

MUST-HAVE CHEAT SHEETS FOR JAVA DEVELOPERS

CONCISE, HELPFUL, BEAUTIFUL, PRINTABLE



 **REBELLABS**
by ZERO TURNAROUND

Java 8 Streams Cheat Sheet

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.

Java Collections Cheat Sheet

For more awesome cheat sheets
visit rebellabs.org!



Notable Java collections libraries

Fastutil

<http://fastutil.di.unimi.it/>

Fast & compact type-specific collections for Java
Great default choice for collections of primitive types, like int or long. Also handles big collections with more than 2^{31} elements well.

Guava

<https://github.com/google/guava>

Google Core Libraries for Java 6+

Perhaps the default collection library for Java projects. Contains a magnitude of convenient methods for creating collection, like fluent builders, as well as advanced collection types.

Eclipse Collections

<https://www.eclipse.org/collections/>

Features you want with the collections you need

Previously known as gs-collections, this library includes almost any collection you might need: primitive type collections, multimaps, bidirectional maps and so on.

JCTools

<https://github.com/JCTools/JCTools>

Java Concurrency Tools for the JVM.

If you work on high throughput concurrent applications and need a way to increase your performance, check out JCTools.

What can your collection do for you?

| Collection class | Thread-safe alternative | Your data | | | | Operations on your collections | | | | | | |
|---------------------------|---|---------------------|-----------------|---------------------------|-------------------|--------------------------------|--------|------|-----------------------------|---------------|----------|----------|
| | | Individual elements | Key-value pairs | Duplicate element support | Primitive support | Order of iteration | | | Performant 'contains' check | Random access | | |
| | | | | | | FIFO | Sorted | LIFO | | By key | By value | By index |
| HashMap | ConcurrentHashMap | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| HashBiMap (Guava) | Maps.synchronizedBiMap (new HashBiMap()) | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ |
| ArrayListMultimap (Guava) | Maps.synchronizedMultiMap (new ArrayListMultimap()) | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| LinkedHashMap | Collections.synchronizedMap (new LinkedHashMap()) | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| TreeMap | ConcurrentSkipListMap | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓* | ✓* | ✗ | ✗ |
| Int2IntMap (Fastutil) | | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ |
| ArrayList | CopyOnWriteArrayList | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| HashSet | Collections.newSetFromMap (new ConcurrentHashMap<>()) | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ |
| IntArrayList (Fastutil) | | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| PriorityQueue | PriorityBlockingQueue | ✓ | ✗ | ✓ | ✗ | ✗ | ✓** | ✗ | ✗ | ✗ | ✗ | ✗ |
| ArrayDeque | ArrayBlockingQueue | ✓ | ✗ | ✓ | ✗ | ✓** | ✗ | ✓** | ✗ | ✗ | ✗ | ✗ |

* $O(\log(n))$ complexity, while all others are $O(1)$ - constant time

** when using Queue interface methods: offer() / poll()

How fast are your collections?

| Collection class | Random access by index / key | Search / Contains | Insert |
|------------------|------------------------------|-------------------|--------------|
| ArrayList | $O(1)$ | $O(n)$ | $O(n)$ |
| HashSet | $O(1)$ | $O(1)$ | $O(1)$ |
| HashMap | $O(1)$ | $O(1)$ | $O(1)$ |
| TreeMap | $O(\log(n))$ | $O(\log(n))$ | $O(\log(n))$ |

Remember, not all operations are equally fast. Here's a reminder of how to treat the Big-O complexity notation:

$O(1)$ - constant time, really fast, doesn't depend on the size of your collection

$O(\log(n))$ - pretty fast, your collection size has to be extreme to notice a performance impact

$O(n)$ - linear to your collection size: the larger your collection is, the slower your operations will be