocation
int[][] matrice; // déclaration
matrice = new int[2][4]; // allocation
int[] x, y[]; // déclaration
x = new int[5]; // allocation
y = new int[3][2]; // allocation
```

Allocation et initialisation

- On peut combiner déclaration et allocation

```
int[] tableau1 = new int[10];
int tableau2[] = new int[35];
int[][] matrice = new int[2][4];
int[] x = new int[5];
int[] y[] = new int[3][2];
```

- Chaque élément doit être initialisé séparément

```
int[] tablo = new int[10];
for (int i = 0; i < 10; i++) {
    tablo[i] = i;
}
```

Allocation et initialisation

- Valeurs initiales peuvent être énumérées
  - `int[] joursParMois = {31, 28, 31,30, 31, 30, 31, 31, 30, 31, 30, 31};`
  - `String[] jours = {"lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"};`
- Déclaration, allocation, initialisation

Accès aux tableaux

- Indexage à partir de 0
- Accès aux éléments par **`[]`**

  `tablo[i]` **i = 0..tablo.length - 1**
- Nombre d'éléments donné par la *variable*

  ***`nom`*`.length`**

```
for (int i = 0; i < tablo.length; i++) {
  System.out.print(tablo[i] + " ");
  tablo[i] = -tablo[i];
}
```