Streams

Parallel Streams

Parallel Streams

- All of the streams thus far have been sequential streams i.e. the streams have processed the data one element at a time.

- Parallel streams can process elements in a stream concurrently i.e. at the same time.

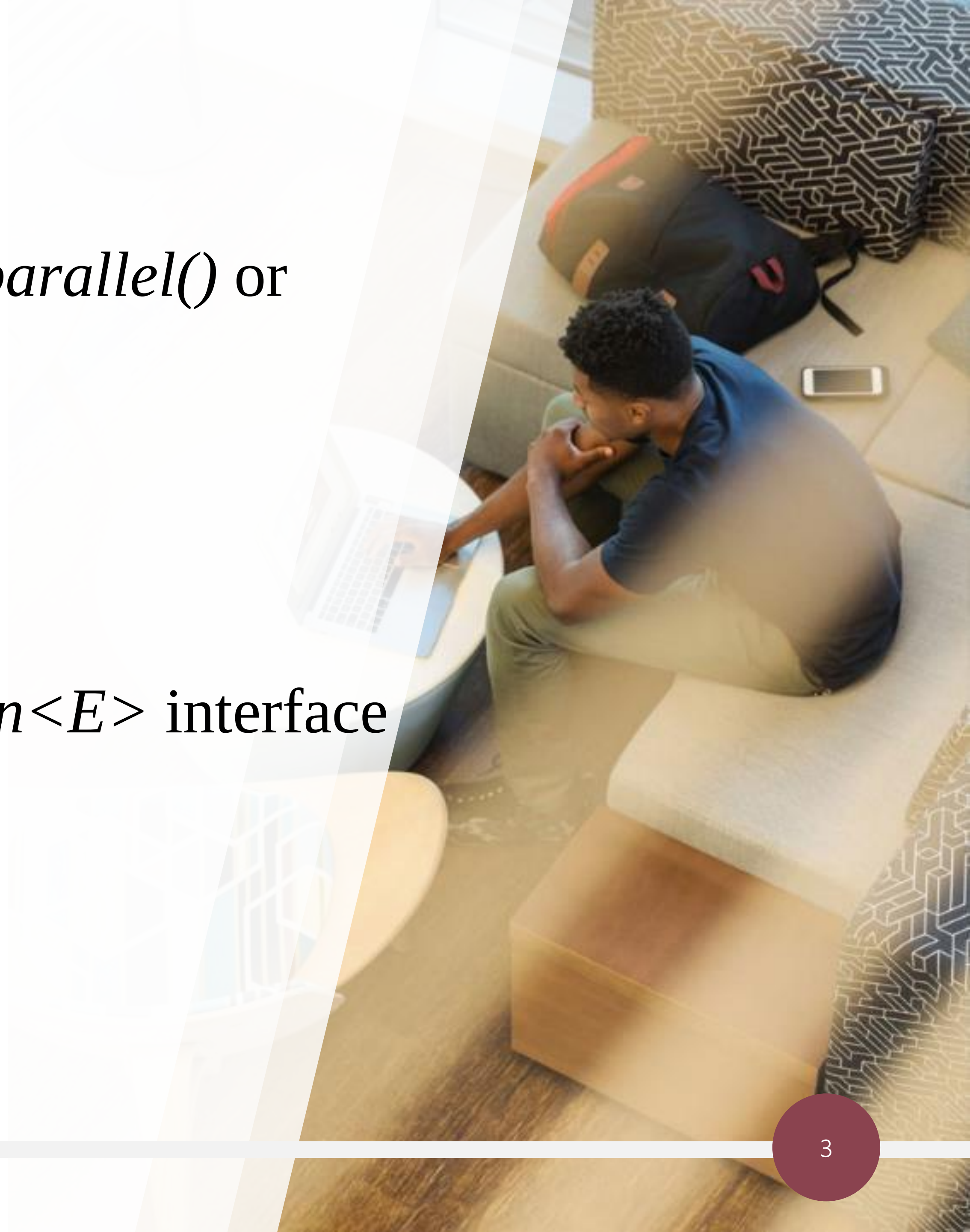
- Java achieves this by splitting the stream up into sub-streams and then the pipeline operations are performed on the sub-streams concurrently (each sub-stream has its own thread).

Parallel Streams

- To make a stream parallel, we can use the *parallel()* or *parallelStream()* methods.

- *parallel()* is available in *Stream<T>*.

- *parallelStream()* is defined in the *Collection<E>* interface



Parallel Streams

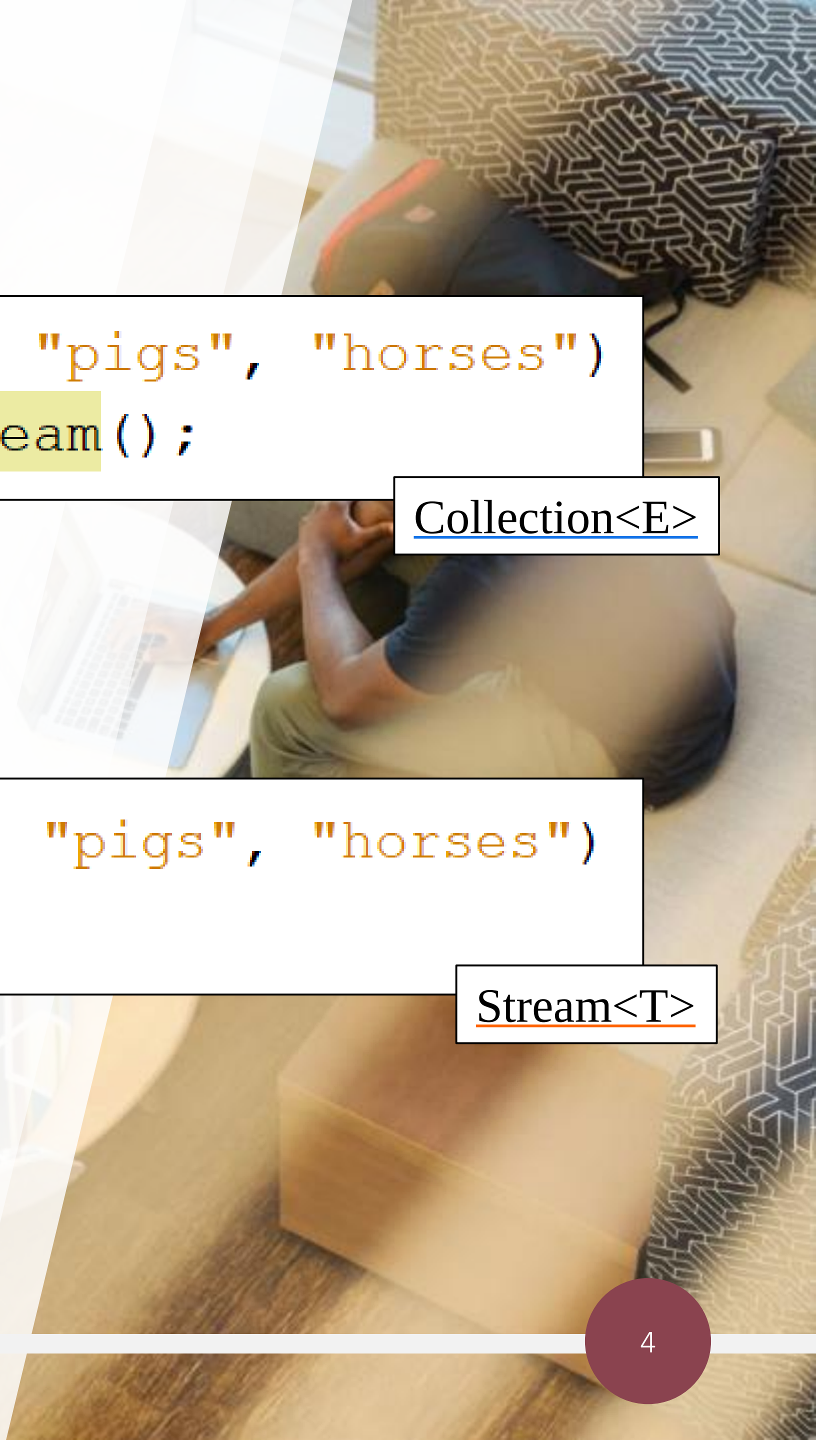
✓

```
Stream<String> animalsStream = List.of("sheep", "pigs", "horses")
                                        .parallelStream();
```

Collection<E>

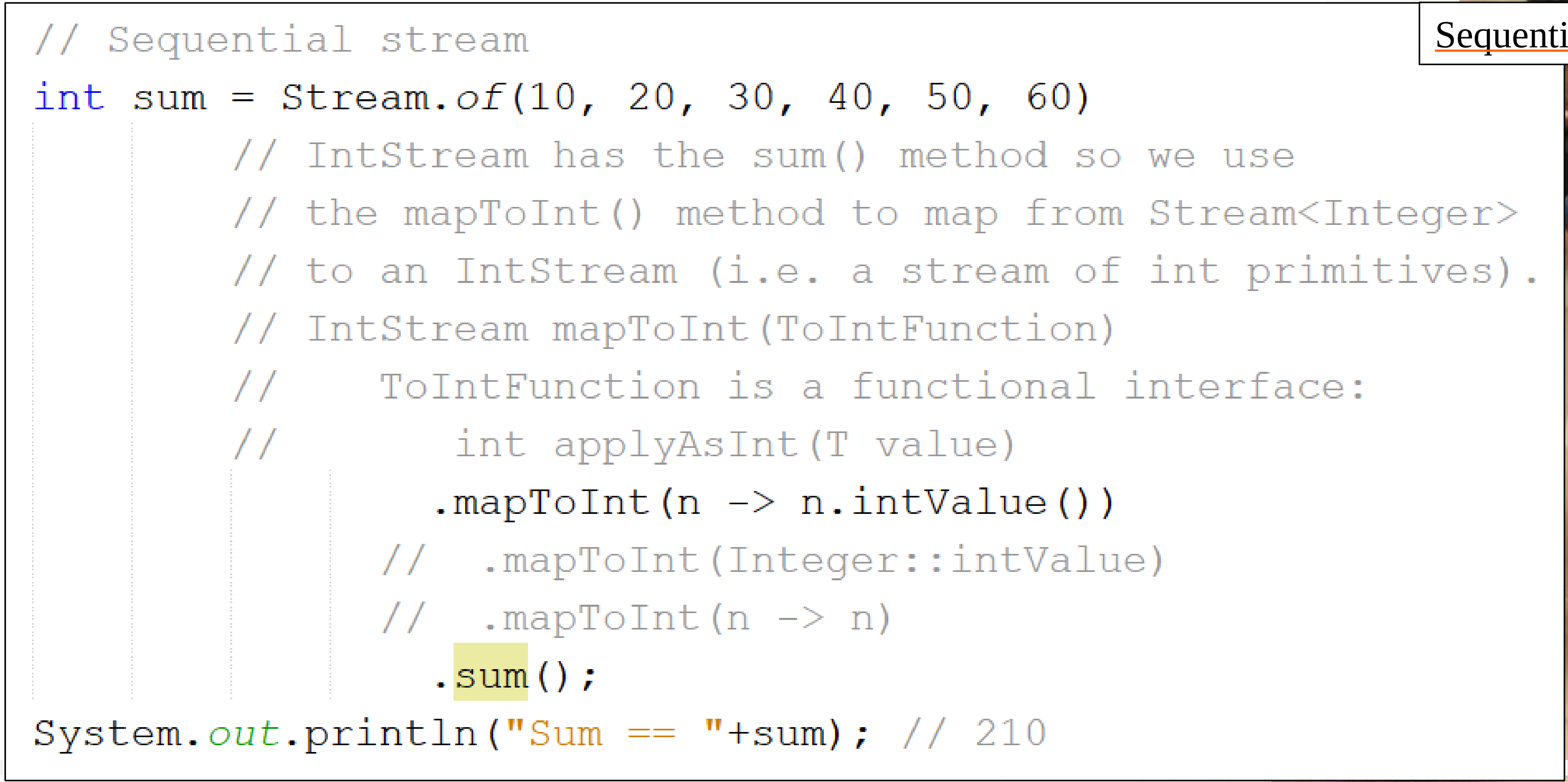
```
Stream<String> animalsStream = Stream.of("sheep", "pigs", "horses")
                                          .parallel();
```

Stream<T>



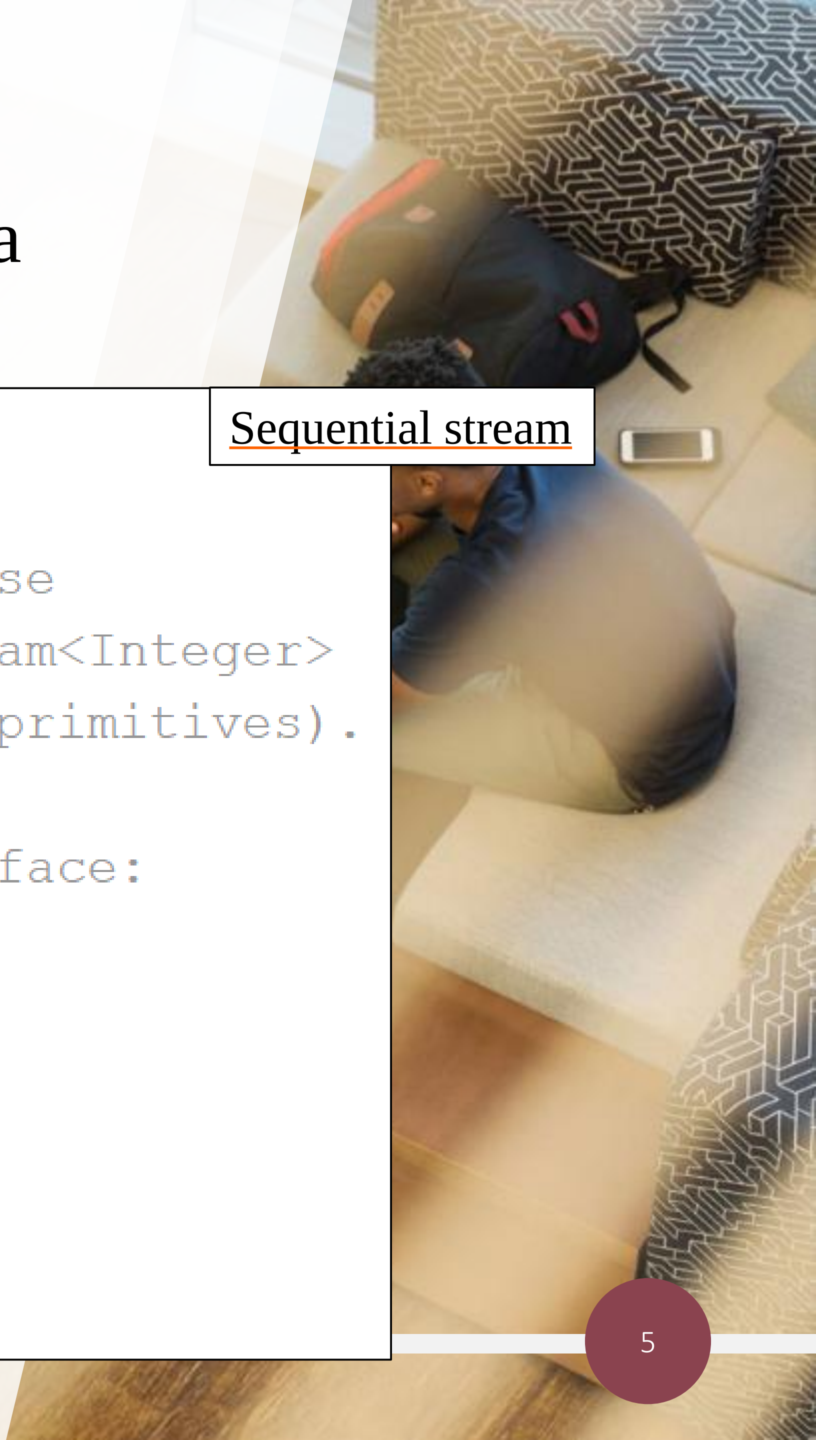
Parallel Streams

- Firstly, let’s look at a sequential stream that sums up a stream of numbers.



```
// Sequential stream
int sum = Stream.of(10, 20, 30, 40, 50, 60)
        // IntStream has the sum() method so we use
        // the mapToInt() method to map from Stream<Integer>
        // to an IntStream (i.e. a stream of int primitives).
        // IntStream mapToInt(ToIntFunction)
        //    ToIntFunction is a functional interface:
        //       int applyAsInt(T value)
                .mapToInt(n -> n.intValue())
              //  .mapToInt(Integer::intValue)
              //  .mapToInt(n -> n)
                .sum();
System.out.println("Sum == "+sum); // 210
```

Sequential stream



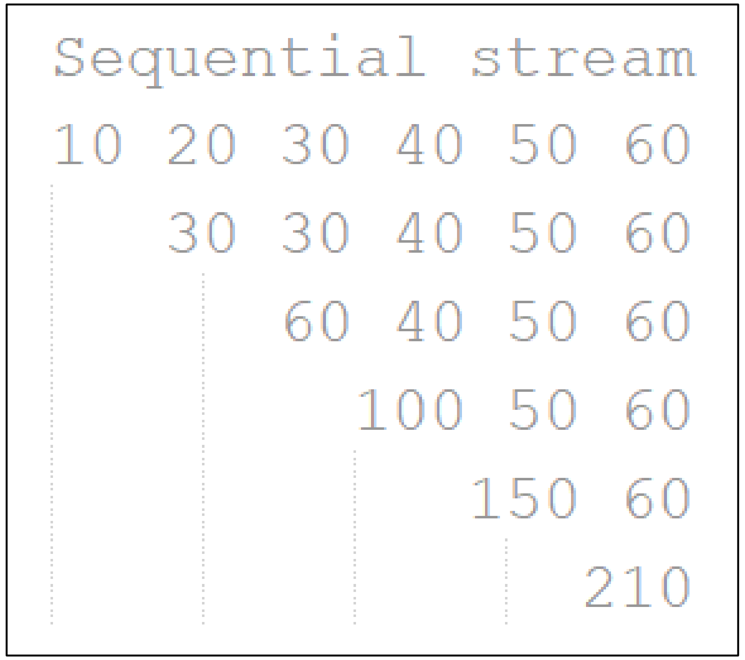
Parallel Streams

```
int sum = Stream.of(10, 20, 30, 40, 50, 60)
                .parallel()
                .mapToInt(Integer::intValue)
                .sum();
System.out.println("Sum == "+sum); // 210
```

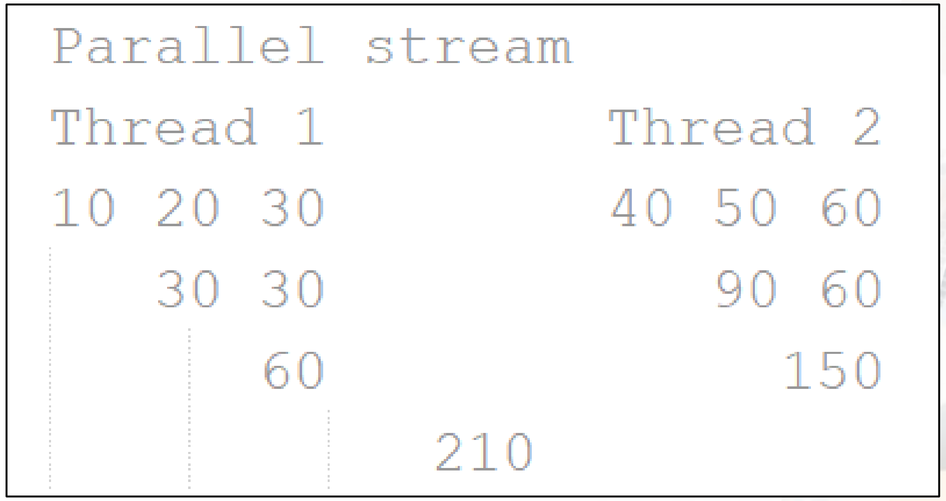
Parallel stream

Parallel Streams

- What is happening in the background?



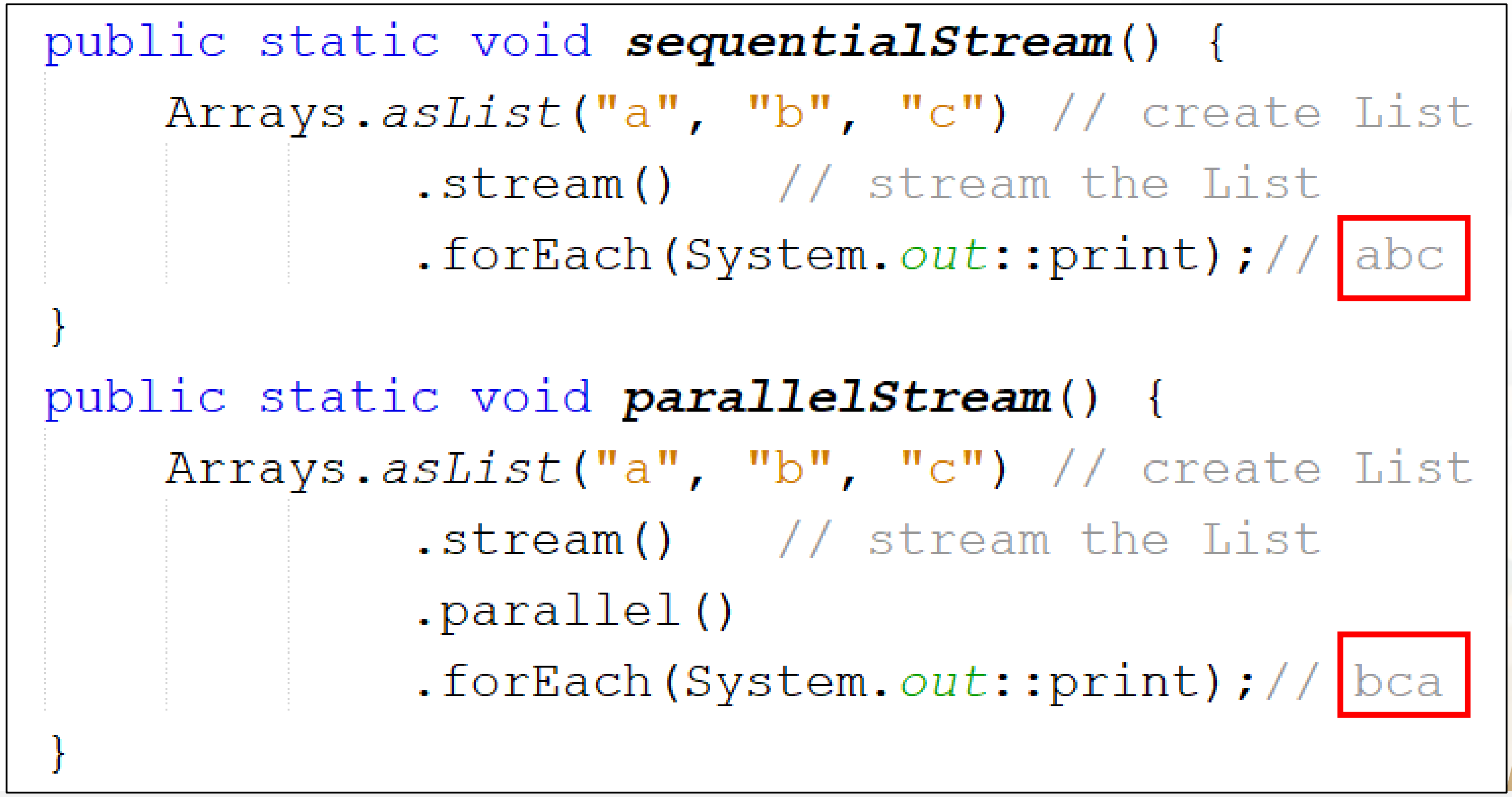
```
Sequential stream
10 20 30 40 50 60
   30 30 40 50 60
      60 40 50 60
        100 50 60
           150 60
              210
```



```
Parallel stream
Thread 1          Thread 2
10 20 30          40 50 60
   30 30             90 60
      60               150
           210
```

Parallel Streams

- Be careful if order is important, as the order of thread completion will determine the final result (not the order in the original collection).



```
public static void sequentialStream() {
    Arrays.asList("a", "b", "c") // create List
            .stream()    // stream the List
            .forEach(System.out::print);// abc
}
public static void parallelStream() {
    Arrays.asList("a", "b", "c") // create List
            .stream()    // stream the List
            .parallel()
            .forEach(System.out::print);// bca
}
```