**Stream API**

* In Stream, datastructure is **computed on demand.**
* Java Stream **doesn’t store data**, it operates on the source data structure (collection and array) and produce pipelined data that we can use and perform specific operations.
* Java Stream operations **use functional interfaces**, that makes it a very good fit for functional programming using lambda expression
* Java 8 Stream **internal iteration** principle helps in **achieving lazy-seeking in some of the stream operations**. For example **filtering, mapping, or duplicate removal** can be implemented lazily, allowing higher performance and scope for optimization.
* All the Java Stream API interfaces and classes are in the **java.util.stream** package.
* Java 8 Stream **support** **sequential as well as parallel processing**, parallel processing can be very helpful in achieving high performance for large collections.
* For primitives like **int, long,** there are specific classes for primitive types – **IntStream, LongStream and DoubleStream**

|  |  |  |
| --- | --- | --- |
| **Functional interface** | **Detail** | **Ex** |
| **Function and BiFunction** | Function represents a **function that takes one type of argument and returns another type of argument**. Function<T, R> is the generic form where T is the type of the input to the function and R is the type of the result of the function. | ToIntFunction, ToLongFunction, ToDoubleFunction, ToIntBiFunction, ToLongBiFunction, ToDoubleBiFunction, LongToIntFunction, LongToDoubleFunction, IntToLongFunction, IntToDoubleFunction etc. |
| **Predicate and BiPredicate** | It represents a **predicate against which elements of the stream are tested**. This is used to filter elements from the java stream. Just like Function, there are primitive specific interfaces for int, long and double. |  |
| **Consumer and BiConsumer** | It represents **an operation that accepts a single input argument and returns no result**. It can be used to perform some action on all the elements of the java stream. |  |
| **Supplier** | Supplier represent **an operation through which we can generate new values in the stream**. |  |

**Optional (java.util.Optional):** It is a container object which may or may not contain a non-null value. If a value is present, isPresent() will return true and get() will return the value. Stream terminal operations return Optional object

**java.util.Spliterator: For supporting parallel execution** in Java 8 Stream API, Spliterator interface is used. Spliterator trySplitmethod returns a new Spliterator that manages a subset of the elements of the original Spliterator.

**Intermediate and Terminal operations**

* **Intermediate:** returns a new Stream, are **lazy** in nature. Ex: filer, map
* **Terminal:** returns result or produce a side effect, are **eager** in nature. Ex: forEach, toArray, min, max, findFirst, anyMatch, allMatch

**Short Circuiting Operations**

* An intermediate operation is called short circuiting, if it may produce finite stream for an infinite stream.
* For example limit() and skip() are two short circuiting intermediate operations.
* A terminal operation is called short circuiting, if it may terminate in finite time for infinite stream.
* For example anyMatch, allMatch, noneMatch, findFirst and findAny are short circuiting terminal operations.