Streams

Stream of Primitive

DoubleStream

Return an `OptionalDouble`:

- `max()`
- `min()`
- `average()`

```
OptionalDouble optionalDouble;
//not get()!!!
double value = optionalDouble.getAsDouble()
```

Do not return an `OptionalDouble`

- `count()`: return a long
- `sum()`: return a double DoubleStream

DoubleSummaryStatistics

```
DoubleSummaryStatistics summaryStatistics = DoubleStream.iterate(1D
    .limit(10L)
    .summaryStatistics();
double average = summaryStatistics.getAverage();
```

DoubleSummaryStatistics

LongStream

There ain't `mapToLong()` method in the `LongStream`.
LongStream

DoubleToLongFunction

```
long applyAsLong(double value);
```

DoubleToLongFunction

IntStream

```
IntStream rangeClosed = IntStream.rangeClosed(0, 9);
//note that IntStream.sum(0 returns an int, not a long
int sum = rangeClosed.sum();
```

IntStream

IntSummaryStatistics

```
IntSummaryStatistics intSummaryStatistics = IntStream.range(1, 100)
    .filter(n -> n % 2 == 0)
    .summaryStatistics();
double average = intSummaryStatistics.getAverage();
long sum = intSummaryStatistics.getSum();
```

Optional

Optional examples

Optional for primitives

- `OptionalInt > getAsInt()`
- `OptionalLong -> getAsLong()`
- `OptionalDouble -> getAsDouble()`

Optional Exception

If a value is present, returns the value, otherwise throws `NoSuchElementException`.

```
opt.orElseThrow();
```

Optional.ofNullable

```
    String message = null;
    //this does NOT throw any exception
    var optOfNullable = Optional.ofNullable(message);
    //this does throw NPE
    var opt = Optional.of(message);
```

Infinite Stream

```
Stream.generate(() -> "1");
Stream.iterate(1, x -> x++);
```

Intermediate Operations

sorted

sorted

```
//default natural order
stream
    .sorted()
    .forEach(System.out::println);

//order set by comparator
stream
    .sorted((c1,c2)->c1-c2)
    .forEach(System.out::println);
```

sorted of elements not Comparable

```
record Name(String name){}
List<Name> list = Arrays.asList(new Name("John"), new Name("Mark"),
//java.lang.ClassCastException: class ...Name cannot be cast to clas
list.stream()
    .sorted()
    .forEach(System.out::println);
```

Wrong usage of Comparator

`Comparator.reverseOrder()` does not implement the `Comparator` interface.

```
stream
    .sorted(Comparator::reverseOrder)  //does not compile!
```

Terminal Operations

FindAny() / findFirst()

```
public Optional<T> findAny()
public Optional<T> findFirst()
```

The `findFirst()` method **always** returns the first element on an ordered stream, regardless if it is serial or parallel.

Ordered and Unordered Stream

How to make an ordered stream unordered:

```
    IntStream.rangeClosed(1, 10)
        .boxed()
        .unordered()
        .parallel()
        .forEach(n -> System.out.print(n + " "));
}
```

Matching

```
public boolean anyMatch(Predicate <? super T> predicate)
public boolean allMatch(Predicate <? super T> predicate)
public boolean noneMatch(Predicate <? super T> predicate)
```

min

For `Stream<T>`

```
Stream<String> stream = Stream.of("a", "ab", "abc");
//min() requires a Comparator
Optional<String> min = stream.min((o1, o2) -> o1.length() - o2.leng
```

For `IntStream`

```
IntStream rangeClosed = IntStream.rangeClosed(0, 9);
OptionalInt optional = rangeClosed.min();
```

Reduce

```
public T reduce(T identity, BinaryOperator<T> accumulator)

public Optional<T> reduce(BinaryOperator<T> accumulator)

public <U> U reduce(U identity,
   BiFunction<U,? super T,U> accumulator,
   BinaryOperator<U> combiner)
```

Collectors

PartitioningBy

```
Map<Boolean, List<String>> map = ohMy.collect(
   Collectors.partitioningBy(s -> s.length() <= 5));
```

It requires a `Predicate<T>`
CollectorsPartitioningBy

GroupingBy

Overloaded with one, two or three arguments.

```
Map<Integer, List<String>> map = ohMy.collect(
   Collectors.groupingBy(String::length));
```

It requires a `Function<T,V>`
CollectorsGroupingBy

Counting

Note that it returns a `Long`.

```
Long cnt = IntStream.rangeClosed(1, 10)
    .boxed()
    .collect(Collectors.counting());  //10
```

Teeing

It has three parameters.

```
public static <T,R1,R2,R> Collector<T,?,R> teeing(Collector<? super
 Collector<? super T,?,R2> downstream2,
 BiFunction<? super R1,? super R2,R> merger)
```

Example:

```
record DataReport(long count, long sum) {}

DataReport dataReport = stream.collect(Collectors.teeing(
        Collectors.counting(),
        Collectors.summingLong(DataWrapper::value),
        DataReport::new));
```

Teeing

Spliterator

```
Stream<String> toys = ...
var spliterator = toys.spliterator();
var batch = spliterator.trySplit();  //batch contains the first two

var more = batch.tryAdvance(x -> {}); //we remove Toy A from batch l
System.out.println(more);  //true - as it still contains Toy B
spliterator.tryAdvance(System.out::println); //here we print the fi
spliterator.forEachRemaining(System.out::println);
```

from Collection

```
List<String> words = Arrays.asList("Hello", "World", "Java", "Progr
Spliterator<String> spliterator = words.spliterator();
```

from Stream

```
Stream<String> fruitStream = Stream.of("Apple", "Banana", "Orange",
//here applied on the stream
Spliterator<String> spliterator = fruitStream.spliterator();
```

Spliterator