Streams

Suppose we want a function which halves values
However a function can only return 1 value
Here is where streams are useful where the values with be inside a stream

However if we were to just use map, map(halver) we would end up with a stream containing streams of each halved value
Thus you should use flatMap here

Source operations:

range

Generate elements from m to n - 1
static IntStream range(int m, int n)

rangeClosed

Generate elements from m to n
static IntStream rangeClosed(int m, int n)

of(T . . . values) / of(List<T>)

Creates a stream with values within it
static Stream<T> of(T . . . values)

of(T t)

Creates stream with one element in it, t
static Stream<T> of(T t)

generate

Produces an infinite stream generated by a supplier
Stream<T>::generate(Supplier<T> supp)

iterate

Produces an infinite sequence by repeatedly applying function, starting with seed value. Can take in a predicate
Stream<T>::iterate(T seed, UnaryOperator<T> next)
e.g Stream.<Integer>iterate(x → x + 1).filter(x → !x % 2 == 0).limit(20)
// finds first 20 odd integers

*iterate and generate produce infinite streams, should be used with limit(int n)

Intermediate methods:

limit

Limits number of elements in stream to maxSize
Stream<T> limit(long maxSize)

sorted

returns a stream with the elements in the stream sorted. Without argument, it sorts according to the natural order as defined by implementing the Comparable interface. You can also pass in a Comparator to tell sorted how to sort.

distinct

Returns a stream with only distinct elements within it
Stream<T> distinct()

map

Maps all elements in the streams given the function
<R> Stream<R> map(Function<? super T, ? extends R> func);
e.g IntStream.range(1, 3).map(x → x + 1);

flatMap

Maps and flattens the elements in the stream given the function
<R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>>

mapper)
e.g IntStream.range(1, 3).flatMap(x → IntStream.range(1, 3))

Stream.of("hello\nworld", "ciao\nmondo", "Bonjour\nle monde", "Hai\ndunia")
.map(x -> x.lines()) // returns a stream of streams

Stream.of("hello\nworld", "ciao\nmondo", "Bonjour\nle monde", "Hai\ndunia")
.flatMap(x -> x.lines()) // return a stream of strings

*sorted, limit and distinct are stateful operations and depend on the current state

Terminal methods:

count

Returns number of elements in the stream
long count()

reduce

Reduces the elements into a single return result based on the identity element and accumulator. Applies a lambda repeatedly on the elements of the stream to reduce it into a single value.
T reduce(T identity, BinaryOperator<T> accumulator)
*Can be done without identity e.g T reduce(BinaryOperator<T> accumulator)
*Done from left to right
e.g IntStream.range(1, 4).reduce(1, (x, y) → x * y + y)

- forEach

Element Matching

noneMatch

allMatch

anyMatch