

## Programming Methodology-Lecture16

**Instructor (Mehran Sahami):** All righty. Welcome back to yet another fun-filled, exciting day of CS106A.

So it's the day before the midterm. This is kind of like, you know, the calm before – well I shouldn't even say the calm before the storm. It's probably a lot more calm for me than it is for you. But hopefully you're getting prepared for the midterm, so a few announcements before we start.

There's one handout. You still have sections this week, so sections will be happening after the midterm. The section problem is the handout for this week. The midterm is tomorrow night and hopefully you should already know this by now, but just one more time just to drive the point home: midterm tomorrow night, Tuesday, October 30th, 7:00 to 8:30 p.m. in Crosby auditorium.

If you had emailed me before and let me know that you had a conflict with the midterm, you should've already gotten an email from me this weekend letting you know about the alternate exam, what time it was, what the instructions are for the alternate exam. So if you think you emailed me about a conflict and you never heard from me this weekend, please let me know right away, like right after class.

Right now, if you heard this and you think you have a conflict and you didn't get an email from me this weekend, you should be panicking because I think I've reached everyone who said they have a conflict. And invariably, as you can imagine, there wasn't just one time that worked for everyone, so we ended up doing a little bit of gymnastics to make it work, but hopefully everyone has a time they can actually make.

If you're an SEPD student, email me by 7:00 p.m. tonight if you plan on taking the exam at your site remotely. If I don't get email from you by 7:00 p.m. tonight, I will just assume that you are coming into campus for the regular exam, which is perfectly fine. If you want to send me an email verifying that you're coming in for the regular exam, that's great too.

I always love to hear from the SEPD students, but you're not required to but you are required to send me email if you're gonna take the exam at your site. Otherwise, I won't know to send the exam to your site. So send me email by 7:00 tonight letting me know if you're taking it remotely, and let me know who your site administrator is, their name and email. That's critical because your site administrator will be administering the exam to you.

All righty. So any questions about anything before we dive into our next great topic? All right, let's get started on our next great topic then. And tonight – today I should say, I think it's tonight because like the lights are on; it's actually daytime. So today, there is a whole new concept that we're gonna be getting into called an array. Okay?

And all an array is, is basically a way to keep track of a whole bunch of information at once. So what we've done so far in class is like when you wanted to keep track of information you had some variable, and you had like an int, or you had a double, or you had a g-oval, or whatever it was. And it was you kept track of some small piece of information, but in the real world, really what you want to do is keep track of a whole bunch of information.

Like say you're gonna take your midterm, we would want to keep track of all of your midterm scores. And it would be a real pain, for example, to say int Midterm 1, int Midterm 2, int Midterm 3. You can just see how monotonous it is. And by the time I got to -int Midterm 150 you'd be ready to kill, but we couldn't stop then because we'd have to go up to like int Midterm 325. Right? And so we just don't want to do that and computers are really good at dealing with lots of information. So the way they allow us to keep track of a whole bunch of information at once is something called an array.

And so the two key concepts for an array is that an array is ordered, so it has some natural ordering, and that means we can refer to the elements of the array using some index based on that ordering, and it's also homogeneous, which means that everything in the array that we store is of the same type. So we can have an array, for example, that stores integers, and we could say, "Hey, we want an array that stores 500 integers." But it's only gonna store integers; it's not gonna store a couple integers and a couple doubles and some strings. All the types are the same so it's homogeneous. Okay?

So how do we actually create one of these things?

We'll erase some announcements. We'll slide this puppy over here. It's actually the first time I'm sliding boards around. And the reason why I always hesitate to slide boards around is all it takes is one really nasty time to get your finger crushed by a board, and you never want to slide boards around again because you can't write for a couple days and it's real fun to do class, especially for you with the left hand.

All right, so how do we actually declare an array? First, we specify the type of the thing we want to have in that array. Then we have an open bracket and closed bracket. That's gonna indicate to the computer that this is an array of this type int. Then we give it a name, so let's just call this - I'm gonna call it My ARR, for My Array. Okay?

And then what we need to do is basically tell the machine, "Hey, create some space for the array for me." Okay? So the way we specify that is we use our friend new, then we say the name of the type again, which may seem a little bit redundant because we just said it over here. But the reason why we have to say it over here again is because when you tell the machine, "Hey, I want you to setup some space for some integers, and the place I'm assigning that to is an array of integers." Okay? So these two types over here actually generally need to match, but you need to have that redundancy.

And then the thing that we specify over here is we actually specify what the size is of our array. So for example, we could have five. So what this is actually giving to us is it's

giving us an array of five integers. Okay? And the way you can actually think about this is an array is basically like a big list that has a bunch of cells in it, or what we refer to as the elements of that array. Okay?

So this is my array, My ARR, and it's spaced for five integers. And the way we refer to these particular integers, or the elements of the array, and each one of these boxes hold one integer, is they're indexed because this is an ordered collection. Right? It's homogeneous because they're all integers, it's ordered, and it's indexed starting from zero. Because as computer scientists, we always start counting from zero.

So if I ask for an array of five elements, they're actually indexed from zero up to the size of the array minus one. [Inaudible]. Just like a string, right, like the characters in a string, where zero up to length of the string minus one. Same idea with an array.

And each one of these things in it is storing an integer. And when you create a new array, the values actually in the array get initialized. They get initialized to whatever the default is for that type. What does that mean? Well, it means for things like integers, the default value is zero. For things like Booleans, the default value is false.

And there's a whole list of what the default values are, but we're gonna just stick with integers right now, so all these things start with the value zero in them. One of the niceties, Java gives you the arrays in this life. So if you've worked with other languages, like C or C++, it's one of those things you have to just let go because the syntax is different, the initialization is different, let go of your old C ways and just adopt the happiness that is Java. Okay?

So how do I refer to these individual elements? The way I refer to the individual elements is through the indexes. Okay? So I could say something like My ARR zero; and I put this inside square brackets, so it's square brackets over here and square brackets over here. And what this is now referring to is this particular cell, which stores one integer. Which means I can use that just like would use a variable of type int anywhere else in my program. I can assign to it, I can use them in use them in expressions, whatever the case may be.

So if I say my array equals five, what it's actually doing is just sticking the value five into this cell. Okay? So any questions about that? Hopefully fairly straightforward? All right.

So the thing you want to think about for these arrays, and what makes them kind of funky, is first of all I get a whole bunch of values very quickly just by specifying the size of the array. The other thing is that this little index that's in here doesn't necessarily have to be a constant value like zero. I could actually put a variable in there.

So I could, for example, have a for-loop that goes through all the elements in my array: for int I equals zero. I, in this case, I'm just gonna size is less than five, which is the size but we'll get to how we actually want to deal with that in just second. I-plus-plus and then inside here I'd say my array sub I which is that particular I element. Maybe I want to

read this, for example, from the user: equals read int, and I'll just, you know, put the little question mark to ask the user for a value.

And so what this does is it goes through and essentially asks the user for five integer values, and stores them in the indexes of the array from zero up to four. Okay? So one quick loop does all the work, right? If I have 300 values, I change this to a 300, I change this to a 300. I'd be good to go, that's all I need to change in my program. So that's kinda the power of arrays. Okay?

So the more general form of arrays, this is how we might actually use them and in this case, I'm just thinking about integer arrays. The more general form of thinking about arrays is sort of like this.

Ooh, a lot of chalk dust.

You'll notice here, we'll just map to kind of general form. This is the type that you want to specify. Okay? Then we have open bracket, close bracket, your angle, then you have the name of your array. Then you say equals new, and you specify the same type again, and end size brackets you have the size, so that's kind of the general form.

So we could have an array of doubles, right? We could have an array of Booleans. We could have an array of strings, for example, by saying string bracket-bracket. I'll call this s-list for string list equals new string. Oh, let's say I want 100 strings. Okay? Life is pretty easy. All right? So any questions about that? All right. Hopefully this is all fairly straightforward.

Now the other thing that's kinda easy is one – or that I should mention is you can have, as you might notice, a string is an object. I could have an array, for example, of g-ovals, or g-rects. So I could actually say something like g-oval, maybe is my type, and then I'm gonna have an array of these things. And I'll call this, you know, my circles because I'm just gonna have a whole bunch of these ovals that maybe represent circles. And then I'll say new g-oval. And here's where things get a little funky. I say new g-oval, and then let's say I wanna have ten of these things, I put a ten.

Notice this looks different, even though I use the new it's different than the syntax for creating one new oval. Right? If I wanted to create one new oval, I would say something like, I'll put it down here, g-oval, and I'll just call this O for oval, equals new g-oval. And then I could give it the parameters for the constructor, like some original X, Y location and some size, maybe 100-comma-100.

What this is doing is creating one object of type g-oval and assigning it to the value O. What this is doing is creating space for ten g-ovals, none of which have been created yet. Okay? So what does that actually mean? Let me make this a little bit smaller so I don't have to draw ten boxes; I'm gonna make this four, okay, but same effect.

When I do this declaration, what I now have is this thing called circles, which is an array. It's an array that's got four cells in it. Okay? And each one of these cells is basically gonna be able to store a g-oval. But each one of these cells right now doesn't actually have a g-oval in it. Why: because I haven't created a new g-oval in each one of those cells. Each one of these cells, or elements, all it has in it is the ability to store a g-oval.

It's kind of like if you just said g-oval, and I'll call this empty. Okay? What happens when I say g-oval empty? I get this thing called empty over here, which is space for a g-oval but I haven't done new on it yet which means this thing doesn't actually point to any object that is a g-oval. Its value, for example, is null. Right? There's no actual g-oval there. I just say, "Hey, get me the box that's gonna store a g-oval in it," but I'm not actually putting any g-oval in it yet.

So when I do this declaration, I'm essentially doing the same thing as this except rather than getting one box, I'm getting four boxes. So each one of these boxes to begin with has a null in it, okay? So I have space to store four g-ovals but I have not yet constructed the g-ovals. So if I actually want to construct the g-ovals, I'm just gonna make use of this little line right here. I could say, for example, circles sub zero equals new g-oval. And what this will now do is say, "Hey, this is circle sub zero over here. Go and create some new oval and stick it in that box." So now, it actually has an oval to deal with. Okay?

Or if you want to think about it from the memory point of view, what really is happening is this is some pointer somewhere that points off to some piece of memory that's storing the information for an oval. Okay?

So any questions about that? That's a critical distinction between creating the array, which is just creating the space for essentially these pointers or references to objects, and actually creating the object. Which you still need to do, you can just create objects in each one of the individual elements of the array if you want to. But if you just create the array, you don't actually get space for the objects themselves, you just get space for the references to the objects. Much in the same way you do when you just declare a variable of some object type.

Uh huh?

**Student:** Is there a more efficient way [inaudible] an array [inaudible] spaces to [inaudible]?

**Instructor (Mehran Sahami):** To create new ovals, for example?

**Student:** Yeah, [inaudible].

**Instructor (Mehran Sahami):** Absolutely. So let me erase this and I will just put in one extra line up here four int I equals zero, I less than, in this case, four I-plus-plus circle sub-I equals new g-oval. So that's the beauty of arrays, right, is the fact that you can use them with objects just in the same way you could do them with ints. And so, you can go

through and, for example, construct all your objects and now you've created a whole – all these ovals are the same, you know, size. Right? They're all zero-zero and they size 100-comma-100 but they're all distinct ovals.

So what I've actually gotten in here is this points off to some oval and this points off to some oval and this, and they're all distinct ovals. They just happen to have the same dimensions. Okay? But that's one way I can create all the objects very quickly if I actually want.

Uh huh?

**Student:**[Inaudible] ovals and then array [inaudible] oval [inaudible] array [inaudible] refer to it [inaudible]?

**Instructor (Mehran Sahami):** Yeah, so once I've created all the ovals, I have all these pointers. You're saying like if I pass it to a function? Yeah, if I – the thing about object is or any time I pass an object, I'm passing a reference. So if I change the object that I'm referring to, I'll always be changing the real actual underlying object.

The important thing that we'll get to in, oh, about ten minutes, is that fact that when we get into primitive types in arrays, I can actually pass an array of primitive types and change the primitive types in the arrays, which I couldn't do before. So that's part of the beauty of arrays. Okay?

So I'm gonna go – give you a brief digression on one topic before you kinda launch into the next thing. And this brief digression is this operator that you've actually been using the whole time with tact and verb, as a matter of fact, we just used it on that board over there. And now you're old enough to sort of understand what it really does.

You're like, "But Mehran, I thought I knew what it did," this little operator called plus-plus. "It was nice, I just added one, I used it since the days of KARO when I had for loops in KARO, I had a plus-plus. I didn't know what it did then. Then you told me what it did and life was good. I could add one to things." Now I'm gonna tell you what it really does and you're like, "It does something besides add one to something?" Yeah, it's this very tiny difference.

The thing with this plus-plus is there's actually a bunch of different places I can put it. And you're like, "Yeah, I can imagine some places I'd want to put it, Mehran." Yeah, no it's not going there. All right? So here is X equals five. That's just terrible and it's on video.

All right, so I have X, it has the value five. Now somewhere in my program, I come along and I say, "You know what?" Let me draw my X five over here so I can have more of my program. My X five, I say X-plus-plus. What is that doing? It turns out plus-plus is actually a method. And you're like, "Whoa, it's a method?" Yeah, and it returns something. And you're like, "Oh that's even weirder."

So what it does is it adds one to this variable, okay, so it turns it into six. But this is actually returning a value, and the value it's returning is the value of variable before you did the plus-plus so it returns a five. Okay? In the past, we just never used that value. We said like `I-plus-plus` and it added one to `I` and said, "Oh here's your return value," and we said, "Yeah, we're not assigning that anywhere, we don't actually care."

But we could actually assign it somewhere, okay? So I could actually do something like say `int Y equals X-plus-plus`, and so `Y` will get the value five, `X` will start off as five, I'll add one to it, it'll become six. And you might think, "Hey, isn't this whole thing six now? Doesn't that get assigned to `Y`?" No. When I put the plus-plus after a variable, I get back the old value of the variable. This form is what we refer to as a post-increment.

Why is it post-increments, because the plus-plus comes after, that's where the post comes from, and it adds one, so increments. So the way you can think of it is I add one but after I return a value. Okay?

So you might say, "Hey Mehran, if there's a post-increment isn't there also a pre-increment?" And you would be right. And the way a pre-increment looks is we take our friend the plus-plus and we put it before the variable. So we say `plus-plus X`. Okay? In terms of the effect it has on `X`, it's exactly the same; it adds one to `X`. So when I say `X equals five`, I put in the five here, then I say –

That looks like just a whole bunch of – let me make that all a bit more clear; I just looked at that and I'm like, "I can't even read what that is."

`Plus-plus X`, so what it does is it adds one to `X` and it returns the value after adding one. So it actually returns six and that's what gets assigned to `Y`. And this is called a pre-increment. And the reason why it's called pre-increment is I do the incrementing first; I add the one then I return the new value. Okay? So any questions about that?

And you're like, "But Mehran, why are you telling me about this?" And the reason why I'm telling you about this is that this little form, this `X-plus-plus` form and the fact that it returns a value, actually gets used a whole bunch with arrays. And I'll show you some examples of that, but you need to – that's why you need to understand what it's actually doing. It's adding one to the value `X` but returning the old value when I do the post-increment. And that's the version you usually see with arrays. Okay? But now you know.

So any questions about post-increment versus pre-increment? All righty.

Then let's actually put a bunch of this stuff together into a slightly larger program. And as part of that slightly larger program, we can discuss a couple more concepts that come up with arrays. Arrays hopefully aren't too hard. Like in the past, every time I've taught arrays, like it seems like, yeah, most people generally get arrays and they generally work the way you think they would work, so hopefully you should be feeling like okay with arrays.

If you're feeling okay with arrays, nod your head. If you're not feeling okay with arrays, and it's okay if you're not feeling okay with arrays, shake your head. All righty, let's go on then.

There's this notion of the actual size of the array versus what we refer to as the effective size of the array. Okay? The notion of the actual size of the array is this is how big I've declared the array to be. So when I setup my array here, the actual size is five. Okay? It's how big it's declared. The effective size is how much of it I'm really using in my program.

So the idea is how much are you really using because sometimes what you do is you say, "Hey, you know what, I'm gonna have some program that computes the average midterm scores. But I don't actually know how many people are in this class," frightening as that may be, right? And I probably won't know until I see how many people actually take the midterm because some people might withdraw or whatever at the last minute. Hopefully you won't but sometimes it happens.

So when I'm writing my program, I can't just stick in some value operator that I know. One thing I do know is there's probably less than 500 people taking the exam, so I can set my actual size to be 500. And then there's some number of midterms that we'll actually grade and I'll have the scores for, and that's how many I'm really gonna be using. That's gonna be my effective size. And I need to keep track of this to distinguish it from this in my program. So let's write a little program that actually does that.

Oh, I think I can fit it all on this board. Let's try anyway.

So what I'm gonna do is write a program that basically asks the user to enter values until they give some sentinel to stop. Okay? So my sentinel: private static final end, you have to say it kinda like you're singing a dirge, sentinel equals minus one. Because I really hope no one's gonna get a minus-one on the exam. I don't know how you would do that. It's like you don't do any of the problems and somehow you manage to like upset the instructor. Right? It's just like, "Okay, you're getting a minus-one. Thanks for playing."

I used to get one free point if you spelled your name right, and someone got it wrong once so I don't do that any more. No joke, it really happened. I think just too much stress or something, or maybe I couldn't read the writing but I was like, "That's is not the name." All right.

So one of the other things that we're actually gonna do is keep track of how large the array's gonna be in it's declared size. So I'll keep track of some maximum size, which I'll just call max size. And maybe I'll set max size to be equal to 500. So I know I'm gonna have at most 500 scores that I'm gonna store.

And so when I'm gonna create my array, I'm gonna create an array whose size is max-size and then keep track of how many elements I'm actually using. So in my program, all

right so let me put this in public void run. And inside here, what I'm gonna say is I'm gonna have some integer array that I'm gonna keep track of.

Actually, there's no space in here, so let me just rewrite that.

Some integer array and I'll call this Midterm; I'll just call it Mid to keep it short. It's gonna be the midterm scores. And what this is, is I'm going to declare, basically, a new array of integers that's size is this constant over here, max size. Okay?

So I set that up and now I get an array of 500 integers just created for me and life is good. And I say, "Oh that's great but I don't know how many scores I actually have," so I want to keep track of the effective size to remember how many scores I have. So I'm gonna keep track of an extra integer in here called Num Scores, and I'm gonna set that equal to zero because to begin with I don't have any scores that the user's given me. Okay?

Now I'm gonna have a while loop that's gonna read in scores from the user. So I might while true, because I'm basically keep reading scores until the user gives me the sentinel value over here and that indicates that I should stop, [inaudible] score, so I'll read one score at a time, equals read int, and I'll ask the user for a score by just putting in a question mark.

Then I say if that score is equal-equal to the sentinel then I'm done. Right? Just gave me a negative one, I know that's not a real score so I can just break. But if they gave me a score that was not a negative one, so it's some valid score, okay, then I want to store that in my array. And so the way I'm gonna store in my array is I'm say mid sub, and here's where I'm gonna get funky with my plus-plus operator, num scores plus-plus.

Let me write that a little more clearly.

Num scores plus-plus equals score. And there's the end of my while loop. Okay? And then my program will go on after that.

So what's going on here, okay? And the reason why I show you this is you will see this on, if you look at real Java programs, this kind of thing happens all the time where you do a plus-plus inside the index for an array. So what happens, this array gets setup, it's got size 500, so zero, one, two, three. I won't write it all out. We have a big break in here. This is 499 over here. All right?

So we have this huge big array and num score starts off as zero. So here's num scores. I'll just abbreviate NS for num scores. It comes in and says read some score from the user, so it's get an integer. And I say, "Oh, someone got 100 on the exam," wonderful thing. Is 100 equal to minus one? No, so I don't break out of the loop. And now I'm gong set midterm sub num scores plus-plus to score. So what's that gonna do?

First it's gonna come over here and say, "What score?" because it always evaluates the right-hand first. That score's 100, that's all you need to do. Where am I gonna store

scores? I'm gonna store it over here. How do I evaluate this? I say, "Okay, I'm gonna store it somewhere in my array Mid," here's the array Mid, "which one of these cells am I gonna store it into?" I'm gonna store it into the cell given by num scores plus-plus. So what do I do? This is a post-increment. I add one to num scores, num scores now becomes one, but what value actually gets returned there from the plus-plus?

**Student:**Zero.

**Instructor (Mehran Sahami):** Zero, right? That's a social, all around, except like there's three pieces of candy right there.

It returns a zero so Mid sub-zero is what gets score, so this gets 100. And I've gotten the nice effect where I've gotten the score stored in the location I want, and at the same time I've incremented num scores to tell me how many scores in this array have actually been entered that I care about. Okay? That's why you see this all the time with arrays, especially in Java or other Java-like languages like C or C++ because it's a shorthand that's so common to say, "Yeah, the next time you get a score put it in the next available spot and increment the number of scores you have."

So the next time I go through the loop, it gets another score. Let's say someone got a 30. Sorry, just didn't work out. Study. You get the 30; it's not equal to the sentinel. Where am I gonna store that? Mid sum num scores plus-plus. Num scores is currently one. I add one to it, I get two, but I return the old value, which is one, and so I stick in the 30 in Mid sub one. Okay?

So num scores here is the effective size of my array. It's how many elements of that array I actually care about using. So if I want to now go compute the average of all the scores, I'm not gonna go through all 500 elements because not all 500 elements have valid scores that I care about. I'm just gonna go through num scores number of scores, the effective size of my array, even though I might have some declared size that's larger. Okay?

So any questions about effective versus actual size or using this plus-plus operator to keep track of things? Uh huh?

**Student:**[Inaudible] num scores [inaudible]?

**Instructor (Mehran Sahami):** Ah, good question. Is that what you were gonna ask back there too?

**Student:**Yeah.

**Instructor (Mehran Sahami):** Yeah, what if like, you know someone just enrolled a whole bunch of times in the class.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** Yeah, I can get to you.

That's an excellent question. What do you think would happen if I try to access, say, mid sub 500? You're like, "I don't know. I get candy?" No. You get an exception. Okay? But this time it's not a little exception, you're getting a big honking exception. And so if I'm getting ready to throw this out somewhere, it's gonna be bad. It's gonna hurt. Okay? So that's an exception you first of all you don't want to get, and second of all, if you get it, you probably don't want to deal with it in your program unless you're writing some very advanced program. You want to figure out that there was some problem with your logic and go fix the problem with the logic. All right?

It's kinda like eventually someone will get this exception somewhere and they're like, "Oh. I am disturbed by your lack of bounds checking on the array." You know, it's just one of those things where you want to check –

Don't worry, I'm not gonna give the whole lecture like this because if I did I couldn't see after a while.

But it's an evil, evil exception to get. It's not that bad but you should be careful of that exception. It's a big one not to be confused with the small exceptions. Thanks for asking. Yeah, it's not a plant. Valid question. Figured someone might ask it.

So let me show you an example of how we actually deal with that in a program. Okay? So in a program, we can get around this in some slightly more interesting way and avoid the exception. So here's a more complicated version of the program we just wrote. Rather than setting a maximum size to be a size that's declared as a constant, I'm actually gonna use – ask the user for a maximum size.

Because let's say maybe I wanna run this program for 106A and then I wanna give the program to some other instructor somewhere and they can run it for their class, and they don't want to have recompile the program. They just want to start the program and say, "Yeah, I don't know how big class is either, but its maximum size is like 4,000," right? It's like ECON1 or something like that. "And so just give me an array."

So here's one of the things that's interesting. I have int max length and I'm gonna ask the user for the maximum size of the array, rather than having it set as a constant. I now declare my array, I'll call it Midterm Scores here, to be a size that's this max length that's given to me by the user.

If you've programmed in another language like C or C++, at this point you should going, "Oh Mehran, I couldn't do that in C or C++." And if you never programmed in C or C++, you should be going, "Yeah Mehran, I don't care what you can do in C or C++ because I'm programming in Java." And Java allows you to declare an array like this where the size is actually something that's dynamic, that's only known at the runtime when the user enters it. Okay?

Here's my number of actual scores, that's gonna be my effective size for the array, how many elements of the array I actually care about. And now rather than having a while loop which may allow me to just potentially go off the end of the array, I'm gonna have a for loop and this for loop is gonna count up to the maximum length. So it's gonna guarantee if I ever get to the maximum size of the array, I'm gonna stop asking the user for any more values because I will have exhausted the for loop.

But if I haven't yet exhausted the for loop, what I'll do inside here is I will get a score from the user, so I'll read in the next score into midterm score, if that score is the sentinel I will break out of the loop. If the score's not the sentinel, then I will increment my number of actual scores and iterate this over and over. Okay?

Now the interesting thing here is if the user actually enters the sentinel before they get to filling in all the elements of the array, the max length, I break. Break always takes you out of the closest encompassing loop. So this whole time you've been using break with, for example, while loops. This will actually break out of a for loop; you can break out of for loops as well.

It's not the greatest style in the world but sometimes in rare occasions like this, it actually makes sense because you wanna say, "The most I'm ever count up to is max length. Guarantee me that I'm gonna go any higher than that, otherwise I'm get the Darth Vader exception and life is gonna be bad, but I might break out earlier if the user actually gives me the sentinel." Okay?

And I'll only increment num scores if they didn't give me the sentinel, so I'm not gonna count num score – I'm not gonna count the sentinel as one of the values I actually care about. Even though it gets stored in the array, I'm not actually gonna say that's a valid value because I break out before I actually increment num scores.

Okay, any questions about that? Uh huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** If they enter a negative score for the maximum? Well, try it and see what happens. All right? It's not never happen in real life and so if – ooh, if you're really interested –

I might just peg her because it makes me angry. I can't reach you but I'm gonna keep trying.

Try it yourself and see what happens. It'll probably give you some something that won't work, right?

So what's something else that I want to do? And I'll get to do that right now but let me do it on the board before I show it to you in this program because there's something else I might actually want to do in the program. What happens if I pass an array as a parameter?

Right? So this whole time we've been talking about arrays, we've sort of had the array inside your run method and life was good. And we were like, "Oh array. Yeah, I can do all this stuff with it."

What if I want to pass an array, which is a common thing to do, as an actual parameter to a function? Okay? So let's say I have public void run. Inside here, I have int bracket bracket. I'll call it ARR, for my array, equals new int ten, so I get an array of ten elements. And now I wanna pass this array. I wanna say, "Hey Mehran, rather than reading in all the values from the user in your main program or in run, I wanna decompose the program. I want to have some function that actually reads all the values for me. Can I do that?"

Yes, you can. And so let me have some method called read array, and what I'm going to pass to it is the array. When I pass an array, if I wanna pass the whole thing, I just specify the name of the array. I do not specify an individual index on the array because if I specify an individual index I would just be passing a single element of the array, which would be a single int. If I want to pass the whole array, I give the name of the array with no other brackets. Okay? So this is gonna pass array method.

So somewhere down here, let's just say that's the end of my run method, maybe I have some more stuff in here, I have private void read array. And read array is gonna take in some parameter. How do I specify a parameter array? I say int bracket bracket. That indicates the thing that is getting passed to me is an array of integers. Okay? And then I give it a name for the parameter. Here I'll call it A, just to distinguish it from ARR. All right?

So inside here I get A, and so I'm gonna have some for loop inside here, for int I equals zero. And you might say, "Oh Mehran, how do you know how big the array is? How do you know how many elements to ask the user for," right, "because you just passed the array? Where is the size of the array? You didn't specify – if you had it specified as a constant, like you did before with max size, then I could refer to that constant over here. But you didn't specify it with a constant, so what do I do?"

I would normally give you concentration music and let you decide. But in fact arrays, and this is one of the nice things about arrays in Java, they carry around their size with them. Okay? So what I can say is array dot length. Okay? And the funky thing here, to distinguish it from strings, when you ask the string for its length you put in an open brace or an open bracket or open paren/closed paren. Here you don't specify open paren/closed paren. Length is actually some internal variable of an array that is public so it is available to you.

So you're actually referring directly to this public variable of the array. So array dot length, I want to say actually I is less than array dot length, so let me set this to be zero. I is less than array dot length, I-plus-plus. And so if I pass arrays of different length here, their length, their declared length, right, I don't know it's effective size is, it's declared

length is ten. And so what I might do inside here is say: A sub I equals read int, to get an integer from the user. And that's supposed to be a question mark inside double quotes.

Now you might see this and say, "Oh Mehran, I thought you told me when I pass around primitive types I get a copy of the primitive type. Right? So when I pass an int, I get a copy of the int, and if I try to muck with that int, I'm just mucking with the copy. And so when I return to my calling – the location from where I called that particular method, that value hasn't changed." Yeah, that's for primitive type not for an array, even if the array is of primitive type.

So the thing you want to think about passing arrays is arrays always get passed by reference, just like object. If you pass an array, you're getting the actual array because it says, "Hey you know what, arrays are big. I don't want to copy ten integers. I'm lazy. I'm just gonna let you know where that array actually lives in memory." And so, when you make any changes to that array, you're actually changing in memory at the place where this array actually lives.

So it does it mainly for efficiency reasons, right? Because if this was an array of a million elements, if you got a copy you would have to copy a million elements every time you made this call. Rather than copying them it says, "Hey, I'm just gonna let you know where those million elements live. You want to mess with them, you mess with them. You're gonna change the actual one. If don't want to mess with them, then don't mess with them."

Uh huh?

**Student:**[Inaudible] variable is public, can you change the length [inaudible] code?

**Instructor (Mehran Sahami):** You shouldn't try.

**Student:**I mean I just wondered if that would happen if you actually did that.

**Instructor (Mehran Sahami):** Yeah. I wouldn't encourage you try it. I would just say don't try it; it's a bad time. But if you really want to see what happens, try it. Just it's never something you would want to do.

Uh huh?

**Student:**Can arrays be ivars?

**Instructor (Mehran Sahami):** Yeah, arrays can be ivars. So if you want to have an array declared inside a class and actually store an array inside a class, that's perfectly fine. So arrays you can use just like any other variables, you just get a whole bunch of stuff. Okay?

So but the important thing is any changes you make to the array, the changes persist; arrays always get passed by reference. Okay? So at this point, you could also say, “Hey Mehran, that means if I wanted to change around some integer, and I just had a single integer, could I actually have an array of one integer and pass that array of one integer and change that one value, and I could change an integer that way?”

Yeah, you could. It would be horrendous style but you could do it. Okay? So don't do it in this class and don't ever do it as a software engineer because if you do this as a software engineer, people will look at that and say, “Where did you learn this?” And you'll say, “Oh, that Sahami guy at Stanford told me this.” And then they'll come and hunt me down, and kill me. And that's bad; at least for me, may not be for you. Don't do it. It's not how we try to pass integers by reference.

Uh huh?

**Student:** Can you make arrays of arrays? Like if you wanted to add all of our test scores divided by [inaudible] tests.

**Instructor (Mehran Sahami):** You can and you're actually one day ahead of me. So next time, we'll talk about multidimensional arrays, where you can have arrays of arrays of arrays of arrays. And it can just – till the cows come home. You have like four-dimensional arrays. You can have like 12-dimensional arrays and just expand your view of the universe. It's like string theory with arrays.

Yeah, the other dimension's just wrapped so tightly you can't actually stick integers in them, but that's not important right now. There's like a few string theory people who are like, “Ho-ho-ho, that's funny Mehran.” Everyone else is like, “What?” All right, so not that I'm a string theorist, but I like to play one on TV.

All right, so what else can we do with arrays? If arrays, when we pass them around we're actually passing references to arrays that means other funky things are sort of possible. So let me show you another example of something that's possible, if we return to our program. This one's not so funky. This one just says, “Hey, what I want to do is after you give me all those scores I want to compute an average score.”

So I'm gonna have some method where I want to pass to this method is the Midterm Scores array that I had before. Right? That's just this array that we declared up here. And I also want to pass to you the actual number of values that I'm using, the effective size of the array. The reason why I'm doing that is because if I don't pass you the effective size of the array, you don't know the effective size of the array. The only thing you know that the array gives you is declare its size; it doesn't keep track of, oh, which element you're using or which element you're not using.

It says, “Hey, I know I'm this big. And if you're not using the whole size of me, then you do need to pass the effective size.” So inside compute average, it's getting passed an array of integers and it's getting passed the act number – actual scores that we care about.

And it just goes through a loop that counts up all the actual scores, adds them to this thing called Average that we're sort of adding up as we go along. Right? So it's adding all the individual elements of the array to average.

And then when we're done going through all the elements we care about, we just set average equal to average divided by num actual scores. And because average is a double, it does real value division so we get the average and we return the average here. Okay? And so all this prints out is average score and then it will actually print out what the real average of those scores were a sum double. Okay?

So important thing to keep in mind, even though you get this facility where you know the declared size of the array, sometimes you still need to pass the actual – the effective size, if you have – if you're not using all the elements of your array. If you're always guaranteed that you're always gonna have, you know, “Yeah, this is the Olympics and there's always seven judges. And I know it's always gonna be seven and that's just the way it is, unless there were some kind of payoff scandal,” then you know it's seven, the declared size and the effective size are the same. Okay?

One of the other things I can also do – now in this case, notice – in this array example case, I didn't actually change any of the values inside the array. Okay? Even though I didn't change any of the values, the array still got passed by reference so I still had the real array. If I changed any values, they would've persisted. But I still get a copy no matter – or still always get the actual array, I don't get a copy whenever I pass it whether or not I change the values. In most cases, you don't change the values, which is why it's actually more efficient and I'll give you a copy to give you the actual thing.

Sometimes what I might want to do is say, “Hey you know what, I want to swap two values around.” And so you might think, “Hey you know what, if I have some array, here's some integer array that I declare here that's got five elements in it. And I set its first two values to be one and two, respectively. So array sub zero is one, array sub one is two.” Okay?

Then I say, “Hey, I want to swap the two values.” So I have some function that Mehran wrote for me, some method called Swap Elements Buggy. And why did he name it that? Yeah not because it's a little baby that's inside a little stroller that we push around, that in the days of yore we used to call a buggy, because it's got an error. This not the way you want to do it. And I do want to make it explicitly clear so this is the buggy version.

The buggy version says, “I pass in these two elements of the arrays as the two parameters.” Which means when I want to do swap, here's the buggy version, the buggy version says, “Hey, I get passed an integer X and an integer Y.” Why do I get an integer X and an integer Y? Because the elements that I passed in were array sub zero and array sub one, those are integers. They are primitive values. I'm not passing the whole array; I'm passing individual elements of the array. The individual elements of the array are primitive types, which means we go back to our old case of passing copies.

Anytime the things here are just straightforward ints, rather than being an integer array, for example, all you're getting are copies. So I get copies of array sub zero and array sub one, and I go through this little exercise where I swap them. So I have some temporary where I put one of them in, then I take that one and I replace it with the other one, and then I assign the temporary back to the other one. It's not a big deal how it actually works.

As a matter of fact, let me just draw it for you, how it works so you know how it works and we're all on the same page. I get X and Y, they have values, let's say four and five to start with. I have some temporary. I start off by saying: temporary equals X, so temporary gets the value four, then X gets Y, so X gets the value five, then Y gets temp so Y gets the value four. And for all intents and purposes, it looks like I swapped the values. Good times, now I'm done with this function and this all goes away because they're copies and they're all temporary. And they just vanish and nothing is actually changed in the array because I passed the individual elements of the array, which are primitive types.

So if you pass primitive types, even if they're individual elements of an array, you are still getting copies. If you really want to do the swap, the right way to do it, is Swap Elements Happy because it will make you happy. What you do is you say, "Hey, I'm gonna pass the whole array." Once you get the whole array, you've gotten a reference to the array, you can change stuff around in the array and it will persist. So you say, "Hey, I'm gonna give you the whole array, and I'm gonna give you the two positions of the elements in the array that I want to swap."

So when I actually do the swap here I'm doing the same logic, except I'm actually doing the swap with values in the array rather than copies of values that are passed in as primitives. So this version actually works because I'm modifying the array itself.

Okay, and questions about that? Uh huh?

**Student:** If there are objects [inaudible]?

**Instructor (Mehran Sahami):** If they're objects are actually getting references to the object so you don't need to do the same thing. It's just for primitives that you need to worry about it.

So let me just run this program just to show you, in its painful detail, what it's actually doing. We're running. We're running. Oh, it's – sometimes it's just slow and painful, but at the end of the day, you know, we get the little warm fuzzy. So I want to do the Swap Example. And so the Buggy Swap, notice Element 1 is still – or Element 0 is still one, Element 1 is still two. Happy Swap, they've actually swapped in place because I passed the array, which was a reference.

Okay, so any questions about that? If that all made sense, nod your head. Exellente.

All right, so a couple more quick things. One quick thing, this is super-minor. You'll probably never end up doing this in your life, but if you see someone else do it you'll know what they're doing. Which is you can create an array that's initialized with particular values you give it by saying `int bracket array`, so we have some array of integers. And rather than specifying new or specifying some size, I just say, "Yeah, this array actually has the values two, four, six, and eight in it."

So inside braces I put the initial values, from the number of values that I give it, the values are separated by commas, it figures out, "Oh, this is an array of four elements, so let me create an array of four elements for you, stick in those initial values to initialize it, and you're good to go from there." You virtually never see this, but just in case you do, now you know what it's like. All right?

So it's time for one last great topic. And this last great topic, I'm gonna give you very briefly just enough for right now that you need to know to do your hangman program. On Wednesday, we'll through all this is excruciating detail. So if you're not starting hangman before Wednesday, you're perfectly fine. You'll see way more detail than you need for hangman on Wednesday. But just in case you were like, "Hey Mehran, yeah I'm studying for the midterm, and right after the midterm I'm totally pumped up. I'm just gonna go do all of hangman," this is what you need for Part 3 of hangman.

So Part 1 and 2 you already know everything you need to know. Part 3, there's this thing called the array list. How many people have already gotten to Part 3 of hangman, by the way? Yeah, just one person in the back; did you read about array list yourself or have you not done it yet?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** I can't hear you but that's okay. If you've already gotten to Part 3 you're so far ahead you just don't even need to worry. It's good times. Just come after class, I'll give you all the chocolate you want. All right.

So array list. Because these array things are sometimes bulky to work with, especially when you think about, "Oh, I need this effective size versus actual size, and wouldn't it be great if when I needed more elements the array just grew for me?" So I said, "Hey, I start off with no elements and I read in like five values for midterm scores and I kinda store that." And I say, "That's great." Oh, I got these – I found these like late midterms somewhere, well there's five more I need to add.

And the array would just come along and say, "Oh that's fine. You've been so good to me this whole time, I'm just gonna expand my size to give you more space." Like wouldn't that be great? And you're like, "Oh yeah, that'd be so good." Like the birds would be singing and everything. But real arrays don't do that. Yeah, sorry, birds all been shot down.

Array lists bring the birds back to life and now they're singing again. It will do this for you, so what you need to know about an array list. First of all, what you need to do is you need to import java-dot-util-dot-star because this thing called an array list comes from the Java util package. Okay?

Once you do that, you get access to our friend the array list. And the array list as described in the book comes in two flavors. It comes in the super-cool Java 5.0, also known as Java 1.5. Because sometimes the numbers for Java is 1-point-something, like 1.x and sometimes it's just X; and it's super confusing but that's the way it is. So Java 1.5 is the same thing is the same thing as Java 5. And you're like, "That makes no sense at all." Yeah, it doesn't. That's just the way it is.

So there's the super-cool Java 5.0 or later version, and then there's yes still kind of useful but not quite as cool pre-Java 5.0 version. And the only thing you need to know is the super-cool Java 5.0 or later version, so you don't need to worry about the pre-5.0 version.

So in the super-cool 5.0 or later version, an array list is something called a template, which means when I'm creating an array list, an array list is actually a class that I specify the type that that class is gonna operate on, which is kind of a funky thing. But the way I declare an array list is I say: array list with AL because this is actually a class that exists inside Java-util-dot-star. And then I use essentially the less-than sign or what we sometimes refer to as angle brackets if we're not using it as a less-than sign. And then the type that I wanna store in this array list.

So essentially, I wanna have an array list of strings. And you can kind of think of this almost like an array of strings but cooler. And so, this is inside the less-than/greater-than signs. Okay? Then you give it the name; I might call this STR List, for string list. And then I say that is a new, and here's where the syntax gets kind of very funky, array list angle bracket string close angle bracket, open paren close paren. And you look at that and you're like, "What? Why would I do that?"

That's just the way the syntax is. What this is saying is I'm gonna have an array list. What am I gonna store in the array list? I'm gonna store strings. What's the name of my variable that is that array list? It is STR List, so that's the name of the variable. The type is an array list of strings. Okay? Once I set this up, all I get is my little space for a STR List and so I need to actually say, "Hey, create the new array list for me."

What kind of array list do I want? I need an array list that stores strings. I'm calling a constructor that takes no parameters. So I still need to specify an argument list for that constructor, which is where the open paren/close paren comes from. So the name of the constructor, the way you can think of it, is actually array list with the type specified, as funky as that may seem, and then the parameters are nothing. Okay?

And so what this now gives me is STR List point somewhere in memory that stores my array list, which is some object that's gonna store strings in it. Okay? How do I use that? Let me show you a real quick example.

So because an array list is an object, it has methods on it. Okay? One of the methods I can do is I can say STR List dot add. And what add does, I would give it some string. So I might have some string line that is Hello, and I add line. What the add method does is it adds that element to the end of the array and grows the size of the array by one. So if I want to add a whole bunch of things to some STR List, I start off by creating a STR List, which is empty. So when I do this I get an array list, which is empty called STR List. And as soon as I do this add it grows to size one, and it now has the line Hello as the first element.

If I were to do another add, like say STR List dot add, and maybe I say There as another string and now it grows to size two and There is the second element. So add adds it, appends to the end, and grows the size by one.

Some other things I can do with STR Lists. I can ask STR Lists, or array lists in general, for their size. So there's a method called size open paren close paren. What this does is it gives you back as an integer how many elements are inside your STR List based on how many you've added. Okay?

Another thing you can also do is you can say, "Well, it's great to be able to add all these elements, Mehran, but how do I actually get information out of my array of STR Lists?" What you say is: STR List dot get. So there is like getter, much like there's a setter, and you pass it in an index, I. And so, what this says is get the element at the I position. So if I've added line here, which is Hello, that's a position zero. When I add There that's a position one. You can think of it an array that's just dynamically growing as I add more stuff to the end. And the way I refer to the individual elements is with get and I still specify an index that starts from zero.

Okay? And this is pretty much all you need to know to be able to do Part 3 of hangman. And you'll see it in much more excruciating detail on Wednesday, but that's all you need for now.

Any questions about array lists? All righty, then good luck on the midterm, I'll see you tomorrow night.

[End of Audio]

Duration: 52 minutes