**LES EXPRESSIONS LAMBDA**

Table des matières

[Chapitre 1 2](#_Toc91001280)

[Titre 1 2](#_Toc91001281)

[Titre 2 2](#_Toc91001282)

[Chapitre 2 2](#_Toc91001283)

[Titre 1 2](#_Toc91001284)

[Chapitre 3 2](#_Toc91001285)

[Titre 1 2](#_Toc91001286)

[Chapitre 4 2](#_Toc91001287)

[Titre 1 2](#_Toc91001288)

[Chapitre 5 2](#_Toc91001289)

[Titre 1 2](#_Toc91001290)

[Chapitre 6 2](#_Toc91001291)

[Titre 1 2](#_Toc91001292)

# 

# INTRODUCTION

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.

**Une expression lambda** est donc une fonction qui peut être créée sans appartenir à aucune classe. Une expression lambda peut être transmise comme s’il s’agissait d’un objet et exécutée à la demande.

Pourquoi utiliser les expressions lambda :

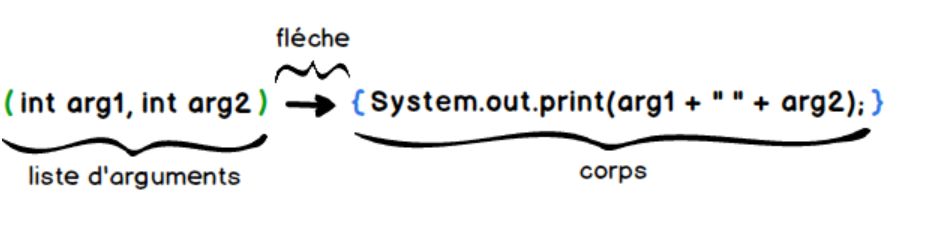
* Pour fournir l’implémentation d’une interface fonctionnelle.
* Moins de code.

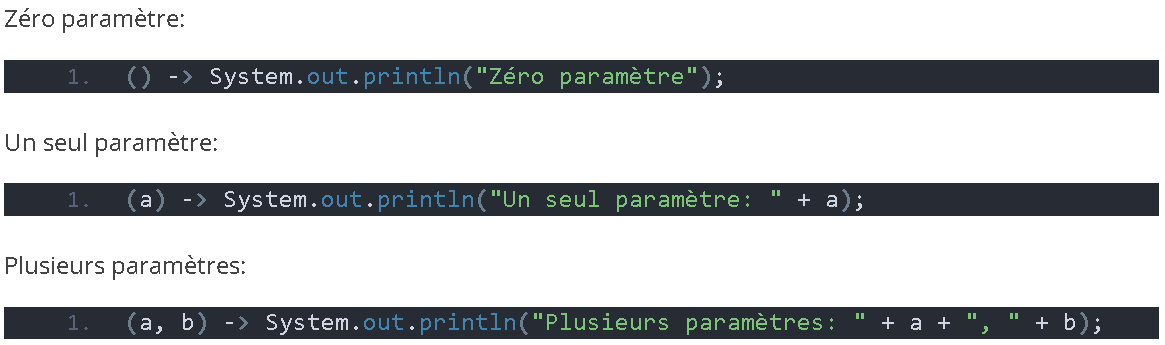
# Interface fonctionnelle

L’expression lambda fournit l’implémentation d’une interface fonctionnelle. Une interface qui n’a qu’une seule méthode abstraite est appelée une interface fonctionnelle. Java fournit une annotation @FunctionalInterface, qui est utilisée pour déclarer une interface comme interface fonctionnelle.

# : SYNTHAXE

**(Liste d'arguments) -> {corps}**





# 

## Titre 1

# 

## Titre 1

# 

## Titre 1

# 

## Titre 1