Chapitre 13
Expressions lambda

Java's first step into functional programming

Expression lamda

- A partir de Java 8
- Fonction qui peut être créée sans appartenir à une classe
  - Ressemble à une déclaration de méthode sans nom
  - Peut être passée comme un objet et être exécutée à la demande

Syntaxe des expressions lamda

- Liste d'arguments
  - Séparés par des virgules
  - Entre parenthèses
    - Si un seul argument: parenthèses facultatives
- -> (flèche)
- Corps
  - Soit une expression
  - Soit un bloc d'instructions entre entre { }

Exemple d'expressions lamda

- **(a,b,c) -> (a+b)*c**

  A la place d'une méthode déclarée avec

  - Arguments : 3 int
  - Type de retour : 1 int

- **(book1, book2) ->**

  { if (book1.getPagesCount() > book2.getPagesCount()) return book1; else return book2; }

  A la place d'une méthode déclarée avec

  - Arguments : 2 objets de la classe Book
  - Type de retour : 1 objet de la classe Book

Exemple d'expressions lamda

- **p -> p.getGender() = = Person.Sex.MALE && p.getAge() >= 18 && p.getAge() <= 25**

  A la place d'une méthode déclarée avec

  - Argument : 1 objet de la classe Person
  - Type de retour : 1 booléen

- **email -> System.out.println(email)**

  A la place d'une méthode déclarée avec

  - Argument : 1 String
  - Type de retour : void

Interface fonctionnelle

- Ne contient qu'une seule déclaration de méthode

- Une expression lambda peut s'utiliser là où on attend le code d'une méthode

  ⇨ Là où une interface fonctionnelle est déclarée comme argument

Exemple d'interface fonctionnelle

```
public interface OperationInterface
   {   int operation (int a, int b);   }
```

→ Interface fonctionnelle car ne contient qu'une méthode

```
public class Calculator
  { public int calculate (int a, int b, OperationInterface interface)
          {return interface.operation (a, b); }
```

→ Classe qui utilise l'interface

Sans expression lambda

⇨ *Il faut créer des classes qui implémentent l'interface pour pouvoir appeler la méthode calculate*

```
public class Addition implements OperationInterface
    { @Override
        public  int operation (int a, int b) { return a + b; }
    }

public class Subtraction implements OperationInterface
    {@Override
        public  int operation (int a, int b) { return a - b; }
    }
```

→ Classes qui implémentent l'interface

Sans expression lambda

Appel de la méthode calculate: création d'objets de type Addition et Subtraction

```
public static void main(String... args) {
    Calculator calculator = new Calculator();

    Addition addition = new Addition();
    System.out.println("40 + 2 = " + calculator.calculate (40, 2, addition));

    Subtraction subtraction = new Subtraction();
    System.out.println("20 - 10 = " + calculator.calculate (20, 10, subtraction));
}
```

Avec expression lambda

⇨ *Pas de création de classes qui implémentent l'interface*

```
public interface OperationInterface
   {   int operation (int a, int b);   }
```

→ Interface fonctionnelle

```
public class Calculator
  { public int calculate (int a, int b, OperationInterface op)
          { return op.operation (a, b); }
```

→ Classe qui utilise l'interface

```
public static void main(String... args) {

    Calculator calculator = new Calculator();

    System.out.println("40 + 2 = " + calculator.calculate (40, 2, (a, b) -> a + b ) );

    System.out.println("20 - 10 = " + calculator.calculate (20, 10, (a, b) -> a – b ) );
```

Expressions lambda