a Stream, transforming each element of type T to another type R through a Function <T,R>. The flatMap() method flattens nested streams into a single level and removes empty streams.

Compare primitive streams to Stream<T>. Primitive streams are useful for performing common operations on numeric types, including statistics like average(), sum(), and so on. There are three primitive stream classes: DoubleStream, IntStream, and LongStream. There are also three primitive Optional classes: OptionalDouble, OptionalInt, and OptionalLong. Aside from BooleanSupplier, they all involve the double, int, or long primitives.

Convert primitive stream types to other primitive stream types. Normally, when mapping, you just call the map() method. When changing the class used for the stream, a different method is needed. To convert to Stream, you use mapToObj(). To convert to DoubleStream, you use mapToDouble(). To convert to IntStream, you use mapToInt(). To convert to LongStream, you use mapToLong().

Use peek() to inspect the stream. The peek() method is an intermediate operation often used for debugging purposes. It executes a lambda or method reference on the input and passes that same input through the pipeline to the next operator. It is useful for printing out what passes through a certain point in a stream.

Search a stream. The findFirst() and findAny() methods return a single element from a stream in an Optional. The anyMatch(), allMatch(), and noneMatch() methods return a boolean. Be careful, because these three can hang if called on an infinite stream with some data. All of these methods are terminal operations.

Sort a stream. The sorted() method is an intermediate operation that sorts a stream. There are two versions: the signature with zero parameters that sorts using the natural sort order, and the signature with one parameter that sorts using that Comparator as the sort order.

Compare groupingBy() and partitioningBy(). The groupingBy() method is a terminal operation that creates a Map. The keys and return types are determined by the parameters you pass. The values in the Map are a Collection for all the entries that map to that key. The partitioningBy() method also returns a Map. This time, the keys are true and false. The values are again a Collection of matches. If there are no matches for that boolean, the Collection is empty.

Review Questions

The answers to the chapter review questions can be found in the Appendix.

1. What could be the output of the following?

```
var stream = Stream.iterate("", (s) -> s + "1");
System.out.println(stream.limit(2).map(x -> x + "2"));
```

A. 12112

B. 212

C. 212112

D. java.util.stream.ReferencePipeline$3@4517d9a3

E. The code does not compile.

F. An exception is thrown.

G. The code hangs.

2. What could be the output of the following?

```
Predicate<String> predicate = s -> s.startsWith("g");
var stream1 = Stream.generate(() -> "growl!");
var stream2 = Stream.generate(() -> "growl!");
var b1 = stream1.anyMatch(predicate);
var b2 = stream2.allMatch(predicate);
System.out.println(b1 + " " + b2);
```

A. true false

B. true true

C. java.util.stream.ReferencePipeline$3@4517d9a3

D. The code does not compile.

E. An exception is thrown.

F. The code hangs.

3. What could be the output of the following?

```
Predicate<String> predicate = s -> s.length()> 3;
var stream = Stream.iterate("-",
  s -> ! s.isEmpty(), (s) -> s + s);
var b1 = stream.noneMatch(predicate);
var b2 = stream.anyMatch(predicate);
System.out.println(b1 + " " + b2);
```

A. false false

B. false true

C. java.util.stream.ReferencePipeline$3@4517d9a3

D. The code does not compile.

E. An exception is thrown.

F. The code hangs.

4. Which are true statements about terminal operations in a stream that runs successfully? (Choose all that apply.)

   A. At most one terminal operation can exist in a stream pipeline.

   B. Terminal operations are a required part of the stream pipeline in order to get a result.

   C. Terminal operations have Stream as the return type.

   D. The peek() method is an example of a terminal operation.

   E. The referenced Stream may be used after calling a terminal operation.

5. Which of the following sets result to 8.0? (Choose all that apply.)

   A.
```
double result = LongStream.of(6L, 8L, 10L)
    .mapToInt(x -> (int) x)
    .collect(Collectors.groupingBy(x -> x))
    .keySet()
    .stream()
    .collect(Collectors.averagingInt(x -> x));
```

   B.
```
double result = LongStream.of(6L, 8L, 10L)
    .mapToInt(x -> x)
    .boxed()
    .collect(Collectors.groupingBy(x -> x))
    .keySet()
    .stream()
    .collect(Collectors.averagingInt(x -> x));
```

   C.
```
double result = LongStream.of(6L, 8L, 10L)
    .mapToInt(x -> (int) x)
    .boxed()
    .collect(Collectors.groupingBy(x -> x))
    .keySet()
     .stream()
    .collect(Collectors.averagingInt(x -> x));
```

   D.
```
double result = LongStream.of(6L, 8L, 10L)
    .mapToInt(x -> (int) x)
    .collect(Collectors.groupingBy(x -> x, Collectors.toSet()))
    .keySet()
    .stream()
    .collect(Collectors.averagingInt(x -> x));
```

E. double result = LongStream.of(6L, 8L, 10L)

```
    .mapToInt(x -> x)
    .boxed()
    .collect(Collectors.groupingBy(x -> x, Collectors.toSet()))
    .keySet()
    .stream()
    .collect(Collectors.averagingInt(x -> x));
```

F. double result = LongStream.of(6L, 8L, 10L)

```
    .mapToInt(x -> (int) x)
    .boxed()
    .collect(Collectors.groupingBy(x -> x, Collectors.toSet()))
    .keySet()
    .stream()
    .collect(Collectors.averagingInt(x -> x));
```

6. Which of the following can fill in the blank so that the code prints out false? (Choose all that apply.)

```
var s = Stream.generate(() -> "meow");
var match = s.______(String::isEmpty);
System.out.println(match);
```

A. allMatch

B. anyMatch

C. findAny

D. findFirst

E. noneMatch

F. None of the above

7. We have a method that returns a sorted list without changing the original. Which of the following can replace the method implementation to do the same with streams?

```
private static List<String> sort(List<String> list) {
  var copy = new ArrayList<String>(list);
  Collections.sort(copy, (a, b) -> b.compareTo(a));
  return copy;
}
```

A.
```
return list.stream()
   .compare((a, b) -> b.compareTo(a))
   .collect(Collectors.toList());
```

B.
```
return list.stream()
   .compare((a, b) -> b.compareTo(a))
   .sort();
```

C.
```
return list.stream()
   .compareTo((a, b) -> b.compareTo(a))
   .collect(Collectors.toList());
```

D.
```
return list.stream()
   .compareTo((a, b) -> b.compareTo(a))
   .sort();
```

E.
```
return list.stream()
   .sorted((a, b) -> b.compareTo(a))
   .collect();
```

F.
```
return list.stream()
   .sorted((a, b) -> b.compareTo(a))
   .collect(Collectors.toList());
```

8. Which of the following are true given this declaration? (Choose all that apply.)

```
var is = IntStream.empty();
```

A. is.average() returns the type int.

B. is.average() returns the type OptionalInt.

C. is.findAny() returns the type int.

D. is.findAny() returns the type OptionalInt.

E. is.sum() returns the type int.

F. is.sum() returns the type OptionalInt.

9. Which of the following can we add after line 6 for the code to run without error and not produce any output? (Choose all that apply.)

```
4: var stream = LongStream.of(1, 2, 3);
5: var opt = stream.map(n -> n * 10)
6:   .filter(n -> n < 5).findFirst();
```

A. `if (opt.isPresent())`
`    System.out.println(opt.get());`

B. `if (opt.isPresent())`
`    System.out.println(opt.getAsLong());`

C. `opt.ifPresent(System.out.println);`

D. `opt.ifPresent(System.out::println);`

E. None of these; the code does not compile.

F. None of these; line 6 throws an exception at runtime.

10. Given the four statements (L, M, N, O), select and order the ones that would complete the expression and cause the code to output 10 lines. (Choose all that apply.)

```
Stream.generate(() -> "1")
  L: .filter(x -> x.length()> 1)
  M: .forEach(System.out::println)
  N: .limit(10)
  O: .peek(System.out::println)
;
```

A. L, N

B. L, N, O

C. L, N, M

D. L, N, M, O

E. L, O, M

F. N, M

G. N, O

11. What changes need to be made together for this code to print the string 12345? (Choose all that apply.)

```
Stream.iterate(1, x -> x++)
  .limit(5).map(x -> x)
  .collect(Collectors.joining());
```

A. Change Collectors.joining() to Collectors.joining(",").

B. Change map(x -> x) to map(x -> "" + x).

C. Change x -> x++ to x -> ++x.

D. Add .forEach(System.out::print) after the call to collect().

E. Wrap the entire line in a System.out.print statement.

F. None of the above. The code already prints 12345.

12. Which is true of the following code?

```
Set<String> birds = Set.of("oriole", "flamingo");
Stream.concat(birds.stream(), birds.stream(), birds.stream())
  .sorted()     // line X
  .distinct()
  .findAny()
  .ifPresent(System.out::println);
```

A. It is guaranteed to print flamingo as is and when line X is removed.

B. It is guaranteed to print oriole as is and when line X is removed.

C. It is guaranteed to print flamingo as is, but not when line X is removed.

D. It is guaranteed to print oriole as is, but not when line X is removed.

E. The output may vary as is.

F. The code does not compile.

G. It throws an exception because the same list is used as the source for multiple streams.

13. Which of the following is true?

```
List<Integer> x1 = List.of(1, 2, 3);
List<Integer> x2 = List.of(4, 5, 6);
List<Integer> x3 = List.of();
Stream.of(x1, x2, x3).map(x -> x + 1)
  .flatMap(x -> x.stream())
  .forEach(System.out::print);
```

A. The code compiles and prints 123456.

B. The code compiles and prints 234567.

C. The code compiles but does not print anything.

D. The code compiles but prints stream references.

E. The code runs infinitely.

F. The code does not compile.

G. The code throws an exception.

14. Which of the following are true? (Choose all that apply.)

```
4: Stream<Integer> s = Stream.of(1);
5: IntStream is = s.boxed();
6: DoubleStream ds = s.mapToDouble(x -> x);
7: Stream<Integer> s2 = ds.mapToInt(x -> x);
8: s2.forEach(System.out::print);
```

A. Line 4 causes a compiler error.

B. Line 5 causes a compiler error.

C. Line 6 causes a compiler error.

D. Line 7 causes a compiler error.

E. Line 8 causes a compiler error.

F. The code compiles but throws an exception at runtime.

G. The code compiles and prints 1.

15. Given the generic type String, the partitioningBy() collector creates a Map<Boolean, List<String>> when passed to collect() by default. When a downstream collector is passed to partitioningBy(), which return types can be created? (Choose all that apply.)

A. Map<boolean, List<String>>

B. Map<Boolean, List<String>>

C. Map<Boolean, Map<String>>

D. Map<Boolean, Set<String>>

E. Map<Long, TreeSet<String>>

F. None of the above

16. Which of the following statements are true about this code? (Choose all that apply.)

```
20: Predicate<String> empty = String::isEmpty;
21: Predicate<String> notEmpty = empty.negate();
22:
23: var result = Stream.generate(() -> "")
24:    .limit(10)
25:    .filter(notEmpty)
26:    .collect(Collectors.groupingBy(k -> k))
```

```
27:   .entrySet()
28:   .stream()
29:   .map(Entry::getValue)
30:   .flatMap(Collection::stream)
31:   .collect(Collectors.partitioningBy(notEmpty));
32: System.out.println(result);
```

A. It outputs {}.

B. It outputs {false=[], true=[]}.

C. If we changed line 31 from partitioningBy(notEmpty) to groupingBy(n -> n), it would output {}.

D. If we changed line 31 from partitioningBy(notEmpty) to groupingBy(n -> n), it would output {false=[], true=[]}.

E. The code does not compile.

F. The code compiles but does not terminate at runtime.

17. What is the result of the following?

```
var s = DoubleStream.of(1.2, 2.4);
s.peek(System.out::println).filter(x -> x> 2).count();
```

A. 1

B. 2

C. 2.4

D. 1.2 and 2.4

E. There is no output.

F. The code does not compile.

G. An exception is thrown.

18. What is the output of the following?

```
11: public class Paging {
12:   record Sesame(String name, boolean human)  {
13:     @Override public String toString() {
14:       return name();
15:     }
16:   }
17:   record Page(List<Sesame> list, long count)  {}
18:
19:   public static void main(String[] args) {
20:     var monsters = Stream.of(new Sesame("Elmo", false));
21:     var people = Stream.of(new Sesame("Abby", true));
22:     printPage(monsters, people);
}

private static void printPage(Stream<Sesame> monsters,
   Stream<Sesame> people) {
  Page page = Stream.concat(monsters, people)
   .collect(Collectors.teeing(
     Collectors.filtering(s -> s.name().startsWith("E"),
       Collectors.toList()),
     Collectors.counting(),
     (l, c) -> new Page(l, c)));
  System.out.println(page);
}}
```

A. Page[list=[Abby], count=1]

B. Page[list=[Abby], count=2]

C. Page[list=[Elmo], count=1]

D. Page[list=[Elmo], count=2]

E. The code does not compile due to Stream.concat().

F. The code does not compile due to Collectors.teeing().

G. The code does not compile for another reason.

19. What is the simplest way of rewriting this code?

```
List<Integer> x = IntStream.range(1, 6)
  .mapToObj(i -> i)
  .collect(Collectors.toList());
x.forEach(System.out::println);
```

A.
```
IntStream.range(1, 6);
```

B.
```
IntStream.range(1, 6)
   .forEach(System.out::println);
```

C.
```
IntStream.range(1, 6)
   .mapToObj(i -> i)
   .forEach(System.out::println);
```

D. None of the above is equivalent.

E. The provided code does not compile.

20. Which of the following throw an exception when an Optional is empty? (Choose all that apply.)

A. opt.orElse("");

B. opt.orElseGet(() -> "");

C. opt.orElseThrow();

D. opt.orElseThrow(() -> throw new Exception());

E. opt.orElseThrow(RuntimeException::new);

F. opt.get();

G. opt.get("");

21. What is the output of the following?

```
var spliterator = Stream.generate(() -> "x")
  .spliterator();

spliterator.tryAdvance(System.out::print);
var split = spliterator.trySplit();
split.tryAdvance(System.out::print);
```

A. x

B. xx

C. A long list of x's

D. There is no output.

E. The code does not compile.

F. The code compiles but does not terminate at runtime.