Programmation orientée objet. Cours 9

Marie-Pierre Béal
BUT 1

Structures de contrôle et chaînes de caractères

Structures de contrôle

`If, ?:, for, while, do ... while`

Les conditions des `if, for, while` doivent être des `boolean`

Par convention, on définit toujours un bloc même si il y a une seule instruction.

```
if (foo == 4) {
  IO.println("ok!");
}
```

Java est plus pointilleux que d'autres langages

Java ne permet pas d'écrire des expressions qui ne font rien.

```
var a = 3;
a + 3;   // ne compile pas
```

On ne peut pas non plus écrire du code après un return, `break/continue`, `throw` ou `yield`.

```
...
return 3;
var a = 3;   // ne compile pas
```

Initialisation des variables

Lorsqu'une variable est utilisée, elle doit être initialisée pour tous les chemins d'exécution possibles.

```
int m(int value) {
  int result;
  if (value < 10) {
    result = 3;
  }
  return result;  // ne compile pas car il y a un chemin
                  // où "result" n'est pas initialisé
                  // il faut ajouter un else avec un bloc
}
```

Switch

Le switch à flèches de Java

Le switch de Java est mieux que celui du C.
On utilise "->" et pas ":", pas besoin de break. Si on a plusieurs instructions, on doit utiliser un bloc.

```
int seats = ...
String type;
switch(seats) {
  case 1, 2 -> type = "small";
  case 3, 4, 5 -> {
    IO.println("debug");
    type = "medium";
  }
  default -> type = "big";  // obligatoire sinon erreur !
}
IO.println(type);
```

Le switch expression

Un switch peut aussi envoyer une valeur

- Lorsque l'on utilise un bloc, on sort avec yield

Le même exemple

```
int seats = ...
var type = switch(seats) {
  case 1, 2 -> "small";
  case 3, 4, 5 -> {
   IO.println("debug");
    yield "medium";
  }
  default -> "big";  // obligatoire sinon erreur !
};  // <- ne m'oubliez pas
IO.println(type);
```

Les chaînes de caractères

`java.lang.String`

Les chaînes de caractères en Java sont représentées par la classe `String`.

La classe est non-mutable.

- On peut passer une `String` en paramètre, on est sûr qu'elle ne sera pas modifiée
- Si on veut changer un des caractères, il faut recréer une `String`

`equals/hashCode/toString`

String possède les méthodes

- `equals()`, teste si deux chaînes sont égales en comparant les longueurs puis les caractères
- `hashCode()`, renvoie un entier "résumé" de la chaîne de caractères
- `toString()` renvoie this

On compare des String avec `equals()`, pas avec ==.

Méthodes de java.lang.String

Méthodes les plus courantes

- `s.length()`
  - obtenir la taille
- `s.charAt(index)` (attention pas de `s[index]`)
  - obtenir le caractère à l'index (commence à 0)
- `s.repeat(times)`
  - répète la même chaîne "times" fois

Méthodes les plus courantes

- `s1.compareTo(s2)`
  - < 0 si s1 plus petit que s2, > 0 si s1 plus grand que s2, == 0 si égales
- `s.startsWith(prefix), s.endsWith(suffix)`
  - est-ce que cela commence par un préfixe, finit par un suffixe

Extraire une sous-chaîne

- `s.indexOf(char)/lastIndexOf(char)`
  - index du caractère par le début/la fin ou -1 si pas trouvé
- `s.substring(beginIndex)`
  - extrait la sous-chaîne à partir `beginIndex` inclus
- `s.substring(beginIndex, endIndex)`
  - extrait la sous-chaîne entre `beginIndex` et `endIndex` (non compris)
  - contrairement à Python, les valeurs négatives ne sont pas permises

split/join

`s.split(regex)` découpe une chaîne de caractères en un tableau suivant une expression régulière.

```
var array = "foo,bar".split(",");
IO.println(array[0]);  // foo
IO.println(array[1]);  // bar
```

`String.join(delimiter, elements)` agrège les éléments d'un tableau avec un délimiteur.

```
var joined = String.join("-", array);
IO.println(joined);  // foo-bar
```

Majuscule / Minuscule

Il ne faut pas oublier d'indiquer `Locale.ROOT`

- `toUpperCase(Locale.ROOT)`
- `toLowerCase(Locale.ROOT)`

Pour comparer des String indépendamment de la casse

- `string1.equalsIgnoreCase(string2)`

Conversion
String <-> type primitif

Parsing

Convertir une String en un type primitif

Plein de méthodes statiques

- `Boolean.parseBoolean(text)`
  - Convertit une chaîne de caractères en boolean
- `Integer.parseInt(text)`
  - Convertit une chaîne de caractères en int
- `Double.parseDouble(text)`
  - Convertit une chaîne de caractères en double

Attention à ne pas confondre `Boolean` et `boolean`, le premier est une classe contenant les méthodes de conversion, le second est le type primitif.
Ne jamais écrire `Boolean b = true;`

Convertir en String

Deux façons :

- Méthodes statiques
  - `Boolean.toString(boolean value)`
  - `Integer.toString(int value)`
  - `Double.toString(double value)`
- Utiliser le + avec la chaîne vide
  - `"" + value`
    C'est la façon idiomatique

Switch sur les String

Switch sur les String

On peut faire un switch sur des String

```
public int computeTax(String vehicle) {
  Objects.requireNonNull(vehicle);
  return switch(vehicle) {
    case "bicycle" -> 10;
    case "car", "sedan" -> 20;
    case "bus" -> 40;
    default -> throw new IAE("unknown " + vehicle);
  };
}
```

Remplacer `IAE` par `IllegalArgumentException`.

Concaténation

Opérateur +

Si on a une String à gauche ou à droite d'un +, le compilateur fait une concaténation et renvoie la String résultante.

```
var s = "hello";
IO.println(s + 3);   // hello3
IO.println(3 + s);   // 3hello
```

Une concaténation implique une allocation d'une nouvelle String.

StringBuilder

```
@Override
public String toString() {
  var builder = new StringBuilder();
  var separator = "";
  for(var item: list) {
    builder.append(separator).append(item);
    separator = "\n";
  }
  return builder.toString();
}
```

Pas de + dans les append()

Si on met un + dans les append(), on ajoute une allocation dans le append()

```
@Override
public String toString() {
  var builder = new StringBuilder();
  for(var item: list) {
    builder.append(item + "\n");   // ahhhhhh
  }
  return builder.toString();
}
```

Formatting

Utiliser le format printf

Attention, comparativement à la concaténation, ces méthodes sont super lentes (comme printf en C)

- L'équivalent de sprintf
  - avec une méthode statique
    - `String.format(String format, Object... objects)`
  - avec une méthode d'instance
    - `format.formatted(Object... objects)`

```
static void main() {
  var name = "toto";
  IO.println(String.format("hello %s", name)); // hello toto
  IO.println("hello %s".formatted(name)); // hello toto
}
```