Interfaces de fonctions

@FunctionalInterface

- Mettre en haut de la déclaration d'une interface.
- @FunctionalInterface

  interface Square {

  int calculate(int x);

  }
- Erreur de compilation si plus d'une méthode abstraite

Quatre @FunctionalInterface vue :

- Supplier<T> : T get()
- Consumer<T> : void accept(T t)
- Predicate<T> : boolean test(T t)
- Function<T, R> : R apply(T t)
- Bonus :
  - BiFunction<T, U, R> : R apply(T t, U u)

Lambdas : syntaxe

```
FunctionalInterface variable =
    ([type] param[, [type] param] ...) -> expression
    ou
    ([type] param[, [type] param] ...) -> {
        instructions;
        [return valeur;]
    };
```

(*) Si un seul paramètre et sans type explicite, ceci est possible :

```
    ParamUnique -> ...
```

(*) Si aucun paramètre : () → ...

Lambdas
(fonctionnent seulement avec les @FunctionalInterface):

- Exemple sur Predicate<T> :

  Predicate<Integer> monTest = (int x) → {

  return x > 5;

  };

- Utilisation : pour un ArrayList<Integer> monArray

  monArray.removeif(monTest) enlève les entier > 5

Lambdas
(fonctionnent seulement avec les @FunctionalInterface):

- Exemple sur Predicate<T> :

  Predicate<Integer> monTest = (int x) → {

  return x > 5;

  };

- Utilisation : pour un ArrayList<Integer> monArray

  monArray.removeif(monTest) enlève les entier > 5

Lambdas (fonctionnent seulement avec les @FunctionalInterface):

- Exemple sur Supplier<T> :

```
Supplier<Integer> monGetter = () → {
    return 5;
};
```

Lambdas (fonctionnent seulement avec les @FunctionalInterface):

- Exemple sur Consumer<T> :

```
Consumer<Integer> consumer = (int x) → {
    System.out.println(x);
};
```

Lambdas (fonctionnent seulement avec les @FunctionalInterface):

- Exemple sur Function<T, R> :

```
Function<Integer, Integer> doubler = (int x) → {
    return x * 2;
};
```