Introduction à la programmation en JAVA

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Table des matières

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda

- Java ne propose pas la possibilité de définir une fonction/méthode en dehors d'une classe ni de passer une telle fonction en paramètre d'une méthode. Depuis Java 1.1, la solution pour passer des traitements en paramètres d'une méthode est d'utiliser les classes anonymes internes.

- Pour faciliter, entre autres, cette mise à œuvre, Java 8 propose les expressions lambda. Les expressions lambda sont aussi nommées closures ou fonctions anonymes : leur but principal est de permettre de passer en paramètre un ensemble de traitements.

- De plus, la programmation fonctionnelle est devenue prédominante dans les langages récents. Dans ce mode de programmation, le résultat de traitements est décrit mais pas la façon dont ils sont réalisés. Ceci permet de réduire la quantité de code à écrire pour obtenir le même résultat.

Les expressions lambda – Simplifications

- Gestion d'un évènement avec classe anonyme:

```
monBouton.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent event) {
        System.out.println(event);
    }

});
```

- Simplification de la classe anonyme avec une lambda:

```
monBouton.addActionListener(event -> System.out.println(event));
```

- Simplification de la lambda par le passage d'une fonction en paramètre:

```
monBouton.addActionListener(System.out::println);
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda – Simplifications

- La mise en œuvre de cette fonctionnalité requiert deux fonctionnalités de Java 8 :
  - Une méthode (ici actionPerformed) qui attend en paramètre une interface fonctionnelle (ActionListener).
    - Ce n'est donc pas possible avec une interface qui a plusieurs méthodes.
  - Les expressions lambda
    - A la condition qu'il n'y ait aucune ambiguïté quand au choix de la méthode.

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda - Simplifications

- Autre exemple de simplification de lambda:

```
a -> a.toLowerCase();
```

- Après:

```
String::toLowerCase;
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda – Ambiguïté

- Exemple d'ambiguïté sur le choix de la méthode:

```
public interface Processor {
    String process(Callable<String> c) throws Exception;
    String process(Supplier<String> s);
}

public class ProcessorImpl implements Processor {
    @Override
    public String process(Callable<String> c) throws
Exception {
        // implementation details
    }

    @Override
    public String process(Supplier<String> s) {
        // implementation details
    }
}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda - Ambiguïté

- Erreur de compilation, *the reference to process is ambigus*:

```
String result = processor.process(() -> "abc");
```

- Par ce que le compilateur ne sait pas quelle méthode nommée *process* il doit choisir. Elles sont deux à porter ce nom et deux à n'avoir qu'un seul argument.

- Remarquez que le *return* est devenu implicite.

Les expressions lambda – Simplifications

- Sans une lambda:

```
for (int i = 0; i < list.size(); i++) {
    System.out.println(list.get(i));
}
```

- Avec une lambda:

```
list.forEach(System.out::println);
```

  - La méthode forEach n'est pas créée par nous, elle vient de l'API Java
  - forEach attend une interface fonctionnelle de type *Consumer*

- Il est également possible de faire:

```
List<String> list = new ArrayList<>();
Consumer<String> afficher = param -> System.out.println(param);
list.forEach(afficher);
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda – Simplifications

- Si on ne souhaite pas créer une interface juste pour faire une lambda, li existe deux interfaces génériques de l'API Java.
  - On spécifie le type d'argument et le type de retour:

```
Function<Integer, String> additionner = val1 -> val1.toString();
```

  - On spécifie les deux types d'arguments et le type de retour:

```
BiFunction<Integer, Integer, Long> additionner = (val1, val2) -> (long) val1 + val2;
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Les expressions lambda - Exercices

1. Reprendre l'exercice avec les villes et pays et remplacer la classe anonyme qui sert de comparateur par une expression lambda.

2. Essayer de remplacer la classe anonyme de *TimeTask* de l'exercice avec la génération de données météorologiques par une expression lambda. Cherchez pourquoi vous avez des difficultés à implémenter la solution.

3. Créer une lambda qui retourne un boolean pour préciser si l'entier passé en argument est pair ou impair.

4. Créer une lambda qui précise si le premier argument est plus petit que le second.