CSC4509 — Interface fonctionnelle et expression lambda

Éric Lallet

Télécom SudParis

26 avril 2021

Interface fonctionnelle

JAVA 8 a introduit les interfaces fonctionnelles :

- ce sont des interfaces ;
- elles n'ont qu'une seule méthode abstraite ;
- elles peuvent définir des méthodes par défaut ;
- elles peuvent être annotées par @FonctionalInterface pour annoncer leur contrat (que le compilateur vérifie).

Interface fonctionnelle

La package *java.util.function* contient des interfaces fonctionnelles prédéfinies. Voici quelques exemples :

Interface	Méthode
Function<T,R>	R apply(T t)
BiFunction<T,U, R>	R apply(T t, U u)
Consumer<T>	void accept(T t)
BiConsumer<T,U>	void accept(T t, U u)
Supplier<T>	T get()
Predicate<T>	boolean test(T t)

Usage d'une interface fonctionnelle

Pour créer un objet qui implémente une interface fonctionnelle il existe trois moyens :

1. Écrire une nouvelle classe nommée qui implémente la méthode, et créer un l'objet à partir de cette classe ;
2. Créer l'objet à partir d'une classe anonyme qui définit la méthode ;
3. Écrire un expression lambda qui fournit le code de la méthode.

Exemple d'une interface fonctionnelle

`Activite<T>` est une interface fonctionnelle qui donne une *durée()* :

```
@FunctionalInterface
public interface Activite<T> {
  /**
   * @param t
   *          loisir de l'activite.
   * @return
   *          duree en minutes.
   */
  int getDuree(T t );
}
```

Elle est paramétré par des classes qui décrivent des loisirs comme :

```
class Livre() {
 private int nbPages;
 (...)
 public int getNbPages() {
   return nbPages;
 }
}
```

```
class Film () {
 private int duree;
 (...)
 public in getDuree() {
   return duree;
 }
}
```

Implémentation avec une nouvelle classe

Implémentation avec l'écriture d'une nouvelle classe.

```
class Lecture implements Activite<Livre> {
  @Override
  public int getDuree(Livre livre) {
      return livre.getNbPages() * 10;
  }
}
```

Création d'une activité :

```
Livre livre = new Livre( ... );
Activite<Livre> activite = new Lecture();
System.out.println("L'activite dure " +
          activite.getDuree(livre) + " minutes");
```

Implémentation avec classe anonyme

Implémentation avec une classe anonyme : on crée l'objet en faisant un new sur l'interface, en lui ajoutant la méthode qu'il faut implémenter.
Création d'une activité :

```
Livre livre = new Livre( ... );

Activite<Livre> activite = new Activite<Livre>() {
 @Override
 public int getDuree(Livre livre) {
     return livre.getNbPages() * 10;
 }
};

System.out.println("L'activite dure " +
          activite.getDuree(livre) + " minutes");
```

Implémentation avec une expression lambda

Sur le code précédent beaucoup d'informations peuvent être déduite du contexte. Les seules informations nécessaires sont les paramètres de la méthode et le code de la méthode. Ce sont ces informations que fournissent l'expression lambda :

```
Livre livre = new Livre( ... );

Activite<Livre> activite = (Livre livre) -> {
     return livre.getNbPages() * 10;
 };

System.out.println("L'activite dure " +
          activite.getDuree(livre) + " minutes");
```

Implémentation avec une expression lambda

Lorsque le code de l'expression lambda ne contient qu'une seule instruction, il est possible de retirer aussi les «{}» et le return. Si la méthode a besoin d'un résultat, la valeur retournée sera la valeur calculée par l'instruction :

```
Livre livre = new Livre( ... );

Activite<Livre> activite = (Livre livre) -> livre.getNbPages() *

System.out.println("L'activite dure " +
            activite.getDuree(livre) + " minutes");
```