

I want to master Java streams  
I want cheat sheet to solve any problem with it.

Can you generate a cheat sheet

Absolutely — here's a **Java Streams "solve-any-problem" cheat sheet** (with templates you can copy-paste). I'll keep it practical and pattern-based.

---

## 1) The Stream Mental Model

**Stream pipeline = Source → 0..N intermediate ops → Terminal op**

- **Intermediate (lazy):** `map`, `filter`, `flatMap`, `sorted`, `distinct`, `peek`, `limit`, `skip`
- **Terminal (executes):** `forEach`, `toList`, `collect`, `reduce`, `count`, `anyMatch`, `allMatch`, `noneMatch`, `findFirst`, `findAny`, `min`, `max`

### Rule of thumb

- Transform element? → `map`
  - Keep/remove? → `filter`
  - Nested list/array? → `flatMap`
  - Need result (List/Map/summary)? → `collect(...)`
  - Need single value? → `reduce`, `min/max`, `sum`, `count`
- 

## 2) Most-used building blocks (templates)

### Create streams

java

```
list.stream()
Arrays.stream(arr)
Stream.of(a, b, c)
IntStream.range(0, n)
Files.lines(Path.of("file.txt")) // remember to close or use try-with-resources
```

## Filter + map + collect

 Copy code

java

```
List<R> out = list.stream()
    .filter(x -> /* condition */)
    .map(x -> /* transform */)
    .toList(); // Java 16+
```

## Sorting

 Copy code

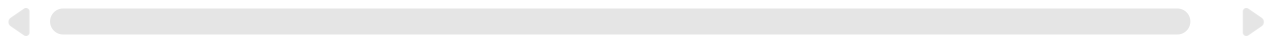
java

```
List<T> sorted = list.stream()
    .sorted(Comparator.comparing(T::getAge))           // asc
    .sorted(Comparator.comparing(T::getAge).reversed()) // desc
    .sorted(Comparator.comparing(T::getA).thenComparing(T::getB))
    .toList();
```

## Remove duplicates

java

```
List<T> uniq = list.stream().distinct().toList(); // needs proper equals/hashCode
```

 Copy code

## Take / skip

java

```
List<T> first10 = list.stream().limit(10).toList();
List<T> page2 = list.stream().skip(10).limit(10).toList();
```

## Match checks (fast)

 Copy code

java

```
boolean any = list.stream().anyMatch(x -> cond);
boolean all = list.stream().allMatch(x -> cond);
boolean none = list.stream().noneMatch(x -> cond);
```

## Find one

 Copy code

java

```
Optional<T> first = list.stream().filter(cond).findFirst();  
T value = first.orElse(defaultValue);
```

---

 Copy code

### 3) flatMap patterns (master this!)

**List<List<T>> → List<T>**

java

```
List<T> flat = listOfLists.stream()  
    .flatMap(List::stream)  
    .toList();
```

### Object has collection field

 Copy code

java

```
List<Item> items = orders.stream()  
    .flatMap(o -> o.getItems().stream())  
    .toList();
```

### Split strings into words

 Copy code

java

```
List<String> words = lines.stream()  
    .flatMap(s -> Arrays.stream(s.split("\\s+")))  
    .filter(w -> !w.isBlank())  
    .toList();
```

---

 Copy code

## 4) Collectors cheat sheet (the “solve-any-problem” section)

### Collect to List / Set

java

```
List<T> L = stream.toList();  
Set<T> S = stream.collect(Collectors.toSet());
```

 Copy code

## Collect to Map (key → value)

java

```
Map<K, V> map = stream.collect(Collectors.toMap(  
    x -> keyOf(x),  
    x -> valueOf(x)  
));
```

## Handling duplicate keys in toMap

 Copy code

java

```
Map<K, V> map = stream.collect(Collectors.toMap(  
    x -> keyOf(x),  
    x -> valueOf(x),  
    (oldV, newV) -> oldV    // keep old  
    // or: (oldV, newV) -> newV // replace  
));
```

## Grouping (most common)

### Group elements by key → Map<K, List<T>>

 Copy code

java

```
Map<K, List<T>> g = stream.collect(Collectors.groupingBy(x -> keyOf(x)));
```

### Group + count

 Copy code

java

```
Map<K, Long> counts = stream.collect(Collectors.groupingBy(  
    x -> keyOf(x),  
    Collectors.counting()  
));
```

### Group + sum

 Copy code

java

```
Map<K, Integer> sumByK = stream.collect(Collectors.groupingBy(  
    x -> keyOf(x),
```

```
Collectors.summingInt(x -> intValueOf(x))
));
```

 Copy code

## Group + max/min item

java

```
Map<K, Optional<T>> maxByK = stream.collect(Collectors.groupingBy(
    x -> keyOf(x),
    Collectors.maxBy(Comparator.comparing(T::getScore))
));
```

## Group + mapping (transform inside group)

java

```
Map<K, List<R>> g2 = stream.collect(Collectors.groupingBy(
    x -> keyOf(x),
    Collectors.mapping(x -> transform(x), Collectors.toList())
));
```

## Group into Set

java

```
Map<K, Set<R>> gSet = stream.collect(Collectors.groupingBy(
    x -> keyOf(x),
    Collectors.mapping(x -> transform(x), Collectors.toSet())
));
```

## Group + join strings

java

```
Map<K, String> joined = stream.collect(Collectors.groupingBy(
    x -> keyOf(x),
    Collectors.mapping(T::getName, Collectors.joining(", "))
));
```

## Partitioning (boolean split)

 Copy code

java

```
Map<Boolean, List<T>> parts = stream.collect(Collectors.partitioningBy(x -> cond)
List<T> yes = parts.get(true);
```

```
List<T> no = parts.get(false);
```

## Downstream + post-processing (collectingAndThen)

 Copy code

Example: group and make list unmodifiable / or unwrap Optional

```
java
```

```
Map<K, T> bestByK = stream.collect(Collectors.groupingBy(
    x -> keyOf(x),
    Collectors.collectingAndThen(
        Collectors.maxBy(Comparator.comparing(T::getScore)),
        opt -> opt.orElse(null)
    )
));
```

## 5) Reduce patterns (when you need one value)

 Copy code

Sum / product / custom fold

```
java
```

```
int sum = ints.stream().mapToInt(i -> i).sum();

int product = ints.stream().reduce(1, (a, b) -> a * b);

Optional<T> combined = stream.reduce((a, b) -> combine(a, b));
```

Prefer built-ins ( sum , min , max ) over manual reduce when possible.

 Copy code

## 6) Numeric streams (avoid boxing)

```
java
```

```
int sum = list.stream().mapToInt(T::getAge).sum();
double avg = list.stream().mapToInt(T::getAge).average().orElse(0);
int max = list.stream().mapToInt(T::getAge).max().orElseThrow();
IntSummaryStatistics stats = list.stream().mapToInt(T::getAge).summaryStatistics()
```

 Copy code

## 7) “Top K” and “Kth”

### Top K by score

java

```
List<T> topK = list.stream()
    .sorted(Comparator.comparing(T::getScore).reversed())
    .limit(k)
    .toList();
```

### Kth largest (safe-ish)

 Copy code

java

```
T kth = list.stream()
    .sorted(Comparator.comparing(T::getScore).reversed())
    .skip(k - 1)
    .findFirst()
    .orElseThrow();
```

---

## 8) Frequency map (classic interview pattern)

 Copy code

java

```
Map<String, Long> freq = words.stream()
    .collect(Collectors.groupingBy(w -> w, Collectors.counting()));
```

### Most frequent

 Copy code

java

```
Map.Entry<String, Long> best = freq.entrySet().stream()
    .max(Map.Entry.comparingByValue())
    .orElseThrow();
```

---

 Copy code

## 9) Distinct by key (not equals/hashCode)

java

```
static <T> Predicate<T> distinctByKey(Function<? super T, ?> keyExtractor) {
    Set<Object> seen = ConcurrentHashMap.newKeySet();
    return t -> seen.add(keyExtractor.apply(t));
}
```

// usage:

```
List<T> uniqById = list.stream()
    .filter(distinctByKey(T::getId))
    .toList();
```

## 10) Null-safety mini patterns

 Copy code

java

```
Stream.ofNullable(value) // Java 9+: 0 or 1 element stream
```

```
List<String> cleaned = list.stream()
    .filter(Objects::nonNull)
    .map(String::trim)
    .filter(s -> !s.isEmpty())
    .toList();
```

## 11) Debugging streams (without breaking them)

 Copy code

java

```
List<T> out = list.stream()
    .peek(x -> System.out.println("before: " + x))
    .map(this::transform)
    .peek(x -> System.out.println("after: " + x))
    .toList();
```

## 12) When NOT to use streams (quick)


 Copy code

- Complicated loops with early breaks + multiple mutable states → loop may be clearer
- Heavy side effects inside stream (especially parallel) → avoid

## 13) Parallel streams (only if it's worth it)

java

```
list.parallelStream()  
    .map(this::cpuHeavyWork)  
    .toList();
```

Use when: CPU-heavy + large data + stateless operations. Avoid when: IO / <sup>shared</sup> mutable state / order-sensitive logic.  Copy code

---

## 14) “Problem → Stream recipe” quick map

- Filter + transform list → filter → map → toList
- Flatten nested collections → flatMap
- Group by field → groupingBy
- Group and aggregate → groupingBy + counting/summing/maxBy
- Split into two buckets → partitioningBy
- Build map → toMap (with merge)
- Frequency → groupingBy(identity(), counting())
- Top K → sorted(desc) → limit(k)
- Any/All checks → anyMatch/allMatch/noneMatch

---

If you want, tell me **what kind of problems you mostly solve** (arrays/strings, objects/DTOs, nested lists, logs/files, etc.), and I'll generate a **second cheat sheet with 20–30 solved examples** in that style (copy-paste ready).

