### 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Preserves count</th>
<th>Preserves type</th>
<th>Preserves order</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>filter</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>distinct</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sorted</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>peek</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

Common examples include:

<table>
<thead>
<tr>
<th>Function</th>
<th>Output</th>
<th>When to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce</td>
<td>concrete type</td>
<td>to cumulate elements</td>
</tr>
<tr>
<td>collect</td>
<td>list, map or set</td>
<td>to group elements</td>
</tr>
<tr>
<td>forEach</td>
<td>side effect</td>
<td>to perform a side effect on elements</td>
</tr>
</tbody>
</table>

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

```java
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.

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

### Parallel streams
Parallel streams use the common ForkJoinPool for threading.

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

or intermediate operation:

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

### Useful operations
- Grouping:

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

- Stream ranges:

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

- Infinite streams:

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

- Max/Min:

```java
IntStream.range(1, e).max();
```

- FlatMap:

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

### Pitfalls
- Don't update shared mutable variables i.e.

```java
List<Book> myList = new ArrayList<>();
library.stream().forEach (e -> myList.add(e));
```

- Avoid blocking operations when using parallel streams.