Interfaces fonctionnelles

Définition 2

Liste des interfaces 2

- (T, U) → V : BiFunction 2
- () -> void : java.lang.Runnable 2
- () -> T : Supplier<T> 2
  - [Int|Long|Double]Supplier 2
- (T) -> void : Consumer 2
  - [Int|Long|Double]Consumer 2
- (T) -> boolean : Predicate 3
  - [Int|Long|Double]Predicate 3
- (T) -> U : Function<T, U> 3
  - [Int|Long|Double]Function<T> 3
  - To[Int|Long|Double]Functio<T> 3
- (T) -> T : UnaryOperator 3
  - [Int|Long|Double]UnaryOperator 3
- (T, U) -> boolean : BiPredicate 3
- (T, U) -> V : BiFunction 4
- (T, T) -> T : BinaryOperator<T> 4
  - [Int|Long|Double]BinaryOperator 4

Définition

Une interface fonctionnelle est une interface avec une seule méthode abstraite.

Liste des interfaces

(T, U) → V : BiFunction

```
BiFunction<String, Integer, Double> fun =
(String s, Integer i) → s.length() + i + 5.0;
fun.apply("TOTO", 10);
```

Méthode :

```
public V apply(T t, U u);
```

() -> void : java.lang.Runnable

```
Runnable code = () -> { System.out.println("hello"); }
code.run();
```

() -> T : Supplier<T>

```
Supplier factory = () -> "hello";
System.out.println(factory.get());
```

[Int|Long|Double]Supplier

```
IntSupplier factory = () -> 42;
System.out.println(factory.getAsInt());
```

(T) -> void : Consumer

```
Consumer printer = s -> System.out.println(s);
 printer.accept("hello");
```

[Int|Long|Double]Consumer

```
DoubleConsumer printer = d -> System.out.println(d);
printer.accept(42.0);
```

(T) -> boolean : Predicate

```
Predicate isSmall = s -> s.length() < 5;
System.out.println(isSmall.test("hello"));
```

[Int|Long|Double]Predicate

```
LongPredicate isPositive = v -> v >= 0;
System.out.println(isPositive.test(42L));
```

(T) -> U : Function<T, U>

```
 Function fun = s -> "hello " + s;
System.out.println(fun.apply("function"));
```

[Int|Long|Double]Function<T>

Type d’argument que prend la fonction

```
IntFunction arrayCreator = size -> new String[size];
System.out.println(arrayCreator.apply(5).length);
```

To[Int|Long|Double]Functio<T>

Type de la valeur de retour de la fonction

```
ToIntFunction stringLength = s -> s.length();
System.out.println(stringLength.applyAsInt("hello"));
```

(T) -> T : UnaryOperator

```
UnaryOperator op = s -> "hello " + s;
System.out.println(op.apply("unary operator"));
```

[Int|Long|Double]UnaryOperator

```
IntUnaryOperator negate = x -> - x;
System.out.println(negate.applyAsInt(7));
```

(T, U) -> boolean : BiPredicate

```
BiPredicate isPrefix = (s, prefix) -> s.startsWith(prefix);
System.out.println(isPrefix.test("hello", "hell"));
```

(T, U) -> V : BiFunction

```
BiFunction concat = (s1, s2) -> s1 + " " + s2;
System.out.println(concat.apply("hello", "Bob"));
```

(T, T) -> T : BinaryOperator<T>

```
BinaryOperator concat = (s1, s2) -> s1 + " " + s2;
System.out.println(concat.apply("hello", "binop"));
```

[Int|Long|Double]BinaryOperator

```
IntBinaryOperator add = (a, b) -> a + b;
System.out.println(add.applyAsInt(40, 2));
```