Domaine Sciences et Technologies
Master Bio-informatique : DLAD

Aix*Marseille université
Socialement engagée

Programmation avancée : Syntaxe Java
Code UE : SBICU05

Année 2023-24

Éléments de syntaxe en Java

Les types primitifs

En Java on appelle *type primitif* un type qui ne correspond pas à un objet.

Les booléens

Le type boolean permet de représenter les booléens, il définit les valeurs `true` et `false`.

Les opérateurs booléens sont :

- négation logique `!`, **Exemple :** `!found`
- et logique : `&&`, **Exemple :** `(n > 0)&& (n % 3 == 1)`
- ou logique : `||`, **Exemple :** `(c == 'a')|| (c == 'A')`

Les caractères

Le type `char` permet de représenter les caractères. Il utilise le codage Unicode sur 16 bits. ex : `'a'`, `'-'`, `'0'`

Note : contrairement au Python, les caractères ne sont pas des chaînes de caractères de longueur 1, les chaînes de caractères sont représentées en Java par des objets de la classe String.

Les entiers

Les entiers signés sont représentés en complément à deux :

- `byte` : entiers sur 8 bits, intervalle de valeurs possible $[-2^7, 2^7 - 1]$
- `short` : entiers sur 16 bits, intervalle de valeurs possible $[-2^{15}, 2^{15} - 1]$
- `int` : entiers sur 32 bits, intervalle de valeurs possible $[-2^{31}, 2^{31} - 1]$
- `long` : entiers sur 64 bits, intervalle de valeurs possible $[-2^{63}, 2^{63} - 1]$ avec $2^{63} \approx 9.10^{18}$, suffixés par L, ex : 2147483648L

Les opérateurs sur les entiers sont :

- addition `+`
- soustraction `-`
- multiplication `*`
- division entière `/`
- moins unaire `-`
- modulo `%` : `x%y = x-(x/y)*y`

On a aussi les « raccourcis » des post/pré incrément/décrément :

— `x++` : délivre la valeur de `x` puis ajoute 1 à `x`
— `--x` : soustrait 1 à `x` puis délivre la valeur de `x`
— symétriquement, on a : `++x` et `x--`.

On a aussi des raccourcis pour des modifications d'une variable avec une opération :

— `x += i` qui ajoute `i` à `x` (équivalent à `x = x + i`)
— `x -= i` qui retranche `i` à `x` (équivalent à `x = x - i`)
— `x *= i` qui multiplie `x` par `i` (équivalent à `x = x * i`)
— `x /= i` qui divise `x` par `i` (équivalent à `x = x / i`)

Note : l'opérateur de puissance n'est pas un opérateur de base. On utilisera la fonction `pow` de la classe `Math`.

Les flottants

Il existe deux types flottants en Java :

— **float** flottants simple précision, littéraux suffixés par f, intervalle $[-10^{-38}, -(10^{38})] \cup [10^{-38}, 10^{38}]$, ex : `112.2e-45f`, `-34E-555f`, `3.14f`
— **double** flottants double précision, littéraux suffixés optionnellement par d, intervalle $[-10^{-308}, -(10^{308})] \cup [10^{-308}, 10^{308}]$, ex : `1.34e56d`, `3.14`

Les opérateurs sur les flottants sont :

— addition `+`
— soustraction `-`
— multiplication `*`
— division flottante `/`
— moins unaire `-`

Prédicats de comparaison de base

Aux opérateurs précédents on peut ajouter les prédicats de comparaison :

— les classiques comparaisons `< <= > >=`
— l'égalité `==`
— la différence `!=`

Chaînes de caractères

En Java, les chaînes de caractères sont des objets, instances de la classe `String` (il s'agit de chaîne de caractères non mutables). Une syntaxe particulière permet de représenter un objet `String` par la séquence de caractères qui le compose encadrée de guillemets `"` : `"Programmation"`, `"Portail Descartes"`. La chaîne vide (sans caractères) correspond donc à `""`.

La classe `String` dispose de nombreuses méthodes. L'étude de sa documentation (lien : https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html) permet de les découvrir. En voici quelques-unes :

— `int length()` renvoie le nombre de caractères de la chaîne qui appelle la méthode
— `char charAt(int pos)` renvoie le caractère situé à la position `pos`, quand elle existe, le premier caractère a la position `0`
— `int indexOf(char c)` renvoie la position de la première occurrence de dans la chaîne, `-1` s'il n'est pas présent (`int indexOf(String str, int fromIndex)` existe aussi, ainsi que `lastIndexOf`)
— `String substring(int beginIndex)` renvoie une sous-chaîne de caractères extraite de la chaîne initiale à partir de `beginIndex`, (`String substring(int beginIndex, int endIndex)` existe aussi)
— `String toUpperCase()` renvoie une chaîne de caractères dont la séquence de caractères est obtenue en remplaçant tous les caractères de la séquence originale par des majuscules.
— `boolean equals(Object o)` renvoie `true` si et seulement si l'objet `o` est une chaîne de caractères contenant la même séquence de caractère que la chaîne appelante.

Les commentaires

```
1 // toute la ligne est en commentaire
2
3 /* commentaire sur
4    plusieurs lignes
5    entre balises    */
```

La déclaration de variables

Déclarer une nouvelle variable. En Java, on ne peut utiliser une variable qu'après l'avoir *déclaré*, pour signaler notre intention. Les variables en Java ont toujours un et un seul type, défini lors de la déclaration. La syntaxe d'une déclaration est l'*identifiant du type* suivi de l'*identifiant de la variable*.

Exemples :

```
1 int n; // une variable n pouvant contenir un entier
2 double x; // une variable x pouvant contenir un nombre à virgule
3 String text; // une variable text pouvant contenir une chaîne de caractéres
4 boolean found; // une variable found pouvant contenir une valeur true or false
```

Par convention, en Java, on utilise la notation dites « camelCase ». De plus les identificateurs de variables commencent par une minuscules.

Les instructions

L'affectation

Affecter une valeur à une variable, avec l'opérateur d'affectation `=`. Comme en Python, ce symbole n'est pas celui du test d'égalité. La syntaxe est *identifiant de la variable* à affecter suivi du symbole `=` suivi d'une *expression* représentant la valeur à stocker.

Exemples :

```
x = 3; // met la valeur 3 dans la variable x
x = x + 1; // augmente de 1 le contenu de la variable x
y + 1 = y; // ERREUR : à gauche du = doit se trouver une variable à affecter !
```

On peut aussi combiner déclaration et affectation :

```
double almostPi = 3.14; // on peut combiner déclaration et affectation
```

Le retour de fonction

Terminer l'exécution d'une méthode, en retournant optionnellement un résultat.

```
return 42; // termine la méthode et retourne 42.
return; // termine la méthode et ne retourne rien.
```

Appel de méthodes (équivalent des fonctions de Python)

La syntaxe est *identifiant du propriétaire de la méthode* (soit une référence d'un objet ou bien une classe pour les méthodes statiques) suivi d'un point suivi de l'*identifiant de la méthode* suivi d'une paire de parenthèses. Entre les parenthèses figurent les arguments passés à la méthode, séparés par des virgules. Voici l'exemple de la méthode `println` appartenant à la classe de `System.out`, et qui permet d'écrire du texte en console.

```
String s = "Arnaud";
s = s.toUpperCase();
System.out.println(s); // affiche ARNAUD
System.exit(0); // termine l'exécution du programme
```

Les structures de contrôle

Les blocs d'instructions

Un bloc est délimité par `{` et `}`. Un bloc définit les règles de portée : toute variable déclarée dans un bloc a une portée (ou visibilité) limitée à ce bloc. Elle n'est connue que dans ce bloc. Toute séquence d'instructions doit être incluse dans un bloc.

```
{ int i = 5; } i = 6; // erreur
```

Structures conditionnelles

Le si... alors... s'écrit : `if` (⟨`condition`⟩) ⟨`code vrai`⟩

Exemple : `if (x == y)z = 0;` ou `if (x == y){ z = 0;}`

Le si... alors... sinon... s'écrit : `if` (⟨`condition`⟩) ⟨`code vrai`⟩ `else` ⟨`code faux`⟩

```
if(x == y){
  found = true;
  x = 0;
}
else {
  found = false;
```

```
7     x = -1;
8   }
```

Les structures itératives

La boucle **tant que** : `while` (⟨condition⟩) ⟨corps⟩

Exemple :

```
1  boolean done = false; // fin de boucle
2  int nb = 0; // nb tours de boucle
3  while (!done) {
4    x = x - 1;
5    nb = nb + 1;
6    done = x < 0;
7  }
```

La boucle **for** : `for` (⟨initialisation⟩ ; ⟨condition⟩ ; ⟨expr progression⟩) ⟨corps⟩

Exemple :

```
1  int j = 2;
2  for (int i=0; i<3; i++) {
3    j += 2;
4  }
```

Boucle faire ... jusqu'à : `do` ⟨corps⟩ `while` (⟨condition⟩)

Exemple :

```
1  int i = 0, j=10;
2  do {
3    i += 2;
4    j++;
5  }
6  while (i < j);
```

→ similaire à une boucle tant-que mais en testant la condition à la fin. Le corps de la boucle est nécessairement exécuté au moins une fois.

Création d'objet

La création d'un objet depuis une classe, ou *instanciation*, se fait toujours avec le mot réservé du langage `new`. La syntaxe est `new` suivi du nom de la classe, suivi d'une paire de parenthèses contenant les arguments d'initialisation de la classe.

```
1  new Point2D(4,3);
2  // En régle générale, on stocke la référence de l'objet dans une variable :
3  Point2D origin = new Point2D(0,0);
```

Les tableaux

Les tableaux permettent de stocker un nombre prédéfini d'objets d'un même type. On aura par exemple un tableau de 10 entiers, un tableau de 2 booléens ... Pour utiliser un tableau il faut le déclarer puis le construire.

Déclaration d'un tableau

Pour déclarer un tableau on précise le type de ses éléments, mais pas le nombre d'éléments qu'il contient : ⟨type elt⟩ [] ⟨nom tab⟩;

Exemples : `int[] tab1; boolean[] tab2;`

Construction d'un tableau

La construction d'un tableau `arrayName` déjà déclaré se fait en utilisant **new** et en précisant sa taille (son nombre d'éléments) : `arrayName = new` ⟨type elt⟩ [⟨expr nb⟩];

Exemples : `tab1 = new int[10]; tab2 = new boolean[x+3]; //en supposant que x est une variable de type int définie`

⇒ `tab1` contient 10 cases numérotées de 0 à 9. `tab2` contient x+3 cases.

Taille

La taille d'un tableau fait partie des attributs fournis par Java : si ⟨nom tab⟩ est un tableau déjà créé, alors ⟨nom tab⟩`.length` est son nombre d'éléments. ex : `tab1.length` vaut `10`.

Initialisation à la déclaration

On peut initialiser un tableau lors de sa déclaration par une liste de valeurs séparées par des , et entourées d'accolades. Dans ce cas, il n'y pas besoin de créer explicitement le tableau avec **new**.

Exemple : `int[] tab = {1, 2, 3, 4, 12};`

On a dans ce cas `tab.length` qui vaut `5`.

Accès indexé

Les éléments d'un tableau sont numérotés (on dit indexés ou indicés) à partir de `0`. Si ⟨nom tab⟩ est un tableau déjà créé, alors on accède à son *i*ème élément par : ⟨nom tab⟩[⟨expr indice⟩] ex : `tab1[0]`, ... ,`tab1[9]` sont des expressions correctes, mais `tab1[10]` est une expression incorrecte (on sort des indices possibles, car le tableau est de taille 10). On écrira par

Exemple :

```
1  tab1[0] = 1;
2  int x = 5;
3  tab1[x-2] = 4;
4  tab1[9] = 10;
```

Attention : l'indice doit être compris entre 0 inclus et tab.length exclu. On écrira donc typiquement :

```
 for (int i=0; i<tab.length; i++) {
     tab[i] = i+1;
}
```

Itération sur les élements d'un tableau

Lorsque l'on souhaite parcourir un tableau et réaliser une manipulation sur chacun de ses éléments on peut utiliser la syntaxe (dite « à la for-each ») suivante :

```
float[] tab = ... ;
float somme  = 0;
for (float element : tab) { // element prend successivement toutes les valeurs
    dans tab
  somme += element;
}
```

qui équivaut à

```
float[] tab = ... ;
float somme  = 0;
for (int i=0; i<tab.length; i++) {
  somme = somme + tab[i];
}
```

Tableaux multi-dimensionnels

Un tableau à deux dimensions peut être vu comme un tableau dont chaque élément est lui-même un tableau à une dimension. Et on peut généraliser pour les tableaux à trois (ou plus) dimensions. La déclaration et la construction de tableaux à plusieurs dimensions suivent le schéma des tableaux à une dimension avec ⟨type elt⟩ qui est un type tableau :

- déclaration : ⟨type elt⟩ []...[] ⟨nom tab⟩
- construction : ⟨nom tab⟩ = **new** ⟨type elt⟩ [⟨expr nb1⟩] ... [⟨expr nbn⟩];

Exemple :

```
int[][] mat;
mat = new int[3][2];
mat[1][0] = 4;
int[][] damier = new int [4][4];
```

mat est un tableau regroupant 3 valeurs (3 cases), chacune contenant un tableau de 2 entiers (2 cases).

Il faut obligatoirement fixer la première dimension à la création. Mais les autres dimensions peuvent être fixées ensuite. Conséquence : un tableau n'est pas forcément rectangulaire.

Exemple :

```
int [][] brokenGameBoard;
brokenGameBoard = new int[4][];
```

```
brokenGameBoard[0] = new int[4];
brokenGameBoard[1] = new int[3];
...
brokenGameBoard[3] = new int[4];
```

Seule restriction par rapport à l'initialisation des tableaux à une dimension : on ne peut pas initialiser un tableau de la seconde dimension en énumérant ses valeurs, sans avoir préalablement fixé sa dimension.

```
brokenGameBoard[2] = new int[2];
//création obligatoire même pour l'initialisation suivante
brokenGameBoard[2] = { 2 , 3 };
```