

Syntaxe

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 20 février 2019

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

Opérations primitives

Opérateurs unaires :

- renvoie le négatif
- ! Négation logique
- ++ Incrémente un entier (affectation implicite)
- Décrémente 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)

`+=`, `-=` affectation et opération

Exemple

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

Tableaux

Tableaux

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

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

Portée

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

Exemple

```
{  
    int a = 3;  
    {  
        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 = 3;  
    {  
        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 = 3;  
    {  
        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 = 3;  
    {  
        int b = 2;  
        // a ? Connue  
    }  
    // a ? Connue  
    // b ? Inconnue  
}
```

Bonne pratique : adopter portée la plus étroite possible

Conditionnelles

- `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 ?

Conditionnelles

- `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

- `public static <type de retour> <nom> (<type param1> <nom param1>, ...)`
- `return` pour renvoyer valeur (exécution méthode cesse)
- `void` pour indiquer absence de retour

Exemple

```
public static int weightedSum (int a, int b,  
    int w1, int w2) {  
    return a * w1 + b * w2;  
}  
  
public static void useWeightedSum () {  
    int result = weightedSum(3, 4, 5, 6);  
}
```

Valeur de result ?

Méthodes : syntaxe

- `public static <type de retour> <nom> (<type param1> <nom param1>, ...)`
- `return` pour renvoyer valeur (exécution méthode cesse)
- `void` pour indiquer absence de retour

Exemple

```
public static int weightedSum (int a, int b,
                               int w1, int w2) {
    return a * w1 + b * w2;
}

public static void useWeightedSum () {
    int result = weightedSum(3, 4, 5, 6);
}
```

Valeur de result ? 39