Java 8 Streams methods

Here's a comprehensive list of Java 8 Stream API methods, along with code examples for each:

1. `stream()`

- Converts a collection into a Stream.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
Stream<String> stream = names.stream();
```

2. `filter()`

- Filters elements based on a predicate.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<String> filteredNames = names.stream().filter(name ->
name.startsWith("J")).collect(Collectors.toList());
```

3. `map()`

- Transforms each element into another form.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<Integer> nameLengths =
names.stream().map(String::length).collect(Collectors.toList());
```

4. `flatMap()`

- Flattens nested structures into a single stream.

```
List<List<String>> names = Arrays.asList(Arrays.asList("John",
"Jane"),Arrays.asList("Jack", "Jill"));
List<String> flatNames =
names.stream().flatMap(Collection::stream).collect(Collectors.toList());
```

5. `forEach()`

- Performs an action on each element.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
names.stream().forEach(System.out::println);
```

6. `collect()`

- Collects the elements into a collection or another data structure.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<String> result = names.stream().collect(Collectors.toList());
```

7. `reduce()`

- Reduces the elements to a single value using an accumulator.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
int sum = numbers.stream().reduce(0, Integer::sum);
```

8. `sorted()`

- Sorts the elements in natural order or using a comparator.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<String> sortedNames = names.stream().sorted().collect(Collectors.toList());
```

9. `distinct()`

- Removes duplicates from the stream.

```
List<Integer> numbers = Arrays.asList(1, 2, 2, 3, 3, 4);
List<Integer> distinctNumbers =
numbers.stream().distinct().collect(Collectors.toList());
```

10. `limit()`

- Limits the number of elements in the stream.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<String> limitedNames = names.stream().limit(2).collect(Collectors.toList());
```

11. `skip()`

- Skips the first `n` elements in the stream.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<String> skippedNames = names.stream().skip(1).collect(Collectors.toList());
```

12. `peek()`

- Allows for inspecting each element in the stream.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
List<String> result =
names.stream().peek(System.out::println).collect(Collectors.toList());
```

13. `count()`

- Counts the number of elements in the stream.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
long count = names.stream().count();
```

14. `anyMatch()`

- Returns `true` if any element matches the predicate.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
boolean hasJName = names.stream().anyMatch(name -> name.startsWith("J"));
```

15. `allMatch()`

- Returns `true` if all elements match the predicate.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
boolean allJNames = names.stream().allMatch(name -> name.startsWith("J"));
```

16. `noneMatch()`

- Returns `true` if no elements match the predicate.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
boolean noneZNames = names.stream().noneMatch(name -> name.startsWith("Z"));
```

17. `findFirst()`

- Returns the first element in the stream.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
Optional<String> firstName = names.stream().findFirst();
```

18. `findAny()`

- Returns any element from the stream (useful in parallel streams).

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
Optional<String> anyName = names.stream().findAny();
```

19. `max()`

- Finds the maximum element according to a comparator.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
Optional<Integer> max = numbers.stream().max(Integer::compare);
```

20. `min()`

- Finds the minimum element according to a comparator.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4);
Optional<Integer> min = numbers.stream().min(Integer::compare);
```

21. `toArray()`

- Converts the stream into an array.

```
List<String> names = Arrays.asList("John", "Jane", "Jack");
String[] namesArray = names.stream().toArray(String[]::new);
```

22. `generate()`

- Creates an infinite stream of elements.

```
Stream<String> infiniteStream = Stream.generate(() -> "Hello");
```

23. `iterate()`

- Creates an infinite stream by iterating over a function.

```
Stream<Integer> numbers = Stream.iterate(0, n -> n + 2).limit(10);
```

24. `of()`

- Creates a stream from a set of values.

```
Stream<String> stream = Stream.of("John", "Jane", "Jack");
```

25. `concat()`

- Concatenates two streams.

```
Stream<String> stream1 = Stream.of("John", "Jane");
Stream<String> stream2 = Stream.of("Jack", "Jill");
Stream<String> combined = Stream.concat(stream1, stream2);
```