# Transcript: Infinite Loops

>> Let's say a few words about infinite loops. When a while loop is executing, eventually its boolean expression should become false. If it doesn't become false of course the loop doesn't end. And we have in this case what's called an infinite loop. In other words, it's gonna execute until the program is interrupted by some external force, usually a user ending the program for example.

There are some situations where you may want an infinite loop, in a control system or something like that. And which runs forever until someone actually presses a hardware button. But, usually, these are just common logical errors, and we don't want infinite loops. So you need to double check the logic in your programs to ensure that your loops will terminate normally.

Here's an example of an infinite loop. Here we have int count = 1, and while count is less than or equal to 25, we want to print out count and then we're decrementing count, and then we go back and check again. Well, it turns out that this is an infinite loop, because we're decrementing count instead of incrementing count.

We could have count minus minus here instead of count = count- 1. But, this loop is gonna continue executing until it's interrupted. If you're in a DOS window or shell, enter in Ctrl-C. Or if you're in jGRASP, for example, you could, click the End button on the Run I/O tab, or in some cases until underflow occurs.

Let's take a look at this particular program. Here we've got some program, and we've got main here, and, I'm just gonna run this. And notice here I've replaced count = count- 1 with just count--. We should have been incrementing probably to print out count 25 times. But let's run this and see what happens with an infinite loop.

Now, this is gonna run a while. In fact since this is an int, it takes on the value of plus or minus 2 gig, and that's a lot, billion. And so this will run a while. So the way we would stop this loop is to click add on it, and it just stops it.

Now let see what happens in the debugger when we run this. In fact we are gonna look at underflow. Let's look at how this can stop with underflow. I'm gonna change int to byte, that way we are only go to 250 rather 128. A byte takes on values plus or minus 128.

So, we'll actually have some underflow here and we'll we'll see things stop. Let's run this in the debugger. In fact, let's let's open a canvas, we'll open a canvas here and we'll run it in the canvas so we can sorta see it run. So here's our bite I'll leave just so we have something on the canvas I'll put that on there.

Now I'm gonna hit Play, and this is, the default is about a half a second here, so let's set that. And so here you see we're going through this loop pretty slowly, but we can speed this up. And if you're a real speed reader, you know, we can go fast, and it stops at after -127.

Since we're decrementing after -128, it actually rolls over and becomes a positive number, we call this, this thing, a byte only. These 16 bits only represent plus or minus 128, and so here when it got to -128 it it rolled over and and stopped there. So, bottom line is, let me just close that, I'llr get rid of it, discard there.

Bottom line is, you wanna be careful with your loops. In fact, just a side note here in interactions, when you're executing, when you're calling your methods, now that we have loops. if you call one of your methods, it's looking through a list, for example, and maybe coming back with an average or something.

When you call that in interactions, if it doesn't come back as interactions appears to be locked up, you probably have an infinite loop. And this happens all the time, so nothing to be concerned about, but what you would do is click End on the interactions tab here Here, here this is the End button, you'd click that and that would stop it.

And then you should go look at that method that you've tried to invoked, or that you have invoked to see if if there's a loop in it. OK, next we'll look at nested loops.