1

```
Stream<String> 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 is the output of the following?

```
Predicate<? super String> predicate = s -> s.startsWith("g");
Stream<String> stream1 = Stream.generate(() -> "growl! ");
Stream<String> stream2 = Stream.generate(() -> "growl! ");
boolean b1 = stream1.anyMatch(predicate);
boolean 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 is the output of the following?

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

A. `false true`
B. `false false`
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? (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 referenced `Stream` may be used after the calling a terminal operation.

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

5

Which terminal operations on the `Stream` class are reductions? (Choose all that apply.)

A. `collect()`

B. `count()`

C. `findFirst()`

D. `map()`

E. `peek()`

F. `sum()`

6

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

```
Stream<String> s = Stream.generate(() -> "meow");
boolean 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) {
   List<String> copy = new ArrayList<>(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 the declaration `IntStream is = IntStream.empty()`? (Choose all that apply.)

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 5 for the code to run without error and not produce any output? (Choose all that apply.)

```
4:    LongStream ls = LongStream.of(1, 2, 3);
5:    OptionalLong opt = ls.map(n -> n * 10).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 5 throws an exception at runtime.

10

Select from the following statements and indicate the order in which they would appear to output 10 lines:

```
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 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 functional interfaces complete the following code? (Choose all that apply.)

```
6:    ________ x = String::new;
7:    ________ y = (a, b) -> System.out.println();
8:    ________ z = a -> a + a;
```

A. `BiConsumer<String, String>`

B. `BiFunction<String, String>`

C. `BinaryConsumer<String, String>`

D. `BinaryFunction<String, String>`

E. `Consumer<String>`

F. `Supplier<String>`

G. `UnaryOperator<String>`

H. `UnaryOperator<String, String>`

13

Which of the following is true?

```
List<Integer> l1 = Arrays.asList(1, 2, 3);
List<Integer> l2 = Arrays.asList(4, 5, 6);
List<Integer> l3 = Arrays.asList();
Stream.of(l1, l2, l3).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 is true?

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

A. Line 4 does not compile.

B. Line 5 does not compile.

C. Line 6 does not compile.

D. Line 7 does not compile.

E. Line 8 does not compile.

F. The code throws an exception.

G. The code compiles and prints 1.

15

The `partitioningBy()` collector creates a `Map<Boolean, List<String>>` when passed to `collect()` by default. When specific parameters are passed to `partitioningBy()`, which return types can be created? (Choose all that apply.)

A. `Map<boolean, List<String>>`

B. `Map<Boolean, Map<String>>`

C. `Map<Long, TreeSet<String>>`

D. `Map<Boolean, List<String>>`

E. `Map<Boolean, Set<String>>`

F. None of the above

16

What is the output of the following?

```
Stream<String> s = Stream.empty();
Stream<String> s2 = Stream.empty();
Map<Boolean, List<String>> p = s.collect(
   Collectors.partitioningBy(b -> b.startsWith("c")));
Map<Boolean, List<String>> g = s2.collect(
   Collectors.groupingBy(b -> b.startsWith("c")));
System.out.println(p + " " + g);
```

A. `{} {}`
B. `{} {false=[], true=[]}`
C. `{false=[], true=[]} {}`
D. `{false=[], true=[]} {false=[], true=[]}`
E. The code does not compile.
F. An exception is thrown.

17

Which of the following is equivalent to this code?

```
UnaryOperator<Integer> u = x -> x * x;
```

A. `BiFunction<Integer> f = x -> x*x;`
B. `BiFunction<Integer, Integer> f = x -> x*x;`
C. `BinaryOperator<Integer, Integer> f = x -> x*x;`
D. `Function<Integer> f = x -> x*x;`
E. `Function<Integer, Integer> f = x -> x*x;`
F. None of the above

18

What is the result of the following?

```
DoubleStream 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.

19

Which of the following return primitives? (Choose all that apply.)

A. `BooleanSupplier`

B. `CharSupplier`

C. `DoubleSupplier`

D. `FloatSupplier`

E. `IntSupplier`

F. `StringSupplier`

20

What is the simplest way of rewriting this code?

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

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

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

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

D. None of the above is equivalent.

E. The provided code does not compile.