Programming Methodology-Lecture06

Instructor (Mehran Sahami):All righty, welcome back. If you haven't turned in the assignment yet and you, at some point want to, turn it into that box up there.

So a couple quick announcements before we start. First of which, first of which is the quarter is already 1/5 of the way over, right? After today it's like two weeks of ten week are over, so that's hard to believe.

But there's three handouts in the back including your next assignment because the fun never stops; when one assignment's due the next assignment goes out, and a couple other handouts. Assignment one, as you know, is due today so please drop it off in the box in front and, congratulations, you'll all – well, assuming you didn't take a late day you're all programmers, right? Because hopefully you all did, Karel, you got him to run around the world and do stuff. Hopefully figured out some interesting out rhythms and now you can turn it in. Question?

Student:I didn't know if you needed a hard copy [inaudible].

Instructor (Mehran Sahami):Un huh, you want to get a hard copy in as soon as you can, like right after class. But hard copies are important because we want you to turn in both because we use the electric submission to be able to actually run it and your section leader makes comments on the hard copy and so it's important to have both.

But just because you asked…

All righty. So I want to take a quick pain pole before we start. So let's actually dive into the real sort of meaningful things. What the pain pole really is – remember I asked you to think about how much time it actually took you to do the assignment? So total it up over all the Karel problems; how many total hours; think about it, it took you to actually do the assignment. Right? And we're just going to go through and do a quick show of hands.

How many people actually got through the assignment in zero to two hours? All right. Maybe like a couple people. I'll make a small bar. How about two to four? A fair number. Four to six? That's good to see. Six to eight; eight to ten; ten to 12; 12 to 14; 14 to 16; 16 plus? Rock on. Thanks for admitting it. It's a good time. Hopefully it was a good time.

But that's – first of all, one thing to note is the world is surprisingly normal. Right? Like everything in the world is just normally distributed, that's just the way it is.

The second thing is that computer programming, right or software engineering is a pretty high variance event, right that you can go from less than two hours to 16 plus. I don't joke when I say it's actually very high variance. But hopefully what this give you is again, what we shoot for, right, is about ten hours per week outside of class for, you know, work. We shoot for here and as I mentioned, you know, it looks like you're all

doing real well because you're sort of below the average point. The truth of the matter is kind of as the quarter goes on, the assignments tend to get a little bit harder which means you'll actually see this curve kind of move down a little bit more into that range. This is good times to actually see it here right now.

It also hopefully gives you a chance to gage for yourself how you're doing sort of relative to expectations. Right? If you're sort of doing real – you know, hopefully you put in your comments and it was good software engineering and everything and I'm totally willing to believe it was and you just wrote your code and it just all worked and it was beautiful and, sort of, if you're done in this end, as long as you're still feeling like you're understanding the concepts and you're plugging away, that's important and just keep plugging and you will do just fine, trust me.

There have been times when I have been down here myself and it wasn't fun when I was there, but you just keep plugging away and it works out.

And but the important thing is if you were sort of in a particular range, even if you're in this range and things just worked but you didn't understand why, that's more dangerous than being in this range and understanding why because all the concepts in this class will build on top of each other. So make sure you understand the concepts not just that, oh Karel happened to do the right thing. Yes, he got to the right spot in the middle of the world, but now he's just spinning around. That's fine if he's just spinning around, he just got that middle spot, right. We're kind of a lock, like you just throw in enough instructions until I kind of did the right thing; that's not real good understanding and you want to talk to me, talk to your section leader, talk to the TA to try to clear that up.

All right, so with that said we're just going to dive in because there's a question in the back.

Student: Is it an honorable violation if you look at someone else's code once you already both handed you assignments in and gotten it back?

Instructor (Mehran Sahami):Once you've gotten it back and it's already graded, it's fine to actually be able to look at someone else's code because at that point you can just kind of, you know, share ideas.

All right, any other questions?

All righty, so a couple things to cover real quickly. Last time we talked all about methods and some more about objects. There's two things you should know in the programs that you're going to be doing, is we talked a little bit about one of them last time in terms of how to get input from the user. There're these functions that you should know about.

One is called READ INT and there's some prompt inside double quotes that you give and what that does is ask the user basically for an integer and gives you back some value that you can say, assign to an integer. There's also a version of this to get doubles, which

surprisingly enough is called RE DOUBLE and has exactly sort of the same properties. So it's called RE DOUBLE; it has some string here as it's parameter or some text here in its parameter inside double quotes which it displays to the screen and then gets you back a value which is a double one you can assign to a double. Those are just two things off the bat that you should know about because that's how you're going to get input, at least for the time being, from the user in a lot of cases.

Now, one thing you want to do once you actually get some input from the user is, you want to do some manipulation on it like some expressions that we talked about last time. We talked about some of the different operators like addition, subtraction or unary minus, it's the same symbol, multiplication, division and my favorite, the remainder. And so we talked about all those except for this little guy last time. All of the operators kind of work the way you would expect them to, okay. And we'll talk a little bit more about division in just a second. The interesting thing about division – so all of these things work with both – or I should say – all of these work with both integers and doubles. The remainder, as we talked about, only works with integers, right because it doesn't make sense to have a remainder when you have real values.

These three guys work exactly the same for integers and double, just the way you would expect addition, multiplication, all that happy stuff, to work. Division kind of rears its ugly head because it actually works slightly differently if you're doing division for integers versus doubles. Okay? The whole point of that is, if you're doing a division and the two arguments that you're dividing, right if both of these things are integers; in this case I have integer constant which is what I mean, the values, right. If both of these integers, what it does is integer division which means it does the division and throws away any remainder. So what you get back is an integer. So 5 divided by 2 when these are integers gives you back the Value 2. That little remainder thing is just gone. If you want to get the remainder you use this guy. Okay?

If either one of these particular values happens to be a real value, like a double, then it will do real-value division and give you back a real value. So if you happen to divide 5, even if 5 is an integer, by the Value 2.0 and so it knows it's a real value because it's got a decimal point in it, this will give you back 2.5 as a double and so you can assign that to a double. Okay?

So if either one of the arguments is a double, you get real-value division; if they're both integers, you get back the integer portion. Un huh?

Student:I'm a little confused about the double; the double is just a real number?

Instructor (Mehran Sahami):It's just a real number. Yes.

So another thing that kind of comes up when you do expressions – yeah, sometimes you're taking notes and you just don't know; it's like candy raining from the sky.

The other thing to keep in mind is just like arithmetic, sometimes you want operators to evaluate in different order. There's an order precedent for how these things actually evaluate in case you have to have some big honking expression. The order of precedent is you can have parentheses. Parentheses are the highest precedent. That means you evaluate everything in parentheses first, then multiplication, division and the remainder operator have the same level of precedents. And so if you have multiple of them; they're evaluated from left to right and then addition and subtraction. Again, if you have multiple, evaluate left to right.

So it's just like regular rules of precedent in algebra, which hopefully you're familiar with, but to make that concrete let's say you have some integer X and we say X equals 1 plus 3 times 5 divided by 2. How does that actually evaluate?

Well first of all, we say do we have any parens? No we don't have any parens. That would be the highest level of precedence. You can always force something to evaluate more highly by putting it in parens. So these guys are all at the same level, so we evaluate left to right. So we come across and we say here, here's multiplication, we evaluate this thing as 15, right? We don't do this addition first; this 3 times 5 becomes 15. Then we divide it by 2. And, you remember what I just said? Hey, this is an integer, this is an integer so this is integer division, so 15 divided by 2 in integer division is?

Student: Seven.

Instructor (Mehran Sahami):Seven. Rock on.

So this whole thing is 7 and then we add the 1 to it because addition has the lowest level of precedence of all the operators here and what we get at the end of the day is that X is equal to 8.

Okay, hopefully you can see that and if you can't, I we'll just tell you X is equal to 8. So those are the rules of precedent. They're exactly the same as, hopefully you know from algebra. Un huh?

Student: Isn't it true [inaudible] should be in parentheses?

Instructor (Mehran Sahami):It's nice to clarify; it's nice to always put in parentheses. Sometimes you have to put in parentheses to get the right thing to compute and I'll show you that in just a second, but it's nice now to just to fully parenthesize everything. Uh huh?

Student: Where do exponents fall in there?

Instructor (Mehran Sahami):There is no built-in exponent operator. Okay? So if you're sort of like a mat-lab person or something like that there're functions that we'll get to later on that compute exponents but there's no build-in primitive operation for exponents. Okay?

So with that said, sometimes there's times in life when you say hey, but Marilyn, I have these two integers but I really want to get some value back which is some real value. Right? So what you can do is – let's say I have INT X, okay? And let me give X some initial value; so I can say X equals 5. Then I want to take like half of that and assign it to some double Y. So if I say double Y equals X divided by 2, you might think, hey, this is a double right, isn't it going to do the right thing and give me back 2.5? No. It evaluates the right hand side first and then assigns it for left hand side. So you have 5 here. That's an integer. This 2 is also an integer. It does integer division which means you get 2. Once you get that 2 it says hey, but that two into a double and says, okay, it's 2.0. So you're like, huh? That was totally weird, but that's because it was doing integer division. So what you need to do is, you need to tell it – you need to say, I want you to do real-value division by temporarily treating one of these things as a double. That's something that we refer to at a cast. It's kind of like you were making a movie and you need to come up with a cast and you know, you get someone who's going to like, play Harry Potter for you, but he doesn't really wear glasses so you say, for the purpose of being in the cast, I'm going to put some glasses on you. For this one thing, you're going to wear glasses and then we're going to take them off. It doesn't change intrinsically who you are, it just makes you appear different for this one operation.

And the way we do that is we specify the type that we want to cast to in front of the thing that we want to cast. So we would say DOUBLE X divided by 2. All this means is, take this thing, which is not normally a double, and for the purpose of this operation, treat it as though it were a double. It does not change X from being an integer intrinsically; X remains an integer after this operation, but now it becomes 5.0 and when we do the division, we do real-value division, we get 2.5 and that's what goes into Y.

Okay? So that's what's called a cast. You can also do that, interestingly enough – let's say you did that and now Y is some box somewhere that has the value 2.5 in it and you say, you know what, I really like the integer part of Y. And so I'm going to have some integer Z and I'm just going to set that to be equal to what I would get if I cast Y to be an integer.

Well, if I cast Y to be an integer, you might say, oh Marilyn, does it round? Like do I get 3 because I remember if it's a .5 we round up, that was always the way it was when life was good. No. You don't round up. As a matter of fact, it could be 2.9; it could be 2.9999, you still don't round up. This is computer science. It's not necessarily forgiving. We always round down. You take the integer of a real value, it truncates it; it drops anything after the decimal point; it doesn't matter how big it is, sorry. It's just like the dollar; we just devalued your currency, right. It just – drop everything after the decimal point. That's life in the city. Okay?

So, if we go to the computer for a second, we can put this all together in a little program and actually see what's going on. So I wrote a little program that averages two numbers and it's running right now. I'll show you the text for that program over here.

So here's some program average two integers. I say this program averages two numbers; I read in one number from the user; I read in another integer from the user and then I come here, and notice I commented for you that it's buggy, I say hey, the average is just adding the two together and diving by 2 right? That's what I remember with a mathematical expression for average of two numbers when I looked it up in my book of mathematical expressions for averaging two numbers. And then I write out the average. So what's the problem here? Uh huh?

Student: The [inaudible].

Instructor (Mehran Sahami):Yeah, so there's two problems here. Da da daa. And then we reveal the comment and there they are.

First, we need to parenthesize the expression because in terms of precedent, the division has precedent over the addition. So if we were to say take the average of 5 and 6, we don't get 5.5, we get 8. Why? Because it took half of 6; it took that 6 and divided it by 2 and then added it to the 5. So it's actually at – doing this operation over here first and we want to say, hey, add some parentheses to force the addition to happen first.

Now ever after we force the addition, this whole expression here is still going to be an integer because it adds two integers, that's just going to give us an integer back and then we divide that by an integer. There's two ways we can fix this. One is, we can change the 2 to a 2.0, so we can explicitly say that that constant in there that we're dividing by is real value and that will force real-value division.

Another way we can do it is to say treat this whole thing as a double for this operation. So after you add those numbers together you get something which is the total of those two numbers, treat that total as though it were a double and then do the division. So if we do that, then we're good to go. Uh huh?

Student: If you put a double in there, do you still need that other double before you add it?

Instructor (Mehran Sahami):If I put a double in where?

Student: Where it is now is now, [inaudible].

Instructor (Mehran Sahami):Uh huh, yeah because this is the type of average. Right? So if average is not – I need to declare average first of all; I need to give it a type. If I don't give it a type double it can't store a real number anyway. So that's what that double's about. Uh huh?

Student: If you cast a double can you expect everything after to double on the line?

Instructor (Mehran Sahami):Yeah, it affects the most immediate thing after it. If it's parenthesized, it's a parenthesized expression; it's just one variable, it's just that variable. Uh huh?

Student: If the user enters a double into the computer, what difference would it make?

Instructor (Mehran Sahami):Ah good questions because someone wrote this little read integer for us so let's run it and see what happens because it's just that smart.

So here's – and one, we're like hey, here's 5.5, I want 5.5 and it says illegal numerical format. Right? It has a big cow; it's in red. Right? All that means is, I want an integer; you didn't give me an integer. Give it to me again and notice it prompts you again and you're like, okay, sorry. I meant the letter A. It's still illegal in numeric format. I would do that with like my friend's computer; like you know, they write their programs and they didn't do like, nice error checking or whatever that all the READ INT stuff gives you and they're like, give me an integer and I'd be like, how about A. And they'd be like, we'll that's not an integer and I'm like, to me it is. I'd put it in and watch their program crash and, you know, I was mean. Yeah, there's 5.5, it's working. You laugh now, wait until I do it to you. Uh huh?

Student:[Inaudible].

Instructor (Mehran Sahami):Pardon:

Student:[Inaudible].

Instructor (Mehran Sahami):There's something you can do in java having to do with formatting. We're not going to get into that now. If you're really interested, you could come talk to me afterward.

Oh, , that's going to be the goal for this class; to see if I can get one to stay up on the camera.

All right, so with that said, hopefully you kind of understand the notices of castings for an operation and the rules of precedence that are actually involved. There's a couple shorthands you should also know. Uh huh?

Student:[Inaudible] can I skip the top [inaudible]?

Instructor (Mehran Sahami):In that particular case with Z, you can skip the cast because if it says, hey if you have Y here and you want to assign it to an INT, the only way it can do that is to cast it automatically to be an INT, so it will do it for you automatically.

Student: The site, the part that I want it to return it contains much information [inaudible]?

Instructor (Mehran Sahami):Yeah, and so over here if you add a double and you try to assign it to an INT it just – you can't because it doesn't have space; it doesn't store what's after the decimal so it just truncates it for you. Yeah. I just put INT to explicit about the truncation.

There's a couple other things we want to do which are shorthands for arithmetical expressions because there are some arithmetical expressions – arithmetical, is that even a word? Arithmetic expressions – you've gotta keep me honest, sometimes I just make up words. I'll tell you stores about that at some point, but not right now.

Let's say we have INT X, our old friend, after I told you all this stuff – good variable name, everything's X. So let's say we say X equals 3. Now there's a lot of things sometimes you want to do with X. Something that we want to do a lot is to add 1 to X. Right, so if we want it to add one to X, how do we do that?

Some people already know. We'll do it the long way. X equals X plus 1. That adds 1 to X and stores it back in X. Turns out, this notion of taking a variable, adding something to it and storing it back to itself is very common and so there's shorthand for doing that which is X plus equals and then some value. So if I change this, say to a 5, and I said X equals X plus 5, that's the same thing as saying X plus equals 5. It just means take that value that's over here and add it to X and store it back into X. In the case of adding 1, it's not adding 1 because often times you want to count. It's something we do so often there's even a special shorthand just for adding 1. Which, is X plus, plus. This may look a little familiar to you from the Karel world. You're like, oh, I remember I plus plus, is that the same thing? Yeah, you were adding 1 to I in Karel's world. Here you're adding 1 to X.

So there's no spaces here or anything, X plus plus just means add 1 to X. If you've ever head of language C and C plus plus, that's where C plus plus gets its name from. They sort of started with C and the made it a little bit better by doing the plus plus. Okay?

You can also do sort of subtraction in a similar way. So if you want to subtract 1 from X, that's the longhand form of doing it, you can say X minus equals 1, which is a slightly shorter hand and there's a complete shorthand for it which is X minus minus, which means subtract 1 from X and store it back in X. Okay?

There's a couple other shorthands, you can also think about multiplication. So if you want to say X equals X times 2, like you want a double 2, you could say X times equals 2. Same sort of effect, sort of a times equal. There is no super shorthand for, you know, times equals 2, it just stops there. There is also divide equal, as you can image; so X equals X divided by 2, X divide equal 2. You know, the value here is basically the value there, so if you want to multiply it by 5, you would just put a 5 there, same kind of thing. But these little shorthands – you can use these with integers, you can use them with doubles, they work for both, you're fine. Okay?

So besides the shorthands, these are just some little syntactic things you should know, right. Hopefully all these things are fairly clear and they make sense.

There's also a notion, kind of time for a new concept; the notion of a constant. I'll write it in big letters. And the idea behind the constant is it differentiates it for a variable and that it does not vary, right. Variables vary, constants remain the same. And so you might wonder why do I care about what is a constant, right? Like isn't the value like 3 here a constant? Yeah but some constants are meaningful and sometimes you want to give them a name. So for example, PI. Right? PI is some constant that has a value like – let's just say for right now, 3.14. So in your program, you could have some double called a PI, all upper case that's the conventional use for constant, separate words have underscores to differentiate them, equals 3.4. And you might say, oh that's great, now anywhere in my program, like if I'm computing like, you know, the area of a circle and it's PI R squared, right, I could just say something like, take the radius, multiply it by PI, multiple it by PI again and this would be, you know, my – assuming I had R somewhere and I have area, I could compute it like that. And oh, isn't that so good. My program's readable now, it's so good, I know – yeah, that's all wrong. .

No one says anything, like sometimes I've gotta throw things at you to be like, is anyone paying attention? Yeah, we just squared PI. PI R squared, remember that? Yeah, there's R times R times PI. I can just do this for computing the area and everything's just fine, right? And it makes it a little bit more readable. But the problem is PI is a variable here right? Nothing prevents someone from coming along and saying, yeah, you know what, I don't really like your PI. I like my own PI and my PI is 6 because it's just bigger. Right? Then you go compute the area and your area got a whole lot bigger and you're like, why's it so much bigger? I don't understand. Right? Because someone changed your PI. Get your hands off my PI, all right.

The way you want to do that is, you want to force PI to not change. You want to tell the machine this is a constant. I will give you a value, it will remain unchanged. And the way we do that just happens to have some sort of bulky syntax associated with it, but I'll show you what that looks like. Okay.

So the way we say that – sort of follow along. First, we say private because we want to keep our constants to ourselves. So what private means is this constant I'm going to define, I'm going to define the constant in a class. It's not defined in a – you can define one in a particular method, but generally we define our constants in our entire class – I'll so you where they go in just a second – and we say private because we don't want anyone to be able to see our constants outside of our class. So first, we say private. Then we say static, which is just kind of a funky word which means that this constant sort of lives for the class and there's only one of these for the WHILE class. So it's not like – you, if you're an object for the class, you don't have your own version of PI and some other object has their own version of PI. The way you can think it is, remember I told you a couple days ago you're all objects? You're all objects and we have one constant, which is the amount of mass in the world, or universe, right. And so you don't have a separate mass and I'm like, hey, I have my different amount of mass for the universe and it's different from your amount of mass for the universe. We share the same mass for the universe. So all of humanity, or you could think of all objects that exists share the same

mass for the universe. That's what static means. Everything of a class shares the same thing. You don't have – each object doesn't have its own version. Okay?

Then, we say final, which means, get your hands off my PI. I'm going to give you a value and it's the final value. No one else is going to be able to give you a value. No one else is going to be able to give you a value to this because the value I give you know is the final value.

Then, we specify the type. Right? So we'd have DOUBLE for PI and then if we can zoom out a little bit this is going to be a two-board extravaganza because it's just that big. Public, or private, static, final, double, name of the constant PI and then its value 3.14. So all of that means you're going to get something who's of type double, it's value's not going to change after I give its initial value, there's only one of these for the entire class and it only lives inside the class, but we just put all those things there just so you know. Okay?

Now the important thing about doing this is you want to give your constants good names and – well, good names is one reason, right, so when someone reads your program they can say oh, he's multiplying by PI, that's a good time. The other thing is that it allows the program to change easily. Right? So I could have an area computed somewhere; I could have could have circumference computed somewhere else that also uses PI. I could go somewhere else and do something somewhere else funky with PI and then someone comes along and like some physicist comes alone and sees my program and says, oh, you must be an engineer because you're like 3.14. Like in physics, if we say 3.14, like particles miss each other when we accelerate them to the speed of light. You're like, well, don't accelerate particles to the speed of light. But they come along and they say, no, no, no, no. PI is 3.14159622, there's always a 2 before the 6. I'll stop there.

I have a friend who actually has PI memorized the 400 digits. I was like, that was just a tremendous waste of your time. . But that's okay, I'm sure it's probably useful somewhere. If I really need to know, I'll go ask him or I'll write a program to compute it. So we can get more exact values of PI, right.

The other thing that kind of comes up is that now, if I use PI all throughout my program, I'll I need to do is change it in that one place and I get more precision all throughout my program. So that's one thing in terms of nice software engineering, right? It changes everywhere so I don't need to go through and hunt down, oh, where did I put 3.14 in my program and I have no inconsistencies because if everyone's referring to that same constant value, they're all referring to the same thing.

The other thing you want to keep in mind, in terms of good names is think about names that are readable. No joke, I actually worked on a program for a large corporation once that will remain nameless, where this constant was in the program. I don't know, he had some extra stuff there and I looked at it and I was like, yeah, that's very descriptive. 72 equals 72 and I was like, I was really tempted, I just wanted to come along and be like, la, la, la, la,la. Like, does the universe end? Like what would happen? I have no idea where

this is getting used. And it turned out, what this – anyone want to venture a guess what this was used for? It's the number of pixels in an inch on most monitor resolutions.

That's what it was. It was like this gooey program, or user-interface program and that's what this was because someone somewhere in some class had told them, hey, if you're going to have any values in your program other than 0 and 1, which are very common and occasionally some other value will come up, like you want to divide by 2, it doesn't make sense to say this is the thing I use to divide by 2 and its value is 2, right. There's some values that come in your program that it's fine to just have the actual number in there. But other than these things, most numbers that appear in a program actually have some meaning and you want to have them assigned to a constant. So this person probably heard some rule somewhere that says oh, you should have constant for every value in your program and they didn't know what to name it so they just called it 72. That's bad. You want to give it real names for what it actually stands for.

All right, so any questions about any of this stuff? Uh huh?

Student: If the rationale like behind name constant and using that whole privacy thing that so nobody can change it but like, to change, if you have like just double PI equals 3.14 you'd still have to go to like the guts of the program. So how's it any better?

Instructor (Mehran Sahami):Well, the difference between this is that programmatically no one changed it. Right? So if I just said double PI equals 3.14, somewhere in my program someone could have come along and written this line and I can't stop them. Right? So I – it's bad software engineering because I'm not preventing them from doing this. This actually prevents them from doing this, so if they go into the program and say PI equals 5, when they try to compile, the compiler's going to uh huh, you told me PI was final now someone's trying to change it. Bad times. So that's the thing. You always want to force other people to also have good practice by doing stuff like this. Uh huh, question?

Student: Is there a library that we can import that has standard constants?

Instructor (Mehran Sahami):There is a math library but for the purpose of this class – if you're really interested in math kind of stuff, come talk to me, but for the purpose of this class, there's some basic functions that are in the book and so you can use those, kind of basic mathy functions but if you are really interested in other kinds of stuff we can talk about it separately.

I need to push on a little bit so I'm just going to hold questions for just one second because we need to go to an entirely different topic.

And the entirely different topic we're going to go to is something called Booleans, and there was a time when I told you about this type called Booleans, which is just the type for variables whose value is true or false. So we can say BOOLEAN P and P will take the

values either true or false. Anyone know where the word Boolean comes from? Yeah, George Boole. So know it and learn it.

I should ask if anyone knows how George Boole died? I asked this once, I'm going to say it, he got sets of false. I thought that was kind of funny. He actually died of pneumonia and the reason why he died of pneumonia – this is a true story – was he was out walking one time from his house to the college where he was a professor and he was in the rain and he got wet and he caught pneumonia and so when he came home, his wife actually, who happened to be the niece of the person who Mount Everest is named for – just a totally random side note – history's just fun – so it turns out she had this belief that the way you get over a particular illness is you experience the same conditions under which you got the illness. So she's like, oh you were in freezing rain, that's why you got a cold, so while he was lying in bed, she would get buckets of ice water and just douse him and he died. So medicine's come a long way; true story.

All right, so returning to our friend, the Boolean; Boolean has the value of true-false which means what we want to do on the – when we assign something a – assign a Boolean value to some Boolean variable, we need to figure out some expression that evaluates to true or false. Right? It doesn't evaluate to 5 or 10 or something, it evaluates to true or false. Like, 3 greater than 5 is a true-false expression, right. You can look at this and say Marilyn, 3's not greater than 5, yeah, and the value of this is false and that's what gets assigned to P.

So there's certain operations we can use called relational operations who, when we evaluate them on two values, give us back something that its value extensively is true or false. So let me show you some examples of that by going to the overhead. Let's see my little – oh, the remote mouse, I love the remote mouse.

So Boolean expressions is just a test, basically, for condition and it evaluates to true or false. Right? You can actually use the words true and false in your program and say P equals true, for example, those are actually parts of language.

So there's a bunch of different value comparisons you can do, like equal equal, two equals in a row, no space, is equals. It means like, does A equal equal B. It is not a single equal. What does single equal mean?

Student: Assignment.

Instructor (Mehran Sahami):Assignment, right? When you say X equals 5, that's an assignment. When you say X equal equal Y, that is a Boolean expression that evaluates the true if X and Y are the same. Not equal is exclamation point equal, or sometimes, now that you're all programmers, after Karel, this is called a bang. Because it's kind of like loud. It's like bang, exclamation point. Right? Caught your attention, hopefully you weren't sleeping. If you were, you're not. Bang equal means not equal and if you're familiar with some other language where you say like less than or greater than means no equal, nuh, uh, not in Java baby, doesn't happen.

All right, so then there's greater than, less than, greater than or equal to and less than or equal to. And the order of the symbols for these two actually makes a difference. But as you can imagine, those are some common things you might want to consider for two values to see whether or not, or two variables to see whether or not, for example, they have equal value. Okay?

Now, Boolean comparisons, order of precedents, it turns out there's a bunch of operations you can do on Booleans. There's an operation which is the bang, or the exclamation point, which means not. That's the logical not. So if you're a logical person, this is familiar to you. If you're not a logical person, I'll tell you what it means. If you put not in front of some Boolean expression, or some Boolean variable like P, if P is true then NOT P is false and vice versa. It just means give me the inverse of the logical value that P has or whatever that expression would have been. Okay?

There's AND, which is ampersand, ampersand. Get to know where your ampersand is on your keys on your keyboard are. This is logical and. What that means, if I have two Boolean expressions, P and Q, they can either be variables or expressions. P and Q is only true if both of the sub-expressions P and Q are both true. If either one is false or both are false, it is false. Okay?

And last but not least, there's OR. If you've never used the vertical bar keys on your keyboard, find them. You're like, I may have never used them in my life. You will probably use them for the first time in this class. Or, if I have two expressions P or Q, it's true if P or Q or both of them are true. So if either P or Q or both of them are true, then what you get when you take the OR of them is also true. These are in order of precedents so you're NOT's get evaluated first, then your ANDs, then your ORs. Much in the same way the parentheses get evaluated first, then multiplication, division, then addition and subtraction; same kind of thing.

So we can look at an expression, for example, Boolean P equals something like this. Notice I've fully parenthesized it to make it a little bit more clear. X is not equal to 1 or X is not equal to 2, so this might be a common thing you might think you want to do if you want to say, well I want to figure out if X is not equal to 1 and X is not equal to 2. So if it's not equal to 1 or it's not equal to two, isn't that the right thing? And in fact, this is not the right thing. This is buggy. P will always be true in this case. Why? Because if P's equal to 1, well the not equal 1 part is false but it's not equal to 2 and that's become true, so yeah, false or true and it becomes true and vice versa if X equals 2.

So what you really want to say is, X is not equal to 1 and X is not equal to 2. That will make sure if you want to make sure that X is not equal to 1 and X is not equal to 2. That's how you would write it although English people tend to say X is not equal to 1 or X is not equal to 2 and so it's confusing if you try to write that out in logic. This is what you really want in that case. This is buggy. Okay?

Hopefully this gave you an example of a Boolean expression. Is there any questions about this?

All righty, then let's move on.

Short circuit – question?

Student: Where do you get the [inaudible]?

Instructor (Mehran Sahami):Just fine it on your keyboard, it depends on your keyboard, but it's a vertical bar, it's somewhere on your keyboard. It will be and if it's not on your keyboard, throw your laptop away and get a new one and get angry at someone, whoever sold it to you. It's gotta be on there. Trust me.

Short-circuit evaluation. We actually stop evaluating Boolean expressions as soon as we know the answer. What does that mean? Okay, so let's consider something like this. P, which is a Boolean, equals 5 is greater than 3 or 4 is less than 2. Well you know what, 5 is greater than 3 so as soon as we evaluate that part, that's true. True or anything is true, right? It doesn't matter what the value of this guy is because we already got a true and true or if anything is going to be true. So in fact, Java does an optimization and this second test is not an evaluated at all. That's why it's called short-circuit evaluation. As soon as we know the answer, we short circuit out and we don't evaluate the rest of it. You might say, oh, yeah Marilyn, that's interesting. Why should I care? And the reason why you care is there are sometimes making use of this actually makes a difference. So here's a useful example. That's a not-so useful example. Where's a useful example?

Let's say you want to avoid dividing by 0 because dividing by 0 is an error. You can say, well P is equal to; X is not equal to zero; if X is equal to 0, this is false. False and anything is false so you want to evaluate the second part over here and you'll never actually divide by the 0. If X is not equal to 0, this is true. So to know the value of true and something else, it needs t actually come over here and evaluate the second part because it got a true for the first part it needs to know that this is true. But if it evaluated the first part and got past it, you know that X is not equal to 0 so you won't divided it by 0. Okay? So these kinds of little tricks are actually used sometimes in code and it's called – sometimes it's called a guard to prevent this from happening. But that's why short-circuit evaluation is something you should actually know about because you can actually use it for usefully things. Okay?

Any questions about that? Any questions about Booleans or the operations or that kind of happy news? Uh huh?

Student: What happens if you divide by 0?

Instructor (Mehran Sahami):If you divide by 0 you – usually you'll get – well, you will get an error, you'll get an exception but we won't talk about exceptions just think of it as an error.

So let's actually move on. We're just going to cruise through tons of stuff today because life is just good.

So it's time to talk about statements. Okay? Just like in Karel when you had statements like, move and turn left and all that happy news, now we're gonna do all that happy stuff in Java as well. Okay?

So one thing we first need to know about is this thing called a statement block, or a compound statement. They are referred to as the same thing. Usually I say block, the book likes to say compound statement, it's just a set of statements enclosed in braces. So you have some opening brace, a bunch of statements and then a closing brace. This is what we would refer to as a block. Why do we care about blocks? The reason why we care about blocks is, remember when we declare variables, a variable has something called a scope. All a scope is – the way you can think about it, it's not a mouthwash – the way you can think about scope is that it's the lifetime of the variable because variables come into the world when they're declared and one thing I didn't tell you is that at some point variables die. It's very sad. But where a variable dies and lives is the block in which it is declared. So when we say X equals 5 up here, X is alive until we get to the end of the block in which X was declared. Which means when we get to a closing brace, that X goes away. That's it's scope, or it's lifetime. Okay?

If we declared X outside of these, the scope, it would not die when it came here. But if X was declared out here, this scope is some which is referred to as an interscope. X is still alive inside the interscope and it does not die when we get it out. It just dies when we get to the closing brace of the scope in which X was declared. Uh huh?

Student: Can you declare a final end [inaudible]?

Instructor (Mehran Sahami):It – well it doesn't die because of where we declare it. So if we declare it inside a method, it could die when we get to the end of the method. We're going to declare it in the class so it's never actually going to be at the end of a method so it won't die, it will be alive for the whole time that class is alive. Uh huh?

Student: When you declare [inaudible]

Instructor (Mehran Sahami):Uh huh.

Student: Will it actually pull out of the sub block?

Instructor (Mehran Sahami):Yeah, as long as the variable's declared outside of the sub block, that's fine. And well – this is kind of technical point and sort of, in most intensive purposes you don't need to really worry about the nuances here but it's just something important to remember and we'll look at some examples as we go along.

So other statements you should know about; the IF statement. You're like, oh, IF, it's just like Karel, and yeah, it's just like Karel, right? We say if some condition, and that condition is something that evaluates to true or false. It can be a Boolean variable or it can be some Boolean expression, anything that evaluates to true or false. So you can use, and, or's, not's, all that happy news, now in a condition and then you have some opening

brace and close brace just like Karel and if the condition is true the statements get executed. So we want to check if something is even, we can say if that number, when it's divided by 2, if it's remainder equal the 0? Its remainder is equal to 0 and it's divided by 2, it's even so we're going to write out NUM is even.

Now notice here, I don't have the braces and it turns out there's this one special case that says if you have an IF statement, and it actually applies to a couple of other things, and it's only – the body is only one statement, you don't have to have it inside braces. If it's more than one statement, like this, then you have to have the braces. So this is just a special case that exists. But I like to think of use braces with IF; you have to if there is more than one statement inside the braces. But I like to think of something that I refer to as the orthodontist rule. Right? What do the orthodontist say when you go to an orthodontist? You need braces. Right? It doesn't matter if your teeth are straight or you only have one statement or whatever, you need braces. It's always a good idea to use braces, which defines a block, right, so this now is a block, even if there's only one statement in the IF. There's only one special case that I'll show you in just a second, but in general always use braces. Right, think orthodontist. Always use braces, it's just a good idea. Okay? Any questions about IF? Hopefully not because you hopefully were using them a lot with Karel.

We have the IF ELSE's just like Karel. Same kind of thing going on here. We have some condition, either we do the statements or we do the ELSEs part if the condition is false. So if we – the remainder of NUM divided by 2 is 0, we say NUM is even; or if the remainder is 0 and if it's not then we know it's odd and we write out NUM is ODD and so are you. All right? So not a whole lot of excitement going on there. You saw this in Karel; same thing exists in Java, now it's just a little bit different because you're not checking a front that's clear anymore, all right. We're not in Kansas anymore. You're actually doing some Boolean expression for that condition.

Let me just give you a couple more and then we'll take some questions. There's something known as the cascading IF. This is kind of funky. A cascading IF says, what I want is I'm going to check for at least just one condition to be true. I'm going to start off with something that looks like a regular IF. This is like high school grading, right? If your score is greater than 90, then print out A. Otherwise ELSE and what are we going to do with the ELSE? We're going to do another IF. ELSE IF your score is greater than 80, print out B; ELSE IF your score is greater than 70, print out C and otherwise it's kind of – bad times. All right?

So if you think about this, only one of these IFs gets executed. Why does that? Because if we come up there and we say score's greater than 90, we do this. We skip the ELSE part. Well, what's the ELSE part. The ELSE part is everything else. Why is it everything else? Because it's an ELSE and then it's one statement, right. There's no brace here. Remember I told you, if you have no braces it's one statement. Well what is that one statement? It's an IF followed by an ELSE and that ELSE has one statement contained within it so it's part of this bigger IF and that has an ELSE which is contained as part of that IF. So all of this stuff, in some sense, all cascades down. So if I do the IF part up

there and I don't do the ELSE, I skip over this whole thing. If I don't do the IF part, then I come and I evaluate the ELSE which means I evaluate this first condition and it's true then I print out B and I skip the rest. If it's false, then I evaluate the next condition. So this is just sort of the idiomatic form – or a pattern form that you see, it's called a cascaded IF for when you want to check a bunch of conditions in sequence and the first one's that true does its IF part and the it skips the remainder of it. It's so useful for things like grades where you're like, is it greater than 90? No. Is it greater than 80? No. Is it greater than 70? As soon as I find one I'm done. Otherwise, I have some catch all at the end if it says things are bad. All right?

So that's the cascading IF. There's something called a SWITCH statement. I'll touch on this very briefly. The SWITCH statement you can get it in excoriating detail in the book. It's just kind of nice for you to have. You can actually do anything with the SWITCH statement – anything you could do with a SWITCH statement you could also do with IF statements so it's just kind of a nicety.

The way the SWITCH statement works is, it says IF you have some integer value, so here we're going to ask user for day of the week, right, and the first day of the week in Day 0, which is Sunday. So they enter some number hopefully between 0 and 6 and we say depending on what number they entered – so that thing that goes inside the parens for a SWITCH statement has to be an integer value. Okay? Has to be an integer value. There're some other things later on in the class where we look at the boil down to integer values but it cannot be a double, interestingly enough. And we specify the syntax goes like this, case, then what value matches that case. So this is Case 0. And you have sequence of statement until you hit something called a break. So notice, this is funky because the only braces here are this brace down here and that brace up there which encloses this whole SWITCH statement. There's no braces in the middle. The way it knows where to stop inside a SWITCH, so if someone enters 0 it says let me find Case 0. Oh, here's Case 0. I start executing from this line and keep going until I hit a break. When I hit a break, I jump out here to the end of the closing brace. So if the user types in, let's say 6, it says it is Case 0, no, so it skips that. Is it Case 6? Yeah, so it prints out Saturday and then it breaks which means it skips to default. If they enter any other value other than 0 and 6 it says does it match 0,no, so it skips that; does it match 6, no, so it skips that; it comes to the default with is kind of the be-all if it doesn't match any other cases and it rights out hey, it's a weekday. Okay?

So you can do this with an IF, right? But this is just a nice way – there's a lot of times – the idea is to think like you have some mechanical switch and you've got to pick one of the options and that's why it's called a SWITCH statement. Uh huh?

Student: Do you have to do in that order, or...

Instructor (Mehran Sahami):No, your order can be anything that it wants. You can have – you know, the things can even be out of order. So they don't even need to be sorted. Was there a question in the back? Uh huh?

Student:[Inaudible].

Instructor (Mehran Sahami):Pardon?

Student: Can you break out of the method?

Instructor (Mehran Sahami):You can't break out of the method all it does is break out of the switch, so it just takes you – you start executing it at that bottom brace again. You don't actually leave the method that you're in.

Student: But if you have like a…

Instructor (Mehran Sahami):Let's take it off line. Let's take it off line. Uh huh? Was there another question over here? All right. Let's press on then.

So in terms of four loops, you've also seen before hopefully in the context of Karel, so we're just zooming through. The way the four loops looks like, here's the general form of the four loop. It has four, just like you saw before in Karel and now it gets a little bit more complicated but not much.

We have something called the INIT. What the INIT is, is when you come to a four loop, it does whatever statements are in that INIT once and only once at the beginning of the loop. Okay?

Then you have something called the condition. The condition is checked every time before we go through the loop. So the first time we come to a four loop, we do the INIT, we check the condition. If the condition, which is a Boolean or it can be any Boolean expression and it's true, we execute what's in the loop. If it's not true, we skip over the loop and just execute immediately after that closing brace. Okay? So similar with what you may have done with Karel, for example, except in Karel you probably executed some number of time. And the step – so we execute the statements inside the loop only if the condition is true. And the step is done every time after the loop. Okay? So what does that mean? This is kind of a whole bunch of stuff to keep in mind. Let me just give you a simple example.

Here's a loop that you may have written sort of in the Karel world except now we're going to print out the value VI instead of making Karel do something, right? So this was a form of syntax used for Karel to do something five times. Remember that? If you remember that nod your head. Yeah. Hopefully.

So what is this really doing? Right now we can pull back the covers. Well the INIT was to say create some new variable named I and set its initial value to be 0. The test you were doing was to say as long as I was less than 5 execute what's inside the loop. And the step says every time you go through this loop I plus, plus. Add 1 to I and store it back to I. So when we execute this, what happens? The first time it comes to execute it says I equals 0, it checks to see if 0's less than 5. It is and it says okay, I execute the loop so it

prints out a 0, adds 1 to I and goes back and checks the test. So you can think of the I plus plus as happening after it's executed the statements in the loop, but before it does the test again. So now it adds 1 to I, I has the Value 1, it's still less than 5, it prints it out, it adds 1 to I, it's now 2; less than 5, it prints it out, adds 1 again, it's 3; it prints it out, adds 1 again, it's 4. Now after it prints out 4, it adds 1 to I again. Now I has the Value 5. It checks the test again. Is 5 less than 5? No. Five is not strictly less than 5 so it's done. Okay, so you get the values from 0 to 4 but notice it still went through the loop five times. It did the body; we just started counting at 0.

And as computer scientists, that's a very common thing to do is, really when you want to count something 10 times, you count from 0 to 9 instead of from 1 to 10. That's just what we do because zero's a real number and we love it, we care about it. Uh huh?

Student: Is the scope of I is the four loop after we're done?

Instructor (Mehran Sahami):Good question. Exactly, the scope of I is the four loop. So when that four loop – oh that was real bad – when this four loop is done, I goes away. Okay? So the lifetime of I is until we get that next closing brace in the scope of which I's the clairton. Sort of, we think of the scope of I's being the four loop so when we get to the end of the, when the four loop is finished executing and we sort of move on down here, I is gone away.

But we can create another one in some other loop, that's fine.

There's other funkier things we can do with the four loop rather than just counting from 0 up to some value. We can actually start with a value like 6 and count down. So we can say I's initial value is 6, as long as it's greater than 0, subtract 2. So we're going to use sort of that minus-equal-to funkiness. And so when we start off, I starts with the Value 6. Six is greater than 0 so it writes out 6, subtracts 2, 4 is great than zero, it writes it out, subtracts 2 again, 2 is great than zero so it writes it out again, and it subtracts 2 form I and so now I have the Value 0. Zero is not greater than zero so, it's done. Okay? So that 0 is not displayed because after we do the step, we always check the test again before we execute the loop one more time. Okay? Any questions about that?

So let's do the WHILE loop super quickly because you've already seen the WHILE loop in Karel's world. Same kind of thing. While condition, if that condition is true we execute some statements. The condition's checked before every iteration of the loop, just like in Karel. Right? So that's why we did all this stuff in Karel because it carries all over directly in Java and we execute the statements only if the condition's true.

Let me show you an example. X starts with a Value 15 while X is greater than 1, every time through we're going to divide X by 2 and write out the value. So first time X has 15, 15 is greater than 1, we divide by 2, we do integer division so we get 7, we write that out and go back up there. Seven's greater than 1, we get 3, we do it again, we get 1, when we do 1 divided equal 2 what do we get?

Student: Zero.

Instructor (Mehran Sahami):Zero, which is not greater than 1 so we're done. Okay? Nope, let me go back. Nope, let me just end the show. All right, so any questions about the WHILE loops and we'll review our friend the loop and a half next time. All right. So I will see you next week.

[End of Audio]

Duration 50 minutes