Cheatography

JAVA Lambda Cheat Sheet

by Peanuts-83 via cheatography.com/140542/cs/47774/

Optional

```
// L'interface Optional permet de gérer les cas de null pointeur de manière élégante.
Option al< Pro duc t> findPr odu ct( pro ducts, product → produc t.g etN ame ().e qu als ("t oma to"))
.map(P rod uct ::g etP rice) // method ref
.ifPre sen tOr Else(
       // 1.Consumer if !empty
        price → System.ou t.p rin tf( "Price of tomato is € %.2f%n",, price),
       // 2.Runnable if empty
       () → System.ou t.p rin tf( "Tomato is not availa ble")
);
```

andThen | compose

```
// Combinaison de 2 lambdas via ces operateurs équivalents. Seul l'ordre des lambdas varie.
// lambda 1
Functi on< Pro duct, BigDec ima l> produc tTo Price = Produc t:: get Price;
// lambda 2
Functi on< Big Dec imal, String> priceT oMe ssage = price → "Tomato price is € " + price;
// 1.andT hen(2)
Function <Pr oduct, String> produc tTo Message = produc tTo Pri ce.a nd Then( priceT oMe ssage);
// 2.comp ose(1)
Function <Pr oduct, String> produc tTo Mes sage2 =price ToM ess age.co mpo se( pro duc tTo Price);
```

Introduction

Fonction anonyme qui implémente une interface fonctionelle.
Consumer<String> consumer = (s) -> System.out.println(s);
Structure
- **Parametres** - optionnels
- **->** - opérateur flêche
- **Corps** - Bloc de code
La lambda capture les variables locales, à condition qu'elle soient **effectivement finales**. Elle récupère le **this** et **super** du context.
Elle ne throw pas d'exception, l'utilisation d'un **try/catch** est recommandé.
Elle peut être remplacée par une **method reference** pour les cas simples (Class::method | object::method | Class::new).

Interface fonctionnelle - Single Abstract Method

Interface disposant d'un seule méthode abstraite.
Pour s'assurer de la validité d'une interface fonctionnelle, on peut utiliser l'annotation **@FunctionalInterface**.
Ces interfaces existent aussi pour **2 types** traités (BiFunction, BiConsumer etc...) sauf Supplier.
On trouvera aussi des **interfaces spécialisées** par type (IntFunction, IntBinaryOperator, LongToDoubleFunction, etc...).

Interfaces fonctionnelles principales

Nom	Méthode	In/Out
Function<T,R>	R apply(T value)	T -> R
Consumer<T>	void accept(T value)	T -> void
Supplier<T>	T get()	void -> T
Comparator<T>	int compare(T val1, T val2)	T -> int
Optional<T>	<T|empty> filter(T value)	T -> <T|empty>
Predicate<T>	boolean test(T value)	T -> boolean

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
http://crosswordcheats.com