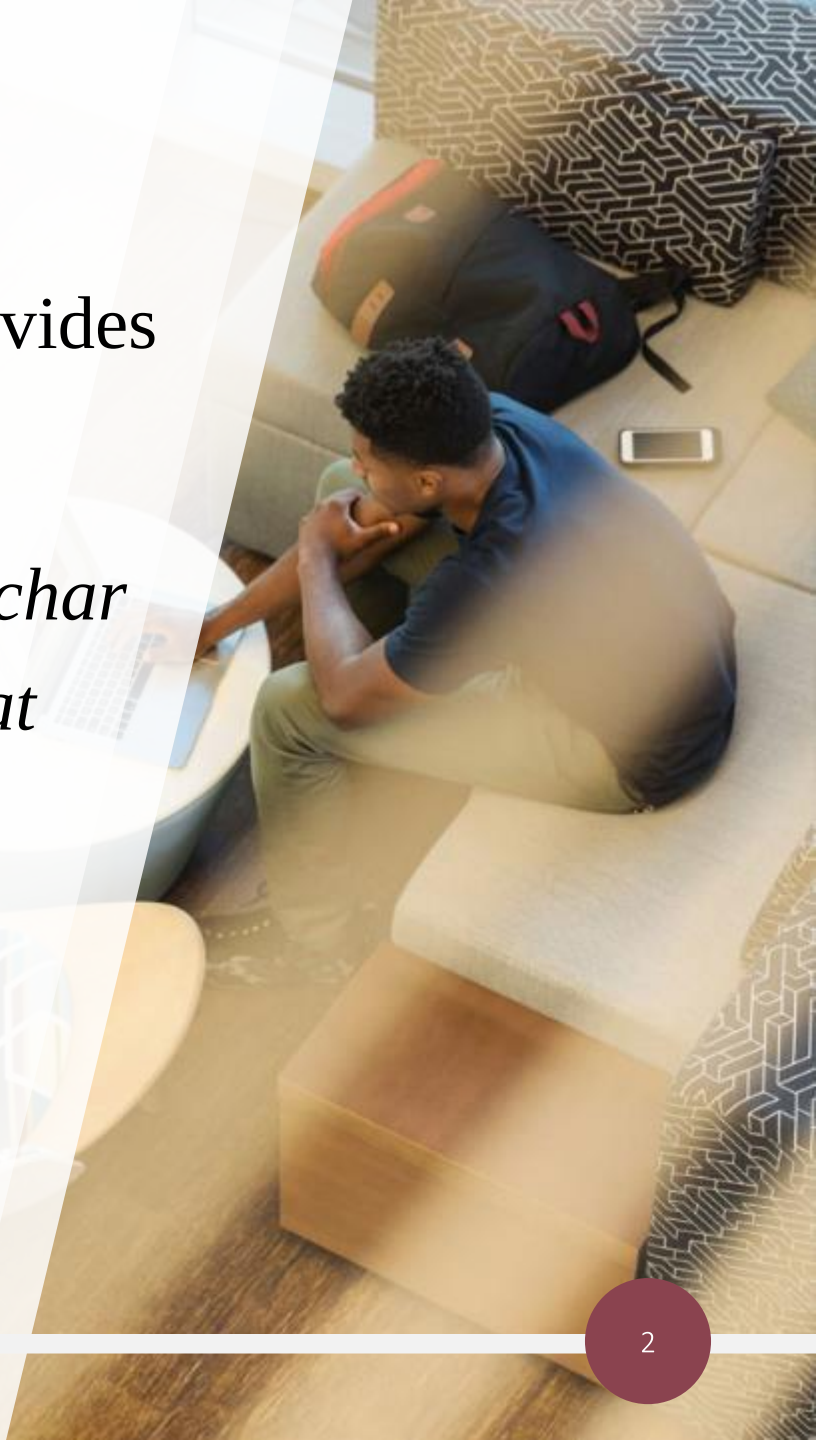
Streams

Primitive Streams

Primitive Streams

- As opposed to Stream<T> e.g. Stream<Integer>, Stream<Double> and Stream<Long>; Java actually provides other stream classes that you can use to work with primitives:
  - *IntStream* – for primitive types *int*, *short*, *byte* and *char*
  - *DoubleStream* – for primitive types *double* and *float*
  - *LongStream* – for primitive type *long*

IntStream	*Stream<Integer>*
DoubleStream	*Stream<Double>*
LongStream	*Stream<Long>*



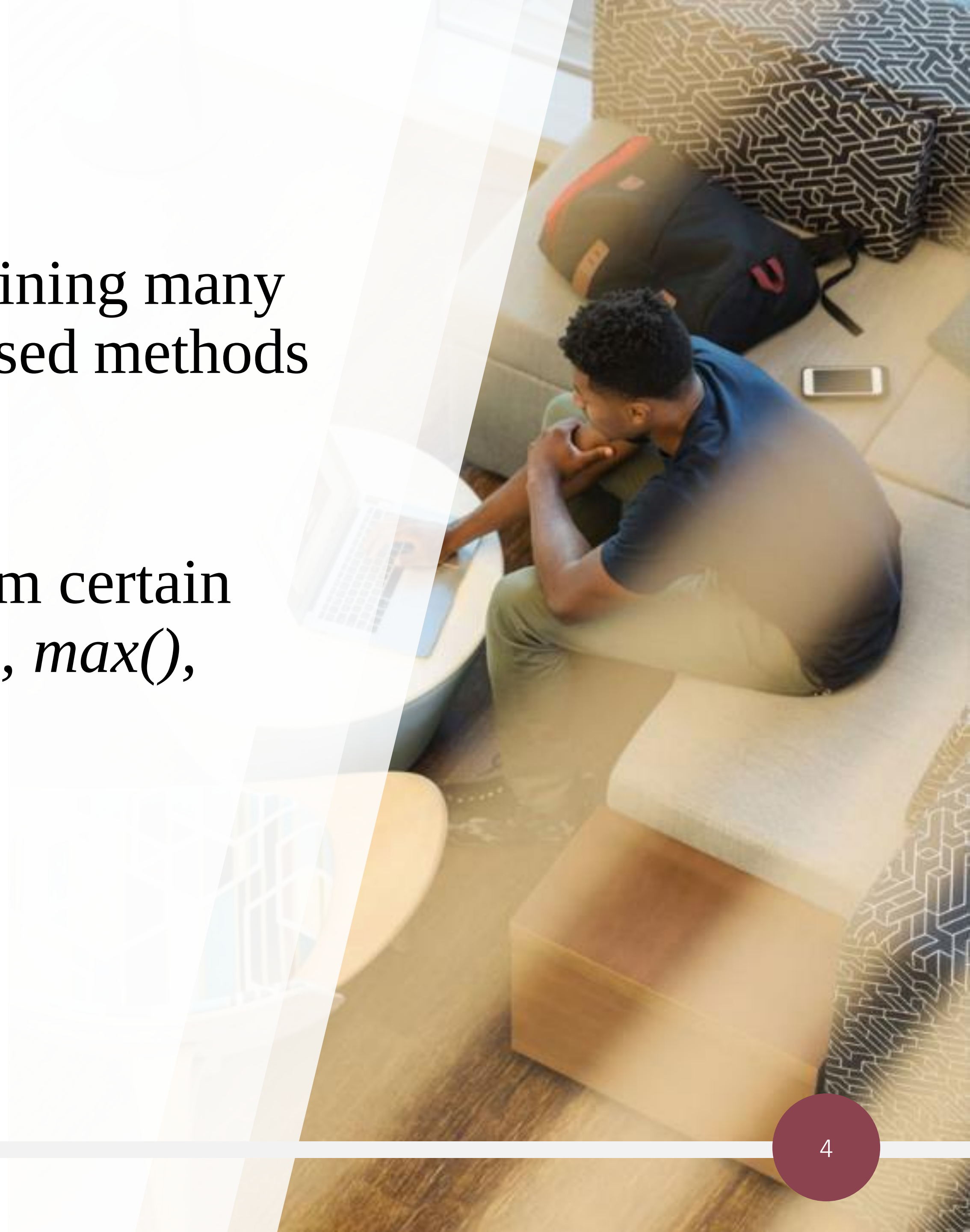
Creating Primitive Streams

```
int[] ia     = {1,2,3};
double[] da = {1.1, 2.2, 3.3};
long[] la    = {1L, 2L, 3L};

IntStream iStream1       = Arrays.stream(ia);
DoubleStream dStream1    = Arrays.stream(da);
LongStream lStream1      = Arrays.stream(la);
System.out.println(iStream1.count() + ", " +
        dStream1.count() + ", " + lStream1.count());// 3, 3, 3

IntStream iStream2        = IntStream.of(1, 2, 3);
DoubleStream dStream2     = DoubleStream.of(1.1, 2.2, 3.3);
LongStream lStream2       = LongStream.of(1L, 2L, 3L);
System.out.println(iStream2.count() + ", " +
        dStream2.count() + ", " + lStream2.count());// 3, 3, 3
```

Primitive Streams

- The primitive streams, in addition to containing many of the *Stream* methods, also contain specialised methods for working with numeric data.

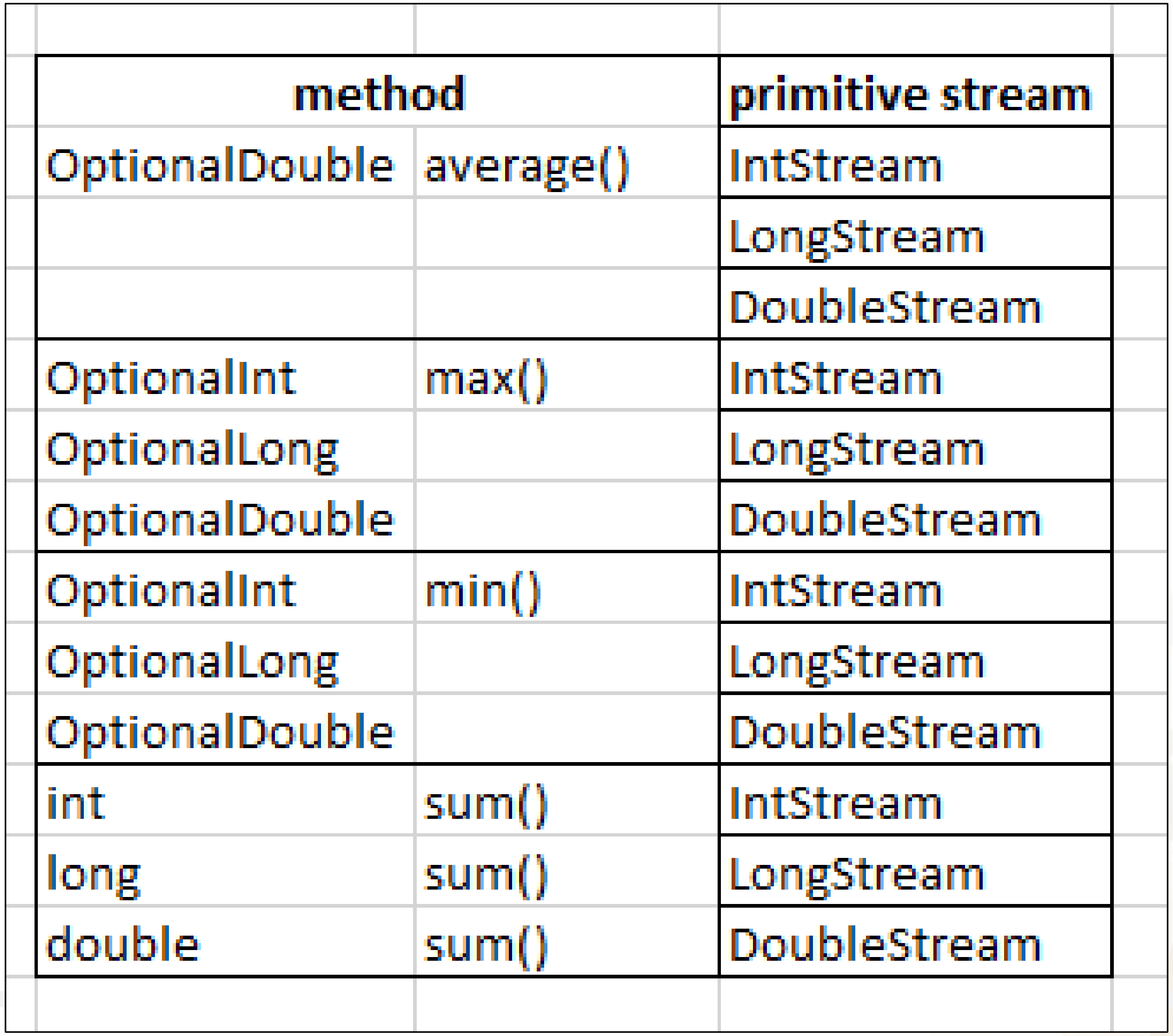
- The primitive streams know how to perform certain common operations automatically e.g. *min()*, *max()*, *sum()* and *average()*.



Primitive Streams

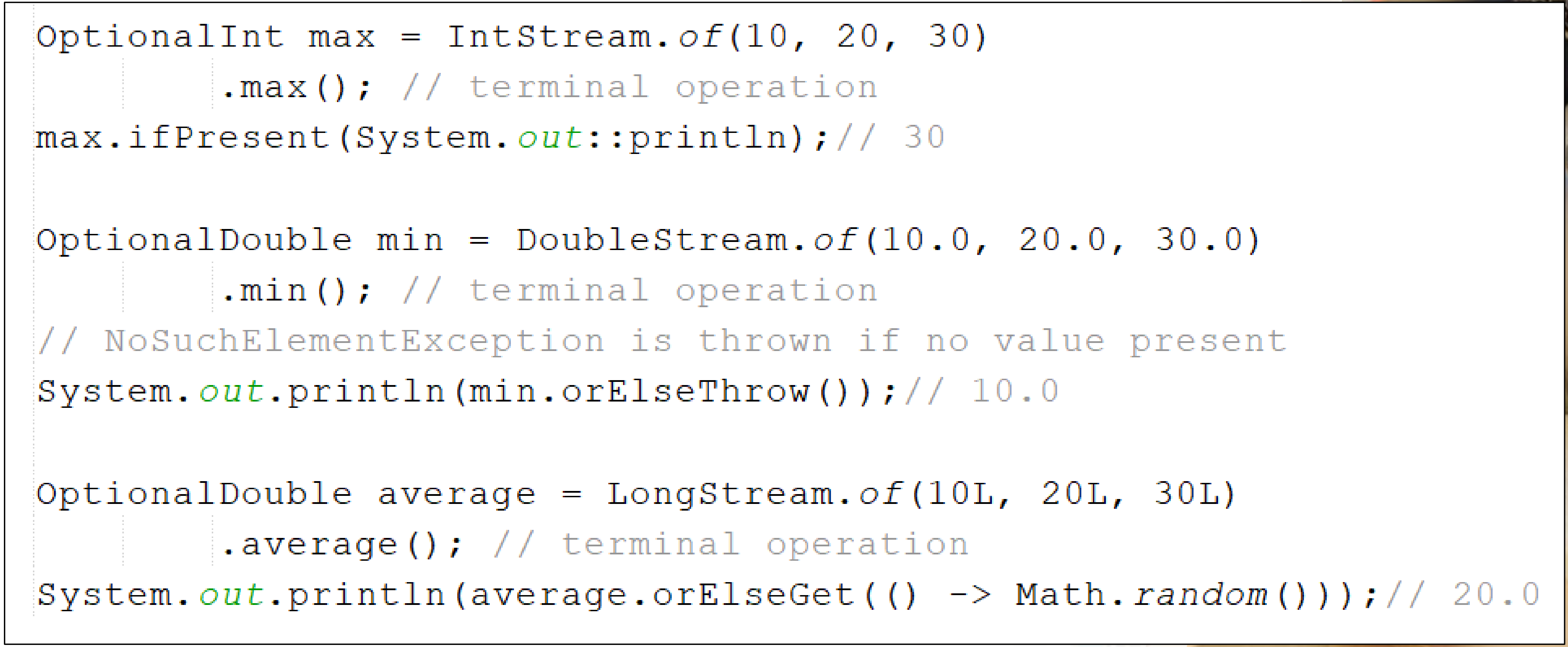
```
// 1. Using Stream<T> and reduce(identity, accumulator)
Stream<Integer> numbers = Stream.of(1,2,3);
// Integer reduce(Integer identity, BinaryOperator accumulator)
//   BinaryOperator extends BiFunction<T,T,T>
//     T apply(T,T)
// starting the accumulator with 0
//       n1 +  n2
//       0  +  1   == 1   (n1 now becomes 1)
//       1  +  2   == 3   (n1 now becomes 3)
//       3  +  3   == 6
System.out.println(numbers.reduce(0, (n1, n2) -> n1 + n2));// 6

// 2. Using IntStream and sum()
// IntStream mapToInt(ToIntFunction)
//   ToIntFunction is a functional interface:
//     int applyAsInt(T value);
IntStream intS = Stream.of(1,2,3)
                      .mapToInt( n -> n);// unboxed
int total = intS.sum();
System.out.println(total);//6
```

Common Primitive Stream Methods

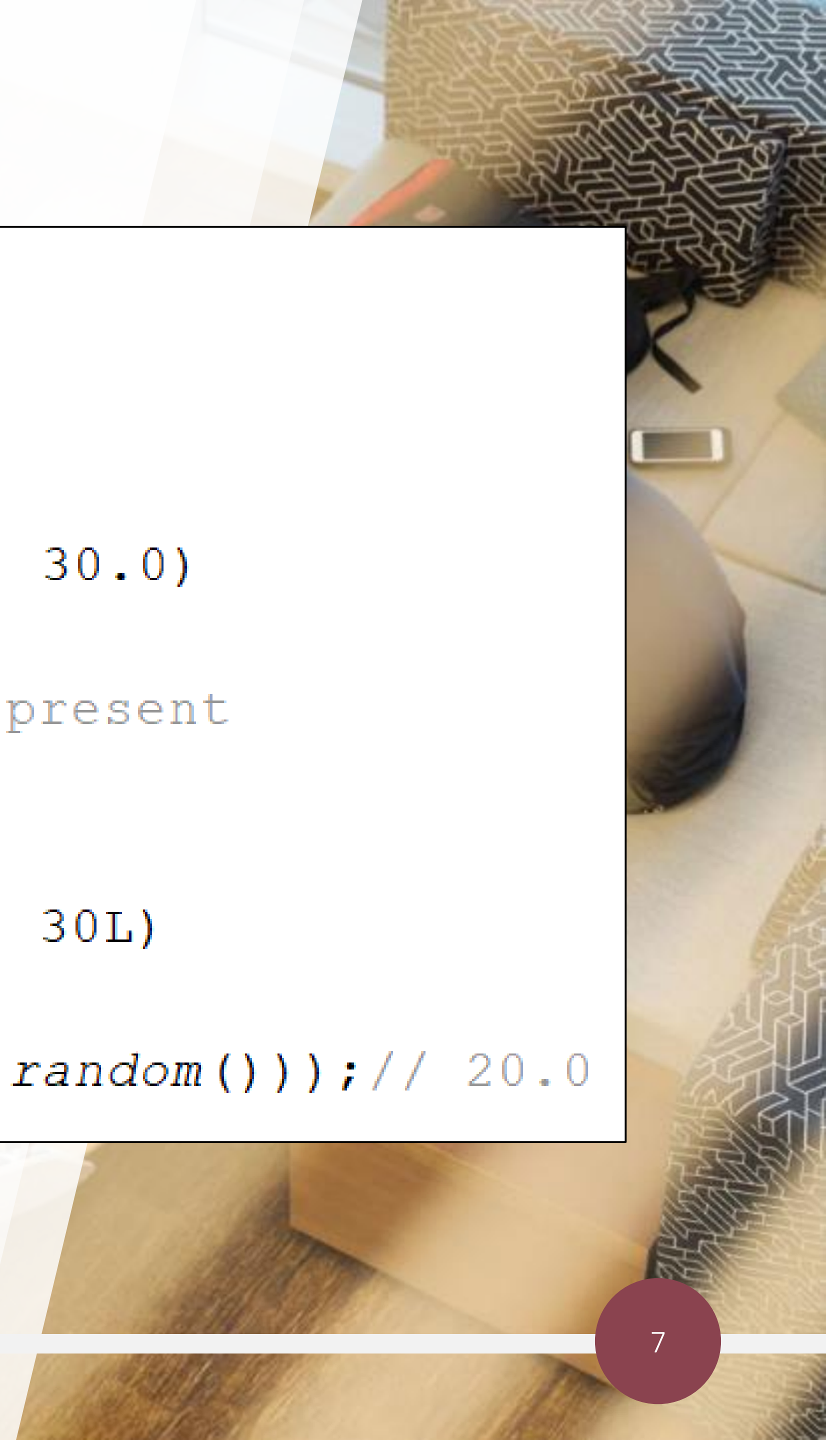


method		primitive stream
OptionalDouble	average()	IntStream
		LongStream
		DoubleStream
OptionalInt	max()	IntStream
OptionalLong		LongStream
OptionalDouble		DoubleStream
OptionalInt	min()	IntStream
OptionalLong		LongStream
OptionalDouble		DoubleStream
int	sum()	IntStream
long	sum()	LongStream
double	sum()	DoubleStream

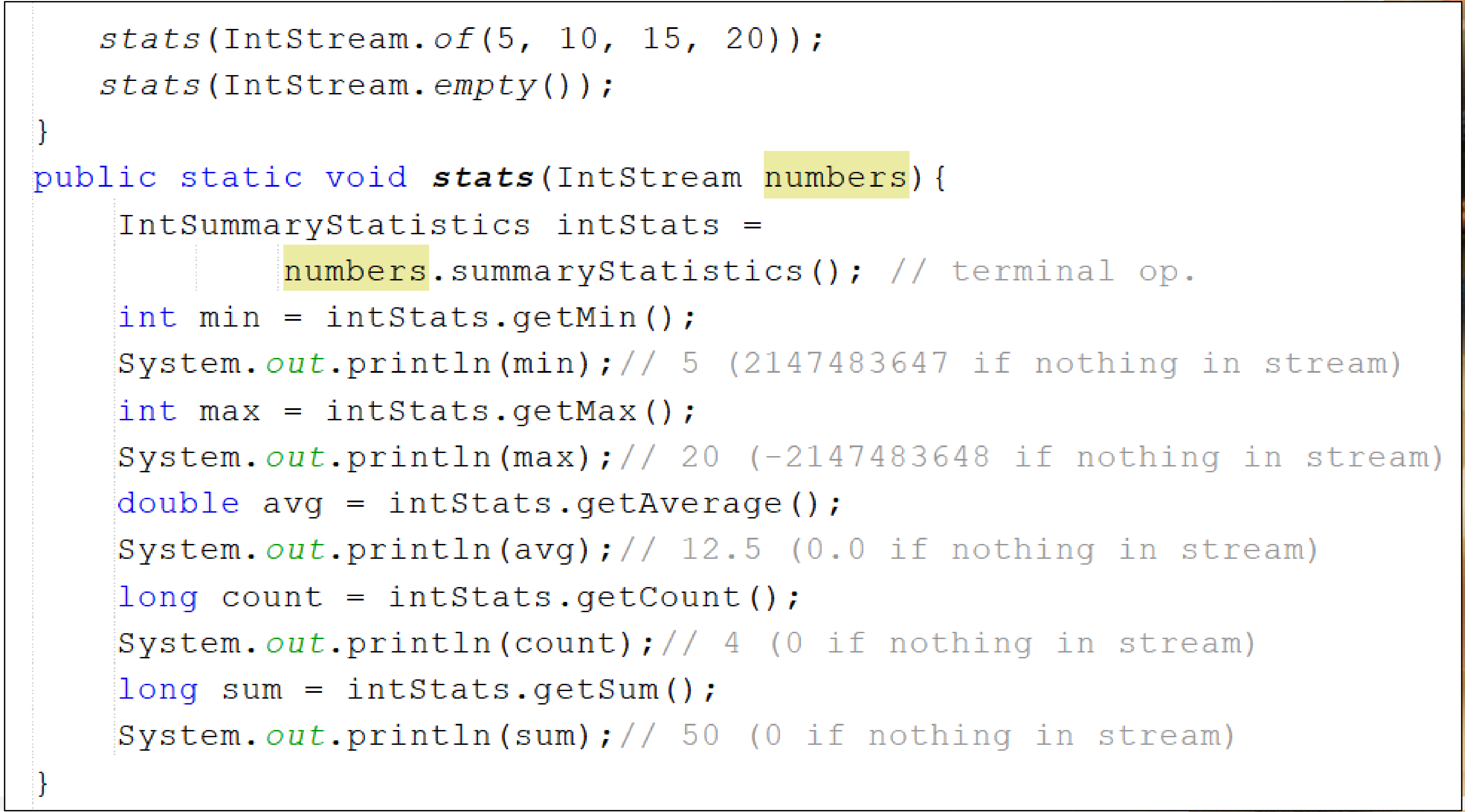
Common Primitive Stream Methods



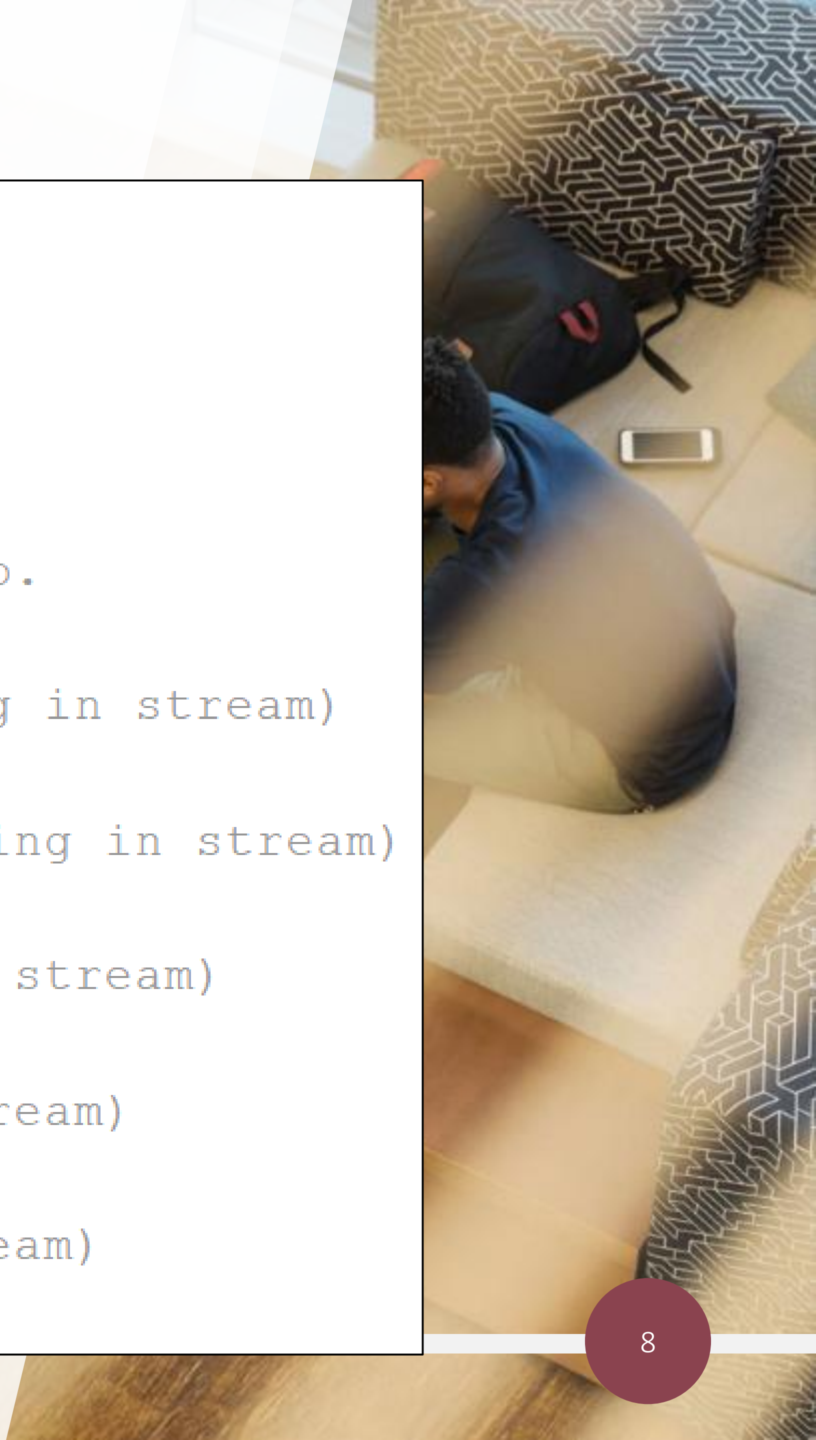
```
OptionalInt max = IntStream.of(10, 20, 30)
          .max(); // terminal operation
max.ifPresent(System.out::println);// 30

OptionalDouble min = DoubleStream.of(10.0, 20.0, 30.0)
          .min(); // terminal operation
// NoSuchElementException is thrown if no value present
System.out.println(min.orElseThrow());// 10.0

OptionalDouble average = LongStream.of(10L, 20L, 30L)
          .average(); // terminal operation
System.out.println(average.orElseGet(() -> Math.random()));// 20.0
```



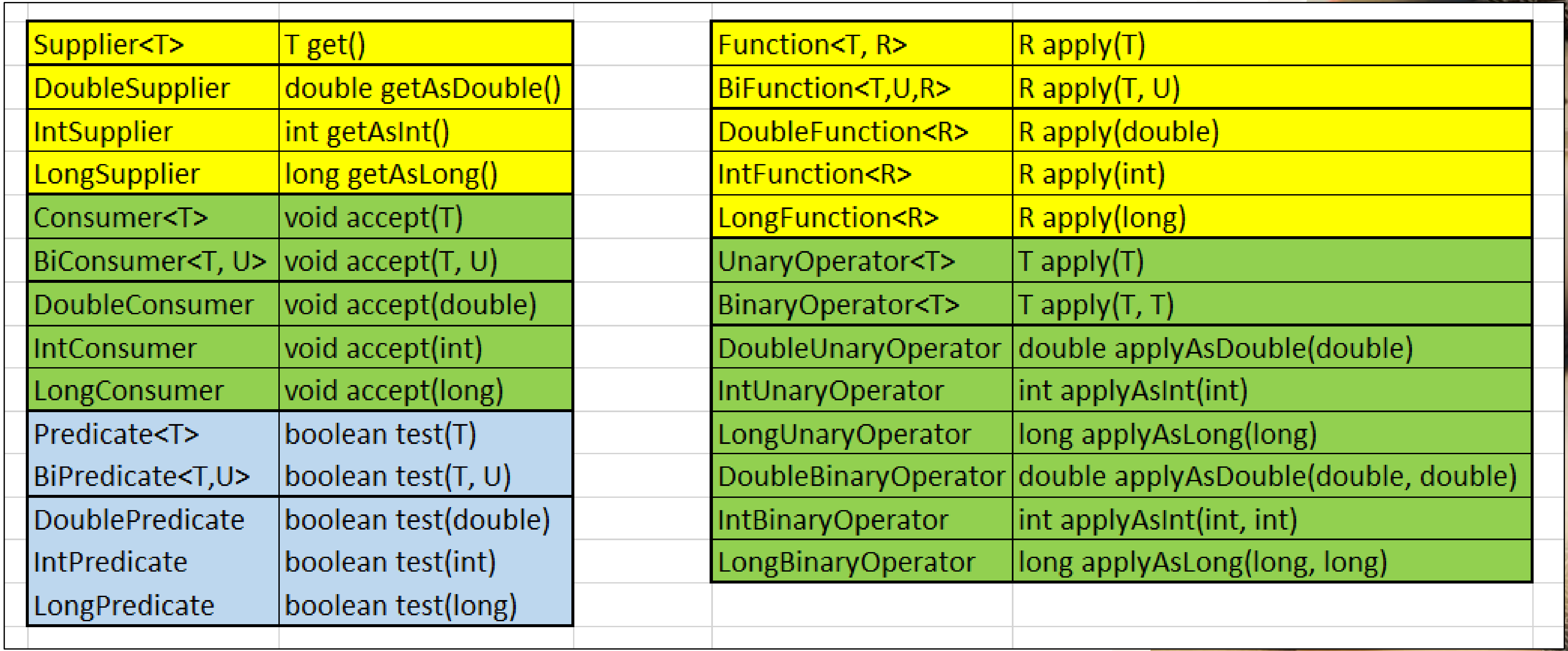
Summarising Statistics



```
    stats(IntStream.of(5, 10, 15, 20));
    stats(IntStream.empty());
}
public static void stats(IntStream numbers){
    IntSummaryStatistics intStats =
            numbers.summaryStatistics(); // terminal op.
    int min = intStats.getMin();
    System.out.println(min);// 5 (2147483647 if nothing in stream)
    int max = intStats.getMax();
    System.out.println(max);// 20 (-2147483648 if nothing in stream)
    double avg = intStats.getAverage();
    System.out.println(avg);// 12.5 (0.0 if nothing in stream)
    long count = intStats.getCount();
    System.out.println(count);// 4 (0 if nothing in stream)
    long sum = intStats.getSum();
    System.out.println(sum);// 50 (0 if nothing in stream)
}
```



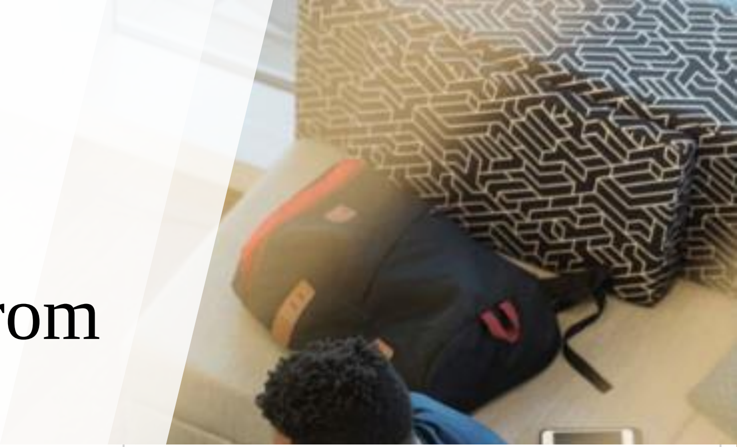
Functional Interfaces



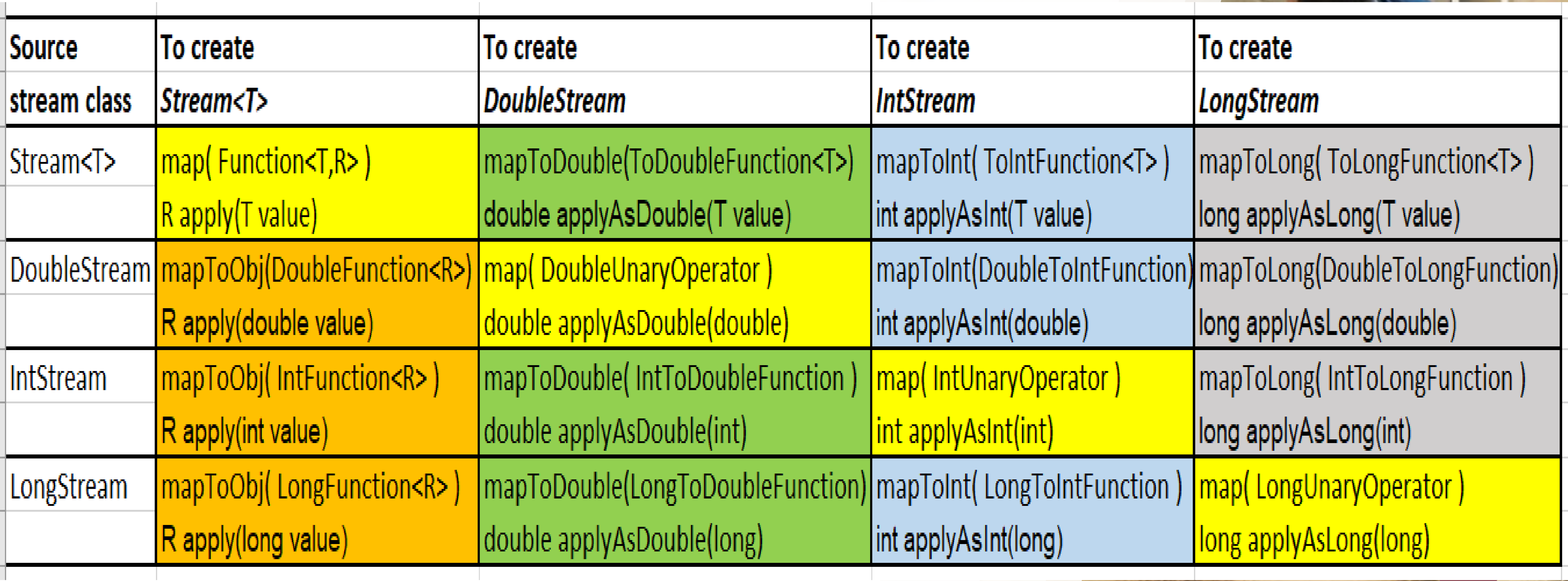
Supplier<T>	T get()
DoubleSupplier	double getAsDouble()
IntSupplier	int getAsInt()
LongSupplier	long getAsLong()
Consumer<T>	void accept(T)
BiConsumer<T, U>	void accept(T, U)
DoubleConsumer	void accept(double)
IntConsumer	void accept(int)
LongConsumer	void accept(long)
Predicate<T>	boolean test(T)
BiPredicate<T,U>	boolean test(T, U)
DoublePredicate	boolean test(double)
IntPredicate	boolean test(int)
LongPredicate	boolean test(long)

Function<T, R>	R apply(T)
BiFunction<T,U,R>	R apply(T, U)
DoubleFunction<R>	R apply(double)
IntFunction<R>	R apply(int)
LongFunction<R>	R apply(long)
UnaryOperator<T>	T apply(T)
BinaryOperator<T>	T apply(T, T)
DoubleUnaryOperator	double applyAsDouble(double)
IntUnaryOperator	int applyAsInt(int)
LongUnaryOperator	long applyAsLong(long)
DoubleBinaryOperator	double applyAsDouble(double, double)
IntBinaryOperator	int applyAsInt(int, int)
LongBinaryOperator	long applyAsLong(long, long)

Mapping Streams



- Another way to create a primitive stream is by mapping from another stream type.



Source stream class	To create *Stream<T>*	To create *DoubleStream*	To create *IntStream*	To create *LongStream*
Stream<T>	map(Function<T,R>) R apply(T value)	mapToDouble(ToDoubleFunction<T>) double applyAsDouble(T value)	mapToInt(ToIntFunction<T>) int applyAsInt(T value)	mapToLong(ToLongFunction<T>) long applyAsLong(T value)
DoubleStream	mapToObj(DoubleFunction<R>) R apply(double value)	map(DoubleUnaryOperator) double applyAsDouble(double)	mapToInt(DoubleToIntFunction) int applyAsInt(double)	mapToLong(DoubleToLongFunction) long applyAsLong(double)
IntStream	mapToObj(IntFunction<R>) R apply(int value)	mapToDouble(IntToDoubleFunction) double applyAsDouble(int)	map(IntUnaryOperator) int applyAsInt(int)	mapToLong(IntToLongFunction) long applyAsLong(int)
LongStream	mapToObj(LongFunction<R>) R apply(long value)	mapToDouble(LongToDoubleFunction) double applyAsDouble(long)	mapToInt(LongToIntFunction) int applyAsInt(long)	map(LongUnaryOperator) long applyAsLong(long)