Programming Methodology-Lecture15

**Instructor (Mehran Sahami):** All right, welcome back to yet another fun-filled exciting day of cs106a. This is the pivot – this is the turning point of the quarter. This is like the pivot of the quarter. After today, it's all downhill, because we've gone through just as many days as we have left. Fairly exciting. As a matter of fact, they actually have fewer days left, because we have like the last class is not happening and stuff.

But a few announcements before we delve into things today. The handouts from last time, if you didn't get the handouts from last time, namely, especially the practice midterm and solutions to the practice midterm as well as assignment No. 4, if you didn't get those, they're available in the back today. If you already got them, you don't need to pick up additional copies. There's no additional handouts for today, but there are just copies of the ones from last week.

So, again, with the practice midterm, I would very highly encourage you to take it and, if you want, also, for the full effect, actually time yourself doing it so you get some notion of what's taking you longer or what's taking less time. That will give you a diagnostic of the kind of things you need to brush up on, so – and it'll also give you a chance to see what we're really kind of expecting for the midterm, and there was a bunch of stuff on there where it explains that the midterm is open book and open note, but closed computer, and all the guidelines for the midterm are all explained on the first page of that handout.

So the midterm is coming up. It's next week, just a few days away. Tuesday, October 30th, 7:00 p.m. to 8:30 p.m. in Kresge Auditorium. If you don't know where Kresge Auditorium is, find out where Kresge Auditorium is. It's your friend, it's big, it's bad, it's Kresge. It's just a huge auditorium. If you've never been there before, it's kind of cavernous. You go in there and you go, my God, I didn't know there was an auditorium this big in Stanford, but there is, and it's ours for that short period of time.

If you have a midterm conflict and you already sent me email – if you haven't already sent me email, you can go ahead and try, but I can't accommodate any more requests. I can hopefully just tell you when the alternate midterm is, but you will get an email from me this weekend letting you know when and where the alternate midterm is, so read your email this weekend. It'll probably come some time in the wee hours of the night either on Saturday or Sunday, which would be the weekend, strangely enough. So watch your email for the alternate midterm.

If you're an SAPD student, SAPD students have two options. If you're a local SAPD student, you can come in for the exam, so just come on down, 7:00 p.m. to 8:30 p.m., Kresge Auditorium, October 30th. Take it along with everyone else, introduce yourself, come by, say hi. I always like to meet the SAPD students.

If you're remote, because I know some of you are saying, not in the state of California, and it would kind of be a stretch to ask you to fly here just to take the midterm exam, you

may take the exam remotely at your site. If you plan on taking the exam remotely at your site, you need to send me email, and in that email you need to let me know your name as well as your site administrator. Find out who your site administrator is if you do not know, because your site administrator will be administering the exam to you, and I need to know their email address.

So send me your name and email, which I'll get when you send me email, but your site administrator's name and email so I can email them the exam and they can administer it to you. And if you're a local student, go ahead and send me email, and just say you're coming in locally, just so I know that you're coming in locally and that way I can keep track of everyone.

But extremely important for most students, if you're a remote student and you don't send me email, so I have no way of contacting your site administrator, I can't give you the midterm, and that's real bad times for just more reasons than you can shake a stick at, but you're certainly welcome to try.

And last but not least, same announcement as last time. There is a syllabus change, a very minor syllabus change, based on just moving the discussion of arrays up by one day. So if you're sort of reading along in the book and you wanna know what to read for Monday, Chapter 11 on arrays is what we'll be covering on Monday, and if you're not following along in the book, you should be following along in the book, but now you know.

All right, so with that said, I wanna do just a tiny little bit of wrap up on what we talked about, memory and this whole notion of pointers last time, and go into our next great topic. All right? So time for our next great topic.

Well we need to do a little bit of pointers first. So the first example is something that we talked about very briefly last time, but I wanna make sure everyone's sort of on board with this. So if I have our friend, the point, that we talked about last time, which is just a class that stores two values, an x and a y that we pass into the constructor, when we do something like this – I'll draw it sort of over on this board.

Now I'm gonna draw it in a – actually, I'll draw it over here, and I'll draw it in a way that we don't need to worry about all the memory addresses and overhead and all that stuff, because that sometimes makes the diagrams a little bit ugly, and so I'll show you the very simple, stylized version of keeping track of the stack and the heap.

So over here is the heap, over here is the stack, and what we're gonna do is say, hey, when we declare point p one before we have done this new, what we get is p one, which is basically just some box on the stack. What does it hold right now? Well, until we do this line, it holds nothing. It holds nothing that we know about, okay?

When we call this new, it says, hey, heap, get me some new memory for a point, and so the heap kind of goes over and goes, boink, here's some memory for a point, and it calls the constructor and says, hey, constructor, put in the values one one. So if somewhere on

the heap we have the values one and one for our little private x and our private y somewhere on the heap, and the value that new gives back, that we assigned to p one, is the address of this thing.

Remember, we don't really care about where in memory this thing's actually stored, and so sometimes, what we like to do is just keep track of addresses by pointer. So we know that this is just some memory address over here, where this particularly object is actually stored on the heap. Okay?

So now, if we do point p two – actually, let me not let – put the semicolon on yet. If we do point p two, what we get is, on the stack, we get some new space set aside for p two and, again, its beginning value is unknown.

Now we have a couple options. One thing we could do is say new point, and give it some x y location, in which case we'll get a new point on the heap and p two will point to that new point. We could do something funky like say, hey, p two equals p one. Well all that's happened there is saying, take the value that's in p one, which is a pointer over to this location on the heap, and set p two equal to that. So if this happens to be a memory location A A A E, this guy gets A A A E, which means, in our sort of stylistic representation, it's pointing to the same place. Okay? Are we sort of onboard with that? If you're onboard with that, nod your head. Excellente.

All right. So now if we do p two dot move, and we tell it to move by an amount like three comma four, it says, well, in order to call the move method, I need to call the move method using the object p two. Where does the object p two live? It looks it up through the pointer and says, here, this is where you live. Call the move method with this particular object. So a path to this pointer to this location in memory, and it says, move three four, and as you saw with move last time, all move does is add those values to the respective x and y. So p x becomes four, p y becomes five, and now that move method is done. And, interestingly enough, what I've done is change not only p two, but also p one, because they were pointing to the same place in memory. They were the same actually point, and that's the important thing to keep in mind.

If I set an object equal to another object like this, they are the same actual object. There's only one of those objects that lives somewhere, and I really have two things that are pointing, or what we really like to refer to as referring, to that object. So often times we refer to these things as references, okay?

Now what happens if I come along and say, point p three. I get p three over here on the stack, right? And again – I'll write the p three over here so as not to interfere with the lines, and I'll draw this line a little bit around p three so it kind of goes like that. It says, p three, I don't wanna interfere with you, I'm just gonna zigzag around.

P three, what value does it start with? I don't know. It's not pointing to an object yet, because it hasn't been initialized to point to an object. It hasn't been initialized to point to a new object. And so if I come along and say, hey, you know what p three? I just wanna

move you. Move to four comma five, because that's where I hear the party's at. That's where p one and p two are. Can you move yourself to four comma five? What happens when I do this?

Bad times, that's what happens. What I get – well first of all, I get a social, because a whole bunch of people said it. But what I get is this thing. Who knows where it's pointing? As a matter of a fact, this pointer could, often times, be something we refer to as a null, which means it's not pointing to any particular object, in which case, when I try to do this, I get what's called a null d reference. Anyone happen to see something called a null d reference while you were working on breakout? A few folks? Yeah, this is what you were getting.

You had some object here that wasn't actually pointing anywhere, and you were trying to call a method on it that says, hey, I'm trying to call the move method on p three and p three says, I got nothing for you, man. I got nothing here. You're trying to move something that doesn't exist, so you're trying to dereference, which means go through the reference, it's a funky name for go through, we say d reference, something that doesn't exist, or maybe this is just – has some random value that's pointing off somewhere into memory that you don't know about, which isn't actually an object. Okay?

So that's what you wanna watch out for. This is real bad times if you actually happen to call a method on some object which doesn't really exist, okay?

Now the other thing that's important about seeing these little pointers is the fact that – remember when we talked about parameter passing? And in the days of yore, we talked about parameter passing and we used to have this little thing where we said, hey, you know what? If you have something like an integer, or a double, or a char, or a Boolean, these are what we refer to as primitive types.

They were like hanging around, they're, you know, lower on the evolutionary ladder. They're just what we referred to as the primitive types. When you pass primitive types as parameters, what you are getting is a copy of the value of the primitive type, okay? What does that mean?

It means, when I call some function, and I say, hey, function, I'm gonna pass to you a three, like in my move method over here I passed in a three, which is an integer, what actually happens when I make this method call is I pass a copy of the three, and so if in this move method I try to change this value around to something else, all I'm doing is changing the copy. I'm not changing the actual parameter that was passed in.

So if this wasn't a value – let's say I actually had some variables here like x and y, and up here I had int x and y, when I make this method call, I get copies of whatever values are in the boxes for x and y, and so if move tries to muck with x and y, all it's changing is the copy. It's not changing the x and y out here, right? That's what we talked about a long time ago, and we talked about little Neron, who went off to France with his mom and the Mona Lisa, and he wanted to take a hacksaw to the Mona Lisa, but the Mona Lisa, he just

got a copy, because they went to the gift shop. Yeah, primitive types, you go to the gift shop.

Now, something funky happens though when you're dealing with objects, okay? So when you're dealing with objects, they are not primitive types, they are object, and so when you make a method call where you pass some object along, what you are actually doing is you are passing the object – or, actually, what we refer to is the object reference. What does that actually mean?

What that means is, let's say I call – were to call some method over here, where I actually pass in a point. So let's say there's some method over here like – well I'll call it change zero, which is supposed to set a point to be zero, and I pass it, p one – and this is just some method that I wrote, and maybe the elements of a point are actually public, let's say, as opposed to private, so this function can actually access those things. What happens when I pass p one?

Well what happens when I pass p one to this change zero method or this change zero function is I'm passing where p one lives, so the parameter I actually pass is the address that's in here. So if the address that's in here is, let's say, the value a zero zero zero, I pass, as my parameter, a zero zero zero, which means what I have is a reference, because I have that pointer, to where p one really lives in memory, which means if this method decides that it's gonna muck around with p one and tell, hey, p one, move yourself to some other location, the values that it's gonna change when it makes method calls on the parameter that's passed in is at this place in memory, because it knows where this place in memory is, right?

You wanna think about, when you're passing objects, this is, as we talked about it before, sort of like the mafia. When you pass around objects, you're talking to the mafia, and when you talk to the mafia, they know where you live, okay? They can come in and mess with you however they want. They're not getting a copy of you to talk. You go to the mafia and you're like, oh, I'm gonna send my clone along to the copy and – or my clone along to the mafia and they'll just talk with him. No. When you're gonna go talk to the like the Godfather or the Godmother, you're not sending in your clone. You're going in yourself, right? And you're like hey, Godfather, and he's like, you come here on the day of my daughter's wedding. Anyway, that's a – anyone seen the movie the Godfather? It's so good. If you haven't seen it, go see it, but we won't talk about that right now.

What we talk about is the fact that, if they know where you live, any changes you make are not to a copy, and that's a critical idea to know. This is what we refer to, up here, as pass by value, because what you're getting is just a copy of the value. You're not getting the reference. This is what we actually refer to as passing by object reference. So I'll just change the the here to a by, which means that when you pass objects around, you're passing a reference to the object, you know where the object lives.

Here's a little way you can think about it. So remember our story about little Neron, who goes to see the Mona Lisa? Think about it this way. When little Neron went to see the

Mona Lisa – again, here is the Mona Lisa smiling, here is little Neron – bowlegged Neron, ponytail, chainsaw. When he wants to actually – if the Mona Lisa were just some integer, right, he's gonna go to the gift shop and he's gonna get copies.

But the Mona Lisa is actually something that's really valuable, so what we do with the Monsa Lisa? It's encapsulated in this big safe that's an object, and so when we wanna say, send the Mona Lisa over to like the MoMA in New York, not that they would actually display it at the MoMA, but let's say we send it to the MoMA in New York, we're sending the actual Mona Lisa. We're not sending a copy of the Mona Lisa, we're sending the actual thing.

So then, in the MoMA in New York, Neron comes along and says, hey, you know what? Security is much more lax here. I'm just gonna chop this thing up. So he takes the chainsaw to the Mona Lisa and chops it up, and that's the real Mona Lisa, so when we're done with it being displayed at the MoMA, and it goes back to our friend the Louvre in Paris – so here it is in Paris, yeah, it's still sliced in half. Bad times. Major international incident, okay?

Don't let this happen to you. If you're gonna mess with an object that's passed around, know that you're changing the actual object, okay?

So one thing you might say about this is, you said, hey, Neron, you were saying talking about Mona Lisa encapsulating that in an object, so if I have some integer, and I actually wanted to change it, could I create an object around an integer and then pass the object and allow someone else to change the integer inside that object, would that work?

Yeah, in fact, that would, and if you're interested in doing that, there's an example that does exactly that on Page 235 of the book. Just because it's important to memorize every single page of the book, but it's there. Okay? I just had to check. Yeah, it's 235. So just something you should know. So any questions about that?

All right so, last but not least, one other thing I should mention since we're on the subject of saying, hey, why not take one of these primitive types and encapsulate it inside a class, it turns out Java actually has a bunch of classes already built in which are encapsulations of these data.

So for an int, there is something called Integer with a capital I and the full word, which is actually a class, that is a class that stores a single integer. The only unfortunate thing about this class is you would say, hey, now I can create integers and I can pass around an integer and change the integer, right? This class doesn't actually give you any methods to change the value, so it doesn't get you the effect you want. We'll use them later on in the quarter for something different, but just so you know these exist. There are the double, and then there's the class version, which is actually upper case double, so Java is case sensitive, and if you ever put upper case double, you would actually have been using the class version of it.

Similarly, with Boolean, there's Boolean and there's upper case Boolean, and then with char it's a little bit different. We have char over here, and here it's the full word, character, is the class. So these are sort of the class equivalent, these are the primitive equivalents. For everything we're doing so far, unless you get told to use the class equivalent, just use the primitive equivalent and life will be good.

This is just kind of an artifact of programming language. Sometimes, in life, things just happen to happen this way, and you get two versions of something just because that's the way life is, okay? But these class versions – remember, like strings? We said strings were immutable and you can't actually change the string in place, all you can do is create a new one. These classes are all also immutable. Once you create an integer and give it some initial value, you can't change that initial value.

If you wanna say, hey, I wanna take that value and add one, you need to create a new object of type integer, which has the old value plus one added to it, and you get a new object. So they're immutable in the same way strings are immutable. You create new ones. You can't change the value once you've gotten an initial value. So, any questions about this, or any questions about this whole notion of pointers and references and passing the stuff around as parameters? Do you? Because, if not, it's time for out next great topic.

And our next great topic, it's something that really is a great topic in Computer Science, because it's something you've been doing this whole time, and now it's time to lift up the covers and say, hey, you know what? There are some much more dangerous things you can do than you've been doing right now. So, before, we sort of gave you the safety scissors, where it has like the rounded corners and they're like rubber. Remember safety scissors? They were fun. I liked safety scissors.

Now, basically, we take the safety scissors, we take them to the mill, we sharpen them up real sharp, we have you hold them up and run with them. Okay? So here's where things get dangerous.

Files. You're going to learn how to read and create files as well, which means the opportunities to erase files or write over them, which is why things get dangerous, but it's really not that dangerous so long as you're careful, right?

When you're running with scissors, just point them down, and things will be – why people ran with scissors just made no sense to me, but – I really need to go cut this piece of paper! Come on, we're all going. It's the paper-cutting marathon.

All right, so files. What is a file? You've used files the whole time, right? When you created some program, like you had My Program dot Java, that was just stored in some file somewhere, and you – when you write a paper in your favorite Microsoft – your favorite Microsoft – your favorite Word Processor, which may or may not be a Microsoft product, you're creating a file. And, in Java, you should be able to create files and read

files as well, so that's something we're gonna do, and the first thing we're gonna thing about is reading files.

Now the files we're gonna think about reading are not necessarily Java files. We're actually gonna think about just plain text files, and often times you'll see these things as – it'll have some name followed by a dot followed by txt, which is kind of the extension that means text, but that's what we're gonna deal with, is just files that contain a bunch of characters, without any of the characters being like special meaning characters or anything like that.

So the way we think about a file – let me just draw a file up here, because life is cheap. As a matter of a fact, I'll show you a little file on the computer. Wake up. Don't worry, it'll come back. It'll just take a moment. Wake up little guy. Yeah, it's time for your starring role. Here's a file. See I just want it to say, A students rock the house, there can be no doubt about it. Doubt about it. Doubt it. I put two doubts in there, I don't know why, I just did. All right.

It's one of those little things, like Springtime in the Paris. Remember that little puzzle? No. All right. Let's not worry about it. When something is at the beginning and the ending of the line, the human brain is just likely to convolve it and think there is only one there. Random psychology for the day. Thank you. Let's move on.

Here's a little file. It's a text file. It just contains a bunch of lines of characters. The way we like to think about files is that we're gonna process a file one line at a time, and we read files sequentially which means we tell the computer, hey, here is the file that I wanna start reading. It says, okay, I'm ready to read the file, and it starts at the very beginning of that file, and then I will ask the computer line by line, give me the next line of the file, and it'll give it to you as a string, and you can do whatever you want with that string, and then when you go to read the next line of the file, you'll get the next line.

So first you'll get cs106 a students, then next time you ask for a line you'll get rock the house, then there'll be there can be no doubt, then you'll get, doubt it, then you'll get some indication that you've reached the end of the file, okay? So what does that look like, all right, and how do you actually do this?

So you're gonna be doing this for assignment No. 4, hangman, so it's a good thing to know. So first thing you wanna do is what's referred to as opening the file. Opening the file basically means you're gonna associate some object – you're gonna create some object, which is actually something we referred to as a buffered reader, and you'll see an example of that in just a second. You're gonna have some object that essentially corresponds to some actual file on your disk. So when you open a file, what you're doing is saying, I'm going to have some object I'm going to create, and when I refer to that object, I'm actually gonna tell you that it's referring to some particular file that's on my disk right now. So that's opening the file, is creating this correspondence.

Then the second thing you do is you basically read the file, and there's multiple ways of reading the file, but we're gonna focus on reading a file line by line until we've read as much of the file as we want, usually the whole file, and then the third thing you need to is, because you open the file, you need to close the file, which basically cuts this correspondence between the object and the file. It says, okay, I'm done with that file now. This object is no longer referring to that file. This object is kind of, I'm gonna be done with it.

So, in order to use some of this function – I'll show you the code in just a second – there's a particular package you need to import, called the Java dot IO, which stands for input output, dot star. So at the top of your program, you'll have an import, then it has Java dot, just the letters IO dot star semicolon, and that will give you all sort of the file stuff.

So how do we actually do one of these things? We're gonna open a file, we're gonna read its contents, and we're gonna close it. It's time to write a little code.

So first thing we're gonna do is we're gonna create on of our friends, the buffered reader. So we're gonna have some object that's of type buffered reader, and this is just a class that exists in Java dot IO. That's why you import that package, and you're gonna be able to refer to objects of this type.

So buffered reader, I'll just call it rd, for reader, for short, and what I'm gonna set that to is a new buffered reader, so this going to be a new buffered reader, and the new buffered reader needs some parameter, and here's where things get funky. The type of parameter it's going to take is something called a file reader. That's a special kind of object that knows how to attach itself to an actual file on disk, but I need to create one of those, so here's the idiom for how I create it. The idiom is just sort of the pattern you'll always see in programs. I say new buffered reader, and the parameter that I passed to my new buffered reader is a new file reader, upper case R, new file reader, and the parameter that I give to new file reader is the name of the file on the disk.

So if the file on the disk is called, like in this example, students dot txt, I'd actually give it here as a string, students dot txt – I need a space for a few extra params there, so let me write that a little bit smaller. Students dot txt. That's the end of the string. Then I have this param to close that param, and another param to close that param.

So what it does, it says create a new file reader, which basically is an object that associates with a particular file on the disk, and that object is what gets passed into a buffered reader, and the buffered reader is what you're gonna ask to get line by line. So you're gonna say, hey, buffered reader, give me a new line, and it's gonna go say, hey, file reader, get the line from that file, and get it back and give it to you. Okay?

So this is just the standard idiom you always see and the way that it's written, and you can put any string in here you want. As a matter of a fact, you can put in a string variable if you want.

Now after you've created that, how do you actually read from the file? So let's have a while loop that will read in every line of the file, so we'll have a while true loop, because we're gonna keep reading until we reach the end of the file, and the way we read this is, we're gonna read the file line by line, and each line is just a string. So I have string line, I tell the buffered reader, this rd object, read line, which kind of looks familiar to you because it sort of looks like reading a line from the user, but in fact, you're passing the read line message to the rd object, which means get me a line from the file. Which file? This file that I created the correspondence with over here, okay?

Now this will read line by line. Every time I call rd read line, I'm gonna get in some – the next line from the file. What do I get when I reach the end of the file? And this is the crucial thing. One thing you might is, hey, do you get an empty string? Like do you get double quote double quote? No, in fact, I don't, because my file could actually create – contain empty lines, and I don't wanna confuse an empty line in a file with the fact that I've reached the end of the file.

So the way I signal reaching the file is – remember a string is just an object. How do I refer to an object that doesn't exist? Null. So what read line gives me back, if I try to read a line past the end of the file, is it gives me back a null. It says, hey, there is no string here for you to read, and the way I'm gonna indicate that to you is to give you back a null. So I can check that. If line is equal equal to null, that means I've reached the end of the file, and so I'm just gonna break out – and here's my little loop and a half. I'm gonna break out of this while loop.

And if I got a valid line, I'm just gonna print it to the screen. So all this code is gonna do is take some file, read it line by line and print every line to the screen, and since every line is just a string, I can just do a println to – of line, and that will write it out to the screen. Okay? So any questions about this?

Now there's one thing I haven't done yet, right? In my steps over here, I've opened the file, because I created the buffered reader, I've read the file. Now I need to close the file. So after I've done reading all the lines from the file – let me move my brace up a little bit to give myself room for this. I just say rd. I refer to the object, and then I say close, and there's no parameters there, and that kind of says, I'm done with you. Thanks. Thanks buffered reader, you did good work, you got me all the lines of the file and now I'm done with you, so I'm just gonna close it off. Okay? Question?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** Yeah, read line keeps track of where it is in the file itself, so every time you read a line, it's automatically keeping track of where it is in the file and will go to the next line. Nice catch.

So now there's another thing that happens, and you're like, good times, I'm all ready to read files, right? Almost. The one extra thing you need to know about – and this is the

part where it's just kind of one of the stickinesses of programming, is what happens when bad things happen to good people. You're like, what does that mean man?

Well what that means is, let's say I come along and I say, hey, buffered reader, open up students dot txt, and some evil person came along and deleted your students dot txt file, and so this guy's trying to create a correspondence between this buffered reader and students dot txt, and it says, there is no students dot txt. I got nothing here. What am I gonna do? It needs to signal that back to you somehow, and the way it signals that back to you is it says, whoa, this is just something that I didn't expect at all, and it gives you something called an exception. Okay?

So what is an exception, and what's involved with actually getting an exception and dealing with exceptions, okay? So the idea here is what we refer to when an exception happens, right? Like when this guy tries to look up this file and the file doesn't exist, is – the term we use is it throws an exception. That's just life in the city, all right? So let's say I'm the buffered file reader, and I'm trying to read students dot txt, and the students dot txt doesn't exist. So I say, that's an exception. What am I gonna do with the exception? It's gonna throw the exception. All right?

So someone gets the exception. You got the exception. You don't just leave it there, you got the exception man. All right. What are you gonna do with the exception?

**Student:**I caught it.

**Instructor (Mehran Sahami):** You caught it, that's important. Hold on to it for the time being, okay? Because it's important, right? That's a valuable commodity, so just hold that exception, hold that thought, okay?

So when an exception gets thrown, there has to be some notion of catching the exception, hopefully, and if no one catches the exception, your program is gonna stop executive, because if there's no one there to catch the exception, that exception is just gonna sort of go up all possible functions in your program and say, hey, is there anyone here to catch me? There's no one here to catch me. Okay, your program's done, because no one caught me and I'm a pretty important thing to catch, okay? That's why I needed you to catch it.

I'll set you up for catching it, because I know it took a little effort, but it's important to catch it. Just hold onto it for the time being, okay?

So how do you actually catch an exception? Or how do you actually tell the computer, I'm going to do something that may involve an exception. So you actually need to set it up and let the computer know, hey, I'm gonna do something that may involve an exception. The way you say that is you say try. I want you to try something. This thing may throw an exception, so be prepared, but I want you to try it.

Inside here you have your code for file access, and that's whatever you're doing with the file, trying to create a new buffered reader with the file, reading in lines one by one or

closing a file. Any time you're trying to do something that involves a file access, you need to put it inside this thing called a try that says, hey, I'm gonna try this out, it may throw and exception.

Now what happens if it does throw an exception? You have something over here where you write catch, and what you're gonna catch, in this case, is something called an IO exception, because it's an exception that happens when you're trying to do IO, or input output, and there's this ex thing which is actually the exception. It's that little envelope that you actually catch in your hands.

And then you have some code here which is how to deal with the exception, okay? So if you are doing something inside here and some place, as soon as you get to it, says, whoa, something bad happened, exception, it does not execute any more of the remaining statements inside this block for try. As soon as it gets an exception inside here, it says, whoa. Here's an exception. It throws it. Where are you gonna catch it? You're gonna catch it here, which means as soon as an exception gets thrown even in the middle somewhere here, the rest of this does not get executed and it comes over here to this code to deal with catching the exception.

If it gets through all of the code where you said, hey, I'm gonna try something dangerous, it gets through all of it and it doesn't throw an exception, this code down here is not executed. So this code down here is only executed if an exception was thrown, okay?

So what does that actually look like, okay? Let's write some code that deals with trying to catch an exception. Just keep holding it. It's a good time. We're kind of running our program in slow motion. It's kind of like, you got the exception, and normally you'd be like, catch, here I'm gonna do something with it. Here we're kind of like – we're taking our time. It's mellow, it's good. Here, I'll give you a little more candy just to keep you awake with the exception.

All right. You never thought just by like sitting there you could just accumulate food. All right, so we have private. We're gonna return a buffered reader, so this a function – or a method, we're gonna write, let's say, private method inside some class. It's gonna return a buffered reader, and what we'll this is we'll call this open file, so the name of the method – let me make this all a little bit smaller, so I can fit it on one line.

Private buffered reader – and what it's gonna get past is basically some string which is a prompt to ask the user for the name of the file to open. So that's a common thing you wanna do. You ask the user, hey, what file do you actually wanna open for reading?

So how might I do that? I'm gonna start off – I wanna return one of these buffered readers, which means I'm gonna need to have a buffered reader object in here at some point. So I'm gonna declare a buffered reader as a local variable rd, and I'm gonna set it initially to be null, which means I created the buffered reader – at least, I created a pointer for it, a reference for it. I have not actually created the object yet, so I can't say I

actually pass – make any method calls on rd yet. I've just sort of created the local variable. I'm gonna actually create the object in just a second.

I'm gonna have a while loop here and say, while rd is equal equal to null. So I know that that's gonna execute the first time through, because I said it equaled to null to begin with, and then what I'm gonna do inside here, so I have a brace there, is I'm gonna say, hey, I'm gonna try something that may be dangerous.

So what I wanna try doing is, I'm going to ask the user to give me the name of the file. So I'm gonna have string name equals read line, and I'll pass it whatever prompt was given to me as a parameter, right? So I'm just asking – all I'm doing here is asking the user to enter the name of the file, right? And whatever prompt I write on the screen is – excuse me, just whatever was passed in here. So it might have been something like, please enter file was the string. So I'll write out please enter file and ask the user for the filename.

Now here comes the dangerous part. I wanna create the buffered reader. So I say rd equals new buffered reader, and the parameter I'm gonna pass to a buffered reader is new file reader, and the parameter I'm gonna give to the file reader is whatever file name the user gave me. So this is just name.

So what this is gonna try to do is, whatever the user gave me, it's going to try to create one of these new buffered reader objects that corresponds to the file with that name that's actually on disk. If that file exists and it creates the correspondence, life is good. rd actually is now pointing to some valid object which is a buffered reader. If that file does not exist, I've thrown an exception. I say bad times. The user mistyped something or they gave me the name of a file that doesn't exist. Here's an exception, buddy. It's not there, I can't create the correspondence.

If that exception thrown -- right, I never created a new object, which means I never returned a value assigned to rd, which means rd still has the value null. That's an important thing, because if I do this try, and the file's not there, I'm gonna catch this IO exception – exception ex, and what I'm gonna do when I catch that exception, I'm not actually gonna do anything with the exception, the parameter ex that's passed in. All I'm gonna do is just write a println to the person saying, like, bad file, and then I'm gonna have a closed param, and – closed param here, or a closed brace for the while loop, so this brace corresponds to while loop, and if all of this works out, I'll just – I'll write it right over here on the side. I will return rd. That line would go right down here.

So what does this code do? It says, create the space for an object, or create one of these pointers to an object, but set it to be null. It's not a point – it's not pointing to a real object yet, and while it remains null, ask the user for a line, try to open up a buffered reader to that file.

If this works, life is good, I don't execute the catch portion, and rd now has a non-null value, because it's actually a valid buffer reader object, and so when I try to do this while

loop again, rd is no longer equal to null which means I'm done with the loop and I will return this object that I just created, which is actually this object that is a valid correspondence to a file. So that's a very convenient way, in one method, to encapsulate all the work of opening up a file and getting a valid reference to a buffered reader that refers to that file.

If I tried to create this file reader and this file, for whatever reason, did not exist, an exception gets thrown. If an exception gets thrown, execution doesn't complete at this line, so rd never gets assigned a new value, and it immediately goes to the catch and says, hey, here is the exception. I couldn't find the file. What does the exception catcher do? It says, hey, bad file, buddy. And then it's done. It just reaches the end of the catch part, and execution continues. So it comes to – the end of the while loop comes back up here and rd still has the value null, so it asks the user, hey, enter a file again, and tries to do this whole process again, and you'll get out of this loop when buffered reader actually gets a valid object.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah, that's a good point. What you wanna do is, when you're trying something dangerous, you don't wanna just tell your whole program – you don't wanna say, hey, program, let me put everything inside a try, because I'm gonna be doing all this dangerous stuff, and I don't know when I'm gonna do it. I might do a little here and a little over there, so put it all inside this huge try block. That's really bad style.

What you really wanna do is only do as little as possible in the try, and say, hey, I'm about to do something dangerous. Let me encapsulate that in one of these things that's a try catch – we refer to it as a try catch block, because we have a try and a catch, and as soon as I'm done with the dangerous part, then let me get out of that try block, because I need to know that, okay, now I'm not doing dangerous stuff anymore, and the programmer needs to know that when they see the files. They need to understand when you're doing dangerous stuff versus not, okay? So any questions about that, this notion of throwing an exception?

All right. So now here's the interesting that happens, okay? Sometimes when an exception gets thrown, you know what you wanna do, like, you wanna say, hey, that file didn't exist. I write out bad file, and I keep executing. Sometimes you get an exception and you have no idea what to do. You try to – you created this correspondence to students. That works just fine. And you start reading lines from the file, and it's just fine, but before you get to the end of the file, what happens?

Your roommate comes along and deleted the file, or like, beats your computer on the head and your disk crashes or whatever and your program's still executing, but your disk is no longer working. This has actually happened before. You try to say, hey, buffered reader, get me a line, and it says – it says, no more line. Like, I thought I had the file and everything, but there's no more line. I don't know what to do. What does it do? It throws an exception.

Sometimes when it throws an exception, what are you gonna do here, right? You just tried to read a line and you're like, what, the line doesn't exist? But the file existed, and now you're telling me the line doesn't exist? What am I gonna do? Sometimes you don't know what you're gonna do, and when you don't know what you're gonna do, what you do is you say, hey, you know that exception that got thrown to me? I'm just gonna keep throwing it.

So throw the exception back to me. And somewhere – now I don't feel so bad about missing you the first time. Someone gets the exception, right? And if I didn't get it, like you throw it to me and I'm just like, what exception, right, and I'm your program. The program just died. Someone comes along and beats me on the head, and says, hey, you didn't catch the exception, you stop executing. But sometimes like, someone gets it, and they're like, hey, I know what to do with the exception. It's all good. All right? And they get something out, and they're like, yeah, really all the exception was was some clothing I needed you to wear. You didn't know how to wear it. I'm sorry, you're not quite as stylish as I am.

So I know how to do with the exception. Once I know how to deal with the exception, I'm done with the exception and I kind of keep executing from where I'm at, but I appreciate you catching the exception to begin with, and if you didn't know what to do with it, you just keep throwing it along its way, okay?

So one more time, just for throwing it. And to the people around you, for getting pelted with candy. All right. So what does this actually look like in code? Okay. So let's look at a little code. Come on little guy. Wake up. Sleepy Mac, sleepy Mac. So what does this actually look like in code?

What's gonna happen is, here is our little function that we just wrote over there, our method, return to buffered reader. So this is all the same code. It creates a buffered reader that's null. While it's null, it asked the user for a file name, it tries to create a new buffered reader with that file name. If it doesn't exist, it says, nice try, punk. That file doesn't exist. And it asks again, all right?

So this is just fine. This is what we wrote. This is an exception that we know how to deal with. We don't actually use this ex portion. You don't need to care about that or whatever. You just say, yeah, I know the file didn't exist, so I'll write out a message to the user and keep executing.

Now down here – don't worry about the set font, that's just making the font bigger. I'm gonna say, hey, I'm gonna call my open file – this is basically a passively aggressive program. It asks you real nice, please enter the file, and if you get it wrong, nice try punk. That file doesn't exist. And then it says, please enter file name, all right? We'll make no more comments about passive aggressive behavior.

But it calls open file, it passes in this prompt, and what it's guaranteed to get back is an rd object that is a valid file, right? Because if it didn't get a valid file, it just keeps looping

here until it got a valid object that it could actually correspond how it corresponded to the file. And you're like, that's great. Now I need to read the file.

Reading the file is a dangerous thing to do, because I'm referring to this file and I don't know if something bad will suddenly happen and the file will go away, so I put this inside another one of these try catch blocks. So not only do I have a try catch block for making the – for opening the file, every time I try to read the file or potentially close the file, I need to have one of these too.

So this guy comes along and says, okay, I'm gonna have a while true loop and read all the lines of the file, kind of like Neron showed me before. So I'll have this variable line that I just read a line from the file. If that line is null, then I know I've reached the end of the file, so I'm done, and I break out of the while loop and I will come here and close the file. Always remember to close your file. It's just good practice. If the line is not null, I'm just gonna write it out to the screen, and I'll say, read line, and I'll write out, inside little brackets just so it's clear what the beginning and ending of the line were, the line that I write in, okay? And I'll just keep doing this, and when I'm done, and I don't get an exception, I close the file, and I will skip over this catch portion and just keep executing, which means my program's done.

If, however, while I'm reading the file, one of my – I say, read line, and it just doesn't exist, I come over here to the exception, and I say, hey, I got this IO exception. I don't know how to deal with that, right? I don't know how to deal with the fact that I just read a line and it doesn't exist, so I'm just gonna throw an exception to someone else.

So what I do inside here is, I will throw a new exception, and the exception you will always throw, if you don't know which exception to throw, which is most of the time, you won't know which exception to throw, is something that we call the error exception. So you say, hey, I got an exception, I don't know how to deal with it, maybe someone else who was trying to access me put me inside a try catch block, so I'm gonna throw an exception up to them and see if they can catch the exception, and the exception I'm gonna throw is an error exception, and error – when you throw an exception, it has a parameter, and that's just the exception object that you're throwing along.

You actually got some exception object past you when the exception was thrown. You didn't wanna look inside, you didn't wanna deal with it, and as a matter of a fact, you don't need to look inside or deal with it. All you're gonna do is just continue to pass that exception object up to the person who may have called you, okay?

So, when you throw an exception, execution would stop in the method that you're at, at that point where you throw the exception the method will end, and it will throw an exception up to whoever called this method, okay? Unless this happens to all be inside some other catch block inside here, in which case you'll catch the exception yourself, but if you're not trying to catch the exception yourself, it'll just get passed up and someone else will catch it, okay?

So any questions about this? Here we know how to deal with the exception, here we don't know how to deal with the exception, and if you wanna actually be able to pass an error exception, that's something that's defined in this thing called ACMU till. So it's an exception that's defined in the ACM libraries. You should import ACM dot u till dot star to be able to throw one of these error exceptions, okay?

So this is – let me just run this program, just so you can actually see that it works. This is another file example. And then, I will show you something that's extremely cool, which is basically, in 10 lines of code, you can copy files, doing it all in Java. And you're like, yeah, man, but I can copy files in like four clicks of my mouse. Yeah, what if I just took your mouse and just busted it? Right? Then what would you do. And you're like, yeah, then I'd write a Java program to copy a file. Don't force me to bust your mouse. One of my other mice busted last night, and it was just a harrowing experience.

So I wanna enter the file name, and I say, the file name is – what was the called? Was it called Stanford dot txt? No. It was called students. I keep pressing the A. Students. Yeah, I can't even type anymore. Wasn't it called students? No, it was called students dot txt. I forget the students dot txt. And then when it finally gets a valid file and creates the correspondence, notice that you're reading the lines from the file one by one and it just them in strings and it's printing them out, okay? Any questions about that?

So, besides reading files, sometimes in life you also wanna be able to write files. So here's the quicky way to be able to write a file, okay? It's similar – very similar, it's actually – writing a file is even easier than reading one, if you can believe that, and you're like, but man, reading one is pretty easy. Yeah, it is, and writing one's even easier, so it's just that much cooler, and if I can find my chalk, it'll be even that much cooler. Here it is.

What you're gonna do is, guess what? You're gonna open a file to write. This is something – rather than using a buffered reader, we call it a print writer. You will create a print writer object. Two, you will write to the file, unlike reading from the file where you use read lin, it's very easy to write to a file. You just do printlns to the file. You use println, but you're actually gonna be passing println to some object here. What object? Your print writer object. And the same ways you've used a println to write on the screen, you can use it exactly the same way to write to a file, and when you are done, you will close the file. That looks real familiar, right? So let's actually see what that looks like in code.

So what I'm gonna show you is a little program that copies a file, line by line. So, once again, here is the exact same function we wrote before to open a file for reading, so this has our buffered reader stuff, and our little while loop, and keeps asking for file names until we read the file, so it's exactly the same code. We're just reusing the code. That's part of the beauty of code reuse, right?

What we're gonna do here is we're gonna open a file by asking the user to enter a file name, and this is the file that's going to be our reader. So our object is rd. Now we're

going to create a file that is a – or a object, which is a writer. So we're going to try, because we're gonna do something dangerous.

Creating a file is dangerous, or writing a file is just as dangerous, as reading one. As a matter of a fact, in some sense, in a more esoteric way, it is more dangerous. Why is it more dangerous? Because, if you try to tell the computer, write to a file which already exists, what's it gonna do? You might say, will it tell you that it already exists? Will it try to give you a different file name? No. It'll go smash the file that exists there to bits, and write a brand new file. Which means, yeah, if you're overwriting students dot txt, not a big deal. If you're overwriting your word processor, big deal. Be careful what filenames you give when you are writing, because if that filename already exists, it will get rid of the one that's already there and overwrite it with a new one, which is whatever you happen to write into it, okay?

So print writer. We're gonna create a new writer, and we want to, inside a print writer, just like we had this little thingy up here where we had a buffered reader and a buffered reader used a file reader over here, when we create a print writer – and this is all in Chapter 12 of the book, so you don't need to worry about scribbling this down quickly – a print writer, you say new file writer is the object you create and pass to a print writer, and you give it a filename.

Here I've – you could have given it like a string variable or whatever. Here I've just put in, directly, copy dot txt, so the thing I'm gonna create is the file called copy dot txt. All your files need to be in the same project as your code, so that's where our new files will get created. That's where, when it looks to read a file, that's where it's gonna look, is that same file that has your project.

So I have a while true loop. What am I gonna do in my while true loop? I'm gonna read, one line at a time, from my input file, from my reader. If my reader has no more lines, I get a null and I say, hey, I'm done reading. I'm gonna break out of the loop. If I'm not done reading, I've just gotten a line, I'm gonna write that line to the screen using println.

So println, without any object over here, writes to the console, because this is a println that is being called on the console program, and it writes to the console.

When I do wr dot println, that's sending println to the write – the print writer object, and it's writing out the line. So it's saying, here's this line. Hey, print writer, write this out to the file that you correspond to it. It says, okay, I'll go write it in copy dot txt. And I keep looping like this. I get a new line. If it's not the end of the file, then I write it to the screen and I write it to the file. If I go through this whole while loop, and finish off everything, so I finish the whole file, I get to where line equals null, I say, hey, I'm done reading the file, so close the file to read. I'm also done writing the file, so I tell the writing file, the print writer, to close itself.

And this whole time, if something bad happens and I get an exception, I catch the exception and say, I don't know what to do with it, I'm just gonna throw it to someone

else. Okay? And that's the whole program. That's all the code you need to copy a file. So let's run this and just make sure it actually works.

Operation in progress. And just to show you that I'm not lying – let me cancel this for a second. We'll open up some folders. Notice no copy dot txt in here. Here's the little project file that I have, okay? There's my copy file class and my copy file dot Java file. Here's my students dot txt file. No copy dot txt file. Now, magic. All right, we run. See, I had to tell you that, because I could have just cheated. I could've been – I just typed up the copy dot txt file.

Running, running, running. Come on, you can do it, in your glowing blue. I love the glowing blue. It's fun. Copy file. Please enter file name. Students dot txt. I'll spare you the incorrect files name. So it's copy – it says copying line. It writes out all these lines to the screen. Let's see if it actually exists. Copy dot txt. It's warm, it's fuzzy, it's fun.

Now if you look over here, just wondering, you might look in your project and say, hey, I don't see. I thought this shows me everything in my folder. I don't see copy dot txt show up here. Yeah, you go up here, you right click, you pick refresh. Copy dot txt. And there it is. Looks identical to students dot txt. There you. All right, any questions? Then I will see you on Monday.

[End of Audio]

Duration: 51 minutes