UMBB/FS/DEP.INFORMATIQUE

Syntaxe et élements de base Java

Ilyes DJAAFRI

1.Présentation de Java

Java est un langage de programmation orientée objet utilisant à la base une syntaxe similaire à celle du langage C. Il utilise des Bibliothèquesstandard assez complète (io, interface homme-machine, BDD, réseaux, web ...)

Java est un langage qu'on peut qualifier de : compilé et interprété:

- Compilation: code source (fichier.java) code intermédiaire - bytecode (fichier.class)
- Interprétation: exécution du bytecode par la machine virtuelle java (JVM)

La machine virtuelle permet notamment :

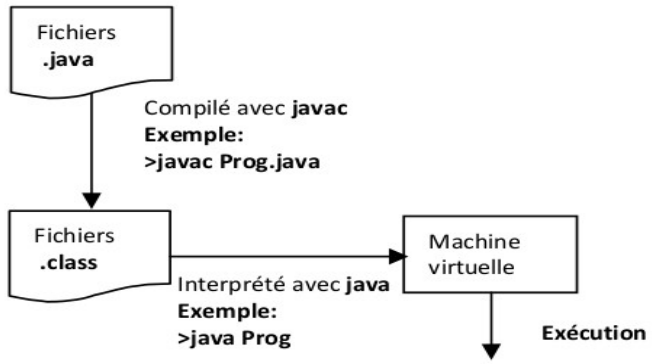
- l'interprétation du bytecode
- l'interaction avec le système d'exploitation
- la gestion de de la mémoire
- ...

Des versions différentes de la machine virtuelle existent pour chaque plateforme (matériel et système d'exploitation)

1.2 Exécution d'un programme JAVA

Pour exécuter des applications en JAVA sur votre système, il faut que le système dispose de « l'environnement de l'exécution JAVA » : JRE(Java Runtime Execution). La JRE est composé de la JVM et d'une bibliothèque standard vous avez besoin de ce qu'on appelle une machine virtuelle. Cette machine virtuelle va interpréter le code compilé, code généré à partir des fichiers .java (et donc des sources du programme).

La compilation des fichiers .java génère des fichiers .class se fait au moyen de la commande javac NomFichier.java. Pour exécuter ensuite ce fichier qui vient d'être généré, vous devez utiliser la commande java NomFichier (sans le .class).



- un fichier '.java' contient seulement des classes
- une seule classe peut être public ,dans ce cas, elle doit porter le même nom du fichier '.java'
- une classe est dite exécutable si elle contient une fonction **main**
- la classe qui contient la fonction main doit porté le même nom du fichier. Java
- la fonction main doit avoir la signature suivante: **public static void main(String[] args)**
- **il est fortement recommandé qu'un ficher source ne contient qu'une seule classe**

2. Types de données

Deux sortes:

- **types primitifs :** c'est des types valeur
  - Entiers: **byte** (8 bits), **short** (16 bits), **int** (32 bits), **long** (64 bits)
  - nombres réelles: **float**(32 bits), **double** (64 bits)
    Les types numériques de java sont tous non signées (Dans C il existe des types signés: valeurs positives seulement ou négatives seulement).
  - caractères: **char** (16 bits) Unicode,
  - booléen : **boolean** (1 bit) ***true*** ou ***false,***
- **Types références :** les tableaux et les objets, c'est des types références (voir les références)
  - Tableaux : différents du langage C voir section tableau.
  - Classe : type abstrait dans les instances sont des objets.

3. Règles de syntaxe générales

- Les blocs de code sont encadrés par des accolades
- Chaque instruction se termine par un ";"
- Java est sensible à la casse, c.-à-d. qu'un identifiant (nom de variable, nom fonction, nom de classe, deprogramme...) contenant des majuscules est différencié du même nom écrit en minuscule
- Une instruction peut tenir sur plusieurs lignes.
- L'indentation (la tabulation) est ignorée du compilateur mais elle permet une meilleure lecture du code par le programmeur

4. Opérateurs

- +, -, *, /, % : opérateurs arithmétiques
- =, +=, -=, *=, /=, %= , ++,--: opérateurs d'assignation
- <, >, <=, >=, ==, != : opérateurs de comparaison

Remarque les opérateurs de comparaison sont utilisé avec des opérants de type primaire. On ne peut pas comparer des objets. Une erreur fréquente est de comparer des objets de type String (chaîne de caractère) avec ces opérateurs.

Exemple

```
String s1="abc", s2="abc";
if (s1==s2) System.out.println("vrai");
else System.out.println("faux");
```

Ce code affichera : faux (voir pourquoi dans la section Objet)

- &&, ||, ! : Opérateurs logiques
- opérateur ternaire : **condition?expression_si_vrai:expression_si_faux**

Exemple :

```
if (a==b) c=val1 ; else c=val2 ;   <=>   c=(a==b)?val1:val2 ;
```

5. Conversion entre types primitifs

La hiérarchie des types numériques primitifs du plus bas vers le plus haut est la suivantes:
byte-> shorte-> int-> long-> float-> double. Le type booléen est incompatible.
Sous java, une affectation d'une expression d'un type donné à une variable d'un autre type compatible nécessite uneconversion de type.

5.1 Constante littérale numérique

Nombres entiers

- Une constante littérale entière est par défaut de type int, c'est-à-dire codée sur 32 bits;
- Si elle est trop grande pour être représentée dans ce type ; elle appartient alors au type **long**.
- La lettre **L** ou **l** indique que la constante doit être considérée comme appartenant au type **long** et codée sur 64 bits, même si sa valeur est petite.

Exemple

1234 // une valeur du type **int**

1234L // une valeur du type **long**

Nombres décimales

- Une constante décimale (c'est-à-dire un nombre comportant un point et/ou un exposant, comme 1.5, 2e-12 ou0.54321E3) représente par défaut une valeur du type double.
- Les suffixes f ou F précise que le type de la constante utilisé est **float**. Ainsi, une simple affectation telle que :

```
float x = 1.5; // ERREUR à la compilation (1.5 est considéré de type double, on ne peut pas l'affecter à unevariable de type réel)
```

•une manière de corriger cette erreur est d'utiliser le préfixe **f** ou **F** :

```
float x = 1.5f; // affectation correcte
```

5.2 Conversion implicite

- Soit l'affectation: **var_Type1=expression_Type2;**

- Si **type2** est moins élevé que **type1**, alors **expression_Type2** est convertie implicitement (automatiquement, sansaucune intervention du programmeur) en **type1** avant l'affectation.

Exemple

```
int a=6;
float b=a;// 6 est converti implicitement 6.0, puis affecté à la variable b
```

5.3 Conversion explicite

- Soit l'affectation: **var_Type1=expression_Type2;**
- Si **type2** est plus élevé que **type1**, dans la hiérarchie des types l'affectation **var_Type1=expression_Type2**,déclenche une erreur lors de la compilation ou lors de l'exécution.
- Dans le cas échéant, il faut prévoir une conversion explicite, en introduisant un opérateur de conversion: (**type1**) devant l'expression à convertir. L'affectation deviendra : **var_Type1=(type1) expression_Type2**

Exemple

```
int a=6;
short b=a; // erreur lors de la compilation, le compilateur exige une conversion explicite
int b=(short)a; //correcte, b<-- 6
```

La conversion explicite peut générer une perte d'information voire des résultats altérés et souvent le compilateurne signale aucune erreur. C'est au programmeur d'anticiper ce genre d'erreurs.

Exemple

```
Int a=40000; // a dépasse la capacité du type short
short b=a; //erreur lors de la compilation, le compilateur exige une conversion explicite
int b=(short)a;//b= -25535!! Résultat faux altéré
```

6. Structures de contrôle

Structure conditionnelle if

```
if (condition) {
        instructions
}
```

si condition est vrai le bloc instructions est exécuté.

Structure conditionnelle else

```
if (condition) {
        instructions1
}
else {
        instructions2
}
```

si condition est vrai le bloc instructions1 est exécuté sinon c'est le bloc instructions2 qui est exécuté

Structure conditionnelle else if

```
if (condition1) {
        instructions1
}
else if (condition2){
        instructions2
}
…
else if (conditionI){
        instructionsI
}
…
else if (conditionN){
        instructionsN
}
else {
        instructionsN+1
}
```

Seul le bloc *instructionsI* est exécuté si *condition1* est vrai. Si toutes les conditions sont fausses alors c'est le bloc *instructionsN+1* qui est exécuté.

Choisir un cas parmi plusieurs: Switch

```
switch (variable){
case valeur1 :
        instructions1
        break;
case valeur2 :
        instructions2
        break;
        …
case valeurN :
        instructionsN
        break;
```

```
case default:
        autres_instructions
}
```

variable doit être de type primitif, seul le cas où variable est égale valeur_i qui est exécuté

Les boucles

La boucle for

for (initialisation; condition de continuation; incrémentation){

        instructions

}

La boucle while

while (condition){

        instructions

}

La boucle do

Do {

        instructions

} while (condition)

6. Tableaux

6.1 Tableau à une dimension

Déclaration et création

- Deux syntaxes équivalentes pour déclarer un tableau sous Java:
    - **<type_de _base> <NomTableau>[];**
    - **<type_de _base>[] <nomTableau>;**

    <nomTableau> désigne un tableau de **<type_de _base>**

Exemple : int[] tab; et **int tab[]** représentent la même déclaration de tab un tableau de int

- On utilise l'opérateur new pour créer les espaces mémoire d'un tableau;

<nomTableau>= new <type_de _base>[];

Exemple: table = new int[5]; *//crée un tableau d'entier de 5 éléments, tous initialisé à 0*

- On peut créer un tableau en lui affectant les valeurs entre {} ;

  Exemple: double[] tableReel={2.5,5,7.1}

Parcours d'un tableau

- les indices d'un tableau java commencent par 0
- l'attribut length d'un tableau donne sa longueur
- accès aux éléments d'un tableau (exemple) : **table[i] = expression; ou var=table[i]**
- un tel accès provoque toujours deux vérifications : **table ≠ null** et que **0<i <table.length**
- on utilise souvent la boucle for pour parcourir un tableau :

```
String persTab={"ali","oussama","brahim","mohamed"};
String nomPersonne;
for (int i=0;i<persTab.length;i++){
    nomPersonne=persTab[i];
    //traitement...
}
```

- A partir de Java 5 (version 1.5), un moyen plus court pour parcourir un tableau est fourni.
- L'exemple suivant réalise le traitement sur tab:
- int tab[]={2,43,1,0,3};

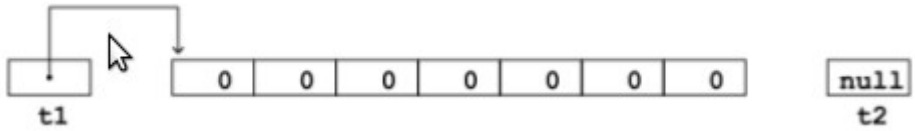
```
for (int element : tab){
        System.out.println(element);
}
```

- Attention néanmoins, la variable element contient une copie de tab[i]. Avec des tableaux contenant des variables primitives, toute modification de element n'aura aucun effet sur le contenu du tableau.

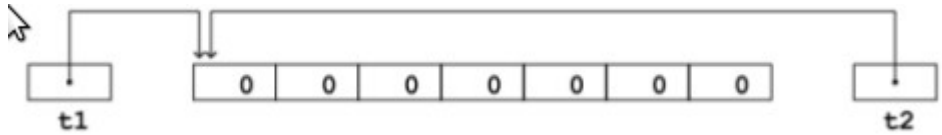
Un tableau est une référence

- **int [] t1,t2;** *//déclaration de deux tableau t1et t2*

- **t1=new int[7];** *//création de 7cases d'entier affecté à t1*



- **t2=t1;***// ne déclenche pas la création d'un nouveau tableau t2 copie de t1, mais permet de pointer t2 sur t1*



6.2. Tableaux à deux dimensions

Java ne supporte pas directement la notion de tableaux à plusieurs dimensions : il faut déclarer un tableau de tableau.

Exemple, pour allouer une matrice de 5 lignes de 6 colonnes :

```
int[][] matrice=new int[5][];
for (int i=0 ; i<matrice.length; i++)
        matrice[i]=new int[6];
```

Java permet de résumer l'opération précédente en :

```
int[][] matrice=new int[5][6];
```

La première version montre qu'il est possible de créer un tableau de tableaux n'ayant pas forcément tous la mêmedimension.

On peut également remplir le tableau à la déclaration et laisser le compilateur déterminer les dimensions des tableaux, en imbriquant les accolades :

```
int[][] matrice =
{
        { 0, 1, 4, 3 } , // tableau [0] de int
        { 5, 7, 9, 11, 13, 15, 17 } // tableau [1] de int
};
```

Pour déterminer la longueur des tableaux, on utilise également l'attribut length :

```
matrice.length // 2
matrice[0].length // 4
matrice[1].length // 7
```

De la même manière que précédemment, on peut facilement parcourir tous les éléments d'un tableau :

```
int i, j;
for(i=0; i<matrice.length; i++) {
	for(j=0; j<matrice[i].length; j++) {
		//Action sur matrice[i][j]
	}
}
```