

## Programming Methodology-Lecture09

**Instructor (Mehran Sahami):** Alrighty, welcome back. Wow, that's pretty loud. Welcome back to CS106a. I hope I didn't just shatter your eardrums. And thanks for making it out in the rain. I think the rain might have stopped a few people from making it today. But, actually, today is one of the most important lectures of the whole quarter, so it's too bad that it would happen to happen that way.

So a couple of quick announcements: One is that there's two handouts, one on coding style, and one on the use of variables. I'd encourage you to read both of them because they are both extremely critical concepts in this class. There are some things that – it's like, "Yeah, it's not so important." These two are really important so please make sure to read the handout.

There is a couple of new concepts I want to show you briefly in the beginning, and then we're gonna begin to put a whole bunch of things together. So so far in class, what we've done is, when you saw Caroline we did a bunch of stuff with Java so far, we've shown you a bunch of bits and pieces of things and put some larger pieces together. And today's the day where it really all comes together, and a whole bunch of stuff that we've talked about before, where I said, "Oh, later on this'll make more sense," today's the day when hopefully that will all make sense.

So first thing to cover, just very briefly, is something called, "Strings." And if you've been reading along in the book you've seen some references to Strings. We haven't talked about them in class, and now it's time that we spent a little bit of time talking about Strings. In about another week and a half we'll spend a whole bunch of time talking about Strings, but you should at least get a little introduction to them.

So what is a String? A String is just a type, like we had ints and we had doubles, and those boolean, stuff like that. String, except String starts with a capital "S," it's not lower case, is actually a type, and so we can declare variables of this type; like, we can have some variable called "str," which is a String.

And what does a String hold? What a String really sort of means is a string of characters, it's a piece of text is all a String is. And the way we usually denote a String is its inside double-quotes, or the " " that we're used to. So we might have some String, and initialize it by setting it equal to, "Hello, space, there." So it's perfectly fine to have spaces and other kinds of characters inside of a String. And, basically, it's just a piece of text that says, "Hello there." It's just one variable that happens to have this value, "Hello there." And so the important thing is it's text enclosed in double quotes.

And you can think of the text that you actually assign to the String. You can use things on it like concatenation, just like you did in a print lin, when you actually want to print some stuff on the screen, we use the plus operation to concatenate pieces of text together. Well, we can do that with Strings.

So we could have some String, we'll call this String, "name." And we might set name = "Bob." And we might have some integer age, which is just set to "20." And then we might have some other String, so some other String s, which we want to set to be something like "name:+," and then whatever the name actually is in this other String, so name. What that'll do is concatenate the word Bob onto name:. And then maybe we want to concatenate onto that something like age, and then concatenate onto that the value of the variable age.

And so what we'll get here is we'll get something that in the end s is just some variable, so it's just some box that contains something. And what it will contain is "name:" then concatenated on with whatever name evaluated to. Well, name:, when we look it up in the box is just Bob, so we'll say, "name: Bob." And then concatenate onto space "age:," and then the value of whatever age was, which is "20." Just like this would work in a print line, if you were to say print line, and have parens around this whole thing, you would expect it to print this out. You could actually just assign that to a String. So same sort of concatenation works.

And then you might wonder, how do you assign these Strings or get these Strings? You can assign what we refer to as "literal," which is an actual value to the String. You can build up a String by concatenating a bunch of other stuff together.

Sometimes you want to read in some text from the user, and there's a little method called "readLine," just like read int and read double, where you give it some piece of text but it's gonna display a question mark because it's gonna ask for something. And what it gives you back is basically the whole line that the user typed in. So it's not just looking for an int or a double, but if they type in a whole line of text, including characters, it gives you that back as a String, that's what it returns, and you can assign it somewhere. Like some variable we might call "line," that's a type String.

So you should just, at this level, see what a String is, understand it, be able to get them from users, in about a week and a half we'll do a whole bunch of funky things with Strings and get into the nitty-gritty. But at this point, this is just to give you a little bit of an idea as to what they are because we're gonna actually use them later on today. And there are some references in the book that hopefully will make a little bit more sense, but up until now, we didn't actually need them. So we just kind of deferred the discussion.

Is there any questions about String? All right.

So the next big thing, here is the real big topic that we're gonna cover today, is writing our own "classes." So this whole time, when we've been writing classes, so we would write something like, "My Program," and it would extend Console Program, and we'd have all of our methods and stuff inside of that class for My Program. Wouldn't it be kind of interesting to actually have multiple classes. Because in the days of yore, we talked about a Java program as really just comprised of multiple classes. So today's the day where you learn how to write other classes other than just the main program that you're actually writing.

So what's the general form for actually writing a class. The idea is we say "public." For right now all of our classes are gonna be public. The word class, and then we give the name of the class. So this should be familiar to you, just like when we wrote programs that extended Console Program, we had some name here, I'll put an underline to indicate that that's not actually a literal name you just give it a name, and then potentially you could also add onto this, "extends," and then some super class.

And so what you're saying is that you're creating a new class with this name that potentially extends some other class. And then inside you have some open brace, you have the body inside here, so this could be all your methods. It could also be variables that are defined, as we talked about before, various instance variables or ivar's in the class, and then the close of the class.

Now, this part over here, I'll put sort of in dashed parens because it's optional. You actually don't have to extend something as part of a class. If you don't extend something as part of a class, what Java sort of says is the default is everything in my universe, at least for the time being, are classes. So if you don't tell me that you're extending some existing class, there are some default class called "object," which by default you will extend if you don't say extend some other existing class. So everything at the end of the day in Java's world sort of moves up the chain, or the hierarchy that we talked about a couple weeks ago, and ends up an object. So everything at the end of the day really is an object, and if you're not extending something in particular then you're just directly an object.

So if you want to write this class, what you end up doing is you want to create a new file. And I'll show you how to do that in Eclipse in just a second. But, basically, that's why I put it in the box here, the name of the file should be – we kind of define it to be whatever the class name was and then a .java at the end. And the .java let's us know that this is a Java program. So if this thing over here, this class, happens to be class Bob, we would have some file Bob.java that would be where all the code that we write for the class Bob actually lives.

And the stuff that's gonna be inside here is all the stuff like the methods, and variables, and constants, which are just a particular form of variable, they're just a final variable so they get no other value. And the stuff that you have inside here, the methods and the variables, have certain visibility associated with them, as we talked about before. These things can either be public or they can be private.

And the difference between those two, hopefully it'll make a little bit more sense, is methods or variables that are public are what we refer to as being "exported out of the class." So if you write a whole bunch of classes, let's say we write three classes. And so I have my class Bob, and Bob might have in it some public method. Well, if I have some other class Mary, Mary can actually call the public methods of Bob. So if it's public anyone can access them.

If they're private, they can only be accessed by other methods inside of the class. So if I have some method inside Bob, or some variable inside Bob that's private, the only things that can refer to it are other methods inside Bob. If I have some other class Mary, the methods in Mary cannot refer to the private members, or the private elements of Bob. So that's the important thing to keep in mind.

There's a couple other notions. There's a notion known as "protected." And we'll talk about that potentially toward the end of class. For right now, you don't need to worry about it. The thing you do want to keep in mind is that most things in classes will actually be private unless there's a good reason to make them public. In some sense think about yourself, you want to have some notion of privacy.

You don't want to just broadcast to the world or let the world be able to come in, and say, "Hey, you have some variable here which is, like, your age," and someone else is gonna go and modify your age. That would kind of bother you, right? And so most things you keep private unless there is a good reason to actually make them public. And so the run method that we talked about so far, I actually sort of told you, it needs to be public. Most things will be private. And I'll show you some examples of classes where we look at things that are private versus public.

Is there any question about that, what public or private actually means or sort of the general notion of how we would define the class? Hopefully, that's familiar to you because you've seen it before.

Um hm.

**Student:** [Inaudible.]

**Instructor (Mehran Sahami):** Right. There is someone else, us, the 106a people, who provided, actually the ACM people who provided the libraries for you to use, that call your run method automatically when a program starts. So that's what's actually going on. And when we get toward the end of the class I'll kind of lift the covers on what's going on there.

Um hm.

**Student:** [Inaudible] any class, like, that somebody's using [inaudible]?

**Instructor (Mehran Sahami):** Yeah, exactly. And unless you want people in Timbuktu touching stuff they shouldn't be touching it should be private. The way I like to think about it is think about your private parts, you like to keep them private, you don't want them to be public. Most things that you have on your body are private. You are an object, so most things in an object are private. All right. Just keep that in mind.

So, with that said, let's actually go and create a "new class." It's kind of fun to create classes. So I want to create a new class. So what I'm gonna do is I'm gonna get my

Eclipse out and I'm gonna fire it up. And the way you can actually create a new class in Eclipse is over here in this little window, that we haven't done all that much stuff with so far, usually we give you some folder that's called Assignment 1 or Assignment 2, and you work on it. If you right click on the name of that folder, and if you happen to be a Mac person that's the same thing as "control clicking" on that, you'll get this big scary menu that comes up, and you just want to pick New. And when you pick New you pick Class. So that's all that's going on. A right click on the name of the folder, pick New and pick Class.

And what happens now is this thing comes up, and it says, "Create a new job of class," and there's all these fields, and things like dogs and cats sleeping together, it's just totally out of control, and the only thing you need to worry about here is what are you gonna name your class.

And so maybe we want to write some class, I'll just call this "my counter." Because I'm gonna write a little class that creates counter. And notice when I started typing, as soon as I started typing, let me get rid of this, I'll start typing again "my counter." Anyone see what happened? Eclipse had a cow, and the cow that it had was up here up at the top. It says, "The use of the default package is discouraged." And you sit there and you think, 1.) What is the default package? And 2.) Why is its use discouraged?

And the important thing to keep in mind is you don't care. This is one of those moments in your life where you're gonna be rebellious, and you're like, "Bring it on, I'm in the default package," that's life in the city. And what the package really is, is the package is just a collection of classes; a set of classes that make sense together. Java provides a facility to say we're gonna put them all in the same package.

Right now we're not gonna define a whole bunch of classes that all work together, so we don't need to worry about packages. So all of our classes that we write are gonna be in the default package, which means, kind of, the un-named package, it's just sort of the package that everyone's a part of unless they're a part of some other package. So it says, "Oh, it's discouraged," and you're like, "I don't care I'm in the default package, I'm living for big gustos." And then you do something extremely complicated, which is you click "Finish." And guess what? You just created a new class.

Now, a couple of things to keep in mind: Over here, in your little project that you have, there is now an entry called, "my counter.java." Remember, we just named the class "counter." Guess what Eclipse did for you, it did this, it created for you a file, automatically ending with .java, that's name matched the name of the class.

What else did it do for you? It created what we refer to as a "stub." It said, "I'm gonna create an empty class for you." You're gonna create a class called, "my counter." So what I'm gonna do is create, for you, the empty shell. Right. It gives you basically two lines of code and one of them is a brace. So it's not providing a whole lot to you, but it's, like, ooh, ooh, you've gotta define a class, like, aren't we excited to write a class. And you're like, all right, let's actually write the class.

So what I want to do is let's go ahead and actually write a class. And the class that we're gonna write is something that's basically a counter, or I'll refer to also as an "incrementor." It's a very simple idea here. What I'm going to create is a class that is going to allow me to get, essentially, numeric values that are sequentially coming from each other.

You might want to say, "Why would I want to do this?" This is the kind of thing, say, that Stanford would actually want to do when they're assigning you ID numbers. Right? They want to have some object somewhere, which is the ID number provider. And they're, like, "Hey, you just came to Stanford, get me a new ID number." And you just came to Stanford, "Get me the next ID number." So what I want to have is some way of saying, "Hey, create this object for me that just gives me numbers, and I'll just ask you for the next number," and you just sequentially give me new numbers.

Or how many of you have actually been at a bakery or something where they have one of those big red wheels that has the little tags that you pull out of it? That's a counter. So that's what we're gonna write as a class, and then we can create objects of this counter type.

Okay. So how do we actually create the class? This is a little bit different than writing a class that extends a program. Because we're not writing a full program we're writing a class. So we don't have a run method here. What we do have is something that's called a "constructor." And what a constructor is, is it's sort of the method that initializes everything that should be going on with this class.

Now, here is the properties of the constructor. First of all, we're gonna make it public. At least for right now, all the constructors you're gonna see are gonna be public. A constructor does not return a value. So we just skip the whole notion of a return type, and we give the name of the constructor the same name as the name of the class. So we say "public my counter." And now, at this point, we have no return type. Then we provide whatever parameters we're gonna take in here.

Now, what I'm gonna do for my little counter is I'm gonna be kind of funky, I'm not necessarily gonna start at one, I'm gonna allow someone to say, "Hey, you know, like, Stanford ID's don't start at one, like, they actually start at, like, 30-some odd million." Right. And most of you actually have one that's in, like, the 40-odd millions or probably 50-odd millions. Is your first ID No. a five now? How many people have a five? Anyone have a six? Anyone have a four? Anyone have a three? Yeah, good times. It's come a long way, evidently. We've admitted 20 million students in the last 15 years.

So what we're gonna allow my counter to do is have some starting values. So we're gonna specify parameter start value that's going to be, essentially, where we want this counter to start. And now you think for a moment, you think, "Hey, we've gotta do something with that start value," right? This is state of the object that we need to keep around. Somehow we need to start off by saying this counter starts at start value, and every time you ask me

for a new value of the counter I'm gonna give you a new number. That means I need to keep track between method calls of what the old number was.

So if I need to keep track of something between method calls what am I gonna need? Instance variable. There's a few people over here, one person lying on the floor down there. So what I'm gonna have is an instance variable. And to find the instance variable down at the bottom I'm going to make the instance variable private because I don't want someone being able to muck with my instance variable. So I'm going to have private int, and I'll just call this "counter." So this is my instance variable over here.

And so what I'm going to do is, in my constructor I'm going to say, "Hey, initialize that counter to be equal to whatever start value the user gave me." So I have my own little instance variable here. Every counter object is gonna have its own little counter variable, and I'm just gonna start it off by giving it whatever value the user told me to set it to when they created an object of this type. And I'll show you in just a second how we create an object of this type.

Now, the other thing we could do, that's kind of funky, is you can actually have more than one constructor. You can have multiple constructors that have the same form, they're all public, at least so far. They return no type. But the thing that differentiates them, and the way the computer knows which constructor to actually use is what the parameters are. So we just created a constructor that has one parameter.

I'm also going to create another constructor that has no parameters. So someone can actually create a counter without telling me what the starting value is. Well, what is the starting value if they don't give me one? I still need to assign one. So I'm gonna say, "Hey, if you didn't give me one, I'm gonna set the starting value to be one." Because most of the time when people want a counter they just want to count from one. But if you want to do something funky, like have ID numbers for a university, you can tell me the starting value.

And so I'll show you in just a second, how we actually create objects of this type, and when we create objects how it differentiates between whether to call this method up here with a parameter or this method over here without a parameter.

A couple of other things we need to do. We need to find and specify some way of saying, "Give me the next value." Right. How do I get new numbers, how do I request new numbers from this counter? Well, I need to have some method. This method's going to return for me an integer, because my counter is just integers, and I'll call this "next value." It takes in no parameters because I don't need to tell you anything. I don't need to tell the object anything to get the next value. I just say, "Hey, buddy, next value." And it gives me the next value.

So the way it's gonna give me the next value is it's going to give me whatever value the counter has right now. So I might be inclined to say, "return counter." What's the problem with doing that? What happens the next time I call next value? I return the same. But

that's not very exciting, right. You go into the bakery, and you're like, "Hey, I got No. 1." And you look around and there's 300 other people that have No. 1. The fights break out, it's bad times.

So what we need to do is we need to say, "Hey, I need to keep track of what value the counter is currently right now because that's what I'm gonna return to you." So I'm gonna have some "temporary." And this temporary is a local variable, it only needs to live inside this method for the lifetime of this method. And I will initialize that temporary to be whatever the current counter value is. Then I'll go ahead and add one to the counter value. So I'll increment the counter so the number next time this gets called is gonna be one higher. And what am I gonna do now when I return? Any ideas? Return the temporary.

The temporary is what the value of the counter started at when this method was first called. So I said, "Oh, store off a copy of it, add one to the counter," because next time it'll be one greater and return whatever the old value was in temporary. And after this function's gone, temporary gets blown away. And it's like, "Hey, I was a local variable, I gave my life for the method." And we're like, "Thank you very much, temporary, that was very courageous of you." But you're gone now but you managed to return this value that you stored. So good times, you actually did what we needed you to do.

And this is my whole class. Is there any questions about this class?

Now, let me show you one other thing before we actually make use of this class. And it's a way that we could have actually done things slightly differently. So when we actually created the class here's our constructor. In this case, what I've done is I've just renamed the class from my counter to be incrementor, because incrementor is kind of the more funky computer science term for it, and incrementor is something that adds one to something and counts. So other than my counter, which is just all fuzzy and good, it's like, "Oh, it's My Counter," but people might think, "Oh, your counter, is that, like, the restaurant in Palo Alto that gives you burgers?" Anyone ever eaten there? It's a good time. And then they didn't even pay for the product placement, I've gotta say. We'll call it incrementor. But same kind of thing here, we're gonna have some constructor for incrementor. It's gonna take some start value. I'm just showing one of the constructors here.

And one thing you might say is, what are the properties, just to reiterate, of constructors? The name of the class is the same as the constructor name. The constructor does not specify return type, it's responsible for initializing whatever values we care about in the object. And this puppy gets called when an object is created. And I'll show you an example. Just like you created graphical objects in the past, we're gonna create an object an object of type incrementor in just a second, and I'll show the code for that. But that's when this implementor method actually gets invoked.

So one thing that people think about when they see this is, "Hey, you have this start value thing, and then over here you call it counter, why don't you just call it counter both

places? Why don't you rewrite the code like that?" What happens when we do that? Is there any issue with doing that? You're, like, ah. There's the one part of you that's, like, "Oh, that looks so clean," like, counter equals counter. And then there's a part of you that get really uneasy.

Because you see this line here? That line does nothing. It assigns the same value back to itself. And you're, like, "But I wanted this counter to be like my instance variable counter, and I wanted this counter to be my parameter counter, can't the computer just figure that out?" No. You can't do that here because sometimes it will prevent you. So the problem here is that the parameter counter is actually getting assigned to itself. You're, like, "Hey, the counter here before was the instance variable, what's going on?"

If you have a parameter that has the same name as one of your instance variables, you get something called, "shadowing." What shadowing is, is when I refer to some variable, the first place I look is do I have any parameters or local variables that have that name. If I do then that's what I'm referring to. If I don't have any parameters or local variables with that name then I check to see if I have an instance variable with that name. So there's actually an order in which I check for these names. And because in this case I have a parameter named counter, this counter's referring to that parameter, and this counter's referring to that parameter.

In the previous case, I didn't have a parameter named counter. So it said, "Do I have a parameter named counter?" "No." "Do I have an instance variable named counter?" "Yes." So it uses the instance variable.

So you could say, "Hey, is there a way I can force it to know the difference between the two counters?" And the answer is actually yes. So even though there's a problem here we can fix this. And the way we fix this is with a notion called, "this." The way you can think of this, is this is always referring to the receiving object.

What does that mean? That's a mouthful. It means that when some method gets called, remember when method gets called there's always some object that that method is getting called on, well, when we call that method, inside the method we can refer to `this.counter`, which says refer to myself, my own object counter, don't refer to my parameter counter. So `this.counter` will look for an instance variable directly. It will not look for a local variable; it will not look for a parameter.

So here when I say, `this.counter`, it says you have an instance variable named counter. That's what `this.counter` is referring to. This counter over here, it says, "Hey, I don't have any kind of qualifier." Let me check the parameters and local variables first. I have a parameter named counter, so that's what the counter over here is actually referring to.

Now, this is, kind of, ugly syntax. There's some reasons to use it here and there, but it's ugly syntax. The way to think about this is just give them different names, that's the easiest way to do it. Keep the names of your parameters distinct from the name of your

instance variables and there'll just be no confusion with this or that or whatever, you just keep the names distinct. And it's the preferred root.

You want to think about writing programs that require the human to think less when they're actually looking at the program. You're, like, "That's really weird, I thought computer science was all about thinking." Yeah, it's about thinking higher up the food chain, you don't want to worry about little details like this and get caught in these weird shadowing effects or aliasing effects, you just want to say, "Yeah, they're distinct," and then it's clear.

Is there are any questions about that? It's kind of a funky concept but it's just important for you to see it.

So with that said, let's actually look at not my counter anymore. We sort of wrote the bare bones code for my counter, but here are the full-fledged incrementor version. And so when we're now computer scientists, and we think about good programming style, we have some comment for the class. So just like the programs you wrote, you want to have some comment for the whole class, you want to have some comment per method you write. So this is all the same code we just wrote, now it's just commented. So good programming style.

**Instructor (Mehran Sahami):** Um hm.

**Student:** [Inaudible.]

**Instructor (Mehran Sahami):** Because if I declare it every time there is no way of me being able to keep track of what it's previous value was. So that's the critical concept of an instance variable, I need to keep track of some information between method calls. If I didn't, like temp, I didn't care what temp was the last time I called next value, it's just the local variable and it goes away. But counter I need to keep track of.

So how do I actually use counter? I wrote a program called "use counter." So use counter is basically just a Java program that is a program. So it extends Console Program, just like you're used to. I'm changing the font. You don't need to worry about this line. All this line does is it just makes the font bigger so when I run the program you can actually see the text. Because there were people who were having trouble seeing the output text when I ran programs.

Now, how do I create objects of this class? This is just like you saw with graphical objects. What's going on is if I want to create some new counter, or some new incrementor, I specify the type of the variable incrementor. I give it a name, I'll call it "count one." And because I want to create an object, I go to the incrementor factory and say, "Give me a new incrementor." When I say new and the name of a class, what I get is a new object of that class. So at this point when I say new incrementor, a new incrementor object is created for me. That is when the constructor for that object gets invoked. So that constructor thing we just wrote, that initializes the object, the place it

gets called is when you ask for a new one of the class, when you get a new object the constructor's invoked.

Which version of the constructor does it invoke if you have multiple versions? It depends on the parameters you give. Right. We created two versions, one without a parameter, if you didn't specify the parameter it set the counter starting at one. One with a parameter, so if you specify parameter, it says, "Hey, there was a version of this that had a parameter," that's the version that's gonna get invoked, and so it's gonna set the beginning counter to 1,000.

Any questions about that?

So what are we gonna do? Now what we're gonna do is write out some values of the counter and count up. So what I'm gonna do is write a method. This method is going to take an object as a parameter. There's no reason why it can't, parameters just have some type and some name. So this count five times method is not gonna return anything, but it's going to take as a parameter something of type incrementor, that I'll just call "counter." And what it's gonna do is very simple, it's going to loop through five times and just print out the next five values of that counter. So that's all it does, it just pulls the next five tickets from the counter and writes them out to the screen.

So I'm gonna say, what are the first five values for count one, and call this function. What are the first five values for count two, and call the function passing and count two, and what are another five values for count one.

So let me go ahead and run this program and you can see what's going on. So if I run this, there's what I get. So I say the first five values for count one. Remember count one started with a counter of one. It gives me one, two, three, four, five. Then I asked for the first five value of count two. Count two started at 1,000, so I get a 1,000 up to a 1,004. Then I ask for the next five values of count one, and I get six, seven, eight, nine, ten.

Now, this all seems fairly straightforward, except for one fact that should slightly disturb you. Is there anything that slightly disturbs you? The thing that should slightly disturb you is the conversation we had last time, where we talked about when you pass a parameter to something you get a copy of that parameter, or you get a copy of that thing that you're passing.

So the interesting thing that's going on is you would think, well, you're passing count one here. When you pass count one there shouldn't you have gotten a copy of the count one that started with its counter of one? So when it counts it up here shouldn't it have only been counting in that copy, and so when it was done here that copy should have gone away, and when you returned over here your counter should have still been one? If we were doing integers that's sort of what would have happened.

But the things that's different between integers and doubles and objects, and this is the important thing to think about objects, when you pass an object as a parameter you are

not passing a copy of the object, you're passing what we refer to as a reference to the object, which you're actually passing the object itself. So what we think of as a reference is really there is a way to point to the object.

So let's say you're an object. When I'm over here and I say give me a new object, you're sort of created, you come into being. And what count one really is, is just a way of pointing to you. And so when I say, "Hey, count five times, count one," what I'm actually passing is a copy of my finger. And you're, like, that's a little weird, but it points to you. So when we count through five times you're the one counting five times. And then when we come back over here, and I call count one again, I'm passing another copy of my finger that's pointing to the same place. So you're the same object that was invoked before, and you give me the next five values.

So critical thing to remember, when you are passing an object as a parameter, you are actually in some sense passing the object itself. The way you can think about this is it's like the mafia. We know where objects live, and if I know where you live when I'm doing something, when I, like, come to your house and put a horse head in your bed or I'm, like, getting the next value or whatever, it's you, and it's you every time because I know where you live. With integers, we don't know where they live, we just pass copies. Here's a copy of a double, here's a copy of an integer, we don't know where they live. That's perfectly fine.

With objects, we know where they live. And so what you're actually passing is you're passing where that object lives, and so that way you're always referring to the same object because you're referring to the same address where that object lives, you're always looking up the object in the same place.

Any questions about that? That's kind of a critical concept.

**Student:** When we write a program that makes use of a class that we've written how does it know to access that class?

**Instructor (Mehran Sahami):** Because when you're creating your classes over here you're creating all your classes in this default package and they're all in the same project. So when you say give me a new incrementor, it goes and looks through these, and says, "Hey, do you have an incrementor class over there," and that's how it actually knows.

**Student:** For the third part of this program, [inaudible]?

**Instructor (Mehran Sahami):** Because we already counted one, two, three, four, five. So the counter of the value that's actually being stored for the counter is currently six.

Any other questions?

So one thing we can also do that's kind of funky is, remember when we have our counter, or our incrementor, we have this instance variable. There's another kind of variable,

which I never told you about before, well, actually, I did tell you about it, I just didn't give you its actual name before, but now you're not old enough so I'll tell you what the real deal is. The basic idea is so far you've seen things like local variables, and you've seen instance variables. There is a new kind of variable, and it is called a "class variable" or a class var.

What a class variable is, it is a variable that is shared by all objects of that class. There is one variable that all objects of that class share, that's the funky concept. In an instance variable each object of that class has a different version of that variable.

So if we have some counter over here, and if we have some count one, and some count two, and they each have an instance variable that we just defined, each of them has its own box. If I were to create a class variable, there is one variable counter that all objects, count one, count two, all objects of the type, or of the class, that this class variable is in, are all referring to the same one single box.

Now, there's sometimes you want to do that. I'll just show you a very simple example of how that works, and how you actually specify it. So the way you specify it, is in instance variable you just specified if it was public or private, and you specified its type and you gave it a name. In class variable you add this word called "static." And sometimes class variables as a result are referred to as static variables.

And you've seen static before. When did you actually see static used in conjunction with a variable? Constants, right. Why did we make them class variables by adding static? Because it doesn't make sense for two different objects that are circles to have their own copy of the constant value pi. Pi is in some sense some universal for all circles. So it would be shared as a class variable among all circles. So even though it was a constant, if we didn't say static, we would have gotten one of those constant values for every circle we created, for example. We want to just have one that's shared, so we call it static. But you can have static variables that are not necessarily constants, they don't have to be final.

So let me show you an example on the computer. We come over here. Here's our incrementor that we just wrote. And we say private static int counter. So what I've now done is, instead of creating an instance variable I have created a class variable. How does this change the program? Let me save this and run this. What this will now do is, all of the things that I create that are incrementors are all referring to the same counter. You're, like, whoa, that's kind of funky. Yeah, let me show you what actually happens.

So when I run use counter, I get a 1,000 through a 1,004, then I get 1,005 through a 1,009, then I get 1,010 through a 1,014. And you're, like, whoa, what happened there? Let me explain to you what happened here and hopefully it'll be clear. Like what did it start at a 1,000, why didn't it start at one, what was actually going on?

What's going on, if I actually look at use counter, is remember there is only one counter variable for all objects of the class. This first count one comes along and say, "Hey, give

me a new object that's an incremator." And we say, "Okay." And we set it's counter to be one, right, that's what the constructor did. Then I say, "Hey, create for me another counter." And it says, "Okay, here's another incremator over here, I'm now setting that counter variable to be a 1,000." And you're, like, uh-oh, that's the same counter variable that count one was using, too. Because now they're both sharing the same variable, and that variable is now a 1,000. The old value of one just got clobbered by this guy because it's referring to that same class variable.

There's only one counter variable among all objects of the class. So now when I count the first five values of count one, it's got a 1,000, it counts up to a 1,004. That 1,004 is shared by all instances of the class. So when I do the next five for count two, it's starting at a 1,005, and it counts from a 1,005 through 1,009. And then when I do count one again, count one's sharing that 1,009, and that's why we get a 1,010 through a 1,014.

Any questions about that?

**Instructor (Mehran Sahami):** Um hm.

**Student:** [Inaudible.]

**Instructor (Mehran Sahami):** Oh, yeah, a microphone would be good. And let me guess what your question is. If you switch the order, yes, you would actually start at zero – or you'd start at one. The first constructor would initialize the value at a 1,000; the second constructor would overwrite the 1,000 with one and then you'd count from one. Every once in a while I have this fleeting ten-second ability to read minds, and it goes away very quickly.

So with that said, it's time for something completely different. So any questions about instance variables versus class variables?

**Instructor (Mehran Sahami):** Um hm.

**Student:** [Inaudible.]

**Instructor (Mehran Sahami):** Maybe we should talk about that offline. Because it'd be good to know what particular thing you're referring to. But most of the time, for a lot of intents and purposes in this class, the main time you'll actually see static variables are when they're actually constants. There would be very few times when you actually have a static variable that's not a final variable, it's not a constant. But if you're interested, we could certainly talk about that offline.

So with that said, there's a funky concept called "Javadoc." And now you're old enough to learn about Javadoc. And all Javadoc really is, is it's basically Java's documentation system. This is just a little side point. And the reason why I'm giving this aside to you is you're gonna see Javadoc in just a second. We're gonna write another class using Javadoc.

The difference between regular comments and Javadoc. comments is very simple. Javadoc comments start with a slash and two stars instead of just one star, but they still end with a star and a single slash. So comments that are in that form, that start with a slash and two stars, are called Javadoc comments. And you'll see what that means in just a second.

In the comments you can put special tags that are understood by the Javadoc system. This is just the system that's gonna generate html pages that explain what your code does via its comment. So your comments actually serve a purpose now. Your comments are going to beautifully get transformed into html that someone else can look at when they're trying to understand what your class does. Instead of having to go through code, they can just look at a webpage.

So there's some special tags, like @ param or @ results, that specify what is a parameter of the function you're writing or the results if the function's actually returning something. And I'll show you examples of that right now.

So here is a method, and this method might be some method we have that just sets the number of units someone earns. So it's called "set units." It takes in some number of units as a double, and let's say we have some instance variables, some are units earned, which is like how many units you've earned at college so far, which just gets set units. This comment here is a Javadoc comments. It starts with slash, two stars, and with a star slash. Inside the comment, we still comment what the function does. Just like a normal comment, we specify whatever we want to explain to the user.

Additionally, we can have some special tags, like @ param. And the way @ param works is, for every one of your parameters you specify @ param. Then you specify the name of the parameter, so that's why the words look kind of funky, because the name of the parameter is units, that's the next token here, or the next word is basically units, and then what that parameter actually means, the number of units earned. So for human reading, and it's not necessarily the most legible thing in the world, but after you've seen it for a while you just kind of internalize it, but the computer can actually understand that very well and produce some nice documentation.

So let me show you an example of this. It turns out, if you're actually interested, all of the ACM libraries, all the libraries that we're using in this class, if you want to dig into them, if you go to this URL, [jtf.acm.org/javadoc/student](http://jtf.acm.org/javadoc/student), because it's the student version of the Javadoc, you can actually see the Javadoc for all of the ACM libraries. So you, until your heart's content, look them over.

But let's just look at one particular example. Remember random generator that we sort of looked at together from the ACM libraries, let me just show you the Javadoc for that so you get a sense for what's actually going on. So all Javadoc is, is it's html. It brings up a web browser. And what it explains in here is here is class random generator. This is stuff that got automatically generated based on the comments and the code. It says, ACM util random generator is actually derived from java util random, so it's actually a subclass of

java.util.Random. And it tells us that by saying that actually this public class RandomGenerator extends Random. So it's in fact a subclass.

And there's a whole bunch of stuff there that explains how this class actually works, and how you get RandomGenerator.getInstance() to get a version of it. But also what's generated for you, very nicely in these nice HTML tables, are things like, "Hey, here's the constructor." Constructor has no arguments, that's how you might potentially create a new RandomGenerator if you wanted to. But, in fact, most of the time what you want to do is get instance. So for all of the methods it has the name of the method, it has, basically, the form of the method, which is the method and then whatever parameters it takes, and then whatever comments you have about that method.

The other interesting thing about this is, it tells you which methods are kind of built in and which methods actually got inherited from your super class. So we mentioned that a RandomGenerator is an object that extends Random. Well, some of the methods are actually defined in RandomGenerator specifically; some of the methods are inherited from this thing called Random. It doesn't actually make a difference which ones are inherited or not, but you get kind of all this text.

You get even more text down here. So if you want to look at the excruciating details of nextBoolean(), that takes in some double p, it tells us what the usage is. How do you use nextBoolean() and what are its parameters. This is where those @param little tag that was in there, there's an @param tag that says p, and then has this comment, and this automatically gets turned into nice HTML.

So anytime you have questions about what something in the ACM library does or what methods are actually available to you for a particular object, you can actually go look at the Javadoc. You don't need to scrounge through code, it's actually just HTML. You can browse in a web browser by going to that URL I gave you and search around.

But the reason for showing you all of this, one, is to let you know that it actually exists, the second is we can now put this all together. We can create a class that has a bunch of stuff in it that we comment using Javadoc, and that also makes use of Strings, and gives you one nice big picture at the end.

So what I'm gonna do is I'm gonna create a class. This class basically is involving a student, something that hopefully you're all extremely familiar with at this point. So what a student class is going to provide, it's gonna have Javadoc comment. So up at the top I have the star, the slash star star, that wasn't a typo, that's actually a Javadoc comment, and explains what this whole class does.

So it says, what I'm gonna do is create a class that keeps track of the student. What is information I care about for a student? Well, this is a very simplified form. I care about a name of a student, their ID No., and how many units they've earned. That's all I care about for right now. The book has an example that's a little bit more involved, but I'm gonna give you a real simple example here.

So what are my constructors? How do I create something as a student? Notice the class student does not extend anything. So all students are just objects, I'm not extending anything else, they're just objects. One of my constructors for a student says specify the name and ID for the student. As a matter of fact, that's gonna be my only constructor, I'm not gonna allow someone to do other kinds of things. Give me a name, which is a String, right, it's just a piece of text, and an integer, which is an ID No. So we'll assume all ID No.'s are integers. And what it's gonna do is keep track of some information about students that are going to be stored as instance variables.

Because every student is going to have their own information for their name, which is just student name that's a String, student ID, which is just an integer, and the number of units they've earned so far, which I'll keep as a double because there are some schools that actually allow half units for whatever classes. They don't here, but other places they actually do. So we're just making it a double for generality as opposed to having it be an int. So these are all private instance variables. No one else outside of a student can get in there and muck with these things. The only way we can potentially change them is through the methods that are involved in this class.

We're also gonna have the number of units that are required to graduate (at Stanford that's 180), and we're going to specify that as a constant that is static and final, which means all students share the same number of units to graduate. And, unfortunately for you, you can't say, "Hey, you know what, I'm just gonna set units to graduate to be like ten." Wouldn't that be fun? Like, you take 106a, you take IHUM and you're done. Final, yeah, you're not changing this, it's 180 until the cows come home. So that's why it's a constant as opposed to just being a class variable that's actually modifiable.

So what are all the things we allow someone to do in here? The things we allow someone to do are, for example, to get the name of a student. Once I've created a student the student name variable is private. I can't actually refer to that variable outside of this class. So how do I ask you for your name after you've been created? So you were just created as a new student at Stanford. And, I'm like, "What's your name?" And I want to kind of come over and get a little close and be like, "Hey, can I touch your variables?" And you're, like, "No, man, this is wrong." So I'm, like, "Oh, what do you have that's public? Oh, get your name." What's your name?

**Student:** Aki.

**Instructor (Mehran Sahami):** Aki. So I get back a String Aki from your object. I can't actually touch any of the private parts. ID number, same kind of deal. I can't go in and see what your ID – like, I'm not gonna go into your wallet and pull out your ID card and be, like, "Oh, ID number, let me get out my magic marker and change it." The only way I can get it is by asking you for it, and if you've made that public then I can get that information from you by making the method public.

A few other things. Now, here are some funky things. Set units I actually made to be public, which means I can allow someone, even though the number of units earned is a

private variable, so I can't directly just go in and modify the variable directly, I can call set units and give it some number and it will just change the number of units you have. I can ask you for how many units you have.

So this is a dangerous thing, if you think about it. And that's the kind of thing you want to think about, in terms of good programming style, what do you make public and what do you make private. And if red flags go off because you're making things public that shouldn't really be public, that's something you want to think about for style.

Get units, this just returns how many units you've actually earned as a double. And, again, you can see we have a return tag here for the Javadoc that says the number of units the student has earned is the things we're actually returning.

A couple other things real quickly, and then I'll show you that we're gonna make use of this class. There's something called "increment units," which basically just says, every time you take a class I want to increment the total number of units you have by the number of units in this class. So I pass in some value for additional units, and I increment your units earned by additional units. Having enough units, that's a predicate method, it returns a boolean. And what it basically returns is the number of units earned greater than or equal to this constant for how many units you need to graduate.

And last, but not least, and this is a critical one, all classes you write should have a method in them called "two String." What two String actually means is get me some textual version of what's the information in this class. So two String for a student returns back a String, which is the students name, plus their ID number in parenthesis. It does not necessarily specify how many units they've earned, that's fine. It doesn't need to include all the information that's actually in the object. What it just should specify in a nice textual way is the information that you care about displaying about an object if someone wants to display it. But all classes you write that are not programs. So something that extends Console program doesn't need this, any other classes should have a two String. And that's the whole class.

So any questions about this class or should I just show you how we use it? Let me show you how we use it.

Well, it's Stanford, you're at Stanford so we've gotta have a program called Stanford that actually creates students. So Stanford is a program that extends Console program. And, again, we have something to increase the font size. But we're gonna create some new student, Ben Newman. And that's pronounced stud. So we have some student, stud, that's a new student. And remember we need to specify the name and ID number. to initialize that student.

Ben Newman, your ID number's now 1,001. I'm gonna set your estimated number of units at 179. And then I want to write out how many units you have. I cannot refer to your name and number of units directly, I need to ask what's the object, that's the receiver of the message. And the message I'm gonna send is the name of the method I just wrote,

stud.getName, that returns to me a String, which is the name that's Ben Newman. I append to that has, I append to that the number of units Ben has, and then add the word units at the end of it and write that out to the screen.

Then I want to see if Ben can graduate. So, again, I print out Ben's name, if he can graduate, his status by calling "has enough units." That will return true or false, that will write true or false out onto the screen as part of the String. Then Ben takes CS106a. So Ben takes CS106a, is what's gonna get written out, and then I increment Ben's units by five because he's now taking CS106a, a five unit class. And then I ask again can he graduate. And if he has enough units to graduate, I'll say, rock on, and write out the String version of the object that I have by calling toString.

So when I actually run this all here's what I get. I'm running. I'm running. I want to run Stanford. It's just that easy to run Stanford. So Ben Newman has 179.0 units, right, because it's a double so it puts that .0 at the end of it. Ben Newman can graduate. We get false back because he doesn't have enough units to graduate. But that 179 is stored inside the object, that stud object has the 179. Ben takes CS106a, so we increment that object's number of units by five, by calling the appropriate method, and then Ben Newman can graduate is now true because he's got 184 units.

And, finally, when we say, "Rock On," and we call toString, what toString gives us back is the name plus the ID number inside paren. So it says, "Rock On Ben Newman paren number 1001."

Any questions about that? Alrighty. Then have a good weekend and I will see you on Monday.

[End of Audio]

Duration: 52 minutes