Programming Methodology-Lecture17

Instructor (Mehran Sahami): All right. So welcome back to yet another fun filled exciting day of cs106a. A couple quick announcements before we start. First of which, there is one handout, which is kind of a quick reference for you on ArrayLists. It's sort of all the ArrayLists you need to know for the hangman assignment part three. If you haven't already done it, now you have a quickie reference for it. If you've already done it, you don't need the quickie reference, but presumably, you saw everything you needed already in the textbook or what we did last time in class. The midterms, it was just a rocking and rolling good time. If you weren't there [inaudible], but two things, if you took the exam, please don't talk to other people about the exam because there are still some people left to take the alternate exam; and if you're taking the alternate exam, you missed out. You'll still take the alternate exam, so you'll get the exam, you just won't get the full effect of the exam under earthquake conditions, but thanks for actually just – it was amazing, it was just like how many people want to continue and everyone was, like, rock on. So thanks for continuing with taking the exam. I'm glad you made it through just fine. Last thing, a little contest. I'm glad to see there was other people besides us – we were a little worried it was gonna be like me and 300 normally dressed people, this is not what I normally dress in around the house, just when I'm in meetings on campus. So the real question is who am I? Anyone know? It's a very specific person. Not King Arthur. What? Student: Sir Lancelot.

Instructor (Mehran Sahami): Not Lancelot. Student: [Inaudible]

Instructor (Mehran Sahami): I almost considered Karel. I almost went for Karel. I got the big box, but it's kind of hard to draw, sort of like, Karel cannot draw that much in the big box. No. Well, I'll give you hints as we kind of go along, and at the very end I'll show you if you can't figure it out. I almost considered giving whoever could figure it out a 100 on the midterm, but I figured that was kind of just a [inaudible] way of giving you a 100 on the midterm, so I'll just give you lots of candy. So anyway, just think about it, and if you want to try to search the web or whatever, it is a one particular character, and it's not a wizard. I actually thought, "Oh, I could just make it easy for you and just come in and be, like, oh, ha, ha," because most people, without the hat, would see it and they'd go, "Oh, you're a wizard, right?" No. So I just brought the hat to kind of fool you. All right. So with that said, time to actually wrap up one of our last topics, which was arrays. We'll look at a few more advanced things we can do with the arrays and then get into the details of our next great topic, which is our friend the ArrayList. So when we talked about arrays, a couple people last time said, "Hey, Miron, can I have multi–dimensional arrays, can I have arrays that have more than one index, and in fact, you can. We do refer to these things as multi-dimensional arrays. And so one way you might want to do it is let's say you wanted to have this dreaded thing called a matrix. How many people know what a matrix is? I'm not talking about the bad movie. Actually, it was a good movie. The sequels, not so great.

It's basically just a grid of information. It's a grid mostly of numbers if you want to think about it in a mathematical sense, but really we think of this thing as a two-dimensional array. And so the way you can think about is if I have a matrix that's a [2] [3] matrix what that means is it's a grid that has two rows and three columns. Okay. And you might look at that and say, "Hey, that almost likes an array, it looks like two arrays stacked on top of each other," and in fact, you would be correct. If we have multi-dimensional arrays, what we're actually creating our arrays of arrays. So really what we want to do here, if we want to create this thing called a matrix, is we specify the type that we're gonna store in all of the boxes in that grid, and then we specify as many pairs of square brackets as we have dimensions of that thing. So if we have two dimensions here like we have rows and columns then we specify two open and closed brackets in a row, if we had three like we wanted to make a little thing that represented a cube we'd have three sets of brackets. So here I'm just having two because I'm just gonna show you two arrays so you can think about the generalization from there. And then we give it a name so we might call this thing a matrix. And then we specify basically the creation of that matrix in the same way we created an array except now we have more than one dimension, so here we say new, we specify the type again, like, indices and then to give it the size rather than just having one size that we specify, we have a size for every dimension. So if this things gonna be [2] [3], we specify the first dimension as two and then the second dimension as three. And each one of these is its own pair of square brackets. We don't use comma notations, so if you've worked with some other language or you say 2, 3, nuh uh. It doesn't work in java. It's [2] [3] no spaces immediately after it. And so what this does when you create that is it sort of creates this thing that's this grid that's a [2] [3] grid. And now you can refer to the individual elements of that grid, so the way you can think about it is this is element [0] [0] or really you want to think of them in brackets, and then this becomes element [0] [1] and this is element [0] [2] and so forth, so this is element [1] [0] and this is [1] [1] and this is [1] [2]. So if you kind of think about assigning to elements of that you could say something like matrix [0] [1] = 5 and what that will do is it'll say, "Oh, where is [0] [1], here is [0] [1]."

Well, what I'm gonna do is, in that particular cell in your grid, is just gonna stick in the value five. And you can use that like an integer anywhere else you could use it like an integer or a single element of an array. Okay. The other thing that's kind of funky about this is that this two-dimensional array is really – the way you can think of it is it's an array of arrays, right? It's an array – think of it this way as an array. It has two entries in it and what is each one of those entries of the array, well, really every entry of the array you can think of this as one cell, that whole thing, and this whole thing as another cell and each entry of that first dimension of the array is just another array with three elements in it. Okay? So what that allows you to do is say something like matrix subzero and matrix subzero, the type of this thing, is an integer array. A single dimensional array because what is matrix subzero, it's this full line here. It's the first row of the entire matrix, and then if I specify something else, like I specify a column, that's how I get [0] [1]. First I specify the row, and then I specify the column. If I don't specify a column this is referring to the whole row, which means it's just referring to a whole array.

Student: Can you refer to [inaudible]

Instructor (Mehran Sahami): You can't. You have to refer to things in order, so there's no way to just pull out a column, so if you're feeling sort of like, "Oh, but in my Math 53 class we do all this stuff with columns." Yeah. You just have to pick, in some sense, what's gonna be your major ordering for this thing and have that one be first. So you could kind of think of it in your head that, "Oh, well, my first index is really my columns and think of this thing as being the transposed and all that," but you just have to keep track of it in your head. Okay? Any questions about that? All right. So we can write a little code that makes use of this just so you see a quickie example. And we could, for example, go through all the elements of this grid and set our initial values to be one, so we could have something like I = 0. "I" is less than – what you want "I" to be less than in the first index is 2. I ++ and then you're gonna have some other four loop. Let's say this is J = 0, J is less than 3 because that's the size of my second index, J++. And then inside here I could just say something like matrix I sub I, sub J = 1. And that would initialize all the cells in my grid to be one. Okay? Is there any questions about that? Hopefully fairly straightforward kind of stuff. You can have as many dimensions of arrays as you want sort of within reason, and within reason being how much memory your computer has right because if you imagine, "Hey, I don't only want a [2] [3] matrix, I want a [2] [3] bi-hundred matrix," so this becomes a three dimensional array. Right? So I basically have this grid repeated a hundred times going forward. Suddenly, you're using up a whole bunch of memory. Right? So you want to be careful doing that, but if you wanted to do that you would just add another pair of brackets here and you'd add another pair of brackets over here with a hundred and now you would have a three dimensional array. So if you want to create, like, your friend the Rubik's Cube or something, with is [3] [3] [3], now you know how. Okay? And if you want to do real funky things with four and five dimensional objects or whatever, be my guest. Okay? But we're not gonna use them a whole bunch in this class, so I might show you some examples of two "D" stuff momentarily, but most of the arrays we're gonna deal with are just what we refer to as one dimensional arrays, right, which is just a list. All right? So any questions about multi-dimensional arrays? If you're feeling okay with multi-dimensional arrays nod your head. Good times. All right.

So moving on from the array we can now go back to our friend that I briefly introduced last time, and now we can go into excruciating detail, our friend the ArrayList. Okay? And the ArrayList – the way you can think about is it's an array just kind of cooler. Okay? So the idea behind an ArrayList is an ArrayList first of all really is an object, and it's an object that's provided for you in the Java Util Package. We talked a little about this last time. I'm just repeating it just for good measure. So you import the Java Util that's star packet to be able to get at the ArrayList class. Okay? And the way you can think about an ArrayList is it's really like an array, except it's an array that dynamically changes its size depending on how you add elements to it. So you can just say, "Hey, add another element to the end," and it will just go, "Hey, you know what, I'll just create extra elements at the end for you and now your size is one greater." And so the way to think about it is in terms of effective and actual size, that we talked about last time, was with regular arrays you set up with the actual sizes that's declared, and then your effective size is how much of it that you're using. With an ArrayList, think of it as the effective size and the actual size, at least stylistically, are exactly the same thing.

Whenever you want to add another element to the end, its effective and actual size both grow. Okay? Now underneath the hood, the implementation actually involves some stuff where the effective and actual size can be different, but that's all abstracted way for you. As far as you're concerned, effective and actual sizes are basically the same thing. Okay? And it provides some other functionality that is nice, and I will show you that in just a moment. But this particular ArrayList thing is called a template. I'll give you a little history of templates. I mentioned this briefly last time, but I will mention it again. Starting in Java, the book refers to it as Java 5.0. A lot of programmers will say Java 5.0. Some programmers will say Java 1.5, and what you have installed on your computer is either Java 1.5 or Java 1.6, which some programmers would say is either Java 5 or Java 6, which you would look at and say, "Hey, man, that makes no sense at all, why is it like that?"

So I'll give you a little bit of history. Quick history lesson for the day. Many years ago, oh about 1995, there was this thing called Java 1.0. It came out, people were happy and went, "Oh, nice," and then Java 1.1 came out and people looked at that and said, "Yeah, nice," and then Java 1.2 came out and people said, "Nice!." So when they said, "Nice!," 1.2 in some people's mind was such a major change in the language Java that they started calling it 2.0, and at that point when 1.3 comes out you can't say, "Oh, well, that's 2.1," right, because then it gets really confusing so then when 1.3 came out people called that Java 3.0. And 1.4 was 4.0, 1.5 became 5.0, 1.6 became 6.0, but in fact, the confusion still exists so if you go and look in your system most of you on your system it will actually have this 1.X number. Most programmers will actually refer to it by just the X as opposed to the 1.X. Don't be confused. They're really referring to the same thing. Okay? So what we're dealing with here, in terms of ArrayLists as templates, is Java 5.0 or later so sometimes that's just written as 5.0 plus, which all of you should have because that's what you installed in the beginning of class. The book actually shows both the pre-5.0 and the post 5.0 version. We're just gonna deal with the post 5.0 because at this point, the older version is just not worth worrying about. Okay? So what is this thing called a template that exists in Java 5.0 and later versions? Okay. What a template is is it's basically a class that allows you to specialize it based on the type. What does that mean? Okay. So let me give you a few examples of what that actually means. What that means is when you see methods for this ArrayList class, and you'll see them listed in the book, you'll see something that looks like this. There's a method called Add and it returns a bullion and you're like, "Oh, that's all good and well," what kind of parameter does Add take and you'll see the spunky thing that looks like this, [T] element, and you look at that and say, "Oh, Miron, you never told me about the [T] type, what type is that?" And the way you can think about it is this is a parameterized type. When you create an ArrayList you say, "I'm gonna create an ArrayList of a particular type." So when you set up your ArrayList you create an ArrayList object. And when you create that object, as you saw last time very briefly, in angle brackets, which are actually less than and great than signs, you specify a type like string. So you say ArrayList string and we call this thing, for example, SList and we might create that to be a new ArrayList. Again, we specify the type string () because we're actually calling the constructor which takes no arguments. Okay? So you create this new ArrayList and this thing that you specify here, the type that you specify there, is what we refer to as the type or the template.

You can think of the template as having all these holes in it, and the way those holes get filled in is they get filled in with whatever type you give it here. And those holes are specified by this [T]. The way you can think of this T is it's the Type, that's why it's a T that this template is what we refer to instantiated as. You create one of these templates of this type. So I create an ArrayList of strings is the way programmers refer to it. Usually, you can think of this as meaning of and this as meaning nothing. So ArrayList of string is how we would actually say it. And when you create an ArrayList of string you get an Add method for this ArrayList of strings that's now [T] is replaced by string. So the method that it actually provides for you is an ad method that takes a string type. Okay? If I were to create an ArrayList of G ovals I would have an Add method for that class that's elements was a G oval. Okay? So it creates for you basically a whole class that has all the holes filled in with whatever type you specify here. Any questions about that? Kind of a critical concept, new in Java 5.0. It's new, it's dynamic, it's cool, it's funky. People like it, now you can, too. All right. So let me show you some of the methods that you get with this ArrayList kind of thing, and then we'll actually go into the details of using it. And you'll notice that when we talk about the methods, this [T] thing will show up all over the place. And the way you want to think about that is this is just getting replaced with whatever type you instantiate your ArrayList to be. So if we go to the computer real quick, drum roll please, it's the methods of the ArrayList. So what are the methods of the ArrayList? The first two methods are Add. What Add does is it takes in some element type of whatever type your ArrayList is instantiated to be, right, the [T] and it adds it to the end of the ArrayList. And you might say, "Hey, Miron, what does that return a bouillon," and it always returns true for the bouillon, so why does it return a bouillon at all? And about in a week it'll be clear why it always returns a bouillon. Okay? And why, in this case, it's always true, but as far as you're concerned you don't really care about the return type. You're not gonna do anything with it because it's always true. You're just gonna say Add and even though it's returning something to you, you're not gonna [inaudible]. It just adds an element to the end of the ArrayList, which means the ArrayList dynamically grows by one. Right? It's an array with dynamic size, so it just creates the space for you and adds the new elements to the end. You can also add in a particular position just like you could add an array at a particular position, you can do that in an array list by specifying the index and the element that you want to enter at that index, and it will just put that element in at that index for you and it's just kind of all abstracted in terms of how it does that. Okay? It's just growing its size as needed. Besides adding things you can remove from the ArrayList, right, which is something that's funky. You didn't really think about removing from an array. Right? An array you could sort of say, "Yeah, read that value and maybe I don't care about the value anymore," but you couldn't actually say, "Yeah, just get rid of that value completely, like, get rid of the space associated with that value." Okay? So when you say remove at a particular index, it actually removes that element and returns it to you from the ArrayList.

And similarly, you can either remove or you give it an index and you say, "Give me back that particular element," or you can remove a particular element if it appears. So this returns a bouillon. So let's say you have an ArrayList of strings and one of your strings may be cs106a, and you're like, "Oh, cs106a, it was so near and dear to my heart until I had to take a midterm in the middle of an earthquake," remove cs106a. And then you try to do this for everyone on campus. Well, some people have 106a and if it appears it'll return true and remove it from the ArrayList, but some people don't. And it doesn't suddenly crash their program or whatever. When you come to that person and you call remove on cs106a, it just returns false to indicate, "Hey, cs106a wasn't in this ArrayList, I didn't make any changes to the ArrayList." Okay. And similarly, if you just get really angry or you decide, "Oh, I'm gonna take some time off from Standard, I want to just clear my ArrayList of classes," you can call "clear" which just removes everything in the ArrayList. It just kind of resets it to be a blank ArrayList. Okay? You can ask an ArrayList for its size, not particularly exciting, but very important to actually be able to do. It just returns how many elements are currently in the ArrayList. Besides putting elements in an ArrayList and removing them it wouldn't be much fun if you couldn't actually get an element, so getting an element at a particular index returns the object at that specified index. It doesn't remove it from the ArrayList, it just gives it back to you. So if there was an object or some element at position one, okay, and you call "Get" it remains at position one. If you call "remove" it's actually removed from the ArrayList. Okay. And then there's set, and so set you can actually say, "Hey, at some particular location I don't care what's there, just slam this thing I'm gonna give you." So what it does is you give it a particular index and you give it some particular value, and it puts that value at that index. So you can say, "Set at position one, you know, this string 106b because now after you've taken 106a, maybe next quarter you're taking 106b, and it goes, "That's great," and what it gives you back is the old value at that index. So it would give you back 106a if that was the old value that was there. Just in case you want to do sort of the quickie slight of hand change, presto change, stick something new in, get the old value out. You can do that if you want. That's just the way it is. Last but not least, a couple of things real quickly. You can get index of. This is kind of funky. This essentially does a search for you on the ArrayList. So you give it some particular value, like, you could say, "Hey, is 106a actually in my ArrayList," so you would pass it to string 106a and it would return to you the index of the first occurrence of cs106a if it was in your ArrayList and minus one otherwise. It's just kind of like the index of operation that you did on strings before when you called it with like [inaudible] or substrings.

It's the same thing here except now you have a whole list of things, and that whole list of things can be any type that you want. Contains, the same kind of thing. It basically just returns a bouillon. You give it some value, and it says, "True or false, this value actually appeared in the list and is empty." It just returns true or false to let you know if the ArrayList contains any elements at all. Okay? So fairly generic, basic functionality and it's all kind of good and well and we're happy. So let's use some of that and actually write a program that makes use of an ArrayList together so we can see how some of this functionality actually looks, and then we'll try it out on the computer and try a few more advanced things with it. So here's our friend the Run Method. We got public, void, run; inside here we're gonna create a new ArrayList of strings. So I have ArrayList of string, ignore that part, although you need it for the syntax, but we just don't say it as programmers. We say ArrayList of strings. My string list, I'm just calling it SList, is a new, very similar to what you saw there, ArrayList that holds strings and there's no parameters for the constructor, so I make sure to have the () even though it looks kind of funky. You have to have it. So I just create one of these, and now what I'm gonna do is just repeatedly read strings from the user until the user enters a blank line to indicate, “Hey, there’s no more lines that I want to give you,” and then I’m gonna write them all out. So I have some wild loop in here and I’m gonna do my friend the wild true loop, right, the old loop and a half construct where I’m gonna have some string line that I’m gonna read in from the user. So I’m just gonna use read line and to keep the prompt short, it’ll just be a question mark. And I’ll just assume the user knows to enter a line there. I want to check to see if that line actually contains anything so I check is the line dot equals to the empty string “” and if it is then I’m gonna break, which means the user just gave me a blank line so there are no more lines that they want to enter. And if they gave me a line that is not equal to the empty string, then I actually got some valid line. I want to stick that string inside my ArrayList at the end and tell my ArrayList to just kind of grow in size. So I do that by just saying SList dot Add, and you’ll notice the parameter that Add’s gonna take up there is something of whatever the type of the ArrayList is. So in this case, it’s going to be a string so I can do an Add on line whose type is string and I’m perfectly fine. And this will return to me true, but I don’t care because I know it always returns true. So I’m just not even gonna assign whatever turns to many variables. And this will keep reading strings from the user until the user gives me a blank line to indicate that they’re done, and then I’ll write a little loop so this is just a continuation of the program to actually print all those strings on the screen so you can see what that looks like. Okay? So I have a little four loop over here for I = – forgot the [inaudible] = 0. “I” is less than – I need to know how many strings are actually entered into my ArrayList. Okay. Well, I can use my methods over there to figure that out with size. So I just say SList dot size I ++, and inside here I’m just going to print line and the line that I’m going to print is sequentially I want to go through my ArrayList and get strings one at a time at the given index. So how do I do that? I just say string list, and what’s the method I use, – [inaudible] the single syllable gets candy. And so did everyone else. Mostly the empty seats – get “I.”

So that will get the “I” string. I got my parens lined up there and that will print everything out. So it just gets a bunch of lines and the user prints out those lines on the screen. Okay? But notice I don’t need to declare an initial size for the ArrayList. I don’t need to worry about is it big enough, did I over write the end of it, do I get the big mean evil exception going off the end of the ArrayList? No, it just keeps expanding space as long as I need space. That’s why it’s so versatile to use and when I’m done, I can just go ahead and say, “Hey, what’s your size now?” And then I can add more if I want, so let me show you a generalization of this program that we just wrote. So if we come back to our friend – all done with you power point. Thanks for playing, and don’t save. All right. So we come over to our friend eclipse, and here’s a simple ArrayList example oddly enough named simple ArrayList dot Java. So what this is gonna do is gonna pass around the ArrayList as a parameter doing the same functionality, but I just want to show you what it looks like when we de-compose the program a little bit and how we pass ArrayLists as parameters. So I create an ArrayList of strings here, the same syntax that I had before. I just called it List instead of SList, and I want to read in that list from the user so I’m going to pass as a parameter the whole ArrayList.

The way I pass the whole Array is just by specifying the name of the parameter, which in this case, is list. So I'm gonna read a list from the user. So what does read list do? Here's the whole read list function. Now here's the funky thing. If you can get to the computer that would be extremely useful. The read list function – how do I pass an ArrayList as a parameter? I just say ArrayList and List and you might look at this and say, "Oh, but Miron, you're not specifying what kind of ArrayList it is. Is it an ArrayList of strings, an ArrayList of integers," the function or the method doesn't care. All it cares about is you're giving me an ArrayList, and I'm gonna assume you're gonna do sort of the right thing with it in terms of putting stuff in and taking stuff out of the ArrayList. But you're just giving me an ArrayList, so you do not specify as a parameter the type that that ArrayList stores. It knows because the actual object you pass underneath the hood knows, "Hey, I'm an ArrayList of strings." So if you try to stick something in it that isn't a string, it'll get upset. But it just trusts you when you actually pass it as a parameter. So here we say, "While true, string item =s read line," so here I'm just calling it item instead of line. I read a line from the user. If that line happens to be equal to "" I'm done and I break out of the loop. Otherwise, what I'm going to do is add the item that was just read, which is the string, to the list. So it's exactly the same code that I had here, I just have decomposed it out into a function. Now notice I don't return anything from this method, and the reason why I don't return anything from the method is an ArrayList, at the end of the day, is an object.

When I pass an object what happens to changes I make to the object? They persist, right? It's an object. It's the like the Mona Lisa that's being shipped around in the truck. If I go and slice the truck in half, I slice the Mona Lisa in half because I really have the real object. And so I don't need to return anything because any changes I make to this ArrayList, because the ArrayList itself is an object, those changes actually persist. And so whether or not you think of an ArrayList as an object or as an array, in both cases if I pass an array to a method or I pass an object to a method, in both cases the changes persist, so it doesn't matter how you think of an ArrayList. If it's easier for you to think of it as just an array, think of it as an array and the changes persist, and if it's easier for you to think of an object, which is what it really is, then changes persist as well. And the truth underneath the hood is to be real quiet. Array is just a regular ray are objects too. Just so you know. But you don't need to worry about that. All right. Just think of them as this thing called an array. So that reads a list. And then after we read the list from the user, and all these changes that we made to list persist, I print out the ArrayList, and I'll print out ArrayList. Okay. And I'll print out an ArrayList again, it gets the ArrayList as a parameter and it just says the list contains list dot size elements, so it tells me how many elements are currently in the list and then it does the little four loop that I just drew up on the board going through all the elements from zero up to size, actually size minus one, and writing them all out on the screen. And I'm gonna do this twice so I'm gonna read the list, print it out and then I'm gonna read to the same list just to show you that the list is gonna continue to grow when I do that, so let me run this. I already compiled it. So it's asking for lines, cs106a. That's over in the psych department. CS106a rocks. And that's the end. It says, "List contains two elements, cs106a and rocks," and I'm, like, "Oh, yeah, I forgot to add heavily. And now the list contains four elements because I continued to add stuff to that same list. It just continues to append onto the end. Okay? Are there any questions about that? It's not like you just add and after you add and you stop adding or it's just frozen, it's not frozen. You can just keep adding more stuff if you want. Okay? Any questions about that? Are we feeling okay about the ArrayList?

Now here's the funky thing. At this point, you should be going, "Hey, ArrayList, kind of cool. How come this time, Miron, you've been doing ArrayLists with strings? I thought the simple type was like ints. Why weren't we doing ArrayLists of ints? Any guesses? No guesses? No guesses. All right. I'll give you a hint. It's a popular science fiction TV show from the 70s. Actually, 60s, I would imagine, 60s and 70s. I think it's actually 60s. Keep thinking. It'll get you closer. Here's the funky things about ArrayLists. An ArrayList, as a template, can only be instantiated, which means its type can only be specified to be objects. So ArrayLists hold objects. As we talked about in the past [inaudible] double bouillon and are friend the little care always comes at the end, just kind of struggling along, are not objects. They're primitive types. So I cannot in fact have an ArrayList of ints. If I try to say ArrayList [ints] I get an error because an ints is not an object. So in order to get around that – remember a few lecturers ago we talked about, "Hey, for everyone of the primitive types there's an equivalence class type, so these are the primitives and the equivalent class types, which are called wrapper classes because they kind of wrap around these. The wrapper classes are INTEGER, W, BOUILLON and then character all written out are the wrapper classes. So if I actually want to have an ArrayList that holds integers, it can't hold ints. I need to create an ArrayList that holds integers. It actually has to hold objects so it has to hold instances of some class. Okay. Now, in the bad old days it was Java, you know, pre-Java 5.0 this made ArrayLists really cumbersome to use because every time you had some integer somewhere you said, "Hey, you know what, I have this thing called int X and maybe it's value I said it originally to be five, and now I want to stick this inside some ArrayList of integers that I created somewhere and I can't do that. So I need to actually go and create a new integer object and assign the value for that integer object to be what X is and it just became a huge pain. So you actually need to do something like this. You need to say, "Integer Y = new integer," and then give it the value X because what you need to do is create a new instance of the integer class and its initial value you gave it as an int, and then it created this nice little box for you which was the integer object and inside the box there is this little space for the actual value and it held the five in there. And that was a pain to have to do this every time you wanted to put some object into an ArrayList of integers. And similarly, if you wanted to get something out you had to do something equally cumbersome, which if I wanted to get the value out. So if I wanted to have some integer Z here and I want to say, "Hey, get the value of that integer, I can't just say Z = Y, at least I couldn't in the bad old days." I needed to actually say Y dot ints value and call a method on it. And then what this gave me back from that object Y was its actual value of an integer, so it would give me back the five as an integer. Okay. All that changed in Java 5.0. So the process of essentially creating an object around a particular primitive, right, so we had our little primitive int five over here, and what this was basically doing was creating Y to be an object around the five.

This process is still called "boxing" because basically what you were doing was drawing another box around your value. And this process over here of going from the box that you had to getting the value out of the box, interestingly enough, was called unboxing because basically when you wanted to get the value out, you cared about the value inside and essentially you wanted to erase the box that was around it. Okay. This whole boxing and unboxing happens for you automatically in Java 5.0 and later, which means if I create an ArrayList of integers I can just add an int to it directly. So if I were to have an ArrayList it still needs to hold object of the integer class, so I still can't say an ArrayList of ints, and I'll call this IList = new ArrayList integer, so I create one of these things. Now I can say IList dot add and give it X directly where here I might have int X = five, and you would say, "But Miron, you told me this ArrayList can only store integer objects and you're passing an ints. You told me that doesn't work." Well, this is what Java 5.0 is doing for you automatically. It says, "Hey, this things expecting an integer and you're giving me an int," which means you really need to box it up. It's kind of like you finished your eating and you still have some food left and you want to store that food somewhere, what do you need to do? It's got to go in a box. And before you used to have to ask like the waiter or the waitress to bring you a box and they would create it in the backroom somewhere, probably out of other food that wasn't eaten, and then you would get the box and you'd put your value in, you could finally go and store it at home. And now Java just says, "Hey, you know what, people were asking for the box so often, yeah, if you need that food stored in a box, I'll just put the box on it for you, and then when you want to get it out over here," like you want to say, "int is Z = IList dot [inaudible], the zeros elements." Before you would just get the box, right, because that's all this thing is storing are boxes that hold integers. Now I'll actually take the box out for you and let you eat the food automatically. Okay. So it does that for you automatically, which makes these things a lot more useful. And the thing to keep in mind is that all these wrappers classes are still immutable objects. They're immutable in the same sense that strings are immutable. When you create one of these guys, like this guy Y has the value five, its value is five. You can't go back and say, "Hey, set its value now to be six," because it does not have a method to set its value. It can get you its value by asking for its integer value, but there is no way to set its value. If you now want to create some new integer that's value is six, then you need to create a new integer and actually say what's its value, its Y + 1. Okay. Just in the same with strings you couldn't change a string in place. If you wanted to actually change the string, what you really needed to do was create a new string that copied over all characters from the old string that you cared about and potentially made some other changes. Okay.

So the wrapper classes are immutable in the same way strings are immutable. Once you set their values, it does not change. If you want to make it look like its really changing what you do is you create a new one and you would assign it to overwrite the value of the old one. Okay. So let me show you an example with that. And we'll use our same little code here and this is gonna be so much fun all we're gonna do is say, "Hey, you know what, rather than having an ArrayList of strings, I'm gonna have an ArrayList of integers." And you're gonna say, "But Miron, this is gonna make your whole program blow up." Well, read list doesn't care what type of ArrayList is getting passed in here. It's just getting passed an ArrayList. I don't want to put strings in there so I'm gonna change my item to be an int and I'll read in ints from the user, and basically the way I'm gonna stop is if that int is == to zero, and Add actually will just remain the same. And here's the coolest part of all. Prints ArrayList doesn't change at all even though I just went from strings to integers. Why does it not change at all? Because this ArrayList parameter here doesn't care what type you're passing in. The list contains the size method doesn't care what type is actually in the list, this list outside doesn't care what type is in the list. And when I get the element from the list, what I'm getting is an integer object, and when I go to print it out, it automatically gets unboxed for me to get turned into an integer and printed out. So even though I changed from an ArrayList of strings to an ArrayList of integers because I never specify the type here and that's just the way life is. That function will just work on change. Okay. So let me run this program again. Yes, go ahead and save and do all the happy things you need to do. One, two, three – [inaudible] numeric format, right, it wasn't in line and it wanted a zero. Now the list contains three elements and then I'll add four and five and then give it a zero and now the list contains five elements. Okay. Again, the beauty of object drawing and programming and the beauty of templates. Some things can just be reused even though you change the type, and you just need to be careful what you're actually storing and not storing. Okay. So any questions about that? All right. Another quick example. When we talked about arrays I told you, "Hey, you know what, you can have an array that contains objects in it too." You can have an array of G objects or an array of G ovals or whatever you want, and that's perfectly fine. And you can do the same thing with ArrayLists. As a matter of a fact, with an ArrayList you don't need to worry about all this boxing/unboxing stuff because if you have an ArrayList of let's say G ovals or G [inaudible] or whatever the case may be.

Those are already objects so there's no boxing or unboxing that needs to go on. Okay. So let's say I was gonna have an array of G labels, I could say G label – actually, let me show you this first of all in just the case of regular arrays and then we'll move on. So I can have an array of labels. So here is labels and I say something like new G label, it has four items in it and what I get here is four elements of my array that all start off no because none of them have been initialized. Okay. That's what I got in the array world. If in the ArrayList world I create an ArrayList of G labels I would say something like ArrayList that's going to store G label, and I'll call this my G list or my GL = new ArrayList of (G label). This creates for me an empty list of G label objects. If I now want to add G labels I still need to create the actual underlying G label object and then add it to this ArrayList, much in the same way that with a regular list that when I created the array, all I got was just an array of no. So if I wanted to have the individual elements, I still needed to call a new on a G label for each one of the elements that I wanted to create. Same kind of idea going on with ArrayLists. All I get is an empty list and every time I want to add a new G label I should've created that G label object already. Okay. So let me show you an example of this with another funky program we call graphic numbers, or I call it graphic numbers. I just wrote it this morning, and all this thing is gonna do is listen for the mouse. Okay. All of the activities are going on when I click the mouse. So what's it gonna do? What it's gonna do when the moue is clicked – well, before the mouse is clicked I have some instance variable in here. The instance variable I have in here is this thing that I just showed you. I have an ArrayList, it's gonna be private, so it's gonna be within my object, it's an ArrayList called labels that holds G labels. So what I get when this line executes, when the program starts running, is I get an empty ArrayList of G labels. There's no G labels in it. The way I'm gonna add G labels is every time – this is the whole program – every time the user clicks the mouse I'm gonna create a new G label and add it to the list. Okay. So what I do when the user clicks the mouse is I'm gonna create a new G label called "lab" and the label or the text that's actually associated with this label is just the pounds sign or the number sign and then however many labels are currently in my list, which means at first it's gonna be zero so it's gonna give me a zero and the labels gonna be number zero, and the next time it's gonna be number one, and the next time it's gonna number two, etcetera. But the first time it'll be number zero.

I set the fonts on that label to be Courier 18 because as the years are passing my eyes are having problems so I need the font to be big. And then what happens – here's the funky thing that I want to do. I want to say, "You know what, where am I gonna put this on the screen," I always want to put the brand new label, like, when it's born in the same place on the screen, but the problem is if I put number zero there and then when I'm ready to put number one, if I put it right on top of number zero it's gonna get real ugly and it's gonna be a mess, it's like dogs and cats sleeping together, and the whole thing is just not gonna work out. So what am I gonna do? I'm gonna say, "Hey, why don't I take all the other numbers I had before and move them down one line so the new guy will always show up on top, and then when there's gonna be a new one, everyone will move down and make room for that new one." It's kind of like a waterfall of numbers. It's just kind of like the movie the Matrix, right, it's kind of like the numbers all just move down, the next number will take its spot. So how do I do that? Star Trek, think Star Trek. All right. So I'm gonna take all the existing labels that I've already put on the screen and move them down. How much am I gonna move them down? I'm gonna move them down in the Y direction, the height of the label. So the reason why I first create the label without a location and set the font is because I want to know how big that label actually is and how much I need to move everyone else before I put this on there. So I say" Label get your height, tell me how big you are so I can move everyone down that much space to put you in," and now I'm just gonna have a four loop. And that four loop is gonna go through my label list, which means it's gonna go through all the existing label objects I have. I'm gonna say, "Hey, in that label list get the [inaudible] label and move it down by the amount DY." Okay. So this four loop goes through all of the objects that are in my list of labels and tells them to all move down, so whatever place they were on the screen before, they all move down one step, and that step is DY pixels. And after they've all moved down they said, "Hey, hey, we made space for the new guy. New guy, come on, come on, come, and the new guy runs over and is, like, ah, I'm part of the number group now." And so we add him to the canvas. We add the label; we added the start X and the start Y location, which is always the same. Start X and start Y are just some consistencies that are given here. So the new guy always goes to the same place, everyone else just moves down to make room for it. And I want to remember, "Hey, new guy, we're not just adding to the screen, we have to add you to the ArrayList, too." If we don't add you to the ArrayList, we're not gonna move you next time we need to move you when you're no longer the new guy. So this add is adding to the canvas, this add is being called on the labels ArrayList, so it's adding this label object to the ArrayList.

So there’s two adds we do. One puts it on the screen; one puts it in the Array List that we keep track of. Any questions about this code? Let me just run this code so I can show you what’s actually going on and how everyone just moved down for the little guy. Okay. Which I realize is vaguely what happens to your life when you have a child. Basically, everything else in your life that you thought was important moves down and the little guy really takes over. So here’s graphic numbers, and it’s just the most wonderful thing in the world. All right. There is zero. I clicked once, and now I’m gonna click again. One comes in. I’m gonna click again. That means zero’s got to move down by DY, ones got to move down by DY, and then two comes in. And if I keep clicking, everyone just keeps moving down and the new guy – or new woman I should say – or new it if we want to be gender neutral – just shows up in the same place, and then you can play games with your friends, like, first to a 100. Yeah, because it’s, like, 4:00 A.M. and you’re, like, “No, no, first to a 1,000.” And number zero is still there. It’s just way off the screen, but it’s still down going, “Yeah, new guy, come on. I’m just moving for you, right, and it doesn’t know it’s not being displayed anymore.” Okay. So any questions about that? All right. So you can have an ArrayList that contains any kind of object. Now there’s one last thing I just want to show you that’s just fun and cool. Okay. And here’s the thing that’s fun and cool. And the thing that’s fun and cool is that now that you’ve learned about all this stuff with arrays and we’re, like, “Oh, you can have an array of numbers or you can have a matrix of numbers,” and you kind of look at that and you’re, like, “Yeah, Miron, yeah, that’s kind of fun, kind of, sort of, but you know what I really like, I like pictures. Can I do stuff with pictures in arrays?” And you think for a while and you say, “Yeah, because if you couldn’t, I wouldn’t be asking you that question and setting you up for the fall.” Sometimes I might, but not right now.

Think about a picture, right – and by a picture I mean a G image. What is a G image, really? A G image is basically made up of a bunch of pixels and those pixels live in an array basically, actually, it’s a grid. They live in a two-dimensional array. Right. So if I have some G image that looks like a smiley face then maybe I have some pixels that look like that, and I just have some G image, and those are the individual pixels. And you look at that and you say, “Hey, that’s kind of cool. Could I actually manipulate that grid of pixels?” Yes, in fact you can. And the value that’s stored in each one of these elements of the G image – there are some pixel arrays, which are actually a two-dimensional array, and I’ll show you how to manipulate it in just a second. There’s an int that’s stored in each one of these cells. You might look at that and say, “An int? Why an int? I thought I could have all these colors. Well, it turns out in fact you can encode a whole bunch of different colors in a single int using something called RGB Values, which stands for red, green and blue. And if you ever look real, real closely on your TV screen, every little dot on your TV screen, unless you have an LCD TV screen, if you look at the old school TV screens they’re really made up of a little green dot, a little blue dot and a little red dot, and the way you get different colors depends on how intense each one of those three things is. So this single integer actually encodes all three of these values using an encoding scheme that’s actually described in the book. It’s not worth going into all the details. You don’t need to worry about it. But all you need to know is that you can get that red, green and blue value out of this single integer. Okay. So if you can do that, what kind of stuff can you actually do with your program? You can take an image, let’s say a color image, and turn it black and white. And you say, “How do you do that, Miron?” So let me show you. We’re gonna read a G image from a file, so this is just the standard G image stuff that you’ve seen before.

We’re gonna create a new image that’s gonna read this file called “milkmaid.gif,” which is basically just the little picture that you’ll see in just a second of a little milk made. And then we’re gonna create a grayscale image of it, and then we’re gonna display both the regular image and the grayscale image at sort of twice their size, we’re gonna scale them by two because it’s actually a small image to begin with. So we’ll scale them just so you can see it more closely. And we’ll show them side-by-side to each other so I just put in some parameters to make them side-by-side. How do I create the grayscale version? What I do is I’m gonna pass that G image, there’s a parameter called “image” to a method called “create grayscale image,” which we’re just gonna write together or I’m just gonna show you here. What I can do from a G image is ask it for its pixel array, that thing over there, it’s a grid; it’s a two-dimensional array of integers. So what I get back in this value that I’m gonna call an array is it’s basically just a two-dimensional array of integers, and I’m getting a copy of all of the pixel values of that image. How do I figure out how high the image is? I can ask that array for its length because remember a two-dimensional array is an array of an arrays, which means that the array itself, or this thing called image, is basically an array that contains one row per element. So what’s the height? It’s the number of rows I have. So if you ask the main array itself how many elements do you have, it tells you how many rows it has. That’s the height of the image. So I get that from array dot length. How wide is the image? It’s how many columns are in one row, so if I say, “Hey, I can pick any row I want,” but I’m just gonna pick the first row because I don’t know how many rows there are and I don’t want to pick some row that doesn’t exist so I say, “Hey, first row, you’re array sub zero, what’s your length?” And it says, “Hey, I’m 40 pixels across.” And then you know what the width of the image is, too. Now once I have that here’s the funky thing. I’m gonna have this double four loop that you saw before when I was going through a grid, right, so I have some index [inaudible], I’m gonna set my individual pixel value to be the element that of the array I sub J, so I’m gonna pull out each one of these individual pixels and integers one at a time. And there are these three methods in the G that are provided for you, static methods of the G image class called get red, get blue, and get green, and they get the corresponding red, green and blue values that are encoded inside this single integer and they return them to you as integers.

So what you get are these little intensity values for the red, green and blue. And based on red, green and blue, there’s these wonderful folks, the National TV Standards Consortium, something like that, NCSC, who said, “Hey, in order to go from a color to a black and white, you want to consider how luminous the image is,” and luminosity depends on the color. So it’s about .3 times the red, the greens a lot brighter so it’s almost .6 times the green, and blue we consider really dark so it’s about .1 times the blue. And these are just numbers that got pulled out of the air by this consortium, but that’s how they determine what the luminosity or how bright something is relative to its color. Funky, but someone does it. And then we just round that value to an integer because we’re returning an integer and that’s what we return, and we say, “Hey, that’s great. You gave me that luminosity, I’m gonna set that same luminosity value for the RG and B images.” If you set the RG and B values to all have the same value what you get is a grayscale because all the color blends out. And essentially you get something that varies from completely white to completely black depending on how intense you’ve given those RGB values, and I store that back in this array, and at the very end I’m gonna create a new G image out of that array and return it. So let me just run this for you in our last minus two minutes together so you can actually see it. Anyone want to take a final guess as to what I am?

Student: Spock.

Instructor (Mehran Sahami): Very close. I’m almost Spock. Student: [Inaudible]

Instructor (Mehran Sahami): I’m Mirror Spock. Yeah. Also known as evil Spock. And it’s because the whole costume is all around the goatee because when my wife saw the goatee she’s, like, “You’re evil Spock,” and so our son is actually Spock. So when we sit next to each other he’s small Spock, he’s about this big, and I’m evil Spock. Yeah. It’s so cute if you’re kind of geeky like me. All right. So here’s the original milkmaid image, and here’s that same image when you created the grayscale of it. So just one last thing. If you shut the camera – because now we’re gonna get something to do if you actually want to see it –

[End of Audio]

Duration: 52 minutes