Chapter 9: Working with Selected Classes from the Java API

1. C. Option A is incorrect because `StringBuilder` does not support multiple threads. In fact, threads aren't even covered on the OCA, which should be your clue that this answer is wrong! You don't need to know this for the exam, but `StringBuffer` supports multiple threads. Option B is incorrect because `==` compares references, not values. Option D is incorrect because both `String` and `StringBuilder` support languages and encodings. Option C is correct and the primary reason to use `StringBuilder`. `String` often creates a new object each time you call certain methods on the object like `concat()`. `StringBuilder` optimizes operations like `append()` because it is mutable.
2. D. A `String` can be created using a literal rather than calling a constructor directly, making Option A incorrect. A string pool exists for `String` reuse, making Option B incorrect. A `String` is final and immutable, making Option C incorrect and Option D correct.
3. D. This question is testing whether you understand how method chaining works. Option A creates an empty `StringBuilder` and then adds the five characters in `clown` to it. Option B simply creates the `clown` when calling the constructor. Finally, Option C creates the same value, just in two parts. Therefore, Option D is correct.
4. B. Since `StringBuilder` is mutable, each call to `append` adds to the value. When calling `print`, `toString()` is automatically called and `333 806 1601` is output. Therefore, Option B is correct.
5. B. `List` is an interface and not a class. It cannot be instantiated. While `Object` is a concrete class, it does not implement the `List` interface so it cannot be assigned to `frisbees`. Note that if you were to add an explicit cast, it would compile and throw an exception at runtime. Of the three options, only `ArrayList` can fill in the blank, so Option B is correct.
6. C. An `ArrayList` does not automatically sort the elements. It simply remembers them in order. Since Java uses zero-based indexes, Option C is correct.
7. C. Calling the constructor and then `insert()` is an example of method chaining. However, the `sb.length()` call is a problem. The `sb` reference doesn't exist until after the chained calls complete. Just because it happens to be on a separate line doesn't change when the reference is created. Since the code does not compile, Option C is correct.
8. A. While the `ArrayList` is declared with an initial capacity of one element, it is free to expand as more elements are added. Each of the three calls to the `add()` method adds an element to the end of the `ArrayList`. The `remove()` method call deletes the element at index 2, which is `Art`. Therefore, Option A is correct.
9. C. On line 12, the value of the `StringBuilder` is `12`. On line 13, it becomes `123`. Since

`StringBuilder` is mutable, storing the result in the same reference is redundant. Then on line 14, the value is reversed, giving us `321` and making Option C correct.

.0. D. Option A is incorrect as it describes autoboxing. Options B and C are not possible in Java. Option D is correct as it describes lambdas. Lambdas use deferred execution and can be run elsewhere in the codebase.

11. D. A `StringBuilder` is mutable, so the length is two after line 6 completes. The `StringBuilder` methods return a reference to the same object so you can chain method calls. Therefore, `line` and `anotherLine` refer to the same object. This means that line 7 prints `true`. Then on line 9, both references point to the same object of length 2, and Option D is correct.

.2. D. The `add()` and `get()` methods are available on `ArrayList`. However, `ArrayList` uses `size` rather than `length` to get the number of elements. Therefore, Option D is correct. If `length` was changed to `size`, Option B would compile if put in the blank. Option A still wouldn't compile in the blank because a cast would be needed to store the value in `str`.

.3. D. Option A is tricky, but incorrect. While a lambda can have zero parameters, a `Predicate` cannot. A `Predicate` is defined as a type mapping to a `boolean`. Option B is clearly incorrect as `->` separates the parts of a lambda. Options C and D are similar. Option C is incorrect because `return` is only allowed when the brackets are present. Option D is correct.

.4. A. Lines 20–22 create an `ArrayList` with two elements. Line 23 replaces the second one with a new value. Now `chars` is `[a, c]`. Then line 24 removes the first element, making it just `[c]`. Option A is correct because there is only one element, but it is not the value `b`.

.5. D. Trick question. There is no reverse method on the `String` class. There is one on the `StringBuilder` class. Therefore, the code does not compile, and Option D is correct.

.6. A. When creating a lambda with only one parameter, there are a few variants. The `pred1` approach shows the shortest way, where the type is omitted and the parentheses are omitted. The `pred2` approach is similar except it includes the parentheses. Both are legal. The `pred4` approach is the long way with both the parentheses and type specified. The only one that doesn't compile is `pred3`. The parentheses are required if including the type.

17. A. This is a correct example of code that uses a lambda. The interface has a single abstract method. The lambda correctly takes one `double` parameter and returns a `boolean`. This matches the interface. The lambda syntax is correct. Since 45 is greater than 5, Option A is correct.

.8. A. Since `String` is immutable, each call to `concat()` returns a new object with the new value. However, that return value is ignored and the `teams` variable never changes in value. Therefore it stays as `694`, and Option A is correct.

9. A. The `ArrayList` class is in the `java.util` package, making I correct. The `LocalDate` class is in the `java.time` package, making II incorrect. The `String` class is in the `java.lang` package, which means you can use it without typing an import, making III incorrect. Therefore, Option A is correct.

20. C. Option A is straightforward and outputs `radical robots`. Option B does the same in a convoluted manner. First Option B removes all the characters after the first one. It doesn't matter that there aren't actually 100 characters to delete. Then it appends `obots` to the end, making the builder contain `robots`. Finally, it inserts the remainder of the string immediately after the first index. Try drawing the flow if this is hard to envision. Option D also creates the same value by inserting `robots` immediately after the end of the `StringBuilder`. Option C is close, but it has an off-by-one error. It inserts `robots` after the letter `l` rather than after the space. This results in the value `radicalrobots` followed by a space. Option C is different than the others and the correct answer.

21. A. Since we are creating the list from an array, it is a fixed size. We are allowed to change elements. At the end of this code, `museums` is `[Art, Science]`. Therefore, it contains `Art`, and Option A is correct.

22. D. Options A and B are not true if the `String` is `"deabc"`. Option C is not true if the `String` is `"abcde"`. Option D is true in all cases.

23. D. Line 25 does not compile. On an `ArrayList`, the method to get the number of elements is `size`. The `length()` method is used for a `String` or `StringBuilder`.

24. B. The `toString()` method call doesn't help in narrowing things down as all Java objects have that method available. The other two methods are more helpful. `String` is the only type of these three to have a `startsWith()` method, making Option B correct. `String` also has the `replace()` method declared here. If you memorized the whole API, you might know that `StringBuilder` also has a `replace()` method, but it requires three parameters instead of two. Please don't memorize the API in that level of detail. We included what you need to know in our study guide. If you do have this outside knowledge, be careful not to read into the questions!

25. B. The `<>` is known as the diamond operator. Here, it works as a shortcut to avoid repeating the generic type twice for the same declaration. On the right side of the expression, this is a handy shortcut. Java still needs the type on the left side so there is something to infer. In the figure, position P is the left side and position Q is the right side. Therefore, Option B is correct.

26. D. The type in the lambda must match the generic declared on the `Predicate`. In this case, that is `String`. Therefore, Options A and B are incorrect. While Option C is of the correct type, it uses the variable `s`, which is already in use from the `main()` method parameter. Therefore, none of these are correct, and Option D is the answer.

27. A. A `String` is immutable so a different object is returned on line 6. The object `anotherLine` points to is of length 2 after line 6 completes. However, the original `line`

reference still points to an object of length 1. Therefore, Option A is correct.

28. C. While it is common for a `Predicate` to have a generic type, it is not required. However, it is treated like a `Predicate` of type `Object` if the generic type is missing. Since `startsWith()` does not exist on `Object`, the first line does not compile, and Option C is correct.

29. B. `LocalDate` only includes the date portion and not the time portion. There is no class named `LocalTimeStamp`. The other two, `LocalDateTime` and `LocalTime`, both include the time elements, making Option B correct.

30. D. Line 4 creates a `String` of length 5. Since `String` is immutable, line 5 creates a new `String` with the value `1` and assigns it to `builder`. Remember that indexes in Java begin with 0, so the `substring()` method is taking the values from the fifth element through the end. Since the first element is the last element, there's only one character in there. Then line 6 tries to retrieve the second indexed element. Since there is only one element, this gives a `StringIndexOutOfBoundsException`, and Option D is correct.

31. D. When you're using brackets, both the `return` keyword and semicolon are needed for the lambda to compile, making Option D correct.

32. B. Java 8 date and time classes are immutable. The `plusDays` method returns a `LocalDate` object presenting Christmas Eve (December 24th). However, this return value is ignored. The `xmas` variable still represents the original value, so Option B is correct.

33. A. Line 3 creates an empty `StringBuilder`. Line 4 adds three characters to it. Line 5 removes the first character, resulting in `ed`. Line 6 deletes the characters starting at position 1 and ending right before position 2, which removes the character at index 1, which is `d`. The only character left is `e`, so Option A is correct.

34. B. While it is common for a `Predicate` to have a generic type, it is not required. When the generic is omitted, it is treated like a `Predicate` of type `Object`. Since the `equals()` method exists on `Object`, this is fine. Option B is correct because the `Predicate` tests as `false`.

35. C. In Java, most things use zero-based indexes, including arrays and a `String`. Months are an exception to this convention starting Java 8. This makes the answer either Option C or D. However, `LocalTime` does not contain date fields, so it has to be Option C.

36. C. `Predicate` is an interface with one method. The method signature is `boolean test(T t)`. Option C is the answer because the method accepts one parameter rather than two.

37. B. Be careful here. The `Period` class uses a `static` helper method to return the period. It does not chain method calls, so `period1` only represents three days. Since three days is less than 10 days, `period2` is larger, and Option B is correct.

38. B. The code starts by correctly creating a date representing January 1, 2017, and a

period representing one day. It then explicitly defines the format as month followed by day followed by year. Finally, the code subtracts a day, giving us the formatted version of December 31, 2016.

39. C. The `trim()` method returns a `String` with all leading and trailing white space removed. In this question, that's the seven-character `String`: `":) - (:"`. Options A and B are incorrect because they do not remove the first blank space in `happy`. Option D is incorrect because it does not remove the last character in `happy`. Therefore, Option C is correct.

40. C. The `Period` class creates immutable objects and is usually used to add/subtract from a `LocalDate` or `LocalDateTime` object. It allows creating date, week, month, or year periods. Since it cannot be used for time, Option C is the answer.

41. D. Line 4 creates a `StringBuilder` of length 5. Pay attention to the `substring()` method `StringBuilder`. It returns a `String` with the value `321`. It does not change the `StringBuilder` itself. Then line 6 is retrieving the second indexed element from that unchanged value, which is `4`. Therefore, Option D is correct.

42. B. This one is tricky. There are two `remove()` methods available on `ArrayList`. One removes an element by index and takes an `int` parameter. The other removes an element by value. Due to the generics, it takes an `Integer` parameter in this example. Since the `int` primitive is a better match, the element with index 2 is removed, which is the value of `1`. Therefore, Option B is correct.

43. C. `ArrayList` has a `size()` method rather than a `length()` method, making Option A incorrect. The `charAt()` and `length()` methods are declared on both `String` and `StringBuilder`. However, the `insert()` method is only declared on a `StringBuilder` and not a `String`. Therefore, Option C is correct.

44. C. The `minusNanos` and `plusNanos` are the smallest units available, making Option C correct. Option D is incorrect because `LocalTime` is not that granular. Note that while you can add milliseconds by adding many nanoseconds, there isn't a method for it. A millisecond is also larger than a nanosecond. Finally, don't be tricked by the fact that `LocalTime` is immutable. You can still add time; it just gets returned as a different object.

45. D. When creating a formatter object, remember that `MM` represents month while `mm` represents minute. Since there are not minutes defined on a `LocalDate` object, the code throws an `UnsupportedTemporalTypeException`. You don't need to know the name of the exception, but you do need to know that an exception is thrown.

46. D. There are two signatures for the `replace()` method. One takes two `char` parameters. The other signature takes a `CharSequence`. Both `String` and `StringBuilder` implement this interface. This makes all three alternatives correct, and Option D is correct.

47. C. Pay attention to the data types. The `print()` method is looping through a list of `String` objects. However, the `Predicate` expects an `Integer`. Since these don't match,

the `if` statement does not compile.

8. D. Line 12 creates an empty `ArrayList`. While it isn't recommended to use generics on only the left side of the assignment operator, this is allowed. It just gives a warning. Lines 13 and 14 add two elements. Line 15 resets to an empty `ArrayList`. Line 16 adds an element, so now we have an `ArrayList` of size 1. Line 17 attempts to remove the element at index 1. Since Java uses zero-based indexes, there isn't an element there and the code throws an `IndexOutOfBoundsException`.

9. C. The declaration of `witch` is incorrect. It tries to store a `char` into a `String` variable reference. This does not compile, making Option C correct. If this was fixed, the answer would be Option B.

0. C. The Java 8 date and time classes are immutable. This means they do not contain setter methods and the code does not compile.