

Java

Introduction

Plan

1. Le langage
2. Structure générale
3. Expressions
4. Instructions
5. Fonctions
6. Classes et objets

1. Le langage

Le langage de programmation Java (nom dérivé de Kawa) a vu le jour en 1995. C'est un langage

- fortement typé
- orienté objet
- compilé-interprété

De plus,

- il intègre des concepts éprouvés, comme les *thread* ou processus légers;
- il évite des concepts délicats comme l'héritage multiple;
- il bannit des concepts discutables comme les macro-définitions;
- il n'offre pas de généricité (classes paramétrées);
- ce n'est pas un langage fonctionnel, mais il est polymorphe.

Le fichier source d'un programme est *compilé* en un langage intermédiaire (*byte code*) indépendant de la machine cible (procédé connu depuis une vingtaine d'années). Le byte code est *interprétable* sur toute machine possédant une *machine virtuelle Java*. Ce peut être fait

- plus tard,
- ailleurs,
- dans un navigateur compatible Java (ils le sont tous).

Le langage Java vient avec un ensemble important de classes prédéfinies (autour de 1000) qui encapsulent des mécanismes de base, comme

- des structures de données: collections, ensembles, “map”, vecteurs, tables de hachage, grands nombres;
- des outils à la base des communications, comme les URL;
- des facilités audiovisuelles, comme le chargement d’images et la gestion du son;
- un ensemble de composants de création d’interfaces graphiques.

Le langage est pour l’essentiel stabilisé. La version actuelle est 1.3.

2. Structure générale d'un programme

Un programme Java est constitué

- d'un ensemble de directives d'importation comme

```
import java.io.*;
```

- d'un ensemble de déclarations de classes.

Chaque classe contient des *attributs* qui sont

- des variables;
- des fonctions (*méthodes*).
- Ces attributs sont
 - des attributs de classe (**static**);
 - des attributs d'objet (ou d'instance).

Les fonctions contiennent des déclarations de variables locales, et des instructions.

Une fonction spéciale est appelée à l'exécution:

```
public static void main(String[] args) {...}
```

Tout identificateur a un type.

Chaque fonction et chaque donnée font partie d'une classe.

- Une fonction ou donnée de classe est appelée en préfixant son nom du nom de la classe:

`Math.cos()` `Math.PI`

- Une fonction ou donnée d'objet est appelée en la préfixant du nom de l'objet.

La classe courante et l'objet courant peuvent être sous-entendus.

Exemple:

`System.out.println()`

Ici `out` est une donnée de la classe `System`. C'est un objet de la classe `PrintStream` et `println` est une méthode de cet objet.

3. Expressions

Types primitifs

`byte` : un octet.

`short`, `int`, `long` ; entier sur 2, 4, 8 octets respectivement.

`float`, `double` : flottant sur 4, resp. 8 octets.

`char` : caractère, sur 2 octets en “unicode”, permettant de représenter “tous” les alphabets.

`boolean` : booléens, de valeur `true` et `false` seulement.

Une variable se déclare en donnant d’abord son type.

```
int i, j;  
float re, im;  
boolean termine;
```

Une variable `static` est une donnée de classe. Une variable `final static` est une constante.

Une variable qui est attribut d’une classe est initialisée par défaut à 0 pour les variables numériques, à `false` pour les booléennes.

Expressions

```
radians = (degres/180) * Math.PI // no comment
"Bonjour" + " Monde" // String
(i != 0) && (i % 2 == 0) // boolean
x = x +1
```

Conversions implicites par promotion:

```
byte → short → int → long → double.
char → int
int → float → double.
```

Opérations arithmétiques sur `int`, `long` et `double`. D'où conversions explicites nécessaires.

Incrémentations

```
++i    i++    --j    j--
```

Exemple:

```
i = 1;
x = ++i;
j = 3;
y = j--;
```

On obtient `x = i = 2` et `y = 3, j = 2`.

```
j = 0
j += 2; // j = j+2
j += j; // j = j+j
```

Expression conditionnelle

```
max = (a > b) ? a : b;  
(x % 2 == 1) ? 3*x+1 : x/2
```

Tableau

Un *tableau* se *déclare*, se *construit* et s'*utilise*.

Une variable de type tableau est une *référence*. Elle contient l'adresse du tableau. Elle se déclare par

```
int[] a; // vecteur d'entiers  
double[][] m; // matrice de doubles
```

La construction d'un tableau par `new`:

```
a = new int[N] ;  
m = new double[N][P];
```

On s'en sert par

```
x = m[i][j];  
for (i = 0; i < a.length; i++)  
    System.out.print( a[i] );
```

On peut fusionner déclaration et construction:

```
int N = 7;  
String[] jours = {"Lundi", "Mardi", "Mercredi",  
    "Jeudi", "Vendredi", "Samedi", "Dimanche" };
```

Provoque toujours une exception:

```
a[a.length]
```

4. Instructions

Affectation

C'est une expression d'affectation se terminant par un point-virgule.

```
x = 1; y = x = x+1;
```

Instructions conditionnelles

```
if ( $E$ )  $S$  (ne retournent pas de valeur)
```

```
if ( $E$ )  $S$  else  $S'$ 
```

Instruction d'aiguillage (**match**)

```
switch(c) {  
  case ' ':  
    nEspaces++; break;  
  case '0': case '1': case '2': case '3': case '4':  
  case '5': case '6': case '7': case '8': case '9':  
    nChiffres++; break;  
  default:  
    nAutres++  
}
```

break permet la sortie d'instructions.

Instructions d'itération

```
while ( $E$ )  $S$ 
```

```
do  $S$  while  $E$ 
```

```
for( $E_1;E_2;E_3$ )  $S$ 
```

La boucle "pour" est très puissante.

5. Fonctions

Chaque classe contient une suite de fonctions non emboîtées.

```
static int next(int n) {
    if (n % 2 == 1)
        return 3 * n + 1;
    return n / 2;
}
```

La *signature* est la suite des types des arguments.

Une fonction qui ne retourne pas de valeur a pour type de retour le type `void`.

```
static int pgcd(int a, int b) {
    return (b == 0) ? a : pgcd( b, a % b );
}
```

Surcharge

Un même identificateur peut désigner des fonctions différentes pourvu que leurs signatures soient différentes.

```
static int fact(int n, int p) {
    if (n == 0) return p;
    return fact( n-1, n*p);
}
```

```
static int fact(int n) {
    return fact(n, 1);
}
```

6. Classes et objets

Une classe est à la fois

- un ensemble de déclarations de variables et de définitions de fonctions;
- la déclaration d'un type non primitif.

De nombreuses classes sont prédéfinies.

Pour créer des objets d'une classe, on utilise l'opérateur **new**.

```
class Gauss {
    int re;
    int im;

    public static void main(String[] args) {
        Gauss a = new Gauss();
        a.re = 3; a.im = 5;
        ...
    };
}
```

La fonction `Gauss()` est le *constructeur par défaut*. On peut déclarer (par surcharge) des constructeurs appropriés:

```
class Gauss {
    ...
    Gauss (int x, int y) { re = x; im = y; }
    ...
}
```

et écrire

```
public static void main(String[] args) {  
    Gaus a = new Gauss(3,5);  
    ...  
};
```

Les méthodes d'objet s'appliquent à l'objet appelant.

```
class Gauss {  
    ...  
    void imprimer() {  
        System.out.println(re + " " + im);  
    }  
    ...  
}
```

On s'en sert dans

```
a.imprimer();
```

De manière équivalente:

```
static void simprimer(Gauss a) {  
    System.out.println(a.re + " " + a.im);  
}
```

s'utilise par

```
simprimer(a);
```

Faut-il écrire des fonctions d'objet ou des fonctions de classe ?

Objets et types primitifs

- Une variable d'un type primitif désigne une valeur de ce type.
- Une variable d'un type non primitif désigne l'emplacement où se trouve l'objet après sa création. On dit que c'est une *référence*.

C'est pourquoi la déclaration d'une référence, et la création de l'objet sont deux actes distincts.

```
Gauss a; // Declaration d'une variables
int[] m;
a = new Gauss(); // Creation de l'objet
m = new int[45];
```

Après sa déclaration, une variable d'objet contient **null**. Toute référence peut avoir pour valeur **null**, mais **null** ne référnce pas un objet.

Copie de références et copie d'objets.

```
Gauss a, b;
a = new Gauss(3, 4);
b = a ; // Copie de reference : un seul objet
b.re = 5;
```

Ici, **a** et **b** désignent le même objet. En particulier, **a.re** vaut 5. Une copie véritable doit être programmée.