Due to the print book page limit, we cannot inlcude all good CheckPoint questions in the physical book. The CheckPoint on this Website may contain extra questions not printed in the book. The questions in some sections may have been reordered as a result. Nevertheless, it is easy to find the CheckPoint questions in the book on this Website. Please send suggestions and errata to Dr. Liang at y.daniel.liang@gmail.com. Indicate the book, edition, and question number in your email. Thanks!

Chapter 30 Check Point Questions

Section 30.1

▼30.1.1

What are the benefits of using aggregate operations on collection streams for processing data?

Using aggregate operations on collection streams simplifies coding and improves performance.

Hide Answer

Section 30.2

▼30.2.1

Show the output of the following code?

```
Character[] chars = {'D', 'B', 'A', 'C'};
System.out.println(Stream.of(chars).sorted().findFirst().get());
System.out.println(Stream.of(chars).sorted(
  java.util.Comparator.reverseOrder()).findFirst().get());
System.out.println(Stream.of(chars)
  .limit(2).sorted().findFirst().get());
System.out.println(Stream.of(chars).distinct()
  .skip(2).filter(e -> e > 'A').findFirst().get());
System.out.println(Stream.of(chars)
  .max(Character::compareTo).get());
System.out.println(Stream.of(chars)
  .max(java.util.Comparator.reverseOrder()).get());
System.out.println(Stream.of(chars)
  .filter(e -> e > 'A').findFirst().get());
System.out.println(Stream.of(chars)
  .allMatch(e -> e >= 'A'));
System.out.println(Stream.of(chars)
  .anyMatch(e -> e > 'F'));
System.out.println(Stream.of(chars)
  .noneMatch(e -> e > 'F'));
Stream.of(chars).map(e -> e + "").map(e -> e.toLowerCase())
  .forEach(System.out::println);

Object[] temp = Stream.of(chars).map(e -> e + "Y")
  .map(e -> e.toLowerCase()).sorted().toArray();
System.out.println(java.util.Arrays.toString(temp));
```

```
A
D
B
C
```

```
D
A
D
true
false
true
d
b
a
c
[ay, by, cy, dy]
```

Hide Answer

30.2.2

What is wrong in the following code?

```
Character[] chars = {'D', 'B', 'A', 'C'};
Stream<Character> stream = Stream.of(chars).sorted();
System.out.println(stream.findFirst());
System.out.println(stream.skip(2).findFirst());
```

A stream can only have one terminal operation. Once you apply stream.findFirst() . the stream is destroyed.

Hide Answer

30.2.3

Rewrite (a) using a method reference and an anonymous inner class and (b) using lambda expression and an anonymous inner class.

```
(a) sorted((s1, s2) -> s1.compareToIgnoreCase(s2))
(b) forEach(System.out::println)
```

```
(a)
Use a method reference:
  sorted(String::compareToIgnoreCase);
Use an anonymous inner class:
sorted(new Comparator<String>() {
  @Override
  public int compare(String s1, String s2) {
    return s1.compareToIgnoreCase(s2);
  }
});

(b)
Use a lambda expression:
  forEach(e -> System.out.println())

Use an anonymous inner class:
forEach(
   new java.util.function.Consumer<String>() {
     public void accept(String e) {
       System.out.println();
     }
```

```
    }
  )
```

Hide Answer

30.2.4

Given a map of the type Map<String, Double>, write an expression that returns the sum of all the values in map. For example, if the map contains {"john", 1.5} and {"Peter", 1.1}, the sum is 2.6.

```
map.entrySet().stream().mapToDouble(e -> e.getValue()).sum()
```

Hide Answer

Section 30.3

30.3.1

Show the output of the following code?

```
int[] numbers = {1, 4, 2, 3, 1};
System.out.println(IntStream.of(numbers)
  .sorted().findFirst().getAsInt());
System.out.println(IntStream.of(numbers)
  .limit(2).sorted().findFirst().getAsInt());
System.out.println(IntStream.of(numbers).distinct()
  .skip(1).filter(e -> e > 2).sum());
System.out.println(IntStream.of(numbers).distinct()
  .skip(1).filter(e -> e > 2).average().getAsDouble());
System.out.println(IntStream.of(numbers).max().getAsInt());
System.out.println(IntStream.of(numbers).max().getAsInt());
System.out.println(IntStream.of(numbers)
  .filter(e -> e > 1).findFirst().getAsInt());
System.out.println(IntStream.of(numbers)
  .allMatch(e -> e >= 1));
System.out.println(IntStream.of(numbers)
  .anyMatch(e -> e > 4));
System.out.println(IntStream.of(numbers).noneMatch(e -> e > 4));
IntStream.of(numbers).mapToObj(e -> (char)(e + 50))
  .forEach(System.out::println);

Object[] temp = IntStream.of(numbers)
  .mapToObj(e -> (char)(e + 'A')).toArray();
System.out.println(java.util.Arrays.toString(temp));
```

```
1
1
7
3.5
4
4
4
true
false
true
3
```

```
6
4
5
3
[B, E, C, D, B]
```

Hide Answer

30.3.2

What is wrong in the following code?

```
int[] numbers = {1, 4, 2, 3, 1};
DoubleSummaryStatistics stats =
  DoubleStream.of(numbers).summaryStatistics();
System.out.printf("The summary of the stream is\n%-10s%10d\n" +
  "%-10s%10.2f\n%-10s%10.2f\n%-10s%10.2f\n%-10s%10.2f\n",
  "  Count:", stats.getCount(), "  Max:", stats.getMax(),
  "  Min:", stats.getMin(), "  Sum:", stats.getSum(),
  "  Average:", stats.getAverage());
```

numbers is an int array, you have to use IntStream.of(numbers) rather than DoubleStream.of (numbers).

Hide Answer

30.3.3

Rewrite the following code that maps an int to a Character using an anonymous inner class?

```
mapToObj(e -> (char)(e + 50))
```

```
mapToObj(
  new java.util.function.IntFunction<Character>() {
    public Character apply(int e) {
      return (char)(e + 50);
    }
  }
)
```

Hide Answer

30.3.4

Show the output of the following code.

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}};
System.out.println(Stream.of(m)
  .mapToInt(e -> IntStream.of(e).sum()).sum());
```

21

Hide Answer

30.3.5

Given an array names in Listing 30.1, write the code to display the total number of characters in names.

```
System.out.println("The number of characters in array names is " +
    Stream.of(names).mapToInt(e -> e.length()).sum());
```

Hide Answer

Section 30.4

30.4.1

What is a stateless method? What is a stateful method?

A stateless method can apply to the elements in the stream independent from the others. A stateful method must consider all the elements in order to produce a result.

Hide Answer

30.4.2

How do you create a parallel stream?

You can create a parallel stream by invoking the parallel() method on a stream or invoking the parallelStream() method from a collection object such as a list or a set.

Hide Answer

30.4.3

Suppose names is a set of strings, which of the following two streams is better?

```
Object[] s = set.parallelStream().filter(e -> e.length() > 3)
  .sorted().toArray();

Object[] s = set.parallelStream().sorted()
  .filter(e -> e.length() > 3).toArray();
```

The former is better than the latter because the stream size is smaller after applying the filter method. This will make the sorted() method to run faster.

Hide Answer

30.4.4

What will be the output of the following code?

```
int[] values = {3, 4, 1, 5, 20, 1, 3, 3, 4, 6};
System.out.print("The values are ");
  IntStream.of(values)
    .forEach(e -> System.out.print(e + " "));
```

The values are 3 4 1 5 20 1 3 3 4 6

Hide Answer

30.4.5

What will be the output of the following code?

```
int[] values = {3, 4, 1, 5, 20, 1, 3, 3, 4, 6};
System.out.print("The values are ");
```

```
IntStream.of(values).parallel()
  .forEach(e -> System.out.print(e + " "));
```

The output is unpredictable due to using a parallel stream with forEach method.

Hide Answer

30.4.6

Write a statement to obtain an array of 1000 random double values between 0.0 and 1.0, excluding 1.0.

```
Random r = new Random();
double[] numbers = r.doubles(1000, 0.0, 1.0).toArray();
```

Hide Answer

Section 30.5

30.5.1

Show the output of the following code.

```
int[] values = {1, 2, 3, 4};
System.out.println(IntStream.of(values)
  .reduce(0, (e1, e2) -> e1 + e2));
System.out.println(IntStream.of(values)
  .reduce(1, (e1, e2) -> e1 * e2));
System.out.println(IntStream.of(values).map(e -> e * e)
  .reduce(0, (e1, e2) -> e1 + e2));
System.out.println(IntStream.of(values).mapToObj(e -> "" + e)
  .reduce((e1, e2) -> e1  + " " + e2).get());
System.out.println(IntStream.of(values).mapToObj(e -> "" + e)
  .reduce((e1, e2) -> e1  + ", " + e2).get());
```

```
10
24
30
1 2 3 4
1, 2, 3, 4
```

Hide Answer

30.5.2

Show the output of the following code.

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}};
System.out.println(Stream.of(m)
  .map(e -> IntStream.of(e).reduce(1, (e1, e2) -> e1 * e2))
  .reduce(1, (e1, e2) -> e1 * e2));
```

```
720
```

Hide Answer

30.5.3

Show the output of the following code.

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}, {1, 3}};
Stream.of(m).map(e -> IntStream.of(e))
  .reduce((e1, e2) -> IntStream.concat(e1, e2))
  .get().distinct()
  .forEach(e -> System.out.print(e + " "));
```

1 2 3 4 5 6

Hide Answer

30.5.4

Show the output of the following code.

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}, {1, 3}};
System.out.println(
  Stream.of(m).map(e -> IntStream.of(e))
    .reduce((e1, e2) -> IntStream.concat(e1, e2))
    .get().distinct().mapToObj(e -> e + "")
    .reduce((e1, e2) -> e1 + ", " + e2).get());
```

1, 2, 3, 4, 5, 6, 1, 3

Hide Answer

Section 30.6

30.6.1

Show the output of the following code.

```
int[] values = {1, 2, 3, 4, 1};
List<Integer> list = IntStream.of(values).mapToObj(e -> e)
  .collect(Collectors.toList());
System.out.println(list);

Set<Integer> set = IntStream.of(values).mapToObj(e -> e)
  .collect(Collectors.toSet());
System.out.println(set);

Map<Integer, Integer> map = IntStream.of(values).distinct()
  .mapToObj(e -> e)
  .collect(Collectors.toMap(e -> e, e -> e.hashCode()));
System.out.println(map);

System.out.println(
  IntStream.of(values).mapToObj(e -> e)
    .collect(Collectors.summingInt(e -> e)));

System.out.println(
  IntStream.of(values).mapToObj(e -> e)
    .collect(Collectors.averagingDouble(e -> e)));
```

```
[1, 2, 3, 4, 1]
[1, 2, 3, 4]
```

```
{1=1, 2=2, 3=3, 4=4}
11
2.2
```

Hide Answer

Section 30.7

30.7.1

Show the output of the following code.

```
int[] values = {1, 2, 2, 3, 4, 2, 1};
IntStream.of(values).mapToObj(e -> e).collect(
  Collectors.groupingBy(e -> e, TreeMap::new,
    Collectors.counting())).
  forEach((k, v) -> System.out.println(k + " occurs " + v
      + (v > 1 ? " times " : " time ")));

IntStream.of(values).mapToObj(e -> e).collect(
  Collectors.groupingBy(e -> e, TreeMap::new,
    Collectors.summingInt(e -> e))).
  forEach((k, v) -> System.out.println(k + ": " + v));

MyStudent[] students = {
  new MyStudent("John", "Johnson", "CS", 23, 89.2),
  new MyStudent("Susan", "Johnson", "Math", 21, 89.1),
  new MyStudent("John", "Peterson", "CS", 21, 92.3),
  new MyStudent("Kim", "Yao", "Math", 22, 87.3),
  new MyStudent("Jeff", "Johnson", "CS", 23, 78.5)};

Stream.of(students)
  .sorted(Comparator.comparing(MyStudent::getLastName)
    .thenComparing(MyStudent::getFirstName))
  .forEach(e -> System.out.println(e.getLastName() + ", " +
                e.getFirstName()));

Stream.of(students).collect(Collectors.
  groupingBy(MyStudent::getAge, TreeMap::new,
    Collectors.averagingDouble(MyStudent::getScore))).
  forEach((k, v) -> System.out.printf("%10s%10.2f\n", k, v));
```

```
1 occurs 2 times
2 occurs 3 times
3 occurs 1 time
4 occurs 1 time
1: 2
2: 6
3: 3
4: 4
Johnson, Jeff
Johnson, John
Johnson, Susan
Peterson, John
Yao, Kim
        21     90.70
```

```
22      87.30
23      83.85
```

Hide Answer

Section 30.8

30.8.1

Can the following code be used to replace line 19 in Listing 30.7?

```
DoubleStream.of(numbers).filter(e -> e >
  DoubleStream.of(numbers).average()).count());
```

No. You have to use:

```
DoubleStream.of(numbers).filter(e -> e >
  DoubleStream.of(numbers).average().getAsDouble()).count());
```

Hide Answer

30.8.2

Can the following code be used to replace lines 15-16 in Listing 30.8?

```
Stream.of(chars).forEach(e -> {
  int count = 0;
  System.out.print(e + (++count % 20 == 0 ? "\n" : " ")); });
```

No. count will always be 0.

Hide Answer

30.8.3

Show the output of the following code?

```
String s = "ABC";
Stream.of(s.toCharArray()).forEach(ch ->
  System.out.println(ch));
```

ABC

Hide Answer

30.8.4

Show the output of the following code? (The toCharacterArray method is presented in Listing 30.9)

```
String s = "ABC";
Stream.of(toCharacterArray(s.toCharArray())).forEach(ch ->
  System.out.println(ch));
```

```
A
B
C
```

Hide Answer

30.8.5

Write the code to obtain a one-dimensional array list of strings from a two-dimensional array matrix of strings.

```
String[] list = Stream.of(m).map(e -> Stream.of(e)).
  reduce((e1, e2) -> Stream.concat(e1, e2)).get().toArray();
```

Hide Answer