

Java 8 Streams Cheat Sheet

For more awesome cheat sheets
visit rebellabs.org!



Definitions

- ✓ A stream **is** a pipeline of functions that can be evaluated.
- ✓ Streams **can** transform data.
- ✗ A stream **is not** a data structure.
- ✗ Streams **cannot** mutate data.

Intermediate operations

- Always return streams.
- Lazily executed.

Common examples include:

Function	Preserves count	Preserves type	Preserves order
<i>map</i>	✓	✗	✓
<i>filter</i>	✗	✓	✓
<i>distinct</i>	✗	✓	✓
<i>sorted</i>	✓	✓	✗
<i>peek</i>	✓	✓	✓

Stream examples

Get the unique surnames in uppercase of the first 15 book authors that are 50 years old or over.

```
library.stream()
    .map(book -> book.getAuthor())
    .filter(author -> author.getAge() >= 50)
    .distinct()
    .limit(15)
    .map(Author::getSurname)
    .map(String::toUpperCase)
    .collect(toList());
```

Compute the sum of ages of all female authors younger than 25.

```
library.stream()
    .map(Book::getAuthor)
    .filter(a -> a.getGender() == Gender.FEMALE)
    .map(Author::getAge)
    .filter(age -> age < 25)
    .reduce(0, Integer::sum);
```

Terminal operations

- Return concrete types or produce a side effect.
- Eagerly executed.

Common examples include:

Function	Output	When to use
reduce	concrete type	to cumulate elements
collect	list, map or set	to group elements
forEach	side effect	to perform a side effect on elements

Parallel streams

Parallel streams use the common ForkJoinPool for threading.

```
library.parallelStream()...
```

or intermediate operation:

```
IntStream.range(1, 10).parallel()...
```

Useful operations

Grouping:

```
library.stream().collect(
    groupingBy(Book::getGenre));
```

Stream ranges:

```
IntStream.range(0, 20)...
```

Infinite streams:

```
IntStream.iterate(0, e -> e + 1)...
```

Max/Min:

```
IntStream.range(1, 10).max();
```

FlatMap:

```
twitterList.stream()
    .map(member -> member.getFollowers())
    .flatMap(followers -> followers.stream())
    .collect(toList());
```

Pitfalls

✗ Don't update shared mutable variables i.e.
`List<Book> myList = new ArrayList<>();`
`library.stream().forEach(e -> myList.add(e));`

✗ Avoid blocking operations when using parallel streams.