Introduction à Java

Nicolas Sendrier

INRIA, Domaine de Voluceau, BP 105,
78153 Le Chesnay CEDEX, FRANCE

2 mai 2001

Qu'est ce que Java ?

– Un langage interprété. En fait le Java pour les humains est compilé en un pseudo-code qui pourra (en principe) être compris par n'importe quelle machine virtuelle Java (JVM).

  → Avantage : portabilité

Aucun intérêt pédagogique.

Avantage de Java

- un langage orienté objet,
- gestion des exceptions,
- garbage collector,
- syntaxe « confortable »,
- (pas de pointeur,)

- motivation des élèves.

Syntaxe de Java

Composants d'un programme de Java

- les identificateurs,
- les mots-clés,
- les constantes,
- les opérateurs.

Les commentaires : `/* ... */` ou bien `//` $\longrightarrow$ fin de ligne.

Identificateurs

Un *identificateur* est une suite de

- lettres (minuscules ou majuscules),
- chiffres,
- “blanc souligné” (*i.e.* '_'),

ne commençant pas par un chiffre. Il sert à désigner

- un nom de variable,
- un nom de méthode,
- un nom de classe ou d'interface,
- une étiquette.

Un *nom* est une suite d'identificateurs joints par l'opérateur '.'.

Exemple

```
class Bonjour {
    public static void main(String [] args) {
        System.out.println("Bonjour!");
    }
}
```

Les mots-clés

– types primitifs

`boolean` `char` `byte` `short` `int` `long` `float` `double` `void`

– définitions d'objets

`class` `interface`

– modificateurs de type et/ou de classe

`public` `protected` `private` `static` `abstract` `final` `native` `synchronized` `transient` `volatile`

– instructions de contrôle

`if` `else` `for` `do` `while` `switch` `case` `default` `break` `continue` `try` `catch` `finally`

– instructions diverses

`import` `package` `instanceof` `new` `return` `super` `this` `throw` `extends` `implements` `throws`

Constantes

Constante	Type		Constante	Type	
`1234`	`int`		`true`	`boolean`	
`02322`	`int`	octal	`false`	`boolean`	
`0x4D2`	`int`	hexadécimal	`'a'`	`char`	
`123456789L`	`long`		`'\t'`	`char`	
`12.34`	`double`		`'\311'`	`char`	É en ISO 8859-1
`12.3e-4`	`double`		`'\xc9'`	`char`	É en ISO 8859-1
`12.34F`	`float`		`"abcdef"`	`String`	
`12.3e-4F`	`float`		`null`	`Null`	

Caractères spéciaux

\b	retour arrière	\0	caractère nul
\f	saut de page	\'	apostrophe
\n	nouvelle ligne	\"	guillemet
\r	retour chariot	*ooo*	octal
\t	tabulation	\x*hh*	hexadecimal
\\\\	backslash	\u*dddd*	caractère unicode

Les opérateurs

<table>
<tr><td>arithmétiques</td><td>+   -   *   /   %</td></tr>
<tr><td>« bit à bit »</td><td>& | ^   ~   <<   >>   >>></td></tr>
<tr><td>relationnels</td><td><   >   <=   >=   ==   !=</td></tr>
<tr><td>logiques</td><td>&& || !   & | ^</td></tr>
<tr><td>d’affectation</td><td>=   +=   -=   *=   /=   %=   &=   ^= |=
<<=   >>=   >>>=   ++   --</td></tr>
<tr><td>conditionnel (ternaire)</td><td>? :</td></tr>
<tr><td>divers</td><td>.   []   ()   instanceof   new</td></tr>
</table>

Les types primitifs

- `boolean` : deux valeurs `true` et `false`
- `char` : caractères (unicode) sur 16 bits
- entiers (signés) : `byte` (8 bits), `short` (16 bits), `int` (32 bits) et `long` (64 bits)
- flottants : `float` (32 bits) et `double` (64 bits)

Tous les autres types sont des objets, les types primitifs sont passés *par valeur*, les objets sont passés *par référence*.

(La classe `String` est particulière, ce sont des objets pour lesquels il existe des constantes)

Les tableaux

Un tableau de taille n est indexé de 0 à $n-1$. Pour déclarer et allouer un tableau on utilise le mot-clé `new` :

```
int [] t = new int[10];
double [][] m = new double[10][20];
String [] jourSemaine = new String[7];
```

Les tableaux de types primitifs sont initialisés à `0` ou à `false`, les tableaux d'objets sont initialisés à `null`.

Java autorise de façon naturelle l'allocation dynamique :

```
static void foo(int n, int k) {
  int [] t = new int[n * k + 2];
  ...
}
```

En fait la taille du tableau peut être n'importe quelle expression qui s'évaluera en un entier à l'exécution.

Les tableaux (suite)

On peut également allouer un tableau à l'aide de constantes, par exemple :

```
int [] jourParMois = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
  30, 31};

String [] jourSemaine = {"lundi", "mardi", "mercredi", "jeudi",
  "vendredi", "samedi", "dimanche"};

String [] moisAnnee = {"janvier", "février", "mars", "avril",
  "mai", "juin", "juillet", "août", "septembre", "octobre",
  "novembre", "décembre"};
```

Les tableaux (suite)

On peut a tout moment connaître la taille d'un tableau `tab` grâce à l'expression `tab.length`.

Par exemple :

```
static void affiche(int [] tab) {
  for (int i = 0; i < tab.length; ++i)
    System.out.print(tab[i]);
  System.out.println();
}
```

Les opérateurs

arithmétiques : `+` `-` `*` `/` `%`

Pour les entiers, `/` et `%` sont le quotient et le reste de la division euclidienne.

relationnels : `<` `>` `<=` `>=` `==` `!=`

Tous s'évaluent en un `boolean`.

logiques : `&&` (et) `||` (ou) `!` (non)

Les expressions sont évaluées de gauche à droite, et l'évaluation s'arrête dés que le résultat est déterminé. Ainsi le code ci-dessous est correct.

```
int [] p = new int[10];
...
if((i >= 0) && (i < 10) &&  (p[i] != 0))
...
```

Les opérateurs

« **bit à bit** » **:** `&` `|` `^` `~` `<<` `>>` `>>>`

Seuls `&`, `|` et `^` (ou exclusif) s'appliquent à des `boolean`. Toutes les clauses d'une expression sont évaluées.

Ces opérateurs appliqués à des *entiers* agissent sur les bits de leurs *représentations en machine* (bits les moins significatifs à droite) :

- `&`, `|` et `^` s'appliquent bit à bit,
- `~` est le complément,
- `<<` est la multiplication par une puissance de 2 **et** le décalage à gauche,
- `>>` est le quotient de la division par une puissance de 2,
- `>>>` est le décalage à droite,

Représentation des entiers

Un entier représenté sur m bits ($m \in \{8, 16, 32, 64\}$) est un élément de $\mathbf{Z}/2^m\mathbf{Z} = [-2^{m-1}, 2^{m-1}[$.

$x \geq 0$ est représenté en machine par	0 | `v`
$x - 2^{m-1} < 0$ et est représenté par	1 | `v`

`i` $\leftrightarrow i \in [0, m[$

`x` $\leftrightarrow x \in [-2^{m-1}, 2^{m-1}[$

`~x` $\leftrightarrow -x - 1$

`x<<i` $\leftrightarrow x \cdot 2^i$

`x>>i` $\leftrightarrow \mathrm{Quo}(x, 2^i)$

`x>>>i` $\leftrightarrow \mathrm{Quo}(x + 2^m, 2^i)$

où $\mathrm{Quo}(x, y)$ est l'unique entier q tel que $x = q \cdot 2^i + r$ avec $0 \leq r < 2^i$

Syntaxe

expression : ensemble de composants ayant un sens, par exemple
« `x = i % p[i]` » ou « `(i >= 0) && (i < 10) && (p[i] != 0)` »

instruction : *expression* suivie d'un ;

bloc : séquence d'*instructions* ou de *blocs* entre { }.

La portée des variables est le bloc dans lequel elles sont déclarées.

Les branchements conditionnels

```
if ( expression )
    instruction
else if ( expression )
    instruction
...
else if ( expression )
    instruction
else
    instruction
```

```
switch ( expression ) {
    case constante:
      liste d'instructions
      break;
    ...
    case constante:
      liste d'instructions
      break;
    default:
      liste d'instructions
      break;
}
```

Les boucles

```
while ( expression )
    instruction
```

ou bien

```
do
    instruction
while ( expression );
```

```
for ( expression1 ; expression2 ; expression3 )
    instruction
```

équivalent à

```
{
  expression1 ;
  while ( expression2 ) {
      instruction
      expression3 ;
  }
}
```

Méthode `Factorielle.fact()`

```
class Factorielle {

  public static void main (String [] args) {
    if (args.length > 0) {
      int n = Integer.parseInt(args[0]);
      System.out.println(n + "! = " + fact(n));
    }
    else
      System.out.println("Il faut un argument!");
  }

  static long fact (int n) {
    long x = 1;
    for (int i = 1; i <= n; ++i)
      x *= i;

    return x;
  }
}
```

Méthode `Pgcd.euclide()`

```
class Pgcd {
  public static void main(String [] args) {
    int a = Integer.parseInt(args[0]);
    int b = Integer.parseInt(args[1]);

    System.out.println(euclide(a, b));
  }

  static int euclide(int a, int b) {
    int r;
    while (b != 0) {
      r = a % b;
      a = b;
      b = r;
    }
    return a;
  }
}
```

Méthode `Pgcd.euclide()`

```
class Pgcd {
  public static void main(String [] args) {
    int a = Integer.parseInt(args[0]);
    int b = Integer.parseInt(args[1]);

    System.out.println(euclide(a, b));
  }

  static int euclide(int a, int b) {
    if (a < b)
      return euclide(b, a);
    else if (b == 0)
      return a;
    else
      return euclide(b, a % b);
  }
}
```