2. Stream Creation

There are many ways to create a stream instance of different sources. Once created, the instance **will not modify its source,** therefore allowing the creation of multiple instances from a single source.

2.1. Empty Stream

The **empty()** method should be used in case of a creation of an empty stream:

```
Stream<String> streamEmpty = Stream.empty();
```

Its often the case that the empty() method is used upon creation to avoid returning null for streams with no element:

```
public Stream<String> streamOf(List<String> list) {
    return list == null || list.isEmpty() ? Stream.empty() : list.stream();
}
```

2.2. Stream of Collection

Stream can also be created of any type of Collection (Collection, List, Set):

```
Collection<String> collection = Arrays.asList("a", "b", "c");
Stream<String> streamOfCollection = collection.stream();
```

2.3. Stream of Array

Array can also be a source of a Stream:

```
Stream<String> streamOfArray = Stream.of("a", "b", "c");
```

They can also be created out of an existing array or of a part of an array:

```java
String[] arr = new String[]{"a", "b", "c"};
Stream<String> streamOfArrayFull = Arrays.stream(arr);
Stream<String> streamOfArrayPart = Arrays.stream(arr, 1, 3);
```

2.4. Stream.builder()

When builder is used the desired type should be additionally specified in the right part of the statement, otherwise the build() method will create an instance of the Stream<Object>:

```java
Stream<String> streamBuilder =
  Stream.<String>builder().add("a").add("b").add("c").build();
```

2.5. Stream.generate()

The **generate()** method accepts a Supplier<T> for element generation. As the resulting stream is infinite, developer should specify the desired size or the generate() method will work until it reaches the memory limit:

```java
Stream<String> streamGenerated =
  Stream.generate(() -> "element").limit(10);
```

The code above creates a sequence of ten strings with the value – “element”.

2.6. Stream.iterate()

Another way of creating an infinite stream is by using the **iterate()** method:

```java
Stream<Integer> streamIterated = Stream.iterate(40, n -> n + 2).limit(20);
```

The first element of the resulting stream is a first parameter of the iterate() method. For creating every following element the specified function is applied to the previous element. In the example above the second element will be 42.

2.7. Stream of Primitives

Java 8 offers a possibility to create streams out of three primitive types: int, long and double. As Stream<T> is a generic interface and there is no way to use primitives as a type parameter with generics, three new special interfaces were created: **IntStream, LongStream, DoubleStream.**

Using the new interfaces alleviates unnecessary auto-boxing allows increased productivity:

```
IntStream intStream = IntStream.range(1, 3);
LongStream longStream = LongStream.rangeClosed(1, 3);
```

The **range(int startInclusive, int endExclusive)** method creates an ordered stream from the first parameter to the second parameter. It increments the value of subsequent elements with the step equal to 1. The result doesn't include the last parameter, it is just an upper bound of the sequence.

The **rangeClosed(int startInclusive, int endInclusive)** method does the same with only one difference – the second element is included. These two methods can be used to generate any of the three types of streams of primitives.

Since Java 8 the Random class provides a wide range of methods for generation streams of primitives. For example, the following code creates a DoubleStream, which has three elements:

```
Random random = new Random();
DoubleStream doubleStream = random.doubles(3);
```

2.8. Stream of String

String can also be used as a source for creating a stream.

With the help of the chars() method of the String class. Since there is no interface CharStream in JDK, the IntStream is used to represent a stream of chars instead.

```
IntStream streamOfChars = "abc".chars();
```

The following example breaks a String into sub-strings according to specified RegEx:

```
Stream<String> streamOfString =
  Pattern.compile(", ").splitAsStream("a, b, c");
```

2.9. Stream of File

Java NIO class Files allows to generate a Stream<String> of a text file through the lines() method. Every line of the text becomes an element of the stream:

```
Path path = Paths.get("C:\\file.txt");
Stream<String> streamOfStrings = Files.lines(path);
Stream<String> streamWithCharset =
  Files.lines(path, Charset.forName("UTF-8"));
```

The Charset can be specified as an argument of the lines() method.