Chapter 5: Using Loop Constructs

1. D. A `while` loop has a condition that returns a `boolean` that controls the loop. It appears at the beginning and is checked before entering the loop. Therefore, Option D is correct. A traditional `for` loop also has a `boolean` condition that is checked before entering the loop. However, it is best known for having a counter variable, making Option B incorrect. Option A is incorrect because the `boolean` condition on a `do-while` loop is at the end of the loop. Option C is incorrect because there is no condition as part of the loop construct.
2. B. A traditional `for` loop is best known for having a loop variable counting up or down as the loop progresses. Therefore, Option B is correct. Options A and D are incorrect because `do-while` and `while` loops are known for their `boolean` conditions. Option C is incorrect because the `for`-each loop iterates through without an index.
3. A. A `do-while` loop checks the loop condition after execution of the loop body. This ensures it always executes at least once, and Option A is correct. Option B is incorrect because there are loops you can write that do not ever enter the loop body, such as `for (int i=0;i<1;i++)`. Similarly, Option D is incorrect because a `while` loop can be written where the initial loop condition is `false`. Option C is incorrect because a `for`-each loop does not enter the loop body when iterating over an empty list.
4. C. While a traditional `for` loop often loops through an array, it uses an index to do so, making Option B incorrect. The `for`-each loop goes through each element, storing it in a variable. Option C is correct.
5. B. The `continue` keyword is used to end the loop iteration immediately and resume execution at the next iteration. Therefore, Option B is correct. Option A is incorrect because the `break` statement causes execution to proceed after the loop body. Options C and D are incorrect because these are not keywords in Java.
6. A. The `break` keyword is used to end the loop iteration immediately, skip any remaining executions of the loop, and resume execution immediately after the loop. Therefore, Option A is correct. Option B is incorrect because execution proceeds at the next execution of the current loop for `continue`. Options C and D are incorrect because these are not keywords in Java.
7. B. A traditional `for` loop is best known for having an initialization statement, condition statement, and update statement. Option B is correct.
8. C. With a traditional `for` loop, you control the order in which indexes are visited in code. This means you can loop through an array in ascending or descending order, and Option C is correct.
9. A. With a `for`-each loop, the loop order is determined for you. With an array, this means starting with index 0, and Option A is correct. A traditional `for` loop allows you to control the order and iterate in either order.

10. A. A `do-while` loop has a condition that returns a `boolean` at the end of the loop. Therefore, Option A is correct. Option D is incorrect because a `while` loop has this condition at the beginning of the loop. A traditional `for` loop is best known for having a loop variable, making Option B incorrect. Option C is incorrect because there is no condition as part of the loop construct.

11. B. A `while` loop requires a `boolean` condition. While `singer` is a variable, it is not a `boolean`. Therefore, the code does not compile, and Option B is correct.

12. B. This is a correct loop to go through an `ArrayList` or `List` starting from the end. It starts with the last index in the list and goes to the first index in the list. Option B is correct.

13. A. The first time through the loop, the index is 0 and `glass,` is output. The `break` statement then skips all remaining executions on the loop and the `main()` method ends. If there was no `break` keyword, this would be an infinite loop because there's no incrementor.

14. A. Immediately after `letters` is initialized, the loop condition is checked. The variable `letters` is of length 0, which is not equal to 2 so the loop is entered. In the loop body, `letters` becomes length 1 with contents `"a"`. The loop index is checked again and now 1 is not equal to 2. The loop is entered and `letters` becomes length 2 and contains `"aa"`. Then the loop index is checked again. Since the length is now 2, the loop is completed and `aa` is output. Option A is correct.

15. D. There are three arguments passed to the program. This means that `i` is 3 on the first iteration of the loop. The program prints `args`. Then `i` is incremented to 4. Which is also greater than or equal to 0. Since `i` never gets smaller, this code produces an infinite loop and the answer is Option D.

16. B. Since `count` is a class variable that isn't specifically initialized, it defaults to 0. On the first iteration of the loop, `"Washington"`, is 11 characters and `count` is set to 1. The `if` statement's body is not run. The loop then proceeds to the next iteration. This time, the post-increment operator uses index 1 before setting `count` to 2. `"Monroe"` is checked, which is only 6 characters. The `break` statement sends the execution to after the loop and `2` is output. Option B is correct.

17. C. At first this code appears to be an infinite loop. However, the `count` variable is declared inside the loop. It is not in scope after the loop where it is referenced by the `println()`. Therefore, the code does not compile, and Option C is correct.

18. D. A `for` loop is allowed to have all three segments left blank. In fact, `for(;;) {}` is an infinite loop.

19. C. It is not possible to create an infinite loop using a `for`-each because it simply loops through an array or `ArrayList`. The other types allow infinite loops, such as, for example, `do { } while(true)`, `for(;;)` and `while(true)`. Therefore, Option C is correct. And yes, we know it is possible to create an infinite loop with `for`-each by

creating your own custom `Iterable`. This isn't on the OCA or OCP exam though. If you think the answer is Option D, this is a great reminder of what not to read into on the real exam!

20. A. This is a correct loop to go through an `ArrayList` or `List` starting from the beginning. It starts with index 0 and goes to the last index in the list. Option A is correct.

21. D. Braces are optional around loops if there is only one statement. Parentheses are not allowed to surround a loop body though, so the code does not compile, and Option D is correct.

22. B. The `for`-each loop uses a variable and colon as the syntax, making Option B correct.

23. C. In this figure, we want to end the inner loop and resume execution at the `letters` label. This means we only want to break out of the inner loop. A `break` statement does just that. It ends the current loop and resumes execution immediately after the loop, making `break;` a correct answer. The `break numbers;` statement explicitly says which loop to end, which does the same thing, making it correct as well. By contrast, `break letters;` ends the outer loop, causing the code only to run the `println()` once. Therefore, two statements correctly match the diagram, and Option C is correct.

24. B. In this figure, we want to end the inner loop and resume execution at the `letters` label. The `continue letters;` statement does that. The other two statements resume execution at the inner loop. Therefore, only the second statement correctly matches the diagram, and Option B is correct.

25. C. A `while` loop checks the `boolean` condition before entering the loop. In this code, that condition is false, so the loop body is never run. No output is produced, and Option C is correct.

26. C. A `for`-each loop is allowed to be used with arrays and `ArrayList` objects. `StringBuilder` is not an allowed type for this loop, so Option C is the answer.

27. B. This is a correct do-`while` loop. On the first iteration of the loop, the `if` statement executes and prints `inflate-`. Then the loop condition is checked. The variable `balloonInflated` is `true`, so the loop condition is `false` and the loop completes.

28. D. Immediately after `letters` is initialized, the loop condition is checked. The variable `letters` is of length 0, which is not equal to 3, so the loop is entered. In the loop body, `letters` becomes length 2 and contains `"ab"`. The loop index is checked again and now 2 is not equal to 3. The loop is entered and `letters` becomes length 4 with contents `"abab"`. Then the loop index is checked again. Since the length 4 is not equal to 3, the loop body is entered again. This repeats for 6, 8, 10, etc. The loop never ends, and Option D is correct.

29. B. In a `for` loop, the segments are an initialization expression, a `boolean` conditional, and an update statement in that order. Therefore, Option B is correct.

30. B. On the first iteration through the outer loop, `chars` becomes 1 element. The inner loop is run once and `count` becomes 9. On the second iteration through the outer loop, `chars` becomes 2 elements. The inner loop runs twice so `count` becomes `7`. On the third iteration through the outer loop, `chars` becomes 3 elements. The inner loop runs three times so `count` becomes `4`. On the fourth iteration through the outer loop, `chars` becomes 4 elements. The inner loop runs four times so `count` becomes `0`. Then both loops end. Therefore, Option B is correct.

31. A. On the first iteration of the outer loop, `i` starts out at `10`. The inner loop sees that `10 > 3` and subtracts 3, making the `7` the new value of `i`. Since `7 > 3`, we subtract 3 again, making `i` set to `4`. Yet again `4 > 3`, so `i` becomes `1`. Then `k` is finally incremented to `1`. The outer loop decrements `i i`, making it `0`. The `boolean` condition sees that `0` is not greater than `0`. The outer loop ends and `1` is printed out. Therefore, Option A is correct.

32. D. Options A and C do not compile as they do not use the correct syntax for a `for`-each loop. The `for`-each loop is only able to go through an array in ascending order. It is not able to control the order, making Option C incorrect. Therefore, Option D is the answer.

33. C. Since there are no brackets around the `for` statement, the loop body is only one line. The `break` statement is not in the loop. Since `break` cannot be used at the top level of a method, the code does not compile, and Option C is correct.

34. C. Multiple update expressions are separated with a comma rather than a semicolon. Tricky, we know. But it is an important distinction. This makes Option C correct.

35. D. There are three arguments passed to the program. This means that `i` is 3 on the first iteration of the loop. The program attempts to print `args[3]`. Since indexes are zero based in Java, it throws an `ArrayIndexOutOfBoundsException`.

36. B. The first time the loop condition is checked, the variable `tie` is `null`. The loop body executes, setting `tie`. Despite the indention, there are no brackets surrounding the loop body so the `print` does not run yet. Then the loop condition is checked and `tie` is not `null`. The `print` runs after the loop, printing out `shoelace` once, making Option B correct.

37. C. The code compiles as is. However, we aren't asked about whether the code compiles as is. Line 27 refers to a loop label. While the label is still present, it no longer points to a loop. This causes the code to not compile, and Option C is correct.

38. C. The `continue` statement is useless here since there is no code later in the loop to skip. The `continue` statement merely resumes execution at the next iteration of the loop, which is what would happen if the if-then statement was empty. Therefore, `count` increments for each element of the array. The code outputs `4`, and Option C is correct.

39. C. A `do-while` loop requires a `boolean` condition. The `builder` variable is a `StringBuilder` and not a `boolean`. The code does not compile, and Option C is correct.

40. A. At first this code appears to be an infinite loop. However, there is a `break` statement. On line 6, `count` is set to `0`. On line 9, it is changed to `1`. Then the condition on line 10 runs. `count` is less than 2 so the inner loop continues. Then `count` is set to 2 on the next iteration of the inner loop. The loop condition on line 10 runs again and this time is false. The inner loop is completed. Then line 11 of the outer loop runs and sends execution to after the loop on line 13. At this point `count` is still `2`, so Option A is correct.

41. C. Option A breaks out of the inner loop, but the outer loop is still infinite. Option B has the same problem. Option C is correct because it breaks out of both loops.

42. B. This code is correct. It initializes two variables and uses both variables in the condition check and the update statements. Since it checks the size of both arrays correctly, it prints the first two sets of elements, and Option B is correct.

43. B. Looping through the same list multiple times is allowed. The outer loop executes twice. The inner loop executes twice for each of those iterations of the outer loop. Therefore, the inner loop executes four times, and Option B is correct.

44. B. The initializer, which is `alpha`, runs first. Then Java checks the condition, which is `beta`, to see if loop execution should start. Since `beta` returns `false`, the loop is never entered, and Option B is correct.

45. B. The initializer, which is `alpha`, runs first. Then Java checks the condition, which is `beta`, to see if loop execution should start. Then the loop body, which is `delta`, runs. After the loop execution, the updater, which is `gamma`, runs. Then the loop condition, which is `beta`, is checked again. Therefore, Option B is correct.

46. C. Option A goes through five indexes on the iterations: 0, 1, 2, 3 and 4. Option B also goes through five indexes: 1, 2, 3, 4 and 5. Option D goes through five iterations as well, from 0 to 4. However, Option C goes through six iterations since the loop condition is at the end of the loop. Therefore it is not like the others, and Option C is the answer.

47. D. The first time the loop condition is checked, the variable `tie` is `null`. However, the loop body is empty due to the semicolon right after the condition. This means the loop condition keeps running with no opportunity for `tie` to be set. Therefore, this is an infinite loop, and Option D is correct.

48. C. Remember to look for basic errors before wasting time tracking the flow. In this case, the label of the loop is trying to use the keyword `for`. This is not allowed, so the code does not compile. If the label was valid, Option A would be correct.

49. D. On the first iteration of the loop, the `if` statement executes printing `inflate-`. Then the loop condition is checked. The variable `baloonInflated` is `true`, so the loop condition is `true` and the loop continues. The `if` statement no longer runs, but the variable never changes state again, so the loop doesn't end.

50. B. In a `for` loop, the type is only allowed to be specified once. A comma separates

multiple variables since they are part of the same statement. Therefore, Option B is correct.