Logical Operator &&

Hannah Wiltsey
Assignment Definition And General Feedback By Michael Panitz
at Cascadia Community College (http://www.cascadia.edu)

Table of contents:

Quick Summary:

In Java, you can tell a program to implement a method if or while a series of tests is true. Rather than nesting another if or while statement, adding the logical operator && means all tests which are therein connected must return true. If one or more of the tests does not return true, then the accompanying methods will not be executed. Also, if the first test does not return true, Java will not bother to run the others (unless there is also an "or" || operator, but I digress.)

When To Use This / Avoid This:

There are several situations where you will want to use this technique. One such time is when you only want a method to run if a number of different tests prove true (or one proves true and one false, if you use the ! negation symbol. But it would still be true for both anyway.)

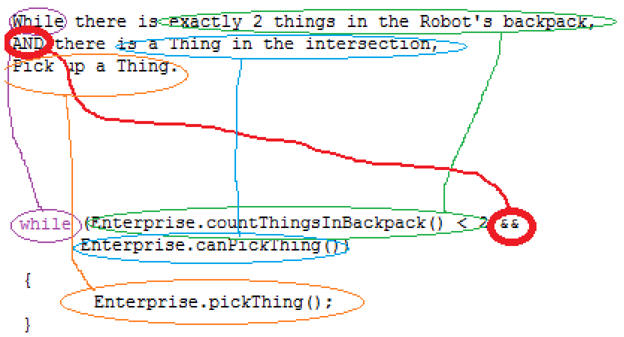
There are several situations where you will want to avoid this technique. If only one of the tests need to be true, in which case, use the "or" operator, ||

Example Of Usage:

Let's say you want the Robot Enterprise to pick up a Thing only while there is less than 2 Things in its backpack, AND there is a Thing in the intersection.

There are, essentially, 2 steps:
First, you determine what tests you wish to run. In this case, we'll start with saying what we want the program to do in "English": While there is less than 2 Things in Enterprise's Backpack, AND there is a Thing in the intersection, pick up a Thing.

Second, “Translate” (for lack of a better word) into Java:



```
While there is exactly 2 things in the Robot's backpack,
AND there is a Thing in the intersection,
Pick up a Thing.


while (Enterprise.countThingsInBackpack() < 2 &&
Enterprise.canPickThing())
{
    Enterprise.pickThing();
}
```

The important thing here is what is highlighted in the bold red arrangement. When saying you want thus-and-so AND such-and-such to be true, the “and” is represented by two ampersands, &&. The ampersands go within the parentheses of the if or while statement, and connect the two (or three, or four, etc.) tests. When there is more than one set of &&, (or any logical operator) the operations are executed in the order of precedence (which we technically don't have to worry about in this section because I'm only dealing with &&) or, if the precedence is the same, then they get executed in order from left to right. (This, also doesn't really apply with the “and” operator because if one test proves to be false, then none of the other tests will be executed anyway.).

Line	Code
1.	import becker.robots.*;
2.	
3.	public class Example_2 extends Object
4.	{
5.	public static void main(String[] args)
6.	{
7.	City UnchartedPlanet = new City(6, 9);
8.	Robot Enterprise = new Robot(UnchartedPlanet, 3, 3, Direction.EAST, 0);
9.	new Thing(UnchartedPlanet, 3, 4);
10.	new Thing(UnchartedPlanet, 3, 5);
11.	new Thing(UnchartedPlanet, 3, 7);
12.	
13.	while(Enterprise.getAvenue() != 8)
14.	{
15.	while (Enterprise.countThingsInBackpack() < 2 && Enterprise.canPickThing())
16.	{
17.	Enterprise.pickThing();
18.	}

```
19.                      Enterprise.move();
20.                 }
21.            }
22. }
```

Example Trace:

In a nutshell, this technique allows you to have two or more criteria for performing an action

In order to go over these details more thoroughly, here is a (partial) trace of the above program, with some additional explanation afterwards

Line#	Program Statement	Enterprise Location	Thing Locations	T/F
7	`City UnchartedPlanet = new City(6, 9);`	-	-	-
8	`Robot Enterprise = new Robot(UnchartedPlanet, 3, 3, Direction.EAST, 0);`	(3,3,E,0)	-	-
9	`new Thing(UnchartedPlanet, 3, 4);`	(3,3,E,0)	(3,4)	-
10	`new Thing(UnchartedPlanet, 3, 5);`	(3,3,E,0)	(3,4) (3,5)	-
11	`new Thing(UnchartedPlanet, 3, 7);`	(3,3,E,0)	(3,4) (3,5) (3,7)	-
13	`while(Enterprise.getAvenue() != 8)`	(3,3,E,0)	(3,4) (3,5) (3,7)	T
15	`while (Enterprise.countThingsInBackpack() <= 2 && Enterprise.canPickThing())`	(3,3,E,0)	(3,4) (3,5) (3,7)	F
19	`Enterprise.move();`	(3,4,E,0)	(3,4) (3,5) (3,7)	-
13	`while(Enterprise.getAvenue() != 8)`	(3,4,E,0)	(3,4) (3,5) (3,7)	T
	`while (Enterprise.countThingsInBackpack() <= 2 && Enterprise.canPickThing())`	(3,4,E,0)	(3,4) (3,5) (3,7)	T
17	`Enterprise.pickThing();`	(3,4,E,1)	(3,5) (3,7)	-
19	`Enterprise.move();`	(3,5,E,1)	(3,5) (3,7)	-
13	`while(Enterprise.getAvenue() != 8)`	(3,5,E,0)	(3,5) (3,7)	T

	`while`
	`(Enterprise.countThingsInBackpack() <= 2`
	`&&`
	`Enterprise.canPickThing())`
		(3,4,E,1)	(3,5) (3,7)	T

17	`Enterprise.pickThing();`	(3,4,E,2)	(3,7)	-

19
 	
13
	`Enterprise.move();`
	
	`while(Enterprise.getAvenue() !=`
	`8)`
		(3,6,E,2)
		
		(3,6,E,2)
			(3,5) (3,7)
			
			(3,7)
				-
				
				T

	`while`
	`(Enterprise.countThingsInBackpack() <= 2`
	`&&`
	`Enterprise.canPickThing())`
		(3,6 ,E,2)	(3,7)	F

19	`Enterprise.move();`	(3,7,E,2)	(3,7)	-

13	`while(Enterprise.getAvenue() !=`
	`8)`
		(3,7,E,2)	(3,7)	T

	`while`
	`(Enterprise.countThingsInBackpack() <= 2`
	`&&`
	`Enterprise.canPickThing())`
		(3,7,E,2)	(3,7)	F

19
13
	`Enterprise.move();`
	`while(Enterprise.getAvenue() !=`
	`8)`
		(3,8,E,2)
		(3,8,E,2)
			(3,7)
			(3,7)
				-
				F

You'll notice that the trace starts at beginning of the **main** function, on line 7, like normal. It proceeds normally until line 15, where we see the && operator. The first time the && operator is called, the first test (things in backpack being less than (or equal to) 2) returns true, but the second test (the robot can pick thing) returns false, so the entire statement returns false. The next two times the && operator is encountered, both tests return true, so the whole statement is true, so the Enterprise picks up the Thing. The third time the && operand is encountered, the first test returns true (things in backpack are less than or equal to 2), but the second test (a thing in the intersection) returns false. The fourth (and final) time the && operand is encountered, the second test returns true (there is a Thing in the intersection), but the first test returns returns false, because there are exactly 2 Things in the backpack, therefore the whole statement is false.

Syntax Explanation:

In regards to syntax, the logic operator && is used to connect two BOOLEAN expressions, something that will output either true or false.

Help With The Logic:

This is good to use whenever you want more than one criteria to be met before a program executes a certain method.

Important Details:

When there is more than one set of &&, (or any logical operator) the operations are executed in the order of precedence (which we technically don't have to worry about in this section because I'm only dealing with &&) or, if the precedence is the same, then they get executed in order from left to right. (This, also doesn't really apply with the "and" operator because if one test proves to be false, then none of the other tests will be executed anyway.).

Licensing

Plagiarism