

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.

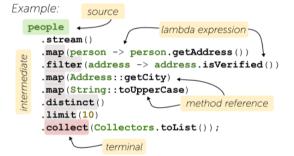
Example:

```
@FunctionalInterface
 interface Comparator<T> {
   int compare (T o1, T o2);←
              functional method
                                       parameter list
 Comparator<Person> c =
    (Person p1, Person p2) ->
     pl.getName().compareTo(p2.getName());
                             - implementation
lambda expression
Predicate < T > \rightarrow boolean test(T t)
Supplier<T> \rightarrow T get()
Consumer<T> \rightarrow void accept(T t)
Function<T, V \rightarrow V apply(T t)
BiPredicate\langle T, U \rangle \rightarrow boolean test(T t, U u)
BiConsumer<T, U > \rightarrow void accept(T t, U u)
BiFunction\langle T, U, V \rangle \rightarrow V \text{ apply}(T t, U u)
BinaryOperatorT> \rightarrow 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.



Intermediate operations:

input	method	output
12345	map (function) $\frac{1 \to a}{2}$	(a b c d e
12345	flatMap (function) $1 \to [a, b] \int_{a}^{b}$	abcdefghij
1 2 3 4 5	filter(predicate) x≠288 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 J	1 2 3
1 2 3 4 5	skip(int) <mark>2</mark> J	3 4 5
1 1 2 1 2	distinct()	1 2
42153	sorted(comparator)	12345

Terminal operations:			
input	method	result	
1 2 3 4 5	<pre>findAny(predicate) x % 2 == 0 J</pre>	2 not guaranteed	
1 2 3 4 5	findFirst(predicate) <mark>x%2==0</mark> ✓	2 guaranteed	
12345	allMatch(predicate) <mark>x≠6 </mark>	true	
1 2 3 4 5	noneMatch (predicate) $x \% 2 == 0 $	false	
1 2 3 4 5	anyMatch (predicate) $\frac{x \% 2 == 0}{\sqrt{100}}$	true	
1 2 3 4 5	count()	5	
1 2 3 4 5	count() min(comparator)	5 Optional(1)	
12345	min(comparator)	Optional(1)	
12345	min(comparator) max(comparator) collect(collector)	Optional (1) Optional (5)	

lambda examples

```
t→ {} ← no parameters, result is void
                -> null ← expression body
                -> { return 42; } •
                -> { System.gc(); } ← block body
(int x)
(int x)
                 -> { return x + 1; }
                                  parenthesis are optional
                                  for single inferred-type
(int x, int y)
                                         parameter
(\mathbf{x}, \mathbf{y})
                -> x + y
(String s)
                -> s.length()
(Thread t)
                -> { t.start(); }
```

