Java Stream API — ████████████ PDF█████

████████, ██ ██████ ████████████ ██ java.util.stream: ████████ ████████, ████████ ██████████████, ████████ █████ ■ ███████ ██ ████████████████████.

███ ██████ Stream API?

Stream API — ███ ████████████████ API ███ ███████ ■ ████████████████████ ███████ ■ Java (████████ ■ Java 8).
██ ██████████ ███████ ████████████████ ███ ■ ████████████████ ██████:
████████████, ████████████████, ██████████ ███████.
Streams ██ ███████ ███████ — ███ ██████████ ███ ████████████ (████████████, ██████████, I/O, ████████████).

██████████ ████████████

- **████████ ███████ (lazy evaluation)**: ████████████████ ██████████ ████████████████ ██ ████████████████ ██████████.
- **████████████████ ■ ████████████████ ██████████**: ████████████████ (map, filter) ████████████ Stream, ████████████████ (collect, forEach, reduce) ████████████ ██████████.
- **████████████████ ██████████ ████████████**: ██████████ Stream ██ ████████████████ ██████████ ████████████.
- **████████████████**: Stream ██████ ████████████████ ███████ ██████ ████ — ██████ ████████████████ ██████████ ██ ████████████ ██████████.

██████████ ██████████ (■ ████████████)

█████ — ██████ ██████ ████████████████ ██████████. ████████ ████████████████ ██ `List` ■ `List`.

████████ ██████

████████:

██████████ stream

```
List<String> names = List.of("Anna", "Bob", "Charlie"); Stream<String> s = names.stream();
Stream<String> pStream = names.parallelStream();
```

████████████ ■ map

```
List<String> filtered = names.stream()     .filter(n -> n.length() > 3)
.map(String::toUpperCase)     .collect(Collectors.toList());
```

FlatMap

```
List<List<Integer>> lists = List.of(List.of(1,2), List.of(3,4)); List<Integer> flat = lists.str
.flatMap(List::stream)     .collect(Collectors.toList());
```

[redacted]

```
Map<String, Long> counts = names.stream()     .collect(Collectors.groupingBy(Function.identity(
Collectors.counting()));
```

Reduce

```
int sum = List.of(1,2,3,4).stream()     .reduce(0, Integer::sum);
```

[redacted]

```
int total = largeList.parallelStream()     .mapToInt(Integer::intValue)     .sum(); // [redacted]:
[redacted]-[redacted]
```

[redacted] Optional

```
Optional<Person> oldest = people.stream()     .max(Comparator.comparingInt(Person::getAge));
oldest.ifPresent(p -> System.out.println(p.getName()));
```

[redacted]

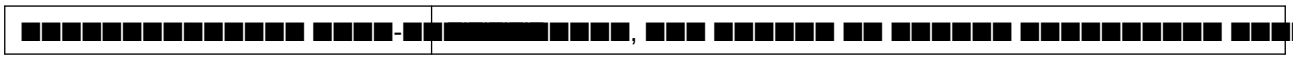
- **[redacted] parallelStream**: overhead [redacted], [redacted] GC.
- **[redacted]-[redacted]**: [redacted] stream-[redacted] (forEach [redacted] pipeline).
- **Boxing/unboxing**: [redacted] IntStream/LongStream/DoubleStream [redacted] mapToInt [redacted].
- **[redacted]**: [redacted], [redacted] — [redacted]. [redacted], toList() [redacted].
- **[redacted]**: [redacted], [redacted].

Best practices

- [redacted], [redacted].
- [redacted]: IntStream, LongStream.
- [redacted] Collectors (groupingBy, partitioningBy, toMap).
- [redacted]-[redacted], [redacted].
- [redacted]: [redacted] pipeline [redacted] — [redacted].

[redacted]

[redacted]	[redacted] / [redacted]
[redacted]	[redacted] parallelStream [redacted], [redacted]
[redacted]	[redacted]: [redacted] overhead parallelStream; [redacted]
ConcurrentModificationException	[redacted].



■■■■■■■■ cheatsheet

• ■■■■■■■ stream: collection.stream(), Arrays.stream(array), Stream.of(...).
• ■■■■■■■■■■■■■: forEach, collect, reduce, count, anyMatch, allMatch, noneMatch, findFirst, findAny.
• ■■■■■■■■■■■■■■: map, flatMap, filter, distinct, sorted, peek, limit, skip.
• ■■■■■■■■■■■: Collectors.toList(), toSet(), joining(), groupingBy(), partitioningBy(), toMap().

■■■■■■■■ ■■■■■■■■ (■■■■■)

```
 // ■■■■■■■: ■■■■■■■■ ■■■■■ ■ ■■■■■■■ Map<String, Long> freq = words.stream()     .flatMap(w
Arrays.stream(w.split("\\s+")))     .map(String::toLowerCase)
.collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));
```