



Java 8

CHEATSHEET

lambdas

A **lambda expression** is like a method: it provides a list of formal parameters and a body - an expression or block - expressed in terms of those parameters.

```
(param1, param2, ...) -> expression
(param1, param2, ...) -> { stmt1; stmt2; ... }
```

Functional interfaces provide target types for lambda expressions and method references. Each functional interface has a **single abstract method**, to which the lambda expression's **parameters and return types** are matched or adapted.

Example:

```
@FunctionalInterface
interface Comparator<T> {
    int compare(T o1, T o2);
}

Comparator<Person> c =
    (Person p1, Person p2) ->
        p1.getName().compareTo(p2.getName());
```

Annotations: functional method, parameter list, implementation, lambda expression

Predicate<T> → boolean test(T t)

Supplier<T> → T get()

Consumer<T> → void accept(T t)

Function<T, V> → V apply(T t)

BiPredicate<T, U> → boolean test(T t, U u)

BiConsumer<T, U> → void accept(T t, U u)

BiFunction<T, U, V> → V apply(T t, U u)

BinaryOperator<T> → T test(T t1, T t2)

stream api

Stream operations are divided into **intermediate** and **terminal** operations, and are combined to form stream **pipelines**. A stream pipeline consists of a **source** followed by zero or more **intermediate** operations and a **terminal** operation.

Intermediate operations return a new stream. They are always **lazy**! **Terminal** operations may traverse the stream to produce a **result** or a **side-effect**.

Example:

```
people
    .stream()
    .map(person -> person.getAddress())
    .filter(address -> address.isVerified())
    .map(Address::getCity)
    .map(String::toUpperCase)
    .distinct()
    .limit(10)
    .collect(Collectors.toList());
```

Annotations: source, intermediate, lambda expression, method reference, terminal

Intermediate operations:

input	method	output
1 2 3 4 5	map(function) 1 → a	a b c d e
1 2 3 4 5	flatMap(function) 1 → [a, b]	a b c d e f g h i j
1 2 3 4 5	filter(predicate) x ≠ 2 & x ≠ 5	1 3 4
1 2 3 4 5	peek(consumer) side effect	1 2 3 4 5
1 2 3 4 5	limit(int) 3	1 2 3
1 2 3 4 5	skip(int) 2	3 4 5
1 1 2 1 2	distinct()	1 2
4 2 1 5 3	sorted(comparator)	1 2 3 4 5

Terminal operations:

input	method	result
1 2 3 4 5	findAny(predicate) x % 2 == 0	2 not guaranteed
1 2 3 4 5	findFirst(predicate) x % 2 == 0	2 guaranteed
1 2 3 4 5	allMatch(predicate) x ≠ 6	true
1 2 3 4 5	noneMatch(predicate) x % 2 == 0	false
1 2 3 4 5	anyMatch(predicate) x % 2 == 0	true
1 2 3 4 5	count()	5
1 2 3 4 5	min(comparator)	Optional(1)
1 2 3 4 5	max(comparator)	Optional(5)
1 2 3 4 5	collect(collector) Collectors.toList()	List(...)
1 2 3 4 5	forEach(consumer) side effect	void
1 2 3 4 5	reduce(binaryOperator) Integer::sum	Optional(15)

lambda examples

```
()
()
()
()
(int x)
(int x)
(x)
x
(int x, int y)
(x, y)
(String s)
(Thread t)

-> {}
-> 42
-> null
-> { return 42; }
-> { System.gc(); }
-> x + 1
-> { return x + 1; }
-> x + 1
-> x + 1
-> x + y
-> x + y
-> s.length()
-> { t.start(); }
```

Annotations: no parameters, result is void, expression body, block body, parenthesis are optional for single inferred-type parameter