17.18 Wrap-Up

In this chapter, you worked with lambdas, streams and functional interfaces. We presented many examples, often showing simpler ways to implement tasks that you programmed in earlier chapters.

You learned how to process elements in an `IntStream`—a stream of `int` values. You created an `IntStream` representing a closed range of `int`s, then used intermediate and terminal stream operations to create and process a stream pipeline that produced a result. We used lambdas to create anonymous methods that implemented functional interfaces and passed these lambdas to methods in stream pipelines to specify the processing steps for the streams' elements. We also created `IntStream`s from existing arrays of `int` values.

We discussed how a stream's intermediate processing steps are applied to each element before moving onto the next. We showed how to use a `forEach` terminal operation to perform an operation on each stream element. We used reduction operations to count the number of stream elements, determine the minimum and maximum values, and sum and average the values. You also used method `reduce` to create your own reduction operations.

You used intermediate operations to filter elements that matched a predicate and map elements to new values—in each

case, these operations produced intermediate streams on which you could perform additional processing. You also learned how to sort elements in ascending and descending order and how to sort objects by multiple fields.

We demonstrated how to store a stream pipeline's results in a collection by using various predefined `Collector` implementations provided by class `Collectors`. You also learned how to use a `Collector` to group elements into categories.

You learned that various classes can create stream data sources. For example, you used `Files` method `lines` to get a `Stream<String>` that read lines of text from a file and used `SecureRandom` method `ints` to get an `IntStream` of random values. You also learned how to convert an `IntStream` into a `Stream<Integer>` (via method `boxed`) so that you could use `Stream` method `collect` to summarize the frequencies of the `Integer` values and store the results in a `Map`.

We introduced infinite streams and showed how to limit the number of elements they generate. You saw how to implement an event-handling functional interface using a lambda. Finally, we presented some additional information about Java SE 8 interfaces and streams. In the next chapter, we discuss recursive programming in which methods call themselves either directly or indirectly.