

### Optional

```
// L'interface Optional permet de gérer les cas de null pointeur de manière élégante.
Optional<Product> findProduct( products, product -> product.getName().equals("tomato"))
.map(Product::getPrice) // method ref
.ifPresentOrElse(
    // 1.Consumer if !empty
    price -> System.out.printf( "Price of tomato is € %.2f\n",, price),
    // 2.Runnable if empty
    () -> System.out.printf( "Tomato is not available")
);
```

### andThen | compose

```
// Combinaison de 2 lambdas via ces operateurs équivalents. Seul l'ordre des lambdas varie.
// lambda 1
Function<Product, BigDecimal> productToPrice = Product::getPrice;
// lambda 2
Function<BigDecimal, String> priceToMessage = price -> "Tomato price is € " + price;
// 1.andThen(2)
Function<Product, String> productToMessage = productToPrice.andThen( priceToMessage);
// 2.compose(1)
Function<Product, String> productToMessage2 =priceToMessage.compose( productToPrice);
```

## Introduction

Fonction anonyme qui implémente une interface fonctionnelle.

```
Consumer<String> consumer = (s) -> System.out.println(s);
```

### Structure

- **Paramètres** - optionnels

- **->** - opérateur flèche

- **Corps** - Bloc de code

La lambda capture les variables locales, à condition qu'elle soient **effectivement finales**. Elle récupère le **this** et **super** du contexte.

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   |

