

Syntaxe

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 4 février 2020

Syntaxe élémentaire

- Instructions terminées par un point virgule
- Blocs d'instructions
- Commentaires

Bloc

```
{  
    instruction1;  
    instruction2;  
}
```

Commentaires

```
//commentaire sur une ligne  
/* commentaire sur  
    plusieurs lignes */
```

Variables

Déclaration variable

```
<type> nom;
```

Affectation variable

```
nom = valeur;
```

Déclaration et affectation variable

```
<type> nom = valeur;
```

⇒ Essayer avec jshell !

Types primitifs et valeurs primitives

Types primitifs

- `boolean`
- `int` (32 bits)
- `double` (64 bits)

Valeurs primitives

- `true`, `false`
- littéral entier : `156`, `100_000`
- littéral flottant : `1.5d`, `2d`

Une fois qu'une variable a une valeur, on peut l'utiliser

Portée

- Variables connues en fonction de leur endroit de déclaration
- Connues dans les blocs inclus
- Inconnues dans les blocs non inclus

Exemple

```
{  
    int a = 2;  
    {  
        int b = 2;  
        // a ?  
    }  
    // a ?  
    // b ?  
}
```

Bonne pratique : adopter portée la plus étroite possible

Portée

- Variables connues en fonction de leur endroit de déclaration
- Connues dans les blocs inclus
- Inconnues dans les blocs non inclus

Exemple

```
{  
    int a = 2;  
    {  
        int b = 2;  
        // a ? Connu  
    }  
    // a ?  
    // b ?  
}
```

Bonne pratique : adopter portée la plus étroite possible

Portée

- Variables connues en fonction de leur endroit de déclaration
- Connues dans les blocs inclus
- Inconnues dans les blocs non inclus

Exemple

```
{  
    int a = 2;  
    {  
        int b = 2;  
        // a ? Connu  
    }  
    // a ? Connu  
    // b ?  
}
```

Bonne pratique : adopter portée la plus étroite possible

Portée

- Variables connues en fonction de leur endroit de déclaration
- Connues dans les blocs inclus
- Inconnues dans les blocs non inclus

Exemple

```
{  
    int a = 2;  
    {  
        int b = 2;  
        // a ? Connu  
    }  
    // a ? Connu  
    // b ? Inconnu  
}
```

Bonne pratique : adopter portée la plus étroite possible

Opérateurs unaires

Opérateurs unaires :

- renvoie le négatif
- ! Négation logique
- ++ Incrmente un entier (affectation implicite)
- Décrmente un entier (affectation implicite)

Exemples

```
int x = -3;  
int z = -(x - y);  
++i;  
boolean b = !true;  
boolean c = !b;
```

Opérations binaires

`+`, `-`, `*`, `/` NB `/` sur entiers \neq `/` sur double : toujours préciser le type pour meilleure lisibilité

`%` reste de la division entière

`<`, `>`, `<=`, `>=` compare et renvoie un booléen

`==` teste égalité et renvoie un booléen

`!=` teste différence

`&&` renvoie vrai ssi les deux opérandes sont vrais (court-circuite)

`||` renvoie vrai ssi au moins un opérand est vrai (court-circuite)

`+=`, `-=` opération puis affectation

Exemple

```
boolean c = (!b && (x > y)) || (a != 3)
```

Tableaux

Tableaux

```
int[] primes = {2, 3, 5, 7, 11, 13, 17};  
int x = primes[0] + 2 * primes[1];
```

```
double[] array = new double[10];  
array[0] = 3d;
```

- Tableaux à plusieurs dimensions possibles
- Dans ce cours on n'utilisera (presque) pas les tableaux

Tests

- `if (test) bloc`
- `if (test) bloc else bloc`
- `if (test) bloc else if (test) bloc else bloc`

Exemple

```
if (value == 3) {  
    openGate = true;  
} else {  
    openGate = false;  
}
```

Amélioration ?

Tests

- `if (test) bloc`
- `if (test) bloc else bloc`
- `if (test) bloc else if (test) bloc else bloc`

Exemple

```
if (value == 3) {  
    openGate = true;  
} else {  
    openGate = false;  
}
```

Amélioration ? `openGate = (value == 3);`

Boucles

While

```
while (openGate) {  
    ...  
}  
do {  
    ...  
} while (openGate);
```

For

```
int a = 4;  
for (int i=0; i < 3; ++i) {  
    a += i;  
}
```

Valeur de a ?

Boucles

While

```
while (openGate) {  
    ...  
}  
do {  
    ...  
} while (openGate);
```

For

```
int a = 4;  
for (int i=0; i < 3; ++i) {  
    a += i;  
}
```

Valeur de a ? 7

Méthodes : utilité

- Tout code doit se trouver dans une méthode
- Méthode peut renvoyer une valeur
- Peut être réutilisée, clarifie le code

Méthodes : syntaxe

- `<type de retour> <nom> (<type param1> <nom param1>, ...)`
- `return` pour renvoyer valeur (exécution méthode cesse)
- `void` pour indiquer absence de retour
- Une telle méthode peut être vue comme une fonction

Exemple

```
double weightedSum(double a, double b,  
    double w1, double w2) {  
    return w1 * a + w2 * b;  
}  
  
void useWeightedSum() {  
    double r = weightedSum(2d, 4d, 0.8d, 0.2d);  
}
```

Valeur de r ?

Méthodes : syntaxe

- `<type de retour> <nom> (<type param1> <nom param1>, ...)`
- `return` pour renvoyer valeur (exécution méthode cesse)
- `void` pour indiquer absence de retour
- Une telle méthode peut être vue comme une fonction

Exemple

```
double weightedSum(double a, double b,  
    double w1, double w2) {  
    return w1 * a + w2 * b;  
}  
  
void useWeightedSum() {  
    double r = weightedSum(2d, 4d, 0.8d, 0.2d);  
}
```

Valeur de r ? 2.4d

Classes

- Toute méthode doit se trouver dans une classe (sauf jshell)
- Syntaxe : `class MyClassName` suivi d'un bloc
- Sert (entre autres) à grouper les méthodes
- En-tête de la méthode commence par `static`

Exemple

```
class MyMathClass {  
    static void sum(double a, double b) {  
        return a + b;  
    }  
    static void mult(double a, double b) {  
        return a * b;  
    }  
}
```

Puis appel avec : `MyMathClass.sum(3d, 1d);`

License

This presentation, and the associated \LaTeX code, are published under the MIT license. Feel free to reuse (parts of) the presentation, under condition that you cite the author. Credits are to be given to Olivier Cailloux, Université Paris-Dauphine.