Programming Methodology-Lecture08

Instructor (Mehran Sahami): Alrighty. Welcome back to yet another day of CS106a. Couple quick announcements before we start – so first announcement, there is one handout which I'm not sure if it's here yet, but it will be here momentarily if it's not here now. You can pick it up on the way out. I think Ben might have just been delayed on the way in. So there is one handout; hopefully, you can pick it up on the way out if you didn't see it back there now. It's already posted online as well. So if you don't get it in class, you can get it online, or you can get it – there's the Handout Hangout as we like to refer to it, which is a bunch of file folders on the first floor of Gates where there's hardcopies of all the handouts that get left over from class.

So another quick point, it's still a little early to talk about plus pluses because some of the programs that you've written are, sort of, simple enough that it's difficult to, sort of, way overshoot what we're expecting, which is what we want for the plus plus, but I just want to say a little note about plus pluses just because pretty soon that's, hopefully, what you're gonna be going for. If you think you're gonna add extensions to an assignment beyond the basic functionality, what we ask is that you do two versions.

You do one version that is your nice, clean, well-decomposed, commented version that is everything that you would need to do to do the assignment well done just as the specification suggests, and then do another – create another file, or you can just copy this file that you started with and start working on that. You might need to give it a slightly different name for the class like, "My extended version of this," or whatever, and that's where you should add all your extensions.

And the reason we do that is because sometimes in the past when people have added extensions to their assignments, they've inadvertently made things, in terms of software engineering, really ugly that they didn't mean to, or they ended up actually breaking some piece of the basic functionality because they were trying to do something way more complicated. And so that's why we ask for two versions if you're gonna go for the plus plus, so we know you were able to do all the basic stuff and do it cleanly, and then if you want to add extensions, we can look at those extensions, and if the extension happen to break something along the way, you won't get penalized because we'll have your basic version as well. So if you're going for the plus plus, two versions of the program that you want to submit for the plus plus, the basic version that has all the functionality, everything nice and clean, and then whatever you're gonna do for the extension.

Alrighty. So with that said, time to get into the real meat of things, and we're gonna do a little bit of review of what we talked about last time. Remember last time we talked about our friend, the method, and one of the things we said about method was, like, a CD player was like a method, and part of the idea was it had this generalization capability, that it had some parameter of the CD you put in, and out came music, and the music that came out depended on the CD that you actually put in.

Now part of the reason, besides just getting generality, is this notion of information hiding. So when we think about having – this is, kind of, the software engineering principle for the day – when we think about having methods, part of the reason to have a method, and to give it a good name, and to generalize it with parameters is because you don't want the person who's using that method to have to worry about all of the internals of that method.

Here is a simple example. Most of you have seen the readInt function or the readInt method, and it reads information from the user, and we did a little example in class where the user typed in something that wasn't an integer, and it said, "Illegal format," and it asked for it again. It does all of that for you. Do you need to know any of the internals of readInts? No. You can just use it because it's hiding information from you. It's hiding all of the complexity of being able to get some data from the user, do error checking on that data, give the data back to you.

And so, as a result, you can write your programs where you just have this one line in there. You get data from the user, and you don't care what all is going on in order to get that data from the user. That's the same kind of mindset you should be thinking about when you write your methods. Your methods should basically think about solving one problem each, and you want to think about that problem, sort of, being something general that makes sense to hide that information from the user. So when the user called your method, for example, there's a nice comment that explains what it does, that explains what the parameters are, so they don't need to go into the code of you method to actually figure out what it does, okay?

Let me give you a simple, real life example, okay? There is this thing that most of you are probably familiar with that looks, oh, like this, and has two slots in it up here, and a little thing that looks like a plunger over here, and it's something we happen to affectionately – anyone that can identify what this object is?

Student: A toaster.

Instructor (Mehran Sahami): Toaster, rock on. That's a social. A toaster is essentially doing information hiding, right? And you're like, "No, really? Like, my – " yeah, your toaster's alive. It's sitting on the counter, and it is hiding information from you, and it's information you don't care about, right? It's got a parameter. The parameter you stick in is bread. You can stick in lots of different bread, but the type that you stick in is bread, right? There can be some specialization of that type like rye bread, or wheat bread, or whole bread, or whatever it is, but you gotta stick in bread. You call the method by sticking the plunger, and some amount of processing later, what you get out of this thing is toast.

You don't care how the toast is produced. One way the toast is produced is there's coils in here that head up, that, sort of, bake the bread, and bread comes out, that's great. Another way the toast is produced is there's Keebler Elves inside here, and they're like, "Hey, we got bread." And they eat the bread, and they excrement toast, right? And that's just what comes out. Elf excrement, but you don't care because it's toast, right? You're like, "That's great. Something's being hidden from me." And in the case of elves, you're glad it's being hidden from you, but you get back what you cared about, right?

And the beauty of this is that this can be implemented in any way possible. Like, someday down the line we'll figure out fusion, right? And your toaster will toast bread in a fraction of a second because there'll be a little fusion reaction in there. You'll put in the bread; you'll press a button, and immediately toast will come out. It'll be charred at around two million degrees, but you'll get out toast, and the stuff in here, you still don't care about.

So information hiding, the other thing it gives you is the ability to change the implementation here, and the person who is using that thing as a black box never has to care about if you found some more efficient way to do it or however you implemented it, okay? And that's, kind of, the key principle that you want to think about. This is actually so powerfully that about 15 years ago I was teaching this class, and I had a student in there – totally true story – and I told him, "Hey – " his name was Afra, by the way. Afra, if you're watching this, this is all about you. I told him methods – at that time I was calling them functions – but methods are toasters, and I ran into him, like, three years ago, so it's been 12 years, and the first thing he said, "Hi, Mehran." So he did say hi, and then the next thing he said to me, no joke, was "A function is a toaster." And I was like, "Oh, I love you." And we hugged, and it was, oh, plus plus, and now he's a professor himself, and I hope he's telling his students about the dreaded toaster, but that's what it is. That's the thing you should just think about, information hiding in the little toaster you have, okay?

Now, along with that, one of the things we also talked about last time is functions returning values, and I told you sometimes a function doesn't actually – or method returning values, and sometimes a method doesn't actually have to return a value, and that's when we have a private void as the return type, right, or void is the return type; private's just the qualifier, and we might call this, for example, something like intro. If we want to just have it write out some, you know, hello to the user, or it explains how the program works, or whatever, and then we have the end of the method.

There is not necessarily a return anywhere inside this method, just like you did with Carol. When you were writing methods in Carol, you were writing methods that didn't return anything, and you never had to worry about return, and that's perfectly fine. In methods that don't return anything you never have to say return, and that's great. If you want to say return somewhere, you can say return and then put a semicolon because there's nothing you're actually returning, so there's no expression over here for what you're returning, but, generally, we don't like to return in the middle of a method. So these are usually out. Usually, we just don't have a return in there, and we expect it to execute all the way through to get to the bottom, that's how it's done, all right?

So when we talk about calling these methods and what's going on parameter passing, a little bit of complexity comes in, and today we're gonna go through that complexity in excruciating detail. Hopefully, it won't be too excruciating, but enough detail that, hopefully, you will know how it works. As we talked about last time, right, you can have some method over here Run. So we'll say Private, Void, Run, and inside here we might have some Integer I that we do something with, and over here we might have some other method, Private – I'll make this one Void too, and we'll call this, just for lack of a better name, Word Cal, and over here Cal might also have in it some other Integer I, and it might do something with it, and these two Integer I's are not the same because the lifetime of a variable exists only in the method in which it is defined for these kind of variables, and I'll show you some different kinds of variables in just a second, but a variable like this that is declared inside of a method is what we refer to as a local variable.

It's, kind of, a variable that, kind of, hangs out in its little hood, and its hood is the method that it's declared in, so it's local to its hood. It's like, yeah, I'm just hanging out with the boys and girls in run, and that's where I live, and so if Run happens to call Cal somewhere inside here, well, I don't live in Cal, so I'm not going over to Cal, right? And Cal comes, and it gets called, and there's some other I here, and it doesn't know about this I. It's, like, hanging out in a different hood. Like, this is Manhattan, and there's an I over here, and this is Brooklyn, and there is another I over there, and they just don't need to know about each other, and, as a matter of fact, they don't know about each other because if they did, life would be bad. Like, they'd get in fights, knives would come out, the whole deal.

So these two things are actually separate. Now, what does that mean in terms of the mechanics of when we call functions parameters? For example, what would happen if Cal actually expected some parameter over here, let's call it N, and I passed into it I; what does that mean? Is that gonna conflict with this I over here; what's actually going on? So to understand what's going on, let's actually go through an example in excruciating detail of what's going on inside the computer, and you can see exactly what's going on.

So let's come to the slides, and here is a program that you saw last time. This is just our little factorial program that we computed last time that you saw an example of different methods, and we have the run method that calls this factorial method a bunch of times, up to four times, right? Because maximum happens to be four, and inside here we have an I, and over here we have an I, and here we have this things called Result, and we're like, "Hey, what's going on with all this?"

Well, here let's trace it through. So let's actually start with Run because that's where the computer starts executing things, right? So it comes to Run, and it says, okay, inside Run I have this four loop, and it's got an I in there. So in the hood or in the method for Run, we have our own little I, and when we start off, we set I equal to zero. We check to make sure that's less than Max Num, which is four, which it is. So we say, oh, we enter the loop, and now we come to execute this line. Well, to execute this line, we need to make a method call over here to factorial because we need to figure out what's the factorial of I in order to be able to print it out on the line, right?

So we call factorial, and we say, hey, factorial, what I'm going to give you is I. Now, I'm not actually going to give you I because I is my little friend. What I'm gonna give you is a copy of I, okay? So when factorial gets called, we get what's called a new Stack Frame, and the reason why it's called a Stack Frame is that these things get stacked on top of each other. Notice the box for back here or the box for Run, sort of, gets occluded by a new box or a new frame for factorial. That's why this is called a Stacked Frame because we frame it inside a box, and we're gonna stack these things together as functions or methods get called, okay?

So when factorial gets called, it's expecting a parameter called N. What it got passed back here was I. What we did at this point was we said what's the value of I? It's zero. We make a copy of that value and pass it over to factorial as its parameter. Factorial says, hey, my parameter's called N. What value did you give me? You gave me a zero. I'm gonna stick that in N, okay? So it's got this thing in here, N, which is totally distinct from I. It's gonna have its own I in here, and this I is in a different hood than the I in Run. So they're completely separate variables. They just happen to have the same name, but they have the same name in different context, so they're actually different. It's just like if we had someone named Mary in this room, and there is someone named Mary taking Psych 1, you're different Marys. That's just the way life is, right? You're in different context even though you have the same name, so same thing going on here.

So factorial fires up and says you gave me that zero, that's N. I start off by having result, which is a local variable. It is a variable that is only defined in my context, and I set that equal to one, and then I have my own variable I, which is only defined in my context, and I start off setting that to one, and it comes in here, and this is a real short loop because it does the test, and it says is one less than or equal to zero? No. So the body of the loop never gets executed, and it immediately returns a result, okay? So what value's gonna come back? What's gonna come back is basically a copy of this one. So the value one gets passed back, and we come back, this whole Stack Frame that we had over here goes away because factorial's now done.

Factorial says, hey, I computed my results; here you go. I'm going away. So what we say is it gets popped off the stack. It and everyone in its hood, we, sort of, say thanks factorial. You did your work. Thanks for playing. You're done. Blammo, you're gone. The only thing that remains of you is the value you returned, okay? So that one gets returned, and we say, hey, now I can actually write something out to the screen because I got a value of one back from factorial of zero, and that's what I write out – zero factorial equals one. All right?

One more time, just for good measure, so – whoa. That was bad measure. Sometimes these things just never work right. You have to hate technology. All right. So we add one to I. I now becomes one, right? I was zero over here. We add one. I becomes one. I is still less than four, so we execute this line once again. Now we're calling factorial passing at the value one. It gets a copy of that one. Where does that one go? That one goes into the parameter that it's expecting, the N. So it gets a one because that's the value we passed in. We passed a copy. We didn’t pass I itself. I is still safe and sound over there, the actual variable I. We just got a copy of its value. That’s what got passed.

So here we set results equal to one. We set I equal to one. We checked the test. The test is true because one is less than or equal to one. We multiply result by one, which is one, and we store it back in results. So that looks like it does nothing, but it really multiplied one by one, and stored it back into one. We add another one to I; I is now two. When we do the test, two is not less than or equal to N, which is one, so it says I’m done, and it returns a result. So this guy goes away again, and factorial of one is one. So that’s what we write out on the screen, okay?

And I won’t go through the process in excruciating detail again other than just to say after we add one to I, we’re gonna have to compute factorial of two and the answer that’s gonna come back is two, and we’re gonna compute factorial of three, and the answer that’s gonna come back is six, and after we write that out, I is now four which is not less than Max Num because Max Num also has the value of four, so we’re done, okay? So any questions about the mechanics of how parameters work when you pass them? When you pass them, you’re passing a copy of the value. You’re not passing the actual variable, okay?

Now, what does that actually mean? It means a couple things. First of all, what it means is that when you are in some method somewhere, you can’t refer to someone else’s variable, okay? So what do I mean by that? I mean when I’m, sort of, back here, if I back up to the last place where you can see a factorial, when I’m in factorial over here, and I’m referring to I, I better be referring to my own I; I better have my own local I. I cannot refer to any other method’s I. I don’t have visibility to any other local variable in any other method.

I can refer to variables that are local to me, like I, a result that I declare. I can also refer to parameters because N when it gets passed in gets a little box. So all parameters get a box that gets a copy of the value passed in, and all my local variables get a box which has the value of whatever I set up my local variables to have, okay? But I can’t refer to variables in any other method. That’s an important consideration. So we’ll, sort of, come back to where we were.

So this also leads to some weird behavior that may be unexpected, and this weird behavior is something we like to refer to as bad times with methods, okay? So this program’s buggy. Let me show you why. Here we have Run, and we have inside Run we have some integer X which has the value three, and we say, hey, you know what, I want to add five to X. So why don’t I pass X over to add five. X will add five to X – or Add Five will add five to X, and then I should be able to write out X equals eight, right? No, right? Because if that was right, this wouldn’t be buggy. What’s going on where this happened?

So let’s actually trace this one on the board, and I’ll show you what’s going on. We can draw out all the Stack Frames. So when Run actually gets going, right, we get a Stack

Frame for Run, so I'm just gonna draw it manually up here so you can see it as these things are happening. When run actually runs, it's got its own local variable X, and X is getting set to the value three, okay? Then it calls Add Five. What happens when it calls Add Five? We get a new Stack Frame. I'm not gonna erase Run; I'm just gonna draw down here. We get Add Five. What is Add Five expecting? It's expecting a parameter called X. So it says, oh, I have my own box for X down here. What are you giving me? I'm giving you a copy of the value passed in. What's the value passed in? Three. You get a copy of three. That's great.

Now, Add Five is over here. It has its X in its own hood that is not the same as this X, and you say, "But you passed X as a parameter." Yeah, the parameter happened to have the same name in terms of the argument that I passed in and the parameter I was expecting, but they live in different hoods. They're actually separate, and what you get is a copy even if the name's the same.

So this guy happily comes along and says, ooh, ooh, I'm gonna Add Five – all kinds of excitement going on. It adds five. This three turns into an eight, and then it says, hey, I'm done. Good times. I don't even need to return anything. Thanks for playing, and I'm like hearty handshake, Add Five, you added five to your X, and now you're done. Thank you for playing. It goes away. We come over here to Run, and now Run is going to do a print on X, okay? So what it's going to print out is three, all right? That's the place where it seems counterintuitive. What you are passing when you are passing parameters are copies. You are not passing the actual variable, all right?

So one way you can potentially fix this – I pressed the wrong button. All right. So this was bad times. Here is good times with methods. Anyone remember the show, Good Times, J.J. Walk – Good Times. All right. I'm old. How can we change it? What we need to do is think about information flow through our program. We need to think about that value that we computed in Add Five, somehow we want to pass it back out so Run can get it. So let's trace through this code, right? Run starts, it has a Stack Frame. It has X equals three in it, and what it says is I'm gonna call Add Five, and I'm gonna pass it X. So Add Five once again happily comes along and says, hey, I'm Add Five. I have a value X which you gave me was three.

It adds five to that three, and it gets the value eight, but now what did it do? It returns that eight, which means when it's going away, this puppy basically goes away, but what it says is, hey, what I'm giving you back, in my dying gasp, is this return value eight. What are you gonna do with it, Run, huh, huh? And it gets a little ornery, but Run says, hey, I'm just gonna assign it to X. Which X am I dealing with? I'm dealing with the X in my hood. So that eight that you just gave me – thanks for giving me the eight; I'll assign it to my X. And now, when I print it out, I actually have what I care about because the computation that happened inside the method that got called, the value that I cared about got returned and assigned to the place I cared about. That's what we refer to as information flow, okay? Question?

Student: Could you do the same thing – instead of having a return, just have in the print line method it concatenated with Add Five of X instead of with X?

Instructor (Mehran Sahami): I'm not sure what you're saying, but – oh, you're saying – oh, and just put it on that line, just whatever Add Five returned.

Student: Yeah.

Instructor (Mehran Sahami): Yeah, but then we wouldn't have actually changed the value of X, okay? We would have just printed it out. All right. So one way you can actually think about this, this is the way I like to remember it is when I was a wee tyke in the days of yore, my momma and daddy took me to a place called the Louvre. Anyone actually been to the Louvre? A fair number of people, it's a museum in Paris, and this thing there called the Mona Lisa, which is probably actually smaller than this square, but it happens to be a woman, very famous, who's smiling. That's my rendition of the Mona Lisa, okay?

And so when we went there, here was mommy Sahami, and here was little Mehran, and little Mehran, when he was small, there was two things. One is he had a ponytail, which I would not recommend. It was the '70s; what can I say? The other thing that little Mehran had was he had a chainsaw, and so when we went to the Louvre, what happened was I said, "Hey, mommy, what I want to do is I want to do some computation on the Mona Lisa with my chainsaw." And what mommy said was, "Oh, that's great, but what we're gonna do is we're gonna call the method for you. That method's called gift shop."

So we went down to the gift shop, and in the gift shop, somewhere along the way, they had said, "Hey, here's the copy of the Mona Lisa; make some prints of it." And so what the gift shop has down here, every time you call the gift shop method is you get a copy of the Mona Lisa. And so when happy Mehran comes along and says, "Oh, that's great. You know what? Mona Lisa looks a lot better split in half." And now what I've done is changed my copy of the Mona Lisa, and, luckily, when you go to the Louvre, you still see the original, okay? So that's the way to think about it. When you are in a method, you are getting a copy. If you somehow do something to that copy, you are not changing the original, and it's a good time for everyone involved, all right?

Student: Is this true for non-primitive data types, like –

Instructor (Mehran Sahami): We'll get into that when we get into objects, but yeah, that's a good question. Right now we're just dealing with things like Ints and Doubles, and we'll talk about other stuff when we get there. All right. So, with that said, it's time to think of eventually writing our own whole classes, right? So, so far, we've been writing, like, classes that have been, like, one program, and now we want to think about writing multiple classes, but before we actually write multiple classes, we need to have a little bit more idea about using classes, okay?

And so when you have classes, there's a notion of being a client of a class and being the implementer of a class. The client of the class is basically the user of the class. So when you use the ACM libraries, and you use readInts, you are a client of readInts. When you write your own class, and you write all the code inside that class, and say what that class is gonna do, you are the implementer. That's the difference, okay? And so it's, sort of, an important distinction to know because so far you've been doing a lot of this, and you're going to be moving to doing this pretty soon, all right?

So in order to get a little bit more comfortable with this, we're gonna actually be clients of yet another library, and that's the library to generate random numbers, okay? And so the basic idea here is what we want to do is it would be fun if there was some way the computer could give us random numbers, right, because, like, for games and stuff. Like, games get real boring if the same thing happens every time, right? Unless you're in Vegas, in which case you know you're in good times, but we won't talk about that.

So the way random numbers work on a computer, okay, is we actually call them pseudo random numbers because there is no real randomness on one of these puppies, in as much as you'd like to believe, "Hey, Mehran, like, 3:00 last night when my computer crashed, there was randomness." It wasn't random. It was some computer programmer somewhere that didn't do a good job of implementing the program.

So when we think about randomness, what we actually think of is what's referred to as pseudo random numbers because they look, for all intensive purposes to human beings, as though they're random, but really there is some process that is generating them in some deterministic way, okay? But we can think of some black box, ala, a toaster, except this one happens to be a Random Generator.

And what you do is you go to this Random Generator, and you say, hey, Random Generator – you send it a message, right, which is give me, in some sense, the next random number, and what it gives you out is some number that looks random to you, and then you say give me the next one; it gives you another one. You say give me the next one; it gives you another one, and that's just the way it works. It's a black box, and you don't care what's implemented inside of here, right? It's a toaster to you. You just ask it for random numbers, and it gives you random numbers instead of toast, okay?

So the way you actually can use one of these Random Generators as a client – this is actually a standard thing that exists in the ACM libraries. There is a library that you will import called acm.util, for utilities, .star, and when you import that, you get this thing called the Random Generator, and I'll show you how to use that in just a second, okay? The funky thing about a Random Generator is a Random Generator is a class, but rather than like most classes before when we wanted an object of that class we said, "New," and you got a new object of that class.

Random Generator's a little bit different. What you actually say is RandomGenerator., and the message you send is Get Instants, which means give me an instant – ah, I can't fit it all on this board. Let me write it on two lines. Oh, I'm just in this hailstorm of chalk.

RandomGenerator.GetInstants, there is no parameters here, instants, and what that means is give me an object of the Random Generator type, and what you're gonna assign that to is somewhere you're gonna have some private variable of type Random Generator, and you need to give it a name; we'll call it Rgen for Random Generator, equals RandomGenerator.GetInstants. So what RandomGenerator.GetInstants gives you is an object that is a Random Generator, so it's type is Random Generator. You're only going to use it inside your own class, which is why it's private, and because you're actually declaring a variable, you need to give it a name. So the name of the object is Rgen, okay? That's the thing you would have control over.

Now, the funky thing that comes up when we think about these things, all right, is that sometimes what we actually want to have is we want to use this same object inside all of our methods, right? And so you just say, "Hey, Mehran, you just told me that when I declare a variable, that variable only lives inside the method in which it is declared. So what's going on there? And if I want to have one of these objects, do I need to actually get one of these inside every one of my methods?" And the answer is no. There is a way that you can actually say I want to have some object or some variable that is shared by all of the methods in my class, okay? And that's what we refer to as an Instance Variable.

So the variables we had before were called Local Variables when they live in a particular hood. So like X over there inside Run is a Local Variable because it's local to that particular class, but – let me erase this. We can have a notion of an Instance Variable, and an Instance Variable refer to as affectionately, an Ivar. So if you're a big fan of seafood, that's probably familiar with you. Ivar for Instance Variable, it's just short for it, all right, is just basically a way of saying what I want to have is a variable that lives for everything in the object, so all methods can refer to the same variable.

Now, let's contrast the notion of an Instance Variable with a Local Variable in terms of how they're declared, how we use them, what's going on, why we use them. So here is Instance Variable over here; here is Local Variable over here, which I'll just call the Local Var. We don't call local variable Lvars, we just call them locals, and Instance Variables are Ivars. So a Local Variable, as we talked about, this is declared – it's declared in a method, okay? Instance Variables are declared in a class but not a method. So what else have you seen that's declared inside a class but not inside a method? Uh huh, Constants. It turns out those Constants are actually Instance Variables, sorry. They're Instance Variables whose value is final. So we say they do not change, but those Instance Variables are visible to everyone inside that object; all the methods can see it. So Constants were an example of Instance Variables. We just didn't tell you at the time they were Instance Variables.

So this guy is only visible in terms of its visibility, which we'll just call Vis in the method, and, as a matter of fact, it's lifetime – it only lives in the method, and when the method is done, the scope of this variable is basically gone, and it goes away. These puppies are visible in entire object. So in the class in which they are declared, when you create an object of that class, you get an Instance Variable; for every Instance Variable declared, you get one, and it's visible in the entire object. That means its lifetime is it lives as long as the object lives, which means once you call some particular method, if that method has local variables, they come into being, and when the method is done, they go away. When you create an object, its Instance Variables come into being, and they stay around until that object goes away, you're done using the object. That's how long they live.

So you might say, "When do I use one versus the other? What's the point of having these two kinds of variables?" What you want to think about is you use Local Variables when the computation that you're gonna do only lives for the lifetime of the method that you actually care about, so you're doing some local computation, okay? So think of this local variable local computation. An Instance Variable is something where you need to store some value in between method calls. You can think of this as the state of the object; what state is actually in?

So let me give you an example to, sort of, make this concrete because I think sometimes a specific example makes it a little more clear. Here's two water bottles. These are objects which were lovingly provided to me by the Stanford Computer Forum, okay? I have some class which is water bottle, and I say, "Oh, get me a new water bottle." So this is Water Bottle 1, and I say, "Get me another new water bottle." This is Water Bottle 2.

Now, sometimes there are some methods I might want to call on this water bottle like unscrew the cap, and when I unscrew the cap, it turns out if I do four rotations of the cap, so I have a four loop, and I do four rotations of the cap – one, two, three, four, five – four, sub two, the cap comes off. I needed an Index Variable to keep track of how many times I turned the cap on the bottle, but I don't care about the value of that Index Variable after the cap is off, or if I say, "Put cap back on," and I screw it on four times, I did some local computation to figure out for the four loop how many times I went through it, but I don't care about that value if I go do something else with the water bottle now.

But there are things I do care about the water bottle in between method calls, which is let's say I had some unnamed soda, and so I have Water Bottle 1, which I might have called unscrew on, and Water Bottle 2 that I called unscrew on, and so I said let me – I don't know if this is actually covered by copyright or not. Let me fill up the water bottles, oh, yay, much over here and a whole bunch over here, okay? So I call fill, the fill method, on each of the respective water bottles. Know that they have different amounts in them because the Instance Variables that I have, each object gets its own version of the Instance Variable. So I have some Instance Variable that basically tells me, let's say, a double what percentage of the bottle is full.

When I say put the cap on, that percentage should still stick around, and when I take the cap off, that percentage better still stick around. That's part of the state of this object. In between method calls, when I send other messages to this object like put the cap on or take the cap off, I still need to be able to store how much is actually in the bottle between method calls. Whereas, how many times I unscrew it is a local computation. I don't care about storing that value after I've finished unscrewing or putting it back on. That would be something I would use the Local Variable. So think of Instance Variable as the thing that state what you need to store in between calls to methods that is actually part of the state of the object that you care about, okay? Any questions about Instance Variables versus Local Variables? Uh huh?

Student: Are Instance Variables the same thing as Global Variables?

Instructor (Mehran Sahami): We're not gonna talk about Global Variables. So if you've used another language that has Global Variables, they're not the same thing, and Global Variables are just real bad style. So one way to have you not use them is to not tell you about them. All right.

So given our little example with the water bottle, one thing we can actually think of is about our little friend, random number generator. How do we use – this is an Instance Variable inside a program, okay? So if we have some program – I'll call this Simple Random which just extends the console program. This is just a program that's gonna generate random numbers. Here is my Run method over here. I'm gonna fill this in in just a second. Where I actually declare my Instance Variable is outside of the scope of any particular method, but it's inside my class just like when you declared Constants, okay?

So here I say Private Random Gen Rgen equals RandomGenerator.GetInstants. So what this does is whenever I create a new object of this class, it will initialize this variable Rgen to be getting an instance of Random Generator, and that variable, Rgen, is visible to all of the methods inside my class, and its lifetime will exist until that object goes away – until I'm actually done with the object, okay?

So there's a bunch of things I can do with a Random Generator, right? So besides the fact here that I want to store it as an Instance Variable, you're like, "Okay, that's great. Now you want to throw all this rigmarole, and you told me about Local Variables and Instance Variables just so I could store this in Instance Variable. What do I do with the Random Generator, Mehran?" All right. So here is the things you can do with a Random Generator. So once you have gotten an instance of the Random Generator – man, I hate technology.

We're going old school because sometimes in life you get a little upset, and if the things you get upset about really are just slides that you have to go through twice in a lecture, it's really not that bad. Like, in the overall scheme of things, it's, kind of, like, yeah, you know, there's other things to get upset about like global warming, but it's like, yeah, I gotta go through these slides twice. Oh, my god, I'm so upset. All right.

Random Generator, here are some of the things you can do with a Random Generator, okay? So once you have this thing called Rgen, there are some methods that you can call on, and one of them says Next Integer, right? We talked about this thing being a black box, say give me the next random number, and it gives it to you. So if you say Rgen.NextInt, you can give it a range. Say, that range being, if you want to simulate, like, rolling a dice, one in six, and what you'll get is a random number between one and six that's returned to you as an integer, or it can take a single parameter, you can call Next

Int just giving it a single value like ten, and what you'll get back is a number between zero and N minus one, or in that case, you would get a value back between zero and nine because, as computer scientists, we always start counting from zero.

Besides integers, sometimes you want doubles or real values, so a similar sort of thing. There is next double where you give it a range, and you get a value, strangely enough, between less than or equal to low but strictly less than high. It's like an open interval kind of thing, but really, for all intensive purposes, you're getting real values, you don't really care because the chance of hitting the interval is not a big deal. So it's inclusive of the low interval but exclusive of the high range of the interval, but what it gives you is a double basically in the range from low to high, and you can also call next double without giving it a range, and you'll get a value based between zero and one, okay? Just if you want because a lot of times people ask for values between zero and one.

There is also next Boolean, which just gives you – if you don't call with any parameters it's basically like flipping a coin. It'll give you back true or false 50/50, but if you care about having your coin be biased, right? There's actually some people in the world, including some professors in our statistics department who can actually flip a coin where they can flip it in such a way that they can affect the probability of how often it comes up heads and tails – pretty frickin' cool, but we won't talk about that. Next Boolean you give it a double, and so what it says is it will return true with probability P. So if you set P to be .5, it's the same thing as this case up here, but you can set it to be different if you want. P has to be between zero and one because it's probability.

Last, but not least, just to be funky there's something called Next Color. So you can, sort of, say, oh, I'm feeling adventurous, and I want to just paint the world random colors. Get me a random color. So just next color and it has no parameters, and it just gives you a random color. It's like there you go, good times. It's fun every once – if you have no fashion sense, like me, that's, like, what I do every morning when I wake up, and I put on clothes. I'm just like, "Next color," and I walk out, and my wife's like, "No, no, no, no, no, no." And she actually has a deterministic algorithm for dressing me. All right.

So here is a little example of how that actually works. So all I did was I took our program that we had before, our simple random, and filled in the Run method. What's going on when this, an object of simple random gets created or when this program is run, Rgen gets initialized to be an instance of the Random Generator, and now I can refer to it, because it's an Instance Variable, in any method, right? So Run is just some other method. I can say Int die roll is a random generated number. Give me the next Int between one and six, and then it will just print out, "You rolled," and whatever the random value was. So it's just simulating one die roll. Not a lot of excitement going on there, but this is just to show you how the syntax actually looks for generating a random number. Okay? So any question about that?

Let me show you a slightly more involved program. So we can get rid of this craziness, and here's a little program that rolls some dice. So what this does – notice now I have the imports and everything in here. I need to import acm.util.star if I actually want to use the random number generator, and now just to be a little bit more complicated, I’m gonna show you an example of having Constants and Instance Variables together because just where we put them and what the convention is in the book is just a little bit funky.

So here’s the whole class. Let me extend the window a little bit so you can see the whole class. Well, you can almost see the whole class. Let’s just assume you saw the top line up there, okay? What we have up at the top is a Constant, which is the number of sides on some dice we’re gonna roll. So I should just ask anyone here ever, like, you know, played a game that involved 20-sided dice? Anyone? You can admit it. It’s okay. I did too. 12-step program, you’ll be fine.

Num Sides, in this case, happens to be six. So we say, “Private Static Final Ints,” all the garbage that we have to save for having a Constant. It’s not garbage, it’s just lovely syntax that’s of type integer. It’s Num Sides is six. So inside this program, what it’s gonna do is simulate rolling some number of dice until you get the maximal value on all the dice. So if you’re rolling one die, it keeps rolling until it gets a six, and it tells me how many times it has to roll to get a six. If I’m rolling three dice, the maximum value’s 18 because I need three six’s. So it keeps rolling until it gets an 18 and tells me how many rolls are involved.

So what’s going on here is I first get the number of dice, the local variable, from the user by asking the user for the number of dice. They give me the number of dice, and I compute what’s the maximum roll? Well, it’s just the number of dice times the number of sides on the dice because that’s the maximum value for each die, which is the singular form of dice if you’re wondering. But that’s the maximum value. I have another local variable, Num Rolls, that I initialized to be zero, and I’m gonna have a loop and a half, basically, where I say, “While true.” It’s not an infinite loop because we’re gonna have a break inside. Int roll equals roll dice, the number of dice I want you to roll.

So over here, I have some other method down here, which I’ll just scroll down to down here so you can see the whole method, called Roll Dice. Roll Dice takes in a parameter called Num Dice. It’s getting a copy of that value. It says I’m gonna have sum totaled at zero. What I’m going to do is have a four loop that counts up from I up to the number of dice you told me to roll, and every time I’m just gonna generate a random number between one and the number of sides on the die, and add it to my total. So that simulates rolling the dice, Num Dice number of times, and when I’m done, I’ve gotten the total, and I’m gonna return that to you.

So up here, when I roll dice, what I get back from my roll is a number that is just the sum of that number of dice rolled in terms of random numbers. I say, okay, now that I’ve rolled the dice, increment my number of rolls by one. If the roll that I got was the maximum roll, then I’m done. Hey, I just rolled an 18 if I have three dice or whatever, and I break out of this loop. If I’m not yet done, I say, hey, you rolled whatever your roll was, and I go through this loop again, and roll the dice again, and write out the value again until I finally get Max Roll, and when I get Max Roll I say, hey, you rolled whatever that Max Roll was after some number of rolls. So it tells the user how many times I had to roll before I got that Max Roll, okay?

Now, one thing you might say is, “Hey, Mehran, you had this thing called Num Dice up there, right, and you have this thing called Num Dice over here, and you happen to be passing a parameter called Num Dice.” Again, we’re making copies, right? I can’t directly – if I didn’t have this value here for a parameter, I can’t directly refer to Num Dice because Num Dice is declared in another method; it lives in another hood. The only way I can refer to Num Dice if it’s passed into me as a parameter, and I give it the same name. So even if Num Dice was passed in as a parameter, if this was Int X here, I still couldn’t say Num Dice. I’d have to say X, because the only things you can refer to in a method – and this is an important rule. This is one of those ones that, sort of, like, you know, just shave it on your arm if you happen to have arm hair, and if you don’t, just write it on a piece of paper.

What you want to do inside a method – what can you refer to? You can refer to the local variables inside that method. You can refer to parameters of that method, and you can refer to Instance Variables of the object, okay? So here is the Instance Variable down here is our little random number generator, and I put it all on one line. It slightly goes – the semicolon, sort of, goes off the screen, but here is my Instance Variable. So you can refer to Local Variables, Instance Variables, and Parameters. And you might say, “But, Mehran, [inaudible] Num Sides; isn’t that a Constant?” Yeah, Constants are just a special form, basically, of Instance Variables, okay?

So the other thing you might say is, “So why did you split them up? Why are the Instance Variables down here and the Constants up there?” That’s just convention. Convention in the Java world is all the Constants get defined up at the top, even though they might be Instance Variables, and all of the Instance Variables that are actual variables, that are not final, they can actually change, are declared down at the bottom. Uh huh?

Student: [Off mic].

Instructor (Mehran Sahami): If it’s a Private Instance Variable, and you extend it, no, but we’ll get into that in a couple weeks, and we’ll talk more about public and private when we get more into classes. Uh huh?

Student: [Off mic].

Instructor (Mehran Sahami): It does print out the number of rolls it took.

Student: Okay.

Instructor (Mehran Sahami): That’s up here, right? It says rolled whatever Max Roll was after Num Rolls. Num Rolls is just the local variable up there that I keep track of. Question in the back?

Student: [Off mic].

Instructor (Mehran Sahami): Yeah, these names, as long as they're consistent, could be anything else. This could be X here and X there because all this guy cares about is I'm getting some value passed into me. What am I getting passed in? I happen to be getting passed a copy of Num Dice, but I don't have to name it the same thing; I could name it whatever I want. The reason why I named it the same thing is I wanted you to know that even if you name it the same thing, you're not getting a copy of the variable you're – yeah, you're getting beamed with a piece of candy. You're just getting a copy; you're not getting the actual variable.

So one other thing I want to mention to you real quickly is when random values come up, it becomes real hard, sometimes, to write programs because every time you run the program, you get a different set of random numbers, right? That's just the whole point of random numbers. If they were the same every time you ran the program, that would be, sort of, bad times, right? You'd be playing the same game over and over.

So how do we actually generate random numbers? What's the random number all about? Well, remember the old black box. The way this black box works – this is the Random Generator, which I'll just say RG. It's, sort of, weird. When you ask it for a number, it gives you back a number like five, and then secretly, sort of, secretly we've replaced our Random Generator's regular coffee with our new flaked coffee. Secretly, what it does – anyone remember that commercial? God, I am so old. All right.

It secretly saves that number and uses that number through some complicated equation that you don't need to worry about because this is the black box, to generate the next number when you need the next number, and when you say next number again, it does the same thing. So when you ask for next number, it might've given you six. It stores that six, and the next time you ask for a number, it does something with the six, and scrambles it, and chops it up, and reformulates it, and it says, oh, 17. okay?

So the thing is, you don't know what number this thing actually started with to begin with. As a matter of fact, the number that it starts with has to do with the time on your computer in sixtieth of a second, or actually, in thousandths of a second. So most people don't know what that was, and to the extent that you don't know what that was, you can't figure out what the sequence is. The only problem is if you don't know what that was, and you run your program, and it's got a bug.

First time you run your program it crashes because you got 17, and you're like, "Oh, I need to go find that bug." So you run your program again, but this time instead of getting 17, you get a six, and your program works just fine. So you're like, "Oh, bummer." And you run it again, and you get eight, and your program works just fine. You say, "Oh, it must just work. Let me get ready to submit it." And the last time you're ready to submit it, you get a nine, and it crashes. That's real frustrating. That makes you want to kill. I don't want anyone to kill, right, because if there's 300 of you and one of me, one of the people who might get killed in this process is me.

So what you can do is you can say I want you to generate the same sequence of random numbers ever time. How do I do that? I need to tell you that the number that you're gonna start with the very first time is some specific value so you will always generate the same sequence given that first number, and that first number is something we refer to as the Seed. It's like the number from which all other numbers sprout, and how do I set this seed? It's extremely complicated – actually, not at all.

If I had Rgen, which is my random number generator, this is my Instance Variable, which is my random number generator, I say .SetSeed, and I give it the value of the seed. A real fun seed to use, by the way, in case you're wondering, and you're having difficulty coming up with a seed is one, okay? So set seed equal to one. You do this somewhere at the beginning of your program, and all the random numbers you generate in each run of your program will be the same set. You won't always get the same random number each time you call for next random; what you'll get is the same random sequence. So if you happen to get the sequence seven, five, then nine, two, three – I don't know, you know, whatever, the next time you run your program, you'll get the same sequence, and that makes it much easier for debugging.

So let me show you a quick example of a program that actually uses that which will take our simple random program, which you saw in a previous slide, and add the little seed, okay? So now if we happen to run this program – we're running; we're feeling good. We want to run simple random. It's a good time. You rolled a five, and we're like, oh, that's great. That's wonderful. Let me run this again. So we run it again. We, sort of, do the quick – well, you rolled a five, right? Notice that the seed was actually set to one, so I'm not getting a one; I'm getting whatever would've happened after I put one through this complicated function.

I could go in here and change the seed to something else if I wanted to. I could change it to three, and the next time I run this program, I'll just happen to get a different initial value for that random number, but the sequence of random numbers I get will be the same based on the number three. So if I do simple random based on three, you rolled five. It just turns out one and three give you the same thing. That's life in the city. Maybe Mehran should've done a little more testing before class. All right. Uh huh, question?

Student: [Off mic].

Instructor (Mehran Sahami): The sequence is always based on whatever that first value was. So if it was one and then you changed it to three and got a different sequence, and you changed it back to one, you're gonna get that same sequence based on one. Uh huh?

Student: [Off mic].

Instructor (Mehran Sahami): No, the seed always just sets one number. It'll automatically do the appropriate conversions to whatever you want, but you should always just set the seed, like, one or whatever you want it to be. Uh huh?

Student: [Off mic].

Instructor (Mehran Sahami): Oh, no. The seed doesn’t actually have to be in your inter – if you have an inter. It’s a good question, but it doesn’t have to be. The value of the seed, in most cases, is actually entirely irrelevant because basically just add this when you do debugging, and when you think your program actually runs, then guess what? You take this puppy out, and you see if it still runs, and then if it doesn’t run after you take that out, you try a different seed because it’s gonna give you a different sequence in numbers, all right? Any other questions? Uh huh, one more question there?

Student: [Off mic].

Instructor (Mehran Sahami): Oh, can you use the microphone, please?

Student: If I use the same seed in different computers, does it give the same sequence?

Instructor (Mehran Sahami): Oh, if you use the same seed on different computers, it should give you the same sequence, yeah. Alrighty. Any other questions? I will see you on Friday then.

[End of Audio]

Duration: 51 minutes