Stream Operations Cheat Sheet

Category	Operation	Description	Example
Stream Creation	`Stream.of(T... values)`	Creates a stream from values.	`Stream.of(1, 2, 3, 4);`
	`Arrays.stream(T[] array)`	Creates a stream from an array.	`Arrays.stream(new int[]{1, 2, 3});`
	`List.stream()`	Converts a collection to a stream.	`list.stream();`
	`Stream.generate(Supplier)`	Creates an infinite stream from a supplier function.	`Stream.generate(Math::random);`
	`Stream.iterate(T seed, UnaryOperator)`	Creates an infinite stream by iterating a seed value.	`Stream.iterate(0, n -> n + 2);`
Terminal Operations	`forEach(Consumer)`	Performs an action for each element.	`stream.forEach(System.out::println);`
	`toArray()`	Converts the stream into an array.	`Integer[] arr = stream.toArray(Integer[]::new);`
	`reduce(BinaryOperator)`	Reduces the elements into a single value using an accumulator function.	`stream.reduce(0, Integer::sum);`
	`collect(Collector)`	Collects the stream elements into a collection (e.g., list, set, map).	`stream.collect(Collectors.toList());`
	`count()`	Returns the number of elements in the stream.	`long count = stream.count();`
	`anyMatch(Predicate)`	Returns `true` if any element matches the predicate.	`stream.anyMatch(x -> x > 10);`
	`allMatch(Predicate)`	Returns `true` if all elements match the predicate.	`stream.allMatch(x -> x > 0);`
	`noneMatch(Predicate)`	Returns `true` if no elements match the predicate.	`stream.noneMatch(x -> x < 0);`
	`findFirst()`	Returns the first element in the stream (optional).	`Optional<Integer> first = stream.findFirst();`
	`findAny()`	Returns any element in the stream (useful in parallel streams).	`Optional<Integer> any = stream.findAny();`
Intermediate Operations	`filter(Predicate)`	Filters elements based on a predicate.	`stream.filter(x -> x > 10);`
	`map(Function)`	Transforms each element into another value.	`stream.map(String::toUpperCase);`
	`flatMap(Function)`	Flattens nested structures into a single stream.	`stream.flatMap(List::stream);`
	`distinct()`	Removes duplicate elements.	`stream.distinct();`

	`sorted()`	Sorts elements in natural order.	`stream.sorted();`
	`sorted(Comparator)`	Sorts elements using a custom comparator.	`stream.sorted(Comparator.reverseOrder());`
	`peek(Consumer)`	Performs an action on each element and returns a new stream.	`stream.peek(System.out::println);`
	`limit(long maxSize)`	Truncates the stream to a specified size.	`stream.limit(5);`
	`skip(long n)`	Skips the first `n` elements of the stream.	`stream.skip(3);`
Numeric Streams	`IntStream.range(int, int)`	Creates a range of integers (exclusive).	`IntStream.range(1, 5); // 1, 2, 3, 4`
	`IntStream.rangeClosed(int, int)`	Creates a range of integers (inclusive).	`IntStream.rangeClosed(1, 5); // 1, 2, 3, 4, 5`
	`IntStream.of(int... values)`	Creates a stream of primitive integers.	`IntStream.of(1, 2, 3);`
	`mapToInt(ToIntFunction)`	Converts objects to primitive `int` values.	`stream.mapToInt(String::length);`
	`boxed()`	Converts a primitive stream to an object stream.	`IntStream.range(1, 5).boxed();`
Collectors	`Collectors.toList()`	Collects elements into a `List`.	`stream.collect(Collectors.toList());`
	`Collectors.toSet()`	Collects elements into a `Set`.	`stream.collect(Collectors.toSet());`
	`Collectors.toMap()`	Collects elements into a `Map`.	`stream.collect(Collectors.toMap(x -> x, x -> x.length()));`
	`Collectors.groupingBy()`	Groups elements by a classifier function.	`stream.collect(Collectors.groupingBy(String::length));`
	`Collectors.partitioningBy()`	Partitions elements into `true` and `false` groups based on a predicate.	`stream.collect(Collectors.partitioningBy(x -> x.length() > 3));`
	`Collectors.joining()`	Joins elements into a `String`.	`stream.collect(Collectors.joining(", "));`