Consider the following code:

```
1  String s = "duke";
2  if (x < s.length() && s.charAt(x) == 'e'){
3    System.out.println("character was e");
4  }
```

If x has the value 4, will the second part of the condition (the part to the right of the &&) be evaluated?

- No

Correct

- Yes

Continue

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
Okay, now you have an algorithm which works with any of the three stopCodons. However, let's explore how you can make this code work if we decided to have find stopCodon return at -1 when there's no valid stopCodon, rather than the length of the string. Note that this is a perfectly valid design choice and mirrors what the .indexOf method does. But we need to learn a new concept to make it work with our code. You'd want to change line six of the gene finding algorithm to reflect this choice. You cannot just take the minimum since -1 is smaller than any valid index. So what you want to do is pick the smallest number that's not -1. Let's look at a few examples. Here, taaIndex is -1, and tgaIndex is 3, and tagIndex is 6. What index should you pick for the stopCodon?

0:58
Well, we want 3 because it's the smallest number which is not -1. Notice, we can't just pick the minimum one because that would give us -1 which is not a valid value here. What if we had 5, -1, and 8? We'd want 5.

1:14
LIkewise, with 10, 4, and -1 we'd want 4. And with -1, -1 and 11 we'd want 11. Let's think about how we can express this decision process algorithmically. In the first example we compared these two numbers first. Even if you think you're comparing all three at once, you're really comparing two at a time very quickly and perhaps without thinking about what you're doing. Of these two values we prefer 3 since the alternative is -1. And then we compare 3 against 6. We choose 3 since it's smaller than 6. In the second example we compare 5 to -1.

2:02
Then we choose 5 and compare this against 8. In this choice you'd pick the smaller, which is 5. In the third example you'd compare 10 against 4. You'd choose 4 because it's smaller.

2:19
And then you'd compare 4 against -1 and choose 4. In the final example, you're comparing -1 against -1. It doesn't really matter which you pick, they're the same. You then compare -1 against 11 and you choose 11. Note, you should think about what you would do if all three values were negative. What would that mean? It would mean that no valid stopCodon exists. We'll have to use that in our code. Now how do we think through how we made these choices? We're only going to look at the logic involved in making the choice between one pair. That same logic will work with the second choice. In the first example we picked tgaIndex because taaIndex was -1. In the second example we picked taaIndex because tgaIndex was -1. In the third example where neither is -1 we picked tgaIndex because it's smaller than taaIndex.

3:26
And in the final example it didn't matter which we picked, since both were -1.

3:32
Let's write down what our logic is for making that selection.

3:36
If taaIndex is -1, we'd want to pick tgaIndex.

3:43
But that's not all there is to it. We'd also choose tgaIndex if both tgaIndex is different than -1 and tgaIndex is less than taaIndex. Notice how we've expressed this logic with or and and. Logical connectives which let us make complex conditionals out of simple ones. Now let's put that logic into the algorithm we're working on.

4:11
We'll use that logic to pick between taaIndex, and tgaIndex. And we'll store the best choice there is in a variable called minIndex. Then we'll choose the same logic to choose between minIndex and tagIndex.

4:29
If minIndex is -1, there were no valid stopCodons so we'll give back the empty string.

4:38
Otherwise, we've found a gene.

4:41
Now let's see how we express these ORs and ANDs in Java.

4:46
You can express AND with two ampersands. This is a Shift+7 on most U.S. keyboards. We can see in this example if (x < y && y < z).

5:03
You can express OR with two vertical bars, also called pipes. On most U.S. keyboards this symbol is Shift+backslash, and we can see an example here. if (a > b || c < d).

5:19
In the particular case the algorithm we're working on, you could express the step as shown here. Notice how the OR corresponds to the two vertical bars and the AND corresponds to the two ampersands.

5:34
AND and OR have special rules called short circuit evaluation. Which basically means that if Java can figure out the result of an entire expression involving AND and OR by evaluating only the first operand, then Java will skip evaluating the second operand. Let's look at an example.

5:56
Suppose x is 8 and y is 1.

6:02
x Is less than y is not true because 8 is not less than 1. So x < y evaluates to false. There's no reason to evaluate y < z, because whether it's true or false the whole AND expression will be false. Because false AND anything is false.

6:25
For OR, consider this example and suppose a is 3, and b is 1. Here, a is greater than b is true, because 3 is greater than 1. So the whole OR expression will be true, regardless of whether c is less than d or not. So there's no point in evaluating the expression c < d because true OR anything is true.

6:53
Why is short circuit evaluation important? If you skip evaluating y < z in this example, it doesn't really matter so much. But it's much more useful when the second operand could crash your program if it's evaluated. For example, here we're checking if x is less than the length of a string AND that the character at the xth index of the string is the letter a.

7:22
If the first condition is false then x is not a valid index, it's beyond the end of the string. So trying to get the x character with the .charAt method would crash the program with a StringIndexOutOfBoundsException. But fortunately this line of code is safe because of short circuit evaluation. The .charAt method will never be called when x is greater than or equal to the length of the string. Relying on short circuit evaluation is a great example of defensive programming, and one of the tools in your Java programming toolkit. Have fun!

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?