

The addition of the `Stream` API was one of the major features added to Java 8. A `Stream` in Java can be defined as a sequence of elements from a source that supports aggregate operations on them. The source here refers to collections or arrays that provide data to a stream.

A few important points about streams are:

1. A stream is not a data structure itself. It is a bunch of operations applied to a source. The source can be collections, arrays or I/O channels.
2. Streams don't change the original data structure.
3. There can be zero or more intermediate operations that transform a stream into another stream.
4. Each intermediate operation is lazily executed (This will be discussed later).
5. Terminal operations produce the result of the stream.

Stream creation#

Streams can be created from different element sources, e.g., a collection or an array with the help of `stream()` and `of()` methods. Below are the different ways to create a stream.

a) Stream.of(v1, v2, v3...)#

In the below example, we are creating a stream of integers at **line 7** using the `Stream.of()` method.

```
1 import java.util.stream.Stream;
2
3 public class StreamDemo {
4
5     public static void main(String[] args)
6     {
7         Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8,9);
8         stream.forEach(p -> System.out.println(p));
9     }
10 }
11
```

Run Save Reset

b) List.stream()#

In the below example, we are creating a stream from a List at **line 14**.

```
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.stream.Stream;
4
5 public class StreamDemo {
6
7     public static void main(String[] args) {
8         List<String> list = new ArrayList<>();
9         list.add("a");
10        list.add("b");
11        list.add("c");
12        list.add("d");
13
14        Stream<String> stream = list.stream();
15        stream.forEach(p -> System.out.println(p));
16    }
17 }
18
```

Run Save Reset

The Stream interfaces#

The Stream API defines a few interfaces such as `Stream`, `IntStream`, `LongStream`, etc.

The `Stream<T>` interface is for object elements. For primitives, it defines `IntStream`, `LongStream` and `DoubleStream` interfaces.

It is a good practice to use primitive streams if you are dealing with primitives because wrapping primitives to objects and auto-boxing is a costly process.

Below is the complete list of methods defined in Stream API.

Interface Summary	
Interface	Description
<code>BaseStream<T,S extends BaseStream<T,S>></code>	Base interface for streams, which are sequences of elements supporting sequential and parallel aggregate operations.
<code>Collector<T,A,R></code>	A mutable reduction operation that accumulates input elements into a mutable result container, optionally transforming the accumulated result into a final representation after all input elements have been processed.
<code>DoubleStream</code>	A sequence of primitive double-valued elements supporting sequential and parallel aggregate operations.
<code>DoubleStream.Builder</code>	A mutable builder for a <code>DoubleStream</code> .
<code>IntStream</code>	A sequence of primitive int-valued elements supporting sequential and parallel aggregate operations.
<code>IntStream.Builder</code>	A mutable builder for an <code>IntStream</code> .
<code>LongStream</code>	A sequence of primitive long-valued elements supporting sequential and parallel aggregate operations.
<code>LongStream.Builder</code>	A mutable builder for a <code>LongStream</code> .
<code>Stream<T></code>	A sequence of elements supporting sequential and parallel aggregate operations.
<code>Stream.Builder<T></code>	A mutable builder for a <code>Stream</code> .

The methods defined by these interfaces can be divided into the following two categories:

Intermediate operations#

These methods do not produce any results. They usually accept functional interfaces as parameters and always return a new stream. Some examples of intermediate operations are `filter()`, `map()`, etc.

Terminal operations#

These methods produce some results, e.g., `count()`, `toArray(..)`, and `collect(..)`.

The streams operations can be further classified as:

1. filtering
2. slicing
3. mapping
4. matching and finding
5. reduction
6. collect