CS 240: Debugging in IntelliJ Transcript

This video shows a split screen of Professor Wilkerson on the right and a PowerPoint screen on the left. Chunks of code will be shown on the screen. Knowing the exact syntax of the code is not necessary. Any other text displayed or action performed that is not verbalized will be included in italics as visual descriptions.

[00:00:00] **JEROD WILKERSON:** Now we'll take a look at my ball clock program that would be an implementation for the tech assessment that I described before for that last company that I worked at.

[00:00:13] I actually have multiple implementations.

[00:00:15] This is one that is not all that fast, but I did it in an object-oriented way so you can see the structure of the program, so that is the best one to look at.

[00:00:25] I have another one that runs a lot faster, and is a lot harder to understand when you look at the code.

[00:00:30] We'll look at the easy-to-understand one.

[00:00:32] First of all, we have a Ball class, and that is just a POJO, just a Plain Old Java Object, and it represents a ball by giving it an ID number, and then it just has some things like toString, so we can print out what ball number it is, equals method, and hash code.

[00:00:49] Just a really simple basic POJO.

[00:00:52] Then this is the actual ball clock.

[00:00:56] I'm not going to describe all the details.

[00:00:58] It's not that important that you know how to write a ball clock.

[00:01:00] What's important is that you learn how to debug, but I want you to understand enough of it that the code that we're going to be debugging will make some sense.

[00:01:07] It has some constants first.

[00:01:10] Notice that it has constants specifying the size of the different tracks, or the queues, and then it has a place, it has instances of the deque or deck, however you pronounce that.

[00:01:23] It has instances.

[00:01:26] There's one to represent the minutes, the five minutes, and the hours.

[00:01:32] This queue is like the bin where all the balls are picked up from.

[00:01:38] Then we have a main method, and as I mentioned, we have to keep track of how long it takes the program to run, so that keeps track of the start time in milliseconds and then I can grab the end time so I can print out how long the program took.

[00:01:54] There are two modes that can run in depending on which parameters that I give it.

[00:02:00] That's the general structure of the main method.

[00:02:03] This main method calls a run method.

[00:02:08] If we look at the run method, there are two modes the method can run in depending on whether it got a number of balls and a number of minutes or just a number of balls.

[00:02:18] With that background, we're ready to start debugging this.

[00:02:22] The main method is the tick method.

[00:02:25] When you call the tick method, that's equivalent to the ball clock grabbing a ball and moving it to a track.

[00:02:30] We will start by seeing how to set breakpoints in a method.

[00:02:35] You do that by just clicking over in the gray area on the line you want the breakpoint to be in.

[00:02:39] I'll put one on both tick methods.

[00:02:42] Now we'll go ahead and run this in debug mode.

[00:02:46] The way you run in debug mode, you already know a few ways to run a program in regular mode.

[00:02:52] You can click this *(the Run button next to the line number of code you're trying to run)* or you can right-click *(on the code itself)* and select Run Clock.

[00:02:58] Notice, we also have a debug here *(in the same right-click menu as Run Clock)*, so I can specify Debug and that will run it in debug mode, or I can click this little bug icon *(the icon for Debug on the menu)*.

[00:03:05] We'll click the bug icon.

[00:03:07] Now that's going to start up my program and notice that it stopped right here.

[00:03:11] That's what a breakpoint will do.

[00:03:12] You probably already knew that, but there are some things that I'll show you that maybe you don't already know about how to debug in IntelliJ.

[00:03:19] That stops at our breakpoint and now I can see several things and I can do some things.

[00:03:26] First of all, I can see right here *(the left window in the debug window, under the Frames tab)*, this is showing me all of the current stack frames.

[00:03:32] Programs will run by using a stack to keep track of what methods have been called throughout the running of the program.

[00:03:41] You always start in the main method, so the bottom method on a stack will always be main, and then we were executing a run method, and that's the one that had the breakpoint in it so run is the top thing on the stack.

[00:03:53] That shows us that's the method that we're currently in.

[00:03:56] Then we have this pane *(the middle window in the debug window, labeled Variables)* where we have instance variables and local variables.

[00:04:04] These are all the local variables that are visible inside of this method.

[00:04:07] We can see all their values, and if we open up this, we can see all of the instance variables and all of their values, so that allows me to see what's in the tracks.

[00:04:18] Then this is watches *(in the right window of the debug window, under the Watches tab)*.

[00:04:19] I had one here that I just deleted because I want you to see how I create it.

[00:04:23] We can specify different Java code here that we want to see while we're running our program, and I'll show you what I mean by that and how that works a little bit later.

[00:04:33] Now that we've seen what information we have, we can do a couple of things here.

[00:04:39] We can hit Resume Program *(in the left sidebar of the debug window)*.

[00:04:42] That will cause the program to run to the next breakpoint or I can step through the program.

[00:04:47] There are different ways to step, but by far the most common are these two. *Begin visual description. Wilkerson is referring to two buttons on the topbar of the debug window. The first an arrow that goes up then down. The second is an arrow that's just pointing down. End visual description.*

[00:04:50] This is the Step Over method *(the up-down arrow)*, and this is a Step Into method *(the down arrow)*.

[00:04:54] If I step over, we'll just go to the next method and we'll just keep stepping as long as we're in that while loop.

[00:05:04] First of all, when I stepped over it, it didn't skip running this method.

[00:05:08] It ran it, but it didn't go inside the method.

[00:05:10] If I want to go inside the method and step through it, I can do Step Into, and watch when I click on this; you're going to see this *(the Frames tab in the left window of the debug window)* change.

[00:05:18] When I stepped into it, now we're in the tick method and so now the tick method is on the top of the stack.

[00:05:24] Now I can step over or step into from there.

[00:05:28] That allows me to just walk through my code and see what's going on with it, and that can be really useful.

[00:05:34] I can also rerun the program.

[00:05:37] Sometimes when you're debugging, you'll find out that you step past what you needed to see, so now you need to rerun it and get there again.

[00:05:43] You can just do that by hitting this icon *(the top icon in the left sidebar of the debug window)*, that will rerun it, and now it has started over and we're seeing a new run of the program.

[00:05:53] One thing that's really useful to do is to be able to set a conditional breakpoint.

[00:05:58] Sometimes we're trying to step through code that's in a loop and maybe the thing that we're trying to see, maybe we're seeing some bug on the 1,000th iteration of some loop.

[00:06:10] Obviously, we don't want to sit here and click and hit Resume 1,000 times, so the way we can avoid that is we can set a conditional breakpoint.

[00:06:17] First, just set a breakpoint, then you right-click it and you can set a condition.

[00:06:22] The condition is Java code, so you write some Java code, a Boolean statement, and whenever that is true, the breakpoint will stop, but it won't stop until that condition is true.

[00:06:34] I can say something like hours.size() equals 2 and what that's going to do is now instead of stopping on every tick, it won't stop until the hours variable has a size of 2.

[00:06:51] In other words, we've ticked enough times that we have two balls in the hour queue.

[00:06:56] If I set that, notice, let's look at these variables.

[00:07:00] Notice that the variables here, our queue is full, our hours, five minutes, and minute tracks are empty because this is the first run of the program, but now when I hit Resume, it's not going to stop at that breakpoint.

[00:07:13] It looked like it happened instantly, but it actually ran several times through the while loop and it didn't stop until it had two balls in the hours queue.

[00:07:21] That's really useful, and you'll find that conditional breakpoints will often save you quite a bit of time and quite a bit of trouble.

[00:07:27] Another thing that we often want to do is we just want to be able to execute some code, some Java code, at a certain place in our program while we're debugging and we do that with a watch.

[00:07:37] I'll show you what we can do with that.

[00:07:39] For this particular program, it would be useful to have an easy way to see what time our clock is representing.

[00:07:46] We can do that with a watch. *(Wilkerson starts typing code in the Watch tab in the right window of the debug window.)*

[00:07:48] This is a little bit involved.

[00:07:49] Usually, these watches are pretty short.

[00:07:51] This one will be a little bit longer than usual, but in order to see this in a standard time format, I can get the value of the hours and I can concatenate that.

[00:08:04] I can just write any Java code that's valid at the point where this program is stopped.

[00:08:08] There's my hours followed by a colon, and now I'm going to do a little bit of a calculation.

[00:08:14] I need to multiply the number of balls in the five-minute queue times 5, and then add that to the number of balls in the minute queue to get the minutes. We'll do that.

[00:08:23] First five minutes.

[00:08:24] Let me see.

[00:08:30] I've got a syntax error here.

[00:08:32] That's why that's not helping me out.

[00:08:37] fiveMinutes.size() times 5.

[00:08:46] Make sure I have that in the right place.

[00:08:53] I did something wrong here.

[00:08:56] There we go. Plus minutes.size().

[00:09:08] If we look at what that prints out, that's the time.

The following line of code is a line in the Watch tab:

*hours.size() + ":" + ((fiveMinutes.size() * 5) + ((LinkedList)minutes).size)*

End of code.

[00:09:11] I can make this a lot longer and make it formatted with two spaces and all that, but it's not worth doing that for simple debugging.

[00:09:18] Now as I tick through this program, you can see it better.

[00:09:27] You can see that this watch is making it really easy for me to tell what time my program thinks it is based on how many ticks I've done.

[00:09:35] That is how you can create watches.

[00:09:39] You can really create any watch that you want in here.

[00:09:42] Any code that's valid, you should be able to put it in as a Java statement in a watch and that allows you to see things that are not necessarily showing up in variables.

[00:09:52] Often, the things that you would put there would be printing out an object, calling its toString method, doing some comparison to see if two objects are equal at a certain point in your program, so different things like that.

[00:10:05] There is a lot more that could be said about debugging in IntelliJ, but that's enough to probably do what you're going to need to do most of the time.