Syntaxe du langage Java

J-F. Kamp
M. Adam

Août 2019

Structure d'un programme Java pour M1102 & M1103

```
public class MonProg {

  // déclaration des variables globales SI nécessaire
  // déclaration des constantes
  …

  // le point d'entrée du programme
  void principal () {
        // appel de la séquence de test des méthodes
        testMaMeth1(); // test de maMeth1
        testMaMeth2(); // test de maMeth2
        …
  }

  // la méthode de test de maMeth1
  void testMaMeth1() {…}

  // la méthode maMeth1, exemple
  float maMeth1 (int[] tab, int param1, float param2) {
        float ret;
        …
        return ret; // DERNIERE instruction
  }

}
```

Déclaration des variables/constantes

« type » nomVar1 **[** = valeur initiale **]** **;**

final « type » NOM_CONST = valeur initiale **;**

En Java : 8 « type » primitifs

- boolean : type booleen (2 bits, true, false)
- char : type caractère ('1', '?', '(', ...)
- byte : type entier sur 8 bits
- short : type entier sur 16 bits
- int : type entier sur 32 bits
- long : type entier sur 64 bits
- float : type réel sur 32 bits
- double : type réel sur 64 bits

!! Le type « chaîne de caractères », « String » en Java n'est pas un type primitif mais une classe !!

Les opérateurs

- affectation : **=**

- opérateurs mathématiques : **∗, /, -, +**
  Uniquement les types entiers et réels.
  !! **+** est aussi utilisé pour concaténer des chaînes de caractères (comme TestAlgo3).

- modulo : **%** (types entiers et réels)

- incrémentation de 1 : **++** (types entiers et réels)

- décrémentation de 1 : **--** (types entiers et réels)

- égalité : **==**

- inégalité : **!=**

- supérieur, inférieur : **>, >=, <, <=**

- et, ou, non : **&&, ||, !**
  Types booléens uniquement.

Transferts Externes : entrée / sortie

AFFICHAGE

System.out.println (String str); // *str* de type String

où *str* = la concaténation (+) de n'importe quelle chaîne de caractères constante ("toto") avec n'importe quel type primitif et n'importe quel autre type *String*.

Transferts Externes : entrée / sortie

SAISIE PAR BOITE DE DIALOGUE « SimpleInput »

- **Saisie d'un type entier *int*, enregistré ensuite dans une variable *result* de type *int***

  result = SimpleInput.getInt (String quest);

  où *quest* est une chaîne de caractères affichée dans la boite de dialogue pour inviter l'utilisateur à rentrer une info. L'info saisie devra forcément être de type entier et sera enregistrée dans la variable *result*.

- **Saisie d'un type réel *float*, variable *result* de type *float***

  result = SimpleInput.getFloat (String quest);

- **Saisie d'un type réel *double*, variable *result* de type *double***

  result = SimpleInput.getDouble(String quest);

- **Saisie d'un type booléen *boolean*, *result* de type *boolean***

  result = SimpleInput.getBoolean (String quest);

- **Saisie d'un type caractère *char*, variable *result* de type *char***

  result = SimpleInput.getChar (String quest);

- **Saisie d'un type chaîne de caractères *String*, *result* type *String***

  result = SimpleInput.getString (String quest);

Alternative (si…alors…sinon…finsi)

```
if (<condition>) {

        <instructions1>;
}

[ else {

        <instructions2>;
} ]
```

Un bloc d'instructions en Java est toujours délimité par {…}.

Le «`;`» est le séparateur d'instructions :

- toujours après une déclaration de variable
- toujours après une instruction
- jamais après un mot-clef (sauf boucle « do »)
- jamais après « } »

Alternative : la clause else if

```
if (<condition1>) {
        <instructions1>;
}
else {
        if (<condition2>) {
                <instructions2>;
        }

        else {
                <instructions3>;
        }
}
```

<=>

```
if (<condition1>) {
        <instructions1>;
}
else if (<condition2>) {

        <instructions2>;
}
else {
        <instructions3>;
}
```

Plusieurs alternatives : la structure switch

```
switch (uneVariable) // uniquement byte, short, int, char

{
        case <valeur1> :
                // exécuté si (uneVariable == valeur1)
                <instructions>;
                // on quitte le switch
                break;

        case <valeur2> :
                // exécuté si (uneVariable == valeur2)
                <instructions>;
                // on quitte le switch
                break;

        default :
                // facultatif
                // exécuté si tout le reste a échoué
                <instructions par défaut>;
                // on quitte le switch
                break;
}
```

Boucle while (tantque) + Boucle do…while (repeter…jusqua)

```
while ( <condition de continuation> ) {

        <instructions>;
}
```

```
do {

        <instructions>;
} while (<condition de continuation>) ;
// !! Condition de continuation, PAS condition
// d'arrêt contrairement à TestAlgo3 !!
```

La boucle « do…while » est exécutée au moins une fois contrairement à la boucle « while ».

Boucle for (pour)

```
for ( <init>; <condition de continuation>; <incr> ) {

        <instructions>;
}
```

La boucle « for » est strictement équivalente à la boucle « while » suivante :

```
<init>;

while ( <condition de continuation> ) {

        <instructions>;

        <incr>; //obligatoirement la dernière instruction
}
```

Les Tableaux (forme simple)

Déclaration (sans nécessairement préciser la taille)

« type »[] monTab; // A ce stade, *monTab* contient la
// valeur « null » et est donc
// INUTILISABLE !

Allocation dynamique du tableau

monTab = new « type »[TAILLE];

// création d'un tableau à 1 dimension de TAILLE cases

Récupération de la taille d'un tableau

laTaille = monTab.length; // *laTaille* : variable de type *int*

Commentaires

type = types primitifs (*int*, *float*, …) ou type *String*

monTab = nom du tableau = adresse du tableau

TAILLE = un entier > 0 qui précise le nombre de cases du tableau à 1 dimension.

Première case accessible à l'indice « 0 », dernière case accessible à l'indice « TAILLE - 1 ».

Déclaration des méthodes : syntaxe

```
<typeRenvoyé> <nomMéthode> ( <typeParam1>
<nomParam1>, <typeParam2> <nomParam2>, … )  {

    <déclaration des variables locales>;
    ...
    <instructions>;
    ...
    return (resultat);
}
```

Commentaires

- Une méthode renvoie toujours soit un type primitif, soit « rien » (type *void*).
- Un type primitif est toujours passé en paramètre par valeur (jamais par référence).
- Un tableau est toujours passé en paramètre par référence. Exemple :

```
int maMethode ( float[] leTab, boolean var ) {
    int res;       // variable locale
    <instructions>;
    return (res);
}
```