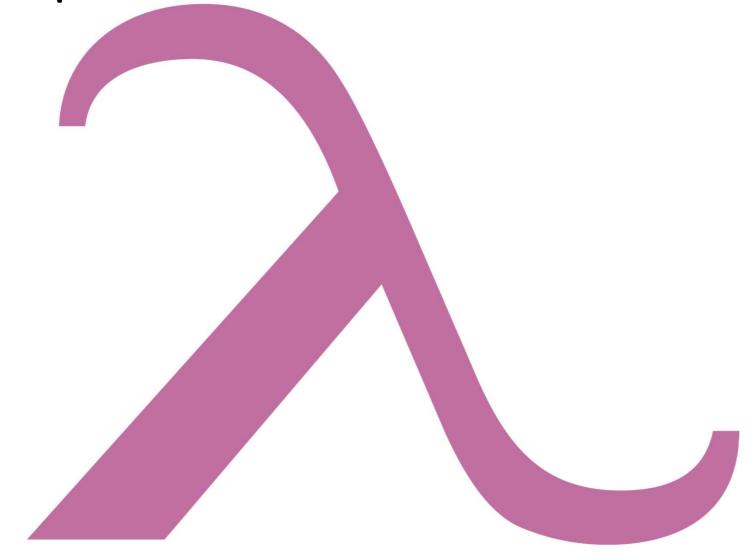
## Les expressions Lambda dans Java 8



# Les expressions Lambda, c'est quoi et pourquoi?

- Issues du lambda-calcul: formalisme mathématique, inventé par Alonzo Church en 1930, pour caractériser les fonctions.
- Objectifs: passer en paramètre un ensemble de traitements.
- On parle **d'expressions Lambda** ou **fonctions anonymes** (pas de nom, pas de modificateur d'accès, pas de déclaration explicite de type de retour).

### Expressions lambda vs classes anonymes

```
Button btn = new Button();
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent evenement) {
        System.out.println("allo");
    }
});

// depuis Java 8, on peut écrire:
btn.setOnAction(evenement -> System.out.println("allo"));
```

### Les expressions lambda: syntaxe

```
    (parametre1, prametre2, ....) -> expression;
    (parametre1, prametre2, ....) { ensemble d'instructions...};
    // trois expressions lambda syntaxiquement équivalentes:
btn.setOnAction((ActionEvent evenement) -> {System.out.println("allo");});
    btn.setOnAction((evenement) -> System.out.println("allo"));
```

btn.setOnAction(evenement -> System.out.println("allo"));

#### Portée des variables

Le corps d'une expression lambda peut utiliser:

- > les variables passées en paramètre de l'expression
- Les variables déclarées dans les corps d'une expression
- Les variables déclarées *final* dans le bloc englobant
- Les variables effectivement *final* déclarées dans le bloc englobant: ces variables ne sont pas déclarées *final*, mais une valeur leur est assignée et elles ne sont jamais modifiées.

## Utilisation des expressions lambda

- Comment affecter une expression lambda à une variable?
- Comment créer une méthode qui accepte des expressions Lambda en paramètres?
- Pour répondre à ces deux questions, nous devons d'abord comprendre la notion d'interface fonctionnelle.

#### Les interfaces fonctionnelles

• Une interface fonctionnelle est une interface qui contient une seule méthode déclarée abstraite (abstract).

• Exemple:

```
public interface TypeOperation {
public abstract int calculer(int x,int y);
}
```

#### Les interfaces fonctionnelles

```
package application;
public class TestInterfaceFonctionnelle {
    public static void main(String[] args) {
        TypeOperation addition = (x,y)-> x+y;
        TypeOperation multiplication = (x,y) \rightarrow x*y;
        System.out.println("addition: "+ addition.calculer(3, 4) );
        System.out.println("multiplication: "+ multiplication.calculer(3, 4));
```

## L'ensemble de classes java.util.function

Cet ensemble propose des interfaces fonctionnelles d'usage courant.

- Function: fonction qui accepte un paramètre et fournit un résultat
- Predicate: fonction dont le résultat est boolean
- Consumer: Prend un argument et ne renvoie aucun résultat
- Supplier: renvoie une valeur sans paramètres en entrée.
- ....

## Les expressions lambda et les collections en Java 8

 Nouvelle API java.util.stream.\* La classe Stream permet d'effectuer une série d'opérations (pipeline) sur des données.

 La série d'opérations est une séquence d'expressions lambda/interfaces fonctionnelles qui modifie ou consulte chacune des données.

## Exemple avec Stream, opérations d'agrégat

```
// création d'une liste de valeurs
List <Integer> listeEntiers = Arrays.asList(23,85,60,67, 90, 22, 55);

System.out.println(listeEntiers);

// déclaration d'une variable non locale qui sera utilisée dans une lambda-expression int seuil = 55;

System.out.println(" valeurs plus grandes que " + seuil+ " converties en hexadecimal");
listeEntiers.stream().filter(valeur-> valeur>=seuil) // utilisation d'un prédicat
.sorted() // tri les éléments filtrés
.map(valeur->Integer.toHexString(valeur).toUpperCase())// transforme en hexadécimal
.forEach(valeur-> System.out.println(valeur)); //affiche les valeurs.
```

#### Résultat de l'exécution

```
[23, 85, 60, 67, 90, 22, 55]
valeurs plus grandes que 55 converties en hexadecimal
37
3C
43
55
```

## Les méthodes default dans les interfaces

- Les méthodes *default* d'une interface ont une implémentation (depuis java 8 seulement).
- Les comportements *default* des interfaces sont automatiquement hérités par les classes qui implémentent ces interfaces.
- Avantage: on peut ajouter des comportements (méthodes) à des interfaces sans avoir à recompiler toutes les classes qui les implémentent.
- Noter que la méthode stream(), utilisée dans l'exemple précédent est une méthode *default* de *l'interface Collection* de Java 8.