SELF CHECK

• 1. Consider the computation

```
long result = words
   .filter(w -> w.length() > 10)
   .limit(5)
   .count();
```

Which of the following statements is true?

- [ ] The result cannot be zero.
- [x] The result is at most 5.
- [ ] The `length` method is invoked on exactly five elements of the stream.
- [ ] The `length` method is invoked on all elements of the stream.

One correct, 0 errors, 100%

• 2. What does the following code do?

```
Stream<String> words = . . .;
long count = words
   .filter(w -> w.length() == 0)
   .count();
```

- [x] It counts how many strings in the stream are empty.
- [ ] It counts how many strings in the stream are not empty.
- [ ] It counts how many strings in the stream are not `null`.
- [ ] It is a syntax error because you can't have a variable and a method with the same name.

One correct, 0 errors, 100%

••• 3. Given a stream of strings, remove all empty strings.

StreamDemo.java

```
1  import java.util.List;
2  import java.util.stream.Stream;
3
4  public class StreamDemo
5  {
6     public static void main(String[] args)
7     {
8        Stream<String> words =
9           Stream.of("Mary", "", "had", "", "a", "little", "lamb", "");
10
11       Stream<String> result = words.filter(w->w.length() > 0);
12
13       Util.print(result);
14    }
15 }
```

CodeCheck Reset

Score: 1/1

••• Collect the first five strings of length greater than ten in a `List<String>` without using streams.

4.

CollectStrings.java

```
1  import java.util.List;
2  import java.util.Arrays;
3  import java.util.ArrayList;
4
5  public class CollectStrings
6  {
7     /**
8      * Return a List<String> containing the
9      * first 5 words longer than 10 characters in strings.
10     */
11    public static List<String> longerThanTen(List<String> strings)
12    {
13       List<String> result = new ArrayList<>();
14       int count = 0;
15       for (int i = 0; i < strings.size(); i++) {
16       if (strings.get(i).length() > 10 && result.contains(strings.get(i)) == false && c
17          result.add(count, strings.get(i));
18          count++;
19          }
20       }
21
22       return result;
23    }
24 }
```

CodeCheck Reset

Calling with Arguments

	Name	Arguments
pass	longerThanTen	Arrays.asList("France", "Italy", "Germany", "Bulgaria", "Romania", "Denmark"
pass	longerThanTen	Arrays.asList("France", "Italy", "Germany", "Bulgaria", "Romania", "Denmark"
pass	longerThanTen	Arrays.asList("France", "Italy", "Germany", "Bulgaria", "Romania", "Denmark"

Score

3/3

••• Given a stream of strings, calculate how many have exactly ten characters.

5.

CountStrings.java

```
1  import java.util.stream.Stream;
2
3  public class CountStrings
4  {
5     public static void main(String[] args)
6     {
7        Stream<String> words = Util.getWords();
8
9        long count = words.filter(w -> w.length() == 10).count();
10
11       System.out.println("Words exactly 10 characters long: " + count);
12    }
13 }
```

CodeCheck Reset

Score: 1/1

SELF CHECK

- **1.** Which of the following makes a stream containing the numbers 1 1 2 3 5 8 11?

  - [ ] `new Stream<int>(1, 1, 2, 3, 5, 8, 11)`
  - [ ] `new Stream<Integer>(1, 1, 2, 3, 5, 8, 11)`
  - [x] `Stream.of(1, 1, 2, 3, 5, 8, 11)`
  - [ ] `(new int[] { 1, 1, 2, 3, 5, 8, 11}).toStream()`

  One correct, 0 errors, 100%

- **2.** How do you get a stream of all lines in a file named `input.txt`?

  - [ ] `Files.lines("input.txt")`
  - [x] `Files.lines(Paths.get("input.txt"))`
  - [ ] `new Scanner(new File("input.txt")).stream()`
  - [ ] `Stream.of(new File("input.txt"))`

  One correct, 0 errors, 100%

- **3.** Given a stream

  ```
  Stream<String> words = . . .;
  ```

  contrast the computations

  ```
  long count = words
     .filter(w -> w.length() > 10)
     .count();
  ```

  and

  ```
  long count = words
     .parallel()
     .filter(w -> w.length() > 10)
     .count();
  ```

  Which of the following statements is *not* true?

  - [x] The second version will not compile unless the computer has multiple processors.
  - [ ] Both versions compute the same result.
  - [ ] When `words` contains many elements and the computer has multiple processors, the second version is likely to be faster.
  - [ ] When `words` has few elements, it is possible that the first version is faster than the second, even on a computer with multiple processors.

  One correct, 0 errors, 100%

••• **4.** Given an array of `String` objects, use streams to count how many have a length less than or equal to three.

StringLengthDemo.java

```
1  import java.util.stream.Stream;
2
3  public class StringLengthDemo
4  {
5     public static void main(String[] args)
6     {
7        String[] names = {"Fred", "Sam", "Ida", "Alice", "Abe"};
8
9        Stream<String> lines = Stream.of(names);
10       long count = lines
11             .filter(w -> w.length() < 4)
12             .count();
13
14       System.out.println("Words: " + count);
15    }
16 }
```

CodeCheck Reset

Testing StringLengthDemo.java

```
Words: 3
pass
```

Score

1/1

••• **5.** Write a statement to count how many lines in the file `input.txt` have length greater than 80.

FileLinesDemo.java

```
1  import java.io.IOException;
2  import java.nio.file.Files;
3  import java.nio.file.Paths;
4  import java.util.stream.Stream;
5
6  public class FileLinesDemo
7  {
8     public static void main(String[] args) throws IOException
9     {
10       long count = 0;
11       Stream<String> lines = Files.lines(Paths.get("input.txt"));
12       count = lines
13             .filter(w -> w.length() > 80)
14             .count();
15
16       System.out.println("Lines greater than 80: " + count);
17    }
18 }
```

CodeCheck Reset

Testing FileLinesDemo.java

```
Lines greater than 80: 2
pass
```

Score

1/1

SELF CHECK

• **1.** Which statement creates a list from a stream of `BankAccount` objects stored in the variable `accounts`?

- [ ] `List<BankAccount> result = accounts.toList();`
- [x] `List<BankAccount> result = accounts.collect(Collectors.toList());`
- [ ] `List<BankAccount> result = accounts.collect(ArrayList<BankAccount>::new);`
- [ ] `List<BankAccount> result = accounts.toList(BankAccount[]::new);`

One correct, 0 errors, 100%

• **2.** Which statement creates a set of `Integer` objects from a stream of `Integer` objects called `numbers`?

- [ ]
```
Set<Integer> result = numbers
    .toSet());
```
- [ ]
```
Set<Integer> result = numbers
    .collect().toSet();
```
- [ ]
```
Set<Integer> result = numbers
    .toSet(new HashSet<Integer>());
```
- [x]
```
Set<Integer> result = numbers
    .collect(Collectors.toSet());
```

One correct, 0 errors, 100%

••• **3.** Collect all strings of length greater than ten from a list of strings and store them in another list.

StreamsDemo.java

```
1  import java.util.List;
2  import java.util.stream.Stream;
3  import java.util.stream.Collectors;
4
5  public class StreamsDemo
6  {
7     public static void main(String[] args)
8     {
9        List<String> list = Util.getList();
10       List<String> result;
11
12       result = list.stream()
13             .filter(w -> w.length() > 10)
14             .collect(Collectors.toList());
15
16       System.out.println(result);
17    }
18 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
[autoincrements, manslaughter, centralization, retransmitting, unifications, retransmitting, unifications]
pass
```

Score

1/1

••• 4. Collect all strings of length greater than ten from a list of strings and store them in a set of strings.

StreamsDemo.java

```
1  import java.util.List;
2  import java.util.Set;
3  import java.util.stream.Stream;
4  import java.util.stream.Collectors;
5
6  public class StreamsDemo
7  {
8     public static void main(String[] args)
9     {
10       List<String> list = Util.getList();
11       Set<String> result;
12
13       result = list.stream()
14             .filter(w -> w.length() > 10)
15             .collect(Collectors.toSet());
16
17       System.out.println(result);
18    }
19 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
[centralization, unifications, retransmitting, autoincrements, manslaughter]
pass
```

Score

1/1

••• 5. Find the first string of length greater than ten in a list of strings. Use `filter` and `limit`, then convert the stream to *a list* and retrieve the result. Assume that there is at least one such string.

StreamsDemo.java

```
1  import java.util.List;
2  import java.util.Set;
3  import java.util.stream.Stream;
4  import java.util.stream.Collectors;
5
6  public class StreamsDemo
7  {
8     public static void main(String[] args)
9     {
10       List<String> list = Util.getList();
11
12       String result = list.stream()
13          .filter(w -> w.length() > 10)
14          .limit(1)
15          .collect(Collectors.toList())
16          .get(0);
17
18       System.out.println(result);
19    }
20 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
autoincrements
pass
```

Score

1/1

••• 1. Given a stream of words, get a stream of all that start with *a* or *A*, converted to lowercase. Apply `map` before `filter`.

StreamsDemo.java

```
1  import java.util.stream.Stream;
2
3  public class StreamsDemo
4  {
5     public static void main(String[] args)
6     {
7        Stream<String> words = Util.getWords();
8
9        Stream<String> result = words
10          .map(w -> w.toLowerCase())
11
12          .filter(w -> w.substring(0, 1).equals("a"));
13
14       Util.print(result);
15    }
16 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
autoincrements
america
advisers
adders
argentina
abc
pass
```

Score

1/1

••• 2. Given a stream of words, get a stream of all that start with *a* or *A*, converted to lowercase. Apply `filter` before `map`.

StreamsDemo.java

```
1  import java.util.stream.Stream;
2
3  public class StreamsDemo
4  {
5     public static void main(String[] args)
6     {
7        Stream<String> words = Util.getWords();
8
9        Stream<String> result = words
10          .filter(w -> w.substring(0,1).toLowerCase().equals("a"))
11
12          .map(w -> w.toLowerCase());
13
14       Util.print(result);
15    }
16 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
autoincrements
america
advisers
adders
argentina
abc
pass
```

Score

1/1

••• **3.** Given a stream of words, produce a stream of `Integer` values containing the lengths of the words.

StreamsDemo.java

```
1  import java.util.stream.Stream;
2
3  public class StreamsDemo
4  {
5     public static void main(String[] args)
6     {
7        Stream<String> words = Util.getWords();
8
9        Stream<Integer> result = words
10          .map(w -> w.length());
11
12       Util.print(result);
13    }
14 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
4
8
4
14
12
8
6
8
10
8
6
```

pass

Score

1/1

••• **4.** Given a list of strings, get a list of the first ten in sorted order.

StreamsDemo.java

```
1  import java.util.List;
2  import java.util.stream.Collectors;
3
4  public class StreamsDemo
5  {
6     public static void main(String[] args)
7     {
8        List<String> words = Util.getList();
9
10       List<String> result = words.stream()
11             .sorted()
12             .limit(10)
13             .collect(Collectors.toList());
14
15       System.out.println(result);
16    }
17 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
[Aeration, Assignable, adders, advisers, autoincrements, ballgown, capacities, centralization, cherries, corpus]
```

pass

Score

1/1

••• 5. Given a list of words, find how many distinct words there are of length equal to four.

StreamsDemo.java

```
1  import java.util.List;
2
3  public class StreamsDemo
4  {
5     public static void main(String[] args)
6     {
7        List<String> words = Util.getList();
8
9        long result = words.stream()
10             .distinct()
11             .filter(w -> w.length() == 4)
12             .count();
13
14
15       System.out.println(result);
16    }
17 }
```

CodeCheck Reset

Testing StreamsDemo.java

```
3
pass
```

Score

1/1

••• 6. Given a list of words, find how many distinct words there are of length equal to four. **Do not use streams.**

DistinctWords.java

```
1  import java.util.List;
2  import java.util.Set;
3  import java.util.HashSet;
4
5  public class DistinctWords
6  {
7     public static void main(String[] args)
8     {
9        List<String> list = Util.getList();
10
11       HashSet<String> storage = new HashSet<>();
12
13       for (int i = 0; i < list.size(); i++) {
14          if (list.get(i).length() == 4 && !storage.contains(list.get(i))){
15             storage.add(list.get(i));
16          }
17       }
18       int result = storage.size();
19
20       System.out.println(result);
21    }
22 }
```

CodeCheck Reset

Testing DistinctWords.java

```
3
```

SELF CHECK

••• 1. Write a lambda expression that tests whether a word starts and ends with the same letter.

LambdaDemo.java

```
1  import java.util.List;
2  import java.util.stream.Stream;
3  import java.util.stream.Collectors;
4
5  public class LambdaDemo
6  {
7     public static void main(String[] args)
8     {
9        Stream<String> words = Util.getWords();
10
11       List<String> result = words
12          .filter(w -> w.charAt(0) == (w.charAt(w.length()-1)))
13          .limit(5)
14          .collect(Collectors.toList());
15
16       System.out.println(result);
17    }
18 }
```

CodeCheck Reset

Testing LambdaDemo.java

```
[area, bulb, madam, specious, tent]
```

pass

Score

1/1

- **2.** What does this lambda expression do?

```
s -> s.equals(s.toUpperCase())
```

- [ ] It turns a string into uppercase.
- [x] It tests whether a string is all uppercase.

  It is a *predicate* that tests whether a string is in uppercase.

- [ ] It tests whether two strings are the same when turned into uppercase.
- [ ] It always returns false.

One correct, 0 errors, 100%

- **3.** Examine this code:

```
List<String> words = Stream.of("a", "A", "Ab", "AB", "ABc", "ABC")
   .filter(s -> s.equals(s.toUpperCase()))
   .collect(Collectors.toList());
```

What is in the list `words`?

- [ ] `[A, Ab, AB, ABc, ABC]`
- [ ] `[a, Ab, ABc]`
- [x] `[A, AB, ABC]`

  The lambda used in the filter determines which strings are passed through to the rest of the stream. Only those strings that are entirely uppercase are retained and collected.

- [ ] `[a, A, Ab, AB, ABc, ABC]`

One correct, 0 errors, 100%

- **4.** Given the stream

```
Stream<String> words = Stream.of("a", "A", "Ab", "AB", "ABc", "ABC");
```

what is contained in the stream

```
words.map(s -> s.equals(s.toUpperCase()));
```

- [ ] `"A", "A", "AB", "AB", "ABC", "ABC"`
- [ ] `"A", "AB", "ABC"`
- [ ] `false true false true false true`
- [x] `Boolean.FALSE, Boolean.TRUE, Boolean.FALSE, Boolean.FALSE, Boolean.TRUE`

  It is a `Stream<Boolean>` with values `Boolean.TRUE` and `Boolean.FALSE` (the wrappers for `true` and `false`), depending on whether the elements of `words` were entirely in uppercase or not.

One correct, 0 errors, 100%