

### 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.

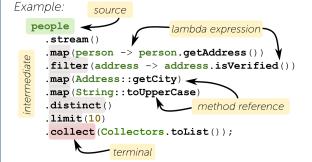
### Example:

```
@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());
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<T, U > \rightarrow boolean test(T t, U u)
BiConsumer<T, U> \rightarrow void accept(T t, U u)
BiFunction<T, U, V> \rightarrow V 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.



output

### Intermediate operations:

input

12345	map(function) $\frac{1 \to a}{2}$	(a b c d e)
12345	flatMap(function) $1 \rightarrow [a, b]$	(abcdefghij)
12345	filter(predicate) x≠288 x≠5	(1 3 4
12345	<pre>peek (consumer) side effect</pre>	(1 2 3 4 5)
12345	limit(int) 3	(1 2 3
12345	skip(int)	( 3 4 5)
(1 (1 (2 (1 (2))	distinct()	(1 (2)
4 2 1 5 3	sorted(comparator)	1 2 3 4 5



## lambda examples

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



