Chapter 15
Java Stream API

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

- ✓**Java Stream API**
- Develop code to extract data from an object using peek() and map() methods including primitive versions of the map() method
- Search for data by using search methods of the Stream classes including findFirst, findAny, anyMatch, allMatch, noneMatch
- Develop code that uses the Optional class
- Develop code that uses Stream data methods and calculation methods
- Sort a collection using Stream API
- Save results to a collection using the collect method and group/partition data using the Collectors class
- Use flatMap() methods in the Stream API

1. Which of the following fills in the blank so that the code outputs one line but uses a poor practice?

```
import java.util.*;

public class Cheater {
   int count = 0;
   public void sneak(Collection<String> coll) {
      coll.stream()._____________________;
   }

   public static void main(String[] args) {
      Cheater c = new Cheater();
      c.sneak(Arrays.asList("weasel"));
  }
}
```

1. `peek(System.out::println)`
2. `peek(System.out::println).findFirst()`
3. `peek(r -> System.out.println(r)).findFirst()`
4. `peek(r -> {count++; System.out.println(r);
   }).findFirst()`

2. Which can fill in the blank to have the code print `true`?

```
Stream<Integer> stream = Stream.iterate(1, i -> i+1);
boolean b = stream._____________(i -> i > 5);
System.out.println(b);
```

1. `anyMatch`
2. `allMatch`
3. `noneMatch`

3. On a `DoubleStream`, how many of the methods `average()`, `count()`, and `sum()` return an `OptionalDouble`?

1. None
2. One
3. Two
4. Three

4. How many of the following can fill in the blank to have the code print `44`?

```
Stream<String> stream = Stream.of("base", "ball");
stream._____________(s -> s.length()).forEach(System.out::print);
```

1. `map`
2. `mapToInt`
3. `mapToObject`

1. None
2. One
3. Two
4. Three

5. What is the result of the following?

```
IntStream s = IntStream.empty();
System.out.print(s.average().getAsDouble());
```

1. The code prints `0`.
2. The code prints `0.0`.
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

6. Which of these stream pipeline operations takes a `Predicate` as a parameter and returns an `Optional`?

1. `anyMatch()`
2. `filter()`
3. `findAny()`
4. None of the above

7. What is the result of the following?

```
List<Double> list = new ArrayList<>();
list.add(5.4);
list.add(1.2);
Optional<Double> opt = list.stream().sorted().findFirst();
System.out.println(opt.get() + " " + list.get(0));
```

1. `1.2 1.2`
2. `1.2 5.4`
3. `5.4 5.4`
4. None of the above

8. Fill in the blank so this code prints `8.0`.

```
IntStream stream = IntStream.of(6, 10);
LongStream longs = stream.mapToLong(i -> i);
System.out.println(_____________);
```

1. `longs.average().get()`
2. `longs.average().getAsDouble()`
3. `longs.getAverage().get()`
4. `longs.getAverage().getAsDouble()`

9. How many of these collectors can fill in the blank to make this code compile?

```
Stream<Character> chars = Stream.of(
   'o', 'b', 's', 't', 'a', 'c', 'l', 'e');
chars.map(c -> c).collect(Collectors._____ );
```

1. `toArrayList()`
2. `toList()`
3. `toMap()`

1. None
2. One
3. Two
4. Three

10. What does the following output?

```
import java.util.*;

public class MapOfMaps {
   public static void main(String[] args) {
      Map<Integer, Integer> map = new HashMap<>();
      map.put(9, 3);
      Map<Integer, Integer> result = map.stream().map((k,v) -> (v,k
      System.out.println(result.keySet().iterator().next());
   }
}
```

1. 3
2. 9
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

11. Which of the following creates an `Optional` that returns `true` when calling `opt.isPresent()`?

1. `Optional<String> opt = Optional.empty();`
2. `Optional<String> opt = Optional.of(null);`
3. `Optional<String> opt = Optional.ofNullable(null);`

1. I
2. I and II
3. I and III
4. None of the above

12. What is the output of the following?

```
Stream<String> s = Stream.of("speak", "bark", "meow", "growl");
BinaryOperator<String> merge = (a, b) -> a;
Map<Integer, String> map = s.collect(toMap(String::length, k -> k, m
System.out.println(map.size() + " " + map.get(4));
```

1. `2 bark`
2. `2 meow`
3. `4 bark`
4. None of the above

13. What is the output of the following?

```
1:   package reader;
2:   import java.util.stream.*;
3:
4:   public class Books {
5:      public static void main(String[] args) {
6:         IntegerStream pages = IntegerStream.of(200, 300);
7:         IntegerSummaryStatistics stats = pages.summaryStatistics
8:         long total = stats.getSum();
9:         long count = stats.getCount();
10:        System.out.println(total + "-" + count);
```

```
11:     }
12:  }
```

1. `500-0`
2. `500-2`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

14. If this method is called with `Stream.of("hi")`, how many lines are printed?

```
public static void print(Stream<String> stream) {
   Consumer<String> print = System.out::println;
   stream.peek(print)
         .peek(print)
         .map(s -> s)
         .peek(print)
         .forEach(print);
}
```

1. Three
2. Four
3. The code compiles but does not output anything.
4. The code does not compile.

15. What is true of the following code?

```
Stream<Character> stream = Stream.of('c', 'b', 'a');       // z1
stream.sorted().findAny().ifPresent(System.out::println);  // z2
```

1. It is guaranteed to print the single character `a`.
2. It can print any single character of `a`, `b`, or `c`.
3. It does not compile because of line `z1`.
4. It does not compile because of line `z2`.

16. Suppose you have a stream pipeline where all the elements are of type `String`. Which of the following can be passed to the intermediate operation `sorted()`?

1. `(s,t) -> s.length() - t.length()`
2. `String::isEmpty`
3. Both of these
4. Neither of these

17. Fill in the blanks so that both methods produce the same output for all inputs.

```
private static void longer(Optional<Boolean> opt) {
   if (opt.___________())
      System.out.println("run: " + opt.get());
}
private static void shorter(Optional<Boolean> opt) {
   opt.map(x -> "run: " + x).___________(System.out::println);
}
```

1. `isNotNull`, `isPresent`
2. `ifPresent`, `isPresent`
3. `isPresent`, `forEach`
4. `isPresent`, `ifPresent`

18. What is the output of this code?

```
Stream<Boolean> bools = Stream.iterate(true, b -> !b);
Map<Boolean, List<Boolean>> map = bools.limit(1)
    .collect(partitioningBy(b -> b));
System.out.println(map);
```

1. `{true=[true]}`

2. `{false=null, true=[true]}`
3. `{false=[], true=[true]}`
4. None of the above

19. What does the following output?

```
Set<String> set = new HashSet<>();
set.add("tire-");
List<String> list = new LinkedList<>();
Deque<String> queue = new ArrayDeque<>();
queue.push("wheel-");
Stream.of(set, list, queue)
   .flatMap(x -> x.stream())
   .forEach(System.out::print);
```

1. `[tire-][wheel-]`
2. `tire-wheel-`
3. None of the above.
4. The code does not compile.

20. What is the output of the following?

```
Stream<String> s = Stream.of("over the river",
    "through the woods",
    "to grandmother's house we go");
s.filter(n -> n.startsWith("t"))
 .sorted(Comparator::reverseOrder)
 .findFirst()
 .ifPresent(System.out::println);
```

1. `over the river`
2. `through the woods`
3. `to grandmother's house we go`
4. None of the above

21. Which fills in the blank so the code is guaranteed to print 1?

```
Stream<Integer> stream = Stream.of(1, 2, 3);
System.out.println(stream.___________);
```

1. `findAny()`
2. `first()`
3. `min()`
4. None of the above

22. Which of the following can be the type for `x`?

```
private static void spot(___________ x) {
   x.filter(y -> ! y.isEmpty())
     .map(y -> 8)
     .ifPresent(System.out::println);
}
```

1. `List<String>`
2. `Optional<Collection>`
3. `Optional<String>`
4. `Stream<Collection>`

1. I
2. IV
3. II and III
4. II and IV

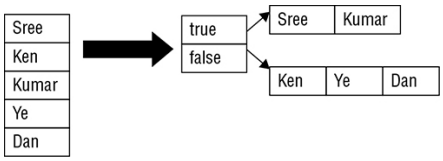
23. Which can fill in the blank to have the code print `true`?

```
Stream<Integer> stream = Stream.iterate(1, i -> i);
boolean b = stream.___________(i -> i > 5);
System.out.println(b);
```

1. `anyMatch`
2. `allMatch`

3. `noneMatch`
4. None of the above

24. What collector turns the stream at left to the `Map` at right?



1. `grouping()`
2. `groupingBy()`
3. `partitioning()`
4. `partitioningBy()`

25. Which fills in the blank for this code to print `667788`?

```
IntStream ints = IntStream.empty();
IntStream moreInts = IntStream.of(66, 77, 88);
Stream.of(ints, moreInts).____________(x -> x).forEach(System.out::
```

1. `flatMap`
2. `flatMapToInt`
3. `map`
4. None of the above

26. Fill in the blank so this code prints `8.0`. Note that it must not print `OptionalDouble[8.0]`.

```
LongStream stream = LongStream.of(6, 10);
LongSummaryStatistics stats = stream.summaryStatistics();
System.out.println(____________);
```

1. `stats.avg()`
2. `stats.average()`
3. `stats.average().get()`
4. `stats.getAverage()`

27. Which can independently fill in the blank to output `No dessert today`?

```
import java.util.*;
public class Dessert {
  public static void main(String[] yum) {
    eatDessert(Optional.of("Cupcake"));
  }
  private static void eatDessert(Optional<String> opt) {
     System.out.println(opt.____________);
  }
}
```

1. `get("No dessert today")`
2. `orElse("No dessert today")`
3. `orElseGet(() -> "No dessert today")`
4. None of the above

28. What does the following output?

```
Stream<Character> chars = Stream.generate(() -> 'a');
chars.filter(c -> c < 'b')
     .sorted()
     .findFirst()
     .ifPresent(System.out::print);
```

1. `a`
2. The code runs successfully without any output.
3. The code enters an infinite loop.

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

29. How many of the following can fill in the blank to have the code print the single digit 9?

```
LongStream stream = LongStream.of(9);
stream._____________(p -> p).forEach(System.out::print);
```

1. `mapToDouble`
2. `mapToInt`
3. `mapToLong`

1. None
2. One
3. Two
4. Three

30. Suppose you have a stream with one element and the code `stream.xxxx.forEach(System.out::println)`. Filling in `xxxx` from top to bottom in the table, how many elements can be printed out?

xxxx	Number elements printed
`filter()`	
`flatMap()`	
`map()`	

1. Zero or one, zero or more, exactly one
2. Zero or one, exactly one, zero or more
3. Zero or one, zero or more, zero or more
4. Exactly one, zero or more, zero or more

31. What is the output of the following?

```
Stream<Character> stream = Stream.of('c', 'b', 'a');
System.out.println(stream.sorted().findFirst());
```

1. It is guaranteed to print the single character `a`.
2. It can print any single character of `a`, `b`, or `c`.
3. The code does not compile.
4. None of the above

32. What is the output of the following?

```
public class Compete {
   public static void main(String[] args) {
      Stream<Integer> is = Stream.of(8, 6, 9);
      Comparator<Integer> c = (a, b) -> a - b;
      is.sort(c).forEach(System.out::print);
   }
}
```

1. `689`
2. `986`
3. The code does not compile
4. The code compiles but throws an exception at runtime.

33. What is the result of the following?

```
class Ballot {
   private String name;
   private int judgeNumber;
   private int score;

   public Ballot(String name, int judgeNumber, int score) {
      this.name = name;
      this.judgeNumber = judgeNumber;
      this.score = score;
```

```
    }
    // all getters and setters
}

public class Speaking {
   public static void main(String[] args) {
      Stream<Ballot> ballots = Stream.of(
         new Ballot("Mario", 1, 10),
         new Ballot("Christina", 1, 8),
         new Ballot("Mario", 2, 9),
         new Ballot("Christina", 2, 8)
      );

      Map<String, Integer> scores = ballots.collect(
         groupingBy(Ballot::getName, summingInt(Ballot::getScore)))
      System.out.println(scores.get("Mario"));
  }
}
```

1. The code prints 2.
2. The code prints 19.
3. The code does not compile due to line `w1`.
4. The code does not compile due to a different line.

34. Which can fill in the blank so this code outputs `true`?

```
import java.util.function.*;
import java.util.stream.*;

public class HideAndSeek {
   public static void main(String[] args) {
      Stream<Boolean> hide = Stream.of(true, false, true);
      boolean found = hide.filter(b -> b).__________();
      System.out.println(found);
   }
}
```

1. Only `anyMatch`
2. Only `allMatch`
3. Both `anyMatch` and `allMatch`
4. The code does not compile with any of these options.

35. What does the following output?

```
Set<String> set = new HashSet<>();
set.add("tire-");
List<String> list = new LinkedList<>();
Deque<String> queue = new ArrayDeque<>();
queue.push("wheel-");
Stream.of(set, list, queue)
   .flatMap(x -> x)
   .forEach(System.out::print);
```

1. `[tire-][wheel-]`
2. `tire-wheel-`
3. None of the above
4. The code does not compile.

36. When working with a `Stream<String>`, which of these types can be returned from the `collect()` terminal operator by passing arguments to `Collectors.groupingBy()`?

1. `Map<Integer, List<String>>`
2. `Map<Boolean, HashSet<String>>`
3. `List<String>`

1. I
2. II
3. I and II
4. I, II, and III

37. Which line can replace line 18 without changing the output of the program?

```
1:   class Runner {
2:      private int numberMinutes;
```

```
3:      public Runner(int n) {
4:         numberMinutes = n;
5:      }
6:      public int getNumberMinutes() {
7:         return numberMinutes;
8:      }
9:      public boolean isFourMinuteMile() {
10:        return numberMinutes < 4*60;
11:     }
12:   }
13:   public class Marathon {
14:     public static void main(String[] args) {
15:        Stream<Runner> runners = Stream.of(new Runner(250),
16:           new Runner(600), new Runner(201));
17:        long count = runners
18:             .filter(Runner::isFourMinuteMile)
19:             .count();
20:        System.out.println(count);
21:     }
22:  }
```

1. `.map(Runner::isFourMinuteMile)`
2. `.mapToBool(Runner::isFourMinuteMile)`

   `.filter(b -> b == true)`
3. `.mapToBoolean(Runner::isFourMinuteMile)`
   `.filter(b -> b == true)`
4. None of the above

38. Which method is not available on the `IntSummaryStatistics` class?

1. `getCountAsLong()`
2. `getMax()`
3. `toString()`
4. None of the above—all three methods are available.

39. Which can fill in the blank so this code outputs `Caught it`?

```
import java.util.*;
public class Catch {
   public static void main(String[] args) {
      Optional opt = Optional.empty();
      try {
         apply(opt);
      } catch (IllegalArgumentException e) {
         System.out.println("Caught it");
      }
   }
   private static void apply(Optional<Exception> opt) {
      opt._________(IllegalArgumentException::new);
   }
}
```

1. `orElse`
2. `orElseGet`
3. `orElseThrow`
4. None of the above. The `main()` method does not compile.

40. A developer tries to rewrite a method that uses `flatMap()` without using that intermediate operator. Which pair of method calls shows the `withoutFlatMap()` method is not equivalent to the `withFlatMap()` method?

```
public static void main(String[] args) {
   List<String> list = new LinkedList<>();
   Deque<String> queue = new ArrayDeque<>();
   queue.push("all queued up");
   queue.push("last");
}

private static void withFlatMap(Collection<?> coll) {
   Stream.of(coll)
      .flatMap(x -> x.stream())
      .forEach(System.out::print);
   System.out.println();
}

private static void withoutFlatMap(Collection<?> coll) {
   Stream.of(coll)
      .filter(x -> !x.isEmpty())
      .map(x -> x)
      .forEach(System.out::print);
   System.out.println();
}
```

1. `withFlatMap(list); withoutFlatMap(list);`
2. `withFlatMap(queue); withoutFlatMap(queue);`
3. Both pairs disprove the claim.
4. Neither pair disproves this claim.

PREV
Chapter 14 Lambda Built-in Functional Interfaces

NEXT
Chapter 16 Exceptions and Assertions