**1. Stream API Interfaces**

* Stream<T> (for objects)
* IntStream, LongStream, DoubleStream (for primitives)

**2. Common Stream Methods**

**Creation**

* stream(), of(), generate(), iterate(), builder()

**Intermediate Operations (return a Stream, can be chained)**

* filter(Predicate<T>)
* map(Function<T,R>)
* mapToInt(ToIntFunction<T>), mapToLong, mapToDouble
* flatMap(Function<T, Stream<R>>)
* distinct()
* sorted(), sorted(Comparator<T>)
* peek(Consumer<T>)
* limit(long n)
* skip(long n)

**Terminal Operations (produce a result or side-effect, end the stream)**

* forEach(Consumer<T>)
* forEachOrdered(Consumer<T>)
* toArray()
* reduce(BinaryOperator<T>)
* collect(Collector<T,A,R>)
* min(Comparator<T>)
* max(Comparator<T>)
* count()
* anyMatch(Predicate<T>)
* allMatch(Predicate<T>)
* noneMatch(Predicate<T>)
* findFirst()
* findAny()

**Short-circuiting Operations**

* limit(), findFirst(), findAny(), anyMatch(), allMatch(), noneMatch()

**3. Example**

List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

List<String> result = names.stream()

    .filter(s -> s.startsWith("A"))

    .map(String::toUpperCase)

    .collect(Collectors.toList());

**4. Primitive Streams**

* IntStream, LongStream, DoubleStream have similar methods, plus:
  + sum(), average(), range(), rangeClosed()

**5. Collectors (for collect())**

* toList(), toSet(), toMap(), joining(), groupingBy(), partitioningBy(), counting(), summarizingInt(), etc.

**Note:**  
Streams are single-use and support functional-style operations for processing sequences of elements.