

## Programming Methodology-Lecture01

**Instructor (Mehran Sahami):** Alrighty. If you could have a seat, please, we need to get started. There are still a bunch of people coming in the back. Come on down and try to find a seat somewhere. If you can't find a seat, sit in the aisle as long as you're not a fire marshal. Anyone here a fire marshal? Good. We're fine. Come on in and sit in the aisles.

So welcome to CS106A. If you don't think you should be in CS106A, you think you should be somewhere different, now is probably a good time to go, not that I would discourage anyone from taking this class. I think we'll have a lovely time in here. But this class is CS106A or E70A, so if you're, like, "Wait. I thought I was in E70A," you're fine. They're the same class; it's the same thing. No worries, okay?

There's four handouts. They're in the back. If you haven't already gotten the handouts because you came in and you sat down, don't worry. You can pick them up on the way out. They're the same handouts. They'll still be there.

So just a quick introduction. That's what the first four handouts actually give you. They give you a little bit of an introduction to the class, what we're gonna cover, some logistics for the class and some other stuff. I'm gonna go over all that today so we can sort of get a good idea for where we're at, okay?

So just a quick show of hands before we get into a bunch of things in the class. This is kind of an intro-programming course; well, it is. I shouldn't say it's kind of an intro-programming course. It is an intro-programming course. And it's always good to get an idea as to how much familiarity you may have beforehand, okay? So just quick show of hands. How many people can recognize a computer that's on? Good, good. That's the prerequisite for this class.

So if you're worried about how much previous experience you've had or your friend who, like, worked their way through high school by programming for Google or whatever, don't worry about it because all you need to know in here is basically either how to turn a computer on or to recognize a computer that's on if you were to walk up to it and it were already to be on, all right?

So but a little bit more seriously, how many people have actually used a computer for anything? All right. I would expect most of you.

So now, we begin to bump it up a notch. How many people have used it for word processing? Okay. Most folks.

How many people have done web browsing? Yeah, I won't ask you what you look at, all right? It's just I don't wanna know.

How many people have actually created a web page? Okay. Fair number.

How many people have done any kind of programming before? Fair number. All right.

How about how many folks have done actually programmed in Java before? All right. A few folks.

How about another language, C, C++, BASIC, anyone program in BASIC? Yeah, oh, I love — that was the first language I learned, and it was kind of like the warm and fuzzy, and I felt good. There were actually people who argued that if you learn BASIC as your first language, you're brain damaged, then you're just beyond help. But if that's the case, we're all in the boat together because I'm probably brain damaged as well. The truth is I probably am, but that's a whole different story.

All right. So one thing you should know kind of up front is actually this course is gonna be provided eventually somewhere down the line as part of Stanford School of Engineering Free Course Initiative, which means not only are we recording this course to broadcast to a bunch of companies and industry who are watching this course, but we're eventually gonna provide it free to the world.

So how does that impact your life? And on the average day, it doesn't at all. The only way it does impact your life is just so you should know, the lawyers told me to tell you that your voice, should you ask a question, may actually be recorded as part of the video. As a result, your voice may end up going out to thousands of people or millions of people in the world. If you have an issue with that, come talk to me. If you don't, everything is just fine, all right?

Don't worry. We're not gonna put your picture up or anything like that. You might wanna be on the video, like, "Hey, ma, I'm on TV." We decided that we're just gonna not show anyone actually on the video, but your voice may actually get recorded, okay?

Now, along those lines, you may also notice there are some microphones in the room. So when you wanna ask a question, please make sure to use the microphone because that's not only good for people in here to be able to hear your question, it's also good for all the folks that this is getting broadcast to because not only are we gonna broadcast to the world, but there's actually some folks who are sort of watching this live now in various companies in Silicon Valley.

So it's real important that you actually use the microphone, so just remember that. And every once in a while, I might get on your case and be, like, "Please use the microphone." I'm not trying to be argumentative or anything. I just wanna make sure we pick up all the audio, all right?

So with that said, a little bit of an introduction. That's kind of a way of background. I didn't give you any sort of introduction. So just to introduce myself, my name's Mehran Sahami. I'm the professor for the class. Don't call my Professor Sahami, way too formal. Don't call me Mr. Sahami. That, I think of my dad. And don't call me Mrs. Sahami, or we're gonna have issues, all right? So just call me Mehran. We'll get along. It's just fine,

all right? It's to keep things a little bit more informal, but that way it's a little bit easier to discuss stuff as you go along.

There is also a head TA for the class, Ben Newman, who's standing up there. Get to know Ben. He has all the real power in this class. I'm just kind of the monkey that gets up here and gives the lectures. But Ben really is the one who's got all the power.

Along with the head TA for the class, we have a large section leading staff. So the section leaders here, could you stand up if you're here? They're kind of all over the place, some over here, some over there, and some over there. As you can see, there's a pretty large number of folks. And this isn't even all of them. We sort of have more — we just can't stuff them all into the room — who are section leaders for the class, and these folks are all here to make sure that everyone in this class has as good an experience as possible when we're sort of going through the class.

And the best way to reach all of us is email. So on Handout No. 1, you get my email and Ben's email. We'll tell you how to sign up for section. That's how you'll meet your section leader and get your section leader's email. That will all be coming soon. But email really is kind of a happy form of communication to get a hold of us, okay?

So with that said, I wanna tell you a little bit about this class and kind of what we're gonna do in here and what you should expect and make sure that you don't feel scared off by this class, okay? Because it really is meant to sort of be an interesting time.

But one question that comes up is why is this class called Programming Methodology, right? Why don't we just call this class, like, Programming with Java? And the real reason for that is that programming methodology is about good software engineering principles. It's about something that's much larger than just programming.

So some people, like, they'll go and get a book somewhere and they'll think they learned how to program by just reading the book. And they're, like, "Oh, I know how to program. Isn't that great?" And it's, like, yeah, you might know the mechanics of the language, but the mechanics of the language are nothing compared to understanding the software engineering principles that go into actually developing a software system.

And that's what you're gonna learn about in this class. You're gonna learn a lot of those principles. But in order to be able to use those principles and apply them, you also need to have the language to program in, and that language that we're gonna use in this class is Java.

So the way I like to think about it and the way I tell a lot of people is writing a good program or learning how to program is like learning to be a good essay writer. And you're, like, "Oh, but part of the reason I'm taking this class, Mehran, is that I don't like writing essays." That's fine. It's okay. Trust me. I didn't like writing essays either.

But the whole point is that when you write an essay, it's not a formulated kind of thing. You're, like, "Well, what about five-paragraph essays?" Yeah, just block that from your mind. That was a bad time, right? That was just, like, '70s education at work. It's not a formulated kind of thing.

There's an art to writing an essay, right? In order to write an essay, you need to know a language. You need to know English or German or Hindi or whatever language you wanna use, but then you use that language to write an essay. Just knowing the language doesn't make you a good essay writer though. Being a good essay writer makes you a good essay writer.

So that's the same difference in programming and software engineering. Knowing the language, in order to be a good programmer, like a good essayist, you need to know a language to write your programs in, whether that be Java or C or C++ or whatever. Here we're gonna use Java.

But just knowing the language doesn't make you a good software engineer and doesn't make you understand what the principles are of writing good software, which is what you're also gonna get in this class in addition to the language, and that's kind of a key thing to stress.

So if you're sort of worried, if you were kind of looking around and you saw a bunch of people raising their hands when I asked, "Do you have any previous programming experience?" and some folks raised their hands, and you got a little worried and you're like, "Oh, am I gonna be in some sense at a disadvantage because I haven't done any programming before?" The answer, plain and simple, is no, okay? You're gonna learn everything you need to learn from the first principle because as a matter of fact, in some cases you might be in slightly better shape. That's not necessarily to say that that's the way it will be.

But how many people are Star Wars fans? Just wondering. Anyone? I'm talking about the old-school, original, like, three movies. Those were so good, and we're not — no George R. Binks here, all right? So if you remember — and sort of I'm a big Star Wars fan, and that's just a whole separate point. But in the second movie, Yoda actually said something which I thought was quite profound, which is he says sometimes you have to unlearn what you have learned.

And one of the things we actually find is that some people who are self-taught programmers, some of them are just fine, and some of them are very good. But some of them have picked up some really bad habits along the way, and it's like being a bad essay writer. And to go from being a bad essay writer to a good essay writer, in some cases, can actually be harder than from not being an essay writer to being a good essay writer because you have to unlearn the bad habits.

So if you're worried about, "Oh, I've had no previous experience," don't worry. You're okay, blank slate, you're just fine. And now if you're thinking, "Oh, I have some previous

experience. Do I have bad habits?" Don't worry. You'll be fine, too, okay? So it's all gonna work out.

So the next question that kind of comes up — hopefully that helps put some of your fears aside. Another one of the things is that we really strive to make everyone successful in this class, okay? At some other schools, people wanna do computer science or they wanna do an engineering major or whatever. And you come into the first day of class, and they say, "Oh, only one third of you are actually gonna make it through this program. And look to the person to your left and look to the person to your right, and only one of you will make it through." And you're, like, "Oh, man, that's real nice." It's not like that here.

As a matter of fact, we want all of you to be extremely successful in this class, which is why we have a huge course staff, which is why over years and years we've refined how we do a lot of the teaching in this class to make sure you have the best possible experience and to make sure that everyone gets through.

And the important thing about that is that you're not competing against anyone except yourself in this class. It's not like we're gonna have a curve and we're gonna say, "Oh, we have a certain number of "F"s and a certain number of "D"s and a certain number of "C"s." All we really have going into it is an expectation that when you get out of here, there's a set of stuff we want you to know. And if you know that stuff well, you get an "A." And if everyone knows that stuff well, everyone gets an "A." And I got no problems with that. Registrar might have a problem with that, but that's okay. You don't need to worry about that.

So you don't need to think about, oh, is someone else doing better than you or whatever. And we'll talk about issues of collaboration in just a little bit. All you need to think about is learning the stuff yourself as well as you possibly can, and you'll be just fine and you'll get a good grade, okay? So that's really all we ask, which is not a trivial amount, right? It requires you to really understand the material.

So another question that comes up is are you in the right place, right? This isn't the only introductory programming class at Stanford. And so I wanna spend a little bit of time making sure you actually are in the right place by going over some of the different options.

So right now, as you know, you're in CS106A. And CS106A, we're sort of happy over here, right? As a matter of fact, we're not only happy, we're happy and we're also a little bit loopy, right? There is no previous programming experience required, as I mentioned, right? All you need to know is basically if you can get to a computer and know how to figure out that it's on, you're in good shape.

But what 106A does is it's a real rigorous class. You learn programming in here, and you learn it in a way that makes you ready to be an engineer if you so choose to be an engineer. That's not to say you're all gonna be engineers. I would love for all of you to be

computer science majors, but statistics in the past show only about 6 percent of you will be computer science majors. That's not because we turn anyone off to computer science; it's because we make programming accessible to so many people that you don't have to be a computer science or a Double E or even an engineering major to do extremely well in the class.

And we actually have sort of a significant percentage of the entire campus undergraduate student body at Stanford actually goes through this class and does well, okay? So don't worry if you're, like, "Oh, but I'm not really a CS person." I hope we'll turn you into one by the end of the class. No, it's okay. But you'll be prepared if that's what you wanna do. So this leads into a whole engineering sequence that can go on to other engineering majors or the computer science majors.

If you're, like, "Huh, I'm not sure if that's really what I wanna do. As a matter of fact, I'm so sure that's not what I wanna do, I only wanna get the general educational requirement out of the way, and I'm positive there is nothing else I wanna do. Really, no matter how much I like it, like, there is no way you're gonna drag me into anything that would involve anything remotely techie." They're the class CS105. And this is happy, yeah, this is kind of, oh, we're happy in our little happy world. And I don't wanna say it's holding hands and singing, "Kumbaya," because that's not what it is. It's a real class. But it's meant to be a general educational requirement, right? It doesn't lead into the 106s. It's meant to be its own self-contained class. You do some Java script in there. You do a little bit of what computers are about.

Computers in society is a good time. We all hold hands. We're all happy. I don't teach the class, so I don't actually hold hands. But it's a fun time, okay? It just doesn't lead to anything else. So think of this as kind of a terminal class, right? So it's sort of like, well, we'll hook you up to the IV drip.

And you're, like, "Well, 106A, you told me I don't need any previous background. Well, hey, Mehran, I got lots of background. I got so much background, it hurts. I got AP background, I got working through school doing software engineering background. I'm not sure I should be here." That could be the case.

We have another class called CS106X, and as the "X" kind of implies, it's sort of the extreme games version of the class. No, it stands for accelerated, right, because "A" was already taken, so we had to come up with something else. So the way CS106X works is it really is a very fast-paced class. It's meant for people who've got previous AP exam credit, like, got a 4 or 5 on the AP, or have had significant and prior programming experience before.

If you're not sure which one of these classes is for you, you can come talk to me afterwards, or I'd also encourage you, you could go to pick up the syllabus for CS106X and compare it to CS106A. This class is all in C++. And if you're thinking, "Hey, Mehran, I'm doing 106A. I wanna learn Java and C++," don't worry. You'll eventually, if you so choose, take a class called CS106B, which is where this class sort of leads to,

which is C++ and all of the other stuff you would have learned in this accelerated class, okay? So you still certainly have that course path.

So don't let anyone make you think — I know a lot of times, and especially for Stanford students, you come in here and you're, like, "Well, every class I took in high school was like an honors or an AP class, or if it wasn't an honors or an AP class, like, I had to tie half my brain before my head because I'm just that hardcore." And so everyone just wants to, like, do the most hardcore thing they can, right? And what I'm here to tell you is that you shouldn't necessarily think about it that way. You should think about it as where you feel most comfortable.

Some number of years ago, let's just say greater than 10, maybe 15, I was sitting where you're sitting right now, literally. I was in CS106A in Terman Auditorium as a freshman, okay? It was perfectly fine. It worked out. I went to grad school, did the faculty thing. It's just fine. It will open your doors to CS. You're not at any kind of disadvantage by starting here. So know where you've been, literally. Like, that seat right there was where I was most of the time. So just something to keep in mind in terms of the different options that are actually available to you.

Now, with that said, let's just assume for the rest of this lecture that this is the right place for you. And if it's not, well, afterwards we can kind of talk about it, or if you really are convinced now that it's not the right place, you can feel free and try to scramble over 20 of your classmates and actually leave the room, which is probably impossible.

All right. So a few other things you should know, some mechanics. So Handout No. 1, should you wanna follow along at home, is the class web page. And so all the stuff that we think of as course materials, including online copies of the handouts, things that you'll need to do for the assignments, announcements related to the class are all on the class web page, which is [www.stanford.edu/class/cs106a](http://www.stanford.edu/class/cs106a). And because that's just kind of a whole bunch to remember, we make your life easy and so there is an equivalent form of the URL, which is just [cs106a.stanford.edu](http://cs106a.stanford.edu), which is the easy thing to remember. You put that in, it'll take you to the class web page, okay?

And you should check that regularly because all the announcements and handouts — we'll give out hard copies of all the handouts in class, but should you happen to miss class for whatever reason, you wanna go print whatever copies of the handouts we're actually giving out, you can find them all on the web page, okay?

Now, there's this funky thing about units. So you may have noticed that this class is for three to five units, and that kind of brings up the natural question, "Should I take it for three or five units?" If you're an undergrad, you take it for five units, end of story. That's life in the city. Congratulations. Five units.

If you're a graduate student, you can have the option of taking it for three units if you want, if you're gonna run into some unit cap. It doesn't change the amount of work you have to do. Welcome to graduate school. Same work, fewer units. So that's just the way

life is. If you have a unit cap and you're a grad student, in three units you can take it if you want. You can take it for five if you want as well. If you're an undergrad, you take it for five, all right?

So why is it five units? And you might think, "Hey, this class only meets three times a week. How come it's five units?" Well, it actually has a fourth meeting every week, which is your section, and that's something you should sign up for. So how you actually sign up for your section is sections are at a bunch of different times. You don't sign up for them in Axess, even though they're all kind of listed in the time schedule. That's not where you sign up for them. In Axess, you just sign up for the class.

How you sign up for a section is you go to a website, [cs198.stanford.edu/section](http://cs198.stanford.edu/section) and this will give us a list of preferences for section times that you wanna sign up for, and there's some matching process that goes on. It takes all your preferences into consideration with the whole system, and eventually you get an email by sometime early next week that tells you what section you're in. And section's 50 minutes, once a week. It's required to go to. It's actually gonna be part of your class participation grade, which we'll talk about in just a bit, okay?

When do these sign-ups happen? They happen between 5:00 p.m. this Thursday is when they go up. So if you try to go there now, you can't sign up. Remember 5:00 p.m. Thursday. So they're up, and then they're down at 5:00 p.m. on Sunday, okay? So make sure you sign up probably this weekend. If you're planning on being out of town this weekend, you wanna sign up before you go. Sign up early, but don't sign up often because you only need one section, okay?

If you're an SCPD student — every once in a while you'll hear me refer to SCPD students. That stands for Stanford Center for Professional Development. They are the folks in industry who actually take this class via broadcast. If you're an SCPD student, you're automatically enrolled for a section, so you don't actually need to do this, and your section will meet at — so for SCPD — and if you're wondering what an SCPD student is, you're not one, okay? So SCPD section meets Friday from 1:15 to 2:05. It meets live, if you wanna go there live, in Skilling Auditorium. But if you're watching it remotely, it meets on Channel E2. I know it seems weird to say it meets on channel — what does that mean? It meets on Channel E2, okay? That is grammatically the correct way of saying it.

All right. So there's a little bit more administrative stuff. Now, textbooks, right? Textbooks, there's nothing quite like the extortion that is textbooks. So there are two textbooks that are required for this class. Well, one's a course reader and one's a textbook.

The course reader is called, Karel the Robot Learns Java. You can pick it up at the bookstore. It's relatively cheap. It was actually written by Eric Roberts here. And surprisingly enough, the textbook for the class was also written by Eric Roberts, The Art and Science of Java, which is available now in your local bookstore, including the bookstore on campus, so you can go and pick up a copy of this.

So both these things you actually wanna have because they're required for the class. We'll go through all of them. We'll go through basically everything except the last chapter of this book. So you sort of get your money's worth. We're just gonna do it a little bit out of order, but we'll go through the whole thing, okay?

So email, how many of you have email accounts? All right. I will ask the reverse question because I think at this point, some people just don't wanna put up their hands. How many people don't have email accounts? Odd how that is not the complement of the folks who had their hands up previously. Email's required for this class. Chances are, by being at Stanford, you've already gotten an email account through your SUNet ID, but if you don't have an email account, get an email account and that's how you'll stay in contact with us. That's how we'll stay in contact with you, except we'll also meet with you live in person, but email is kind of the general method for communication.

As a matter of fact, for your first assignment, and part of your first assignment is to send us an email, just because we love you and we don't get enough email as it is. So you need to have an email account to be able to do that. So if you have not already, you can kind of get ahead of the game and go set up your email account. Now, don't worry. You'll get the first assignment next time. So you still get, like, two days of breathing space before your assignment goes out, okay?

There is also gonna be lots of handouts in the class. They'll be either given out in class, well, they will be given out in class, but we'll also post them online in case you miss them.

And how much real work do you do in this class? That's always kind of an interesting question. So let's talk a little bit about assignments and a little bit of other logistical things. So assignments, we'll just call them the dreaded assigns. There are seven programming assignments. And if you look at the syllabus Handout No. 2, it tells you when all of them are due all the way through by day, so you can plan out your whole quarter. It's just that much fun, okay?

And these seven programming assignments are weighted slightly more toward the last assignments because the assignments will tend to get more complicated. That doesn't necessarily mean there'll be more programming; it just means conceptually, they'll become more complicated, so we tend to weigh them more toward the end of the class. So the later assignments count more than the early assignments.

How you're gonna be actually doing your programming is using a little tool called Eclipse. And Eclipse thankfully is free, so you don't have to pay for it. As a matter of fact, you can download it from the CS106A website. And if you're wondering how you do that, don't worry. We'll give you a handout next class that explains to you the whole grueling process of downloading and installing Eclipse.

And you can use this either on the Mac or the PC. So if you have your own computer, you can certainly work on this yourself. You just download it to your own machine. We'll

explain the whole process in a handout. If you don't have your own computer, the public computer clusters on campus will have Eclipse installed on them, and so you can use Eclipse there. So you're sort of happy to go either way, okay?

Now, the important thing, remember I mentioned that whole notion of software engineering in the class, and that's something we take really seriously, so seriously as a matter of fact that when you turn in your assignments, one thing we could do is we could take your assignments and we could just kind of look at it and go, "Yeah, interesting, 'B.' Here you go. Thanks for playing." And you don't learn a whole lot from them. So in order to actually learn a lot from your assignments, we could take your assignment and write a whole bunch of comments on it and hand it back to you. Even that's kind of not enough.

What really is a little bit more that makes it more fun is every week after you turn in your assignment and your section leader looks it over and grades it, you'll actually meet with your section leader for about 10 to 15 minutes every week or every time an assignment is due to actually go over in something referred to as interactive grading.

And it's a chance to sit there and talk with an actual human being about what's good in your assignment, what are some of the things you need to work on, what are some of the software engineering principles you need to develop. And that way, you can really sort of get more detailed information and be able to ask questions to develop yourself as a programmer as well as get help if you need help, okay?

And that's in addition to going to section, going to class and all that stuff. So it's another 15 minutes a week. You'll actually schedule that time with your section leader on a regular basis when you're gonna have interactive grading or just affectionately referred to as IGs because at Stanford, everything's just short and we just can't say, like, psychology; we have to say psyche. So it's IG. Just remember that, all right?

And then how are these things graded? So the other thing we could do is I told you we could just write "B" and hand it back to you. But we found that that's not really great because people get all wrapped around the axle about the grade.

And so for a while, we did numbers and we're, like, huh, why don't we give a number between 1 and 20? And so what happens there? People get all wrapped around the axle about numbers.

So then we thought, huh, what was a happier time when we were in school? I remember when we were in school, and we used to get back assignments and they had, like, smiley faces on them. Well, we can't do that because then it doesn't appear to be a rigorous Stanford class.

So instead of the smiley face, we come up with something else, which looks surprisingly like this. It's kind of involved to actually draw, so I need to erase the board to do it. Check. That's kind of the beginning of our grading scale, okay?

And the way our grading scale works is we start off with a check in the middle, which says this is a pretty solid program. It meets all the requirements for the program. Maybe it's got a little problem here or there, but it's a check. Then we have sort of two grades on the two sides of it: check plus and check minus.

Check plus is, like, solid. You did a great job; you got everything right; things look good, a nice style in your program, nice software engineering, and the program works flawlessly. Good job. This is like total "A." Check is kind of like, yeah, you're sort of there. It's kind of like "A" minus, "B" plus, maybe on some occasions "B." But it's kind of like it's pretty good work; you're in pretty good shape here. And so a lot of grades in this class ends up being check pluses and checks, and if that's the case, you're perfectly fine grade-wise.

Check minus, as you can imagine, this is kind of thinking about "B," "B" minus. It's, yeah, there are some slightly more significant problems with your program.

But that's not where it ends, right, because we wanna be able to even shoot for in some sense bigger gustoes. There was a plus and a minus. So plus is like, oh, nice job, kind of a hearty pat on the back. If you get pluses all the way through on all your assignments, you're in a pretty good candidate to get an "A" plus.

And minus, like, just take good over here and replace it with bad, it's kind of like, oh, bad times, right, or maybe, you know — but even there was, like, more significant problems with this program or just the style on the program is just really bad. But even there, we don't stop. And you're, like, "Come on, man. Like, I thought the whole reason was to simplify this." Don't worry.

And it gets even better because we have a plus-plus and a minus-minus. And at this point, we've run out of board space, so we can't go any further. But a plus-plus is just outrageous, right? It's the kind of thing — so this is the kind of thing your section leader can't actually give you without coming and talking to Ben and I first because they get a program that just goes — it has to actually exceed the requirements for the assignment. It's by a long shot. Like, you'll get all your assignment requirements, and what we encourage you to do is you can do a grade assignment and get everything right and have good style, and you'll be in this category. And for the later assignments, you may be in this category if it's flawless.

But we'll actually — if you want to go for the plus-plus, go beyond the assignment requirements. And the way we think about the plus-plus, it's a program that makes you weep in a good way. It's just like your section leader sees it, and they're just, like, this is so good, I've gotta show someone else. And they come and show Ben and I, and we're, like, sitting there looking at this on a monitor, and, like, tears are just welling in our eyes, and there was, like, soft violin music playing in the background and we get out the wine and cheese.

So this is just, like, this is the kind of thing that gets you, like, remembered and the, oh, if you want a letter of recommendation, just ask because you got a plus-plus. Like, oh, it's awesome, right?

There are very few of these in a quarter. So just by sort of way of comparison, in a class this size, probably throughout the span of the whole quarter, I'd expect there to be maybe ten plus-pluses, I mean, ten assignment plus-pluses, not ten students who get plus-pluses across the board. So it's really something to strive for, but if you strive for it, like, we're giving you the credit for it. And this gets remembered and you get, like, extra credit and everything.

So we're left with this, right? This assignment also makes you weep, but not in the good way, right? It makes you kind of weep in the sense, like, I look at them and I'm, like, oh, man, like, what did I teach? Like, where did I go wrong, right? I, like, blame myself. I blame you a little bit, but I blame myself. And this is really just, like, the program is just, like, it's a shell. Like, there really wasn't much effort that was put into it. Yeah, you slapped something together or it doesn't really work, that whole deal.

And then if you don't turn anything in, we do kind of reserve the zero to distinguish from the "made really bad effort" versus "didn't make any effort at all." And we just won't talk about these, right? Let's just hope we can avoid those if possible. But that's kind of how the grading scale works now.

Now, at the same time, I trust all of you to be responsible people. And every once in a while, something bad happens to a good person, and there's an assignment that you'd like to be able to turn in, but for whatever reason, you can't turn in on time. And I just wanna treat you like adults. I don't want you to have to worry about coming in and asking for an extension or, like, "Oh, I had this really hard thing in another class, and I couldn't do it at the same time." Up front, everyone gets two free extensions, okay? So in terms of late days — we refer to these as late days, strangely enough — you get two free ones.

What a late day is, is a class day. They're not 24-hour days, but class days. So if something is due on a Wednesday, you turned in on a Friday, that's a late day. That's one. You turn it in on the following Monday, that's two late days. You can split up your two late days among two different assignments. You can use them both on one assignment.

But we encourage you to not use them at all because if you use your late days, you fall behind in the class. The way you should think about these things are these are pre-approved extensions. They're not the kind of thing where you just think, "Oh, yeah, I'm not gonna do the assignment because I wanna go and play Frisbee golf," right? Think of it, well, you wouldn't come ask me for an extension — you might, but you probably wouldn't ask me for an extension if you're, like, "Hey, hey, Mehran, can I turn in the assignment, like, on Wednesday because I'm playing Frisbee golf this afternoon," right? If you would feel embarrassed asking that question, you probably don't wanna use one of your free late days.

But something happens like, oh, it's a tough week, you've got midterms in other classes and you got this assignment due or whatever, that's a good time to use it. So we just trust you. And most people, we actually encourage you not to use them because it just makes you fall behind in the class.

Because we trust you and we give you these two up front, getting extensions beyond your two free class days is virtually impossible because we sort of up front said, hey, it's your responsibility. We're giving you two freebies. We're not gonna give you a third extension. Imagine if you had to come ask us for three extensions. By the third one, we'd be, like, okay, what's going on, which is why we don't necessarily give extensions beyond these two.

The only time we might give an extension beyond the two free ones is for something major, like death in the family or, like, serious medical problems that might require surgery or something like that. Every once in a while, unfortunately, that happens. I hope it doesn't happen in this class. But those are the only kinds of things that we give extensions to beyond the two free late days.

Importantly, don't ask your section leader for extensions. They cannot grant you extensions. Only Ben, who has all the power in this class, can give extensions, which is why you should get to know Ben and then hopefully you won't need to talk to him about extensions, okay?

So other thing to keep in mind is that three days late is the max. Beyond three days late, which is basically one class week, if you think about late days being class days, we will not accept an assignment. And the reason for that is at a certain point, you're so late, you're better off just doing the next assignment, letting the old one go. So to sort of enforce that policy, after three days, we don't accept that assignment late anymore. It's just gonna be a zero if it's not turned in, okay? And that just kind of forces you to keep up.

Couple other minor things, well, I shouldn't say they're minor things. They're actually kind of important. Exams: There's two exams in this class. There's a midterm and a final. Both of them are, well, I shouldn't say both. The midterm is out of class. It's from 7:00 to 8:30 p.m. on Tuesday, October 30. And it's on the syllabus. It's there. It's on the syllabus; it's on Handout No. 1. We repeat it multiple times. The date will eventually be announced when we get close to the midterm.

But if you have a conflict with this time, you need to send me email, okay? You can send me email a little closer to the midterm because I'll announce it again for people who have conflicts. But since it's an out-of-class exam, you need to send me email if you have a conflict. I'll get all the constraints from people who have conflicts and try to schedule an alternate time if there's enough people with conflicts. But 7:00 to 8:30 is when you need to know about the midterm.

And to make up for the fact that we have an out-of-class midterm, I actually give you sort of a belated free day, which is the Friday of the week of the midterm, we don't have class to make up for the fact that we made you come to the midterm outside of class. But the midterm's an hour and a half, and we can't compress time. If we could, we'd have different issues. We can't compress time and fit it into a 50 minute class, which is why it's out of class, but you get a free day for it, all right?

Last but not least, few things about grading. Grading, one of those things as you might be able to tell from this little board over here or something, if I didn't have to do it, I wouldn't do it because honestly, as corny as this sounds, I just believe in the love of learning. Like, I think if you're passionate about something, you just go do it and you learn it. But I'm naïve, and so that's not the way learning always works. So sometimes we actually need grading to make sure that learning takes place.

And so this is how your grade breaks down: Forty-five percent of your grade is on the programming assignments, okay? Fifteen percent is the midterm, which we'll just call the mid because we like to abbreviate everything. Thirty percent is the final. It's a three-hour final exam in the regular final time slot for this class.

If you think or are under the delusion that you should take two classes at the same time, that's a bad idea because their final exams are at the same time, okay? So you should not take two classes at the same time because our final exam is scheduled for — I believe it's December 13, which is a Thursday, 12:15 to 3:15. That's the regular final exam slot for this class. And any other class at the same time will conflict with that slot. Thirty percent of your grade is the final.

And that, if you add it all up, it's not just that I'm bad with math. It's because 10 percent of your grade is actually participation. And this is things like did you go to your interactive grading sessions? Did you regularly attend section? Did you participate in section? Did you participate in class, right?

And so, in order to help you participate in class, there's a little incentive to participate in class, which is sugar in the afternoon. So someone raise their hand. All right. Yeah, sometimes I'm not a good shot. And this will tell you, if you're sitting in the back of the room, I can't throw a Kit Kat back there because they're a little too light. Oh, yeah, sorry. If you sit in the back of the room, the roof prevents me from actually being able to hit you. So if you want the food, come up. But if you ask questions in class, hey, that's a good time. It's just a little way to be able to reward you for actually participating in class or to keep your blood sugar up if you need it, all right?

So that's participation. It's 10 percent of your grade, and as a matter of fact, at the end of the quarter, I ask every one of your section leaders to actually tell me how much you participated in class, and some of them just say, "Oh, this person was wonderful. They came every time. They participated. It's just a great thing." And that helps your grade out a lot, okay?

Now, the final thing, and as you can kind of tell, most of the time, I'm not the most serious person in the world. I just like to have fun with things, and I think it's important for you to have fun with things. There is just one place where I get real serious, and it's one place where Stanford gets real serious. Anyone wanna guess what that is? Plagiarism and the honor code. As a matter of fact, that's what we call a social. So we had someone down here who got it and then a whole bunch of people who I don't know, so we just spray. All right.

So the honor code, in terms of the honor code, the question comes up is what is the honor code all about, and how does that affect working in groups and computer science, etc.? Does that mean we shouldn't talk to each other? No. The answer to all those is no, okay? If you look at Handout No. 4, which is all about the honor code, we encourage you to talk to each other. We encourage you to talk about concepts in the class, talk about different strategies to problems, to think about the ways that you could potentially approach some problem or the way different control constructs when we eventually get to them work in the class. And discussion is perfectly fine, especially among the course staff, but also amongst yourselves. That's a great thing.

So where do we draw the line? And we try to make a bright line for where you've crossed the line for the honor code, which is don't share code, plain and simple, in any respect, okay? Don't give a file to someone else that's got your code in it. Don't get code from someone else. Don't look at someone else's printout. Don't give them a printout.

If you have two people who are sitting looking at the same screen together, that code can't belong to both of you. It belongs to one of you. I don't know which one, but it becomes an honor code violation. So you shouldn't both — two people shouldn't be staring at the monitor together. If it ever gets to the point where you're looking at someone else's code, that's where you're gonna reach an issue, okay? Discuss as much as you want. That's great. Write your own code. That's all we care about.

And you're, like, "Well, what is code, Mehran? What does that word mean?" Code is geek speak for your program, so when you program, the program that you write is what we affectionately refer to as code. And the idea of programming is what we refer to as coding, strangely enough. Computer scientists need to make everything more complicated than it really is so we can get people under the illusion that they should pay us lots of money to do what we do. I mean, you're, like, "Oh, I just write programs." And they're, like, "Oh, yeah, I should pay you half." And you're, like, "No, no, no. I write code." And they're, like, "Oh, yeah." Suddenly, it's much more impressive. So don't share code.

The other thing is if you talk to other people, like if you have a study group to talk about solution approaches or you go, let's say, talk to the TA or your section leader to how you should approach a problem, and they give you a lot of hints as to how to do it, cite collaboration. So cite and collaboration gets you out of trouble. Any collaboration that you cite you cannot be held responsible for under the honor code.

You can actually copy someone else's program and say, "I copied this program from Mary Smith." And I'll look at that and say, "They cited it," and it will warm the cockles of my heart. And Mary Smith will get full credit, and you'll get a zero because you copied your program from Mary Smith, but it's not an honor code violation because you cited the work, okay? So the bottom line is keep yourself safe and cite your collaborations. And I guarantee you most of the time, you'll be just fine.

Now, you might wonder why do I make such a big deal about this. And the reason I make a big deal about this is for a while, thankfully it's not true anymore, but for a while, the computer science department actually had more honor code violations than the rest of the university combined. Take everything else in the university, put them all together, they were like over here. And we're, like, we're computer science, which is not a fun distinction to have, let me tell you.

And you might wonder why is that? Is that because computer science people are just mischievous and dishonest? No. It's because it's easier to catch honor code violations in computer science. We have a whole bunch of tools that allow us — then we take all your programs and we run them through this tool, and it compares them not only to everyone else in here, but, like, to everyone from the last, like, X years where X is the large number of people who've ever gone through the classes, right? And it's an extremely good tool from finding where honor code violations happen, from where they don't. And it doesn't find spurious violations.

To be honest, I've never lost an honor code case. When I find an honor code case, it is blatant. And you take it to judicial affairs, and they look at it, and they're, like, yeah, this is blatant. And I take it to the student, and every student I've ever confronted them with never said, "No, no, no. I didn't cheat." They said, "You caught me," okay? So it's blatant.

It's not like, oh, there's some little line in it, "Oh, am I gonna need to worry about an honor code violation?" Remember those rules, you have nothing to worry about in this class. It's people who go and, like, fish out printouts from the recycle bins and copy other people's code that are the people we catch, right? It's blatant cheating that we catch. But we catch it. We catch it all the time. So I hope, I pray it doesn't happen in this class.

But the reason I make a big deal about it is historically if I look at the evidence, it happens and we catch it. And when we catch it, we're required by the university to prosecute. And I feel bad because usually it's someone who just made a bad call, like, they were up way too late the night before working on something else, and they're not thinking straight. And rather than just taking a late day or turning in their assignment late and getting a slight penalty on it beyond their two free late days, they decide to cheat. And that's just always the wrong call, okay? So you just don't wanna put yourself in that situation. So I get real serious about it for a moment, and hopefully it won't be an issue and we can just kind of go on, okay?

So with that said, that's a whole bunch of logistical stuff. Any questions about the logistics of this class or anything I just talked about? Uh huh?

**Student:** You had briefly mentioned the late penalty.

**Instructor (Mehran Sahami):** Oh, the late penalty, good point. So remember our little bucket scale. If you go beyond your two free late days, every day you turn in an assignment late beyond those, it drops down one bucket. So let's say you already used your two free late days on Assignment No. 1. And on Assignment No. 2, you turned in something one day late and you would have gotten a check normally, it becomes a check minus. So that's how it is. It's one bucket per late day beyond your two free ones. Uh huh?

**Student:** Are the sections first come, first served?

**Instructor (Mehran Sahami):** Yeah, the sign-ups, well, they take into consideration your preference, but part of your preference is to do the match is first come, first served. So you wanna sign up early. Oh, thanks for your honesty. As a matter of fact, I dig honesty, all right? Any other questions? It's just honesty's cool. Uh huh?

**Student:** How much time should we plan on studying [inaudible]?

**Instructor (Mehran Sahami):** Oh, good question. How much time should you plan? And this is something that I say for classes in general at Stanford, which is not always true, which is take the number of units that a class is, multiply it by three. That's how many hours you'll spend per week in that class, total, on average. So what that means is in 106A, a 5 unit class, you multiply by 3, you get 15. Five of those hours are roughly spent between class, section, interactive grading, other stuff. That means on average about ten hours a week will be spent on your assignments in this class. Again, that's an average.

Sometimes when I go to computer science conferences, I sit there and joke around with plans. And we're, like, "Oh, how long did your assignments take?" And I say, "Oh, on average, ten hours." And what I really mean when I say on average 10 hours is they take between 3 and 45, okay? It's a large variance event, right? Ten is the average. Some people take a really long time. Some people get through it really quickly, but that's about the average you can plan for. Uh huh? Another question?

**Student:** [Inaudible] late days [inaudible] class days?

**Instructor (Mehran Sahami):** Yeah, all late days are class days, so the free ones — the halfway mark's really my reach. That's about it. All right.

So I do wanna give you your very beginning of an introduction to programming before we sort of break for the day. How are we doing on time? And so in order to kind of see this, there's a few things that we wanna keep in mind.

Actually, let me show you a little picture, okay? Sometimes when we talk about writing programs, we talk about debugging programs, right? How many people ever heard the term debugging or bugs in programs? A bug in a program is an error in a program, so

sometimes when you hear us say, "Oh, come see," like, your section leader to help debug or see the helpers in LaIR.

That's another thing. In the Tresidder computer cluster is the LaIR. It's a computer cluster that we have helpers there to help you get through this class. What is it? Sunday through Thursday, every week, from around 2:00 in the afternoon 'til midnight every day, okay, to help you get through the class. So that's a good place if, you know, you can work in your dorm room certainly, but if you also want help, go to the Tresidder computer cluster, and there will be helpers there. There's a little queue you sign up for to get help, and that's a great place, and it's all explained in Handout No. 1, but that's just something to keep in mind.

Where the term debugging comes from, it turns out this is an apocryphal story, but I'll tell you anyway. Back in the days of yore, in 1945 actually, there was a computer called the Mark II at Harvard. And there was a woman named Grace Murray Hopper. Anyone ever heard of Grace Murray Hopper? A few folks. She was actually the first woman who was an admiral in the navy. And she was also one of the very early pioneers of computer programming. She did a lot of computer programming when she was actually a captain, and she was stationed at Harvard as part of some sort of navy thing. I don't know why, but that's what happened.

And they had this huge computer there, and they were noticing the computer was on the fritz, and they couldn't understand what was wrong. And this is one of these big old machines in the days of yore that has vacuum tubes and stuff inside it. So they walked inside the computer, right, because then you could actually open it up and walk inside your computer.

And they saw this, and I don't know if you can see that, but that's a moth. It was a moth that had sort of given its life to be immortalized because it had actually shorted out across two relays in the computer and was causing these sort of errors to happen on the fritz. And so they took the bug out, and once they actually plucked this little charred bug out of there, the computer started working fine again, and she taped it in her log book.

And this log book's actually preserved in the Smithsonian Institution now, which is where all this comes from. Here's all the standard disclaimer information: "Image used under fair use for education purposes. Use of this image is exempt from Creative Commons and other licenses," just so you know. Now the lawyers are happy. But this is where we think of sort of the modern term debugging actually came from.

Now, it turns out the actual story is that the term debugging came from the 1800s, in the late 1800s from mechanical devices. People actually referred to debugging as fixing mechanical devices. But this is kind of the apocryphal story for how it comes up in computer science.

Now, with that said, what is the platform in which you're gonna sort of do your first debugging or your first work on? We talked about Java, but in fact in this class, we're not

gonna start with Java. We're gonna start with something even sort of simpler than Java because as I mentioned, sometimes what happens in computer science is people learn all the features of some language. And they think just knowing the language makes them a good software engineer. And they get so worried about all the features of the language that they don't kind of think about the big picture.

And so there was a guy named Rich Pattis, who oddly enough was actually a grad student at the time at Stanford, and he said, "You know what? If we're gonna teach computer science, when we first start out, why don't we have people not worry about all of the different commands of the language and all the different things they can do? Let's start with something really simple so you can learn all the commands real quick. And then you've mastered everything there is to master about that language, and you can focus on the software engineering concepts." And it turns out to be a brilliant idea, which has actually been adopted by a bunch of people.

And so Rich, who's a wonderfully friendly guy — sometime if we get him to come to Stanford, I'll introduce you; he's just very nice — came up with this thing called Karel the Robot. And the term, "Karel" actually comes from Karel Capek. Anyone know who he is? Oh, free candy. Uh huh?

**Student:**He coined the term, "robot."

**Instructor (Mehran Sahami):**He coined the term, "robot." He was a Czech playwright who actually wrote a play called, "RUR," which was about robots. And the word robot actually comes from a Czech word, the Czech word for work. And so the robot is named after Karel. And some people say Karl, which is kind of actually closer to I believe if — I don't know if there's anyone who speaks Czech in the room — but closer to the actual pronunciation. But we say Karel these days because it's kind of like gender neutral, okay?

And so Karel the Robot is basically this robot that lives in a really simple world. And so I'll show you all that you can meet Karel the Robot. He's friendly; he's fun. I'll show you Karel the Robot. So we gotta get Karel running. He's at the factory. He's getting souped up. We're energizing Karel. You gotta add some color to it. Otherwise — all right. We're begging for him. Come on, Karel. There he is. Oh, yeah. That's Karel the Robot. He looks like one of the old Macintoshes if you remember the original Macintoshes that look like a lunch pail, except he's got legs. One sticks out his back. That's just the way it is.

And the way Karel works is he lives in a grid. To you, it may not be exciting, but to Karel, it's way exciting. So Karel lives in this little grid, and the way the grid works is there are streets and avenues in the grid. Streets run horizontally, so this is First Street, Second Street, Third Street. And then over here, we have avenues, First Avenue, Second Avenue, Third Avenue, Fourth Avenue, Fifth Avenue. It's kind of like Karel lives in Manhattan if you wanna think about it that way, okay? So Karel always is on one of these corners. So right now, he's at the corner of First Street and First Avenue, or we just refer to it as 1 1 if you wanna think about sort of Cartesian coordinates, right? But just think of them as streets and avenues. That's where Karel lives.

And Karel can move around in this world. There's a bunch of things that Karel can do. He can take steps forward. He can turn around to face different directions, and he can sense certain things about his world. So there's some things that exist in Karel's world, okay? Things like walls that Karel cannot move through, right, so his world has walls all around it that he can't go through, so he can't fall off the end of the world. And there's other walls like this one if Karel were over here, he can't step through that wall.

There's also something referred to as beepers in Karel's world. And what a beeper is, is it's like a big diamond, okay? But what a beeper really is, is basically just some marker that he puts in the world. You can think of a beeper like a piece of candy. And Karel just goes around, like, putting pieces of candy in the world. As a matter of fact, not only does he put pieces of candy in the world, he carries around a whole bag of candy.

So he has a beeper bag with him, and sometimes that bag has a whole bunch of beepers in it; sometimes it only has one beeper; sometimes, it's sad Karel, and he has no beepers. But he's still got the bag. There just don't happen to be any beepers in it. So he can potentially, if he come across a beeper in his world, he can pick it up and put it in his bag, or he can take, if he's got beepers in his bag, he can take them out of his bag and put them places in the world. And corners in the world can have either zero — if they have no beepers, they just appear like a little dot — or one or more beepers on them that Karel can potentially pick up, okay?

So any questions about beepers or Karel having a little bag of beepers? And that's it. That's Karel. That's his world. His world, we can make it larger if we want. We can put in walls in different places. We can put beepers in different places. We can have Karel be in a different place.

But starting next time, what you're gonna realize is with this extremely simple world, there's actually some complicated things you can do. And after about a week — so this first week, we're gonna focus on Karel — you'll notice that Karel is actually a very nice, gentle introduction into Java. And a lot of the concepts that we learn, sort of software engineering concepts using Karel, will translate over to the Java world, okay? So any questions about Karel or any of the other logistics that you've actually heard about in the class?

Alrighty then. Welcome to 106A. I'll see you on Wednesday.

[End of Audio]

Duration: 50 minutes

## Programming Methodology-Lecture02

**Instructor (Mehran Sahami):** Alrighty, welcome back to CS106A. If you're stuck in the back, just come on down, have a seat. Originally, I thought maybe we would have slightly fewer people today than last time, but that appears not to be the case. So while we're waiting, everyone loves babies, so I decided to put – that's Karel, the Robot, the early days.

No, this actually – my son, and yeah, I know. He's a little bit older now but, like, he's got these little robot pajamas and he runs around all the time. So I'm like, oh, it's Karel, The Robot, and my wife looks at me like, it's your son. But that's a whole different issue.

Anyway, a few administrative announcements before we start. A couple things, there are four more handouts today, just because we like consistency. Four last time, there's four this time. They're all in the back. If you didn't already pick them up, you can pick them up after class, but they have all the information about downloading Eclipse, which is the environment that you're going to use for programming in this class, both for Karel and for Java.

There is also a special handout in there just on using Karel, so it talks about how Karel works in the Eclipse environment, and so you can get all set up with that. And so after today, you'll know how Karel works and you'll have the environment to use it. So surprisingly enough, you also get your first assignment today, which is actually in two parts.

So the assignment, the real assignment is the programming part, which is due on Friday of next week, October 5th. There is also an email part to it, where the email part is, we ask all of you kind folks to send an email to Ben, the head TA, myself, and also your section leader. So Ben and I get a whole bunch of mail, your section leader will hopefully get a little bit less mail.

But you won't actually know your section leader until after you go to your first section next week, which is why that part of the assignment is not actually due until basically one minute before midnight on Sunday. So I didn't make it midnight, just because then there's always confusion, midnight, which night? So 11:59 p.m. on Sunday, October 7th is when you should send that email.

That's not really the critical part of the assignment. The critical part of the assignment is the programming problems, which are actually all due in class on Friday. Okay, and then there's one last assignment, or one last handout, which is about how you actually submit your work in this class. You'll submit your work both electronically and in hard copy. The electronic copy, so that your section leader can run it and verify it. The hard copy is so that they can actually write comments on it, and then you'll do interactive grading with them as well.

A couple other quick announcements. The website, in case you weren't here on Monday and you don't know what handouts I'm talking about, and you don't know where to get them, go to the class website, cs106A.stanford.edu. There will be electronic copies in PDF format of all the handouts there, so you can get caught up if today is your first day.

You also need to sign up for a section. As we mentioned last time, section signups start tomorrow, 5:00 p.m. Sign up early if you want to have the most flexibility in terms of time, because look around, there's a whole bunch of people in this class. It's going to fill up quickly. So if you have constraints, sign up quickly, and the place you sign up is CS 198, not CS 106A, but CS198.stanford.edu/section. The 198 folks are the folks who lovingly run all of the sections and coordinate the whole program.

Last, but not least, just a word on the readings that I didn't mention last time. On the syllabus, you'll notice that pretty much for every day, or most days in the quarter, there is some sort of reading associated with it. That's the reading assignment that you should have done by that day's lecture, because that's what we will cover in that day's lecture.

So for today it should be the first three chapters of the Karel book, or the course reader. And if you're like, oh my God, I'm already three chapters behind, don't worry, it's like 20 pages or something. It's pretty lightweight, but you should do the readings by that time in class. So any questions about anything we covered last time, which is mostly logistics, or any of the sort of administrivia?

All righty, then let's get started on the real content. So you remember from the time before, we talked a little bit about Karel, the Robot, and here is Karel in that world that I showed you before. And there were avenues that run north-south, and streets that run east-west, and little beepers in the world. And Karen can face different directions, and there's walls.

And so now, we want to think about how do we actually program Karel. How do we get this little guy to do something interesting in the world? Okay, and it turns out there are four commands that Karel understands, okay, and those are pretty straightforward.

So here they are. You're going to get all of Karel's vocabulary in, like, one minute. There's a command called move, and move basically moves Karel one spot forward in the direction he's facing. So if he's on corner one, one, and he moves one spot forward, he's facing east. So he'll move over to two, one, basically. So one corner forward in the direction he's facing.

Karel also started being a good democrat, knows how to turn left. So he turns left. This is a lower case T, this is an upper case L. So turning left changes his direction by 90 degrees in the left hand of whichever way he's facing. So if he's facing east now, and he turns left, he will be facing north. And then he can turn left again and he'll be facing west.

Okay, now, question?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):** There is no space here. These are all one word. Good question, and I love it when you make it easy. All right, so besides turning left, there's also B-person Karel's world, and if he couldn't do anything with beepers, they wouldn't be interesting. So he can pick up beepers, which interestingly enough is called pickBeeper, again lower case P, upper case B, all one word, and there is putBeeper.

And what these do is pickBeeper, if Karel happens to be on a corner that has a Beeper on it, he picks up that Beeper, and stuffs it in his Beeper bag, in which case the count in his Beeper bag goes up by one. Or if it's infinite, it remains infinite, because he's got a real big Beeper bag, and sometimes he can infinitely many Beepers in there. It's just – we can talk about the infinity of Karel's Beeper bags some other time, but if you're interested, it's fascinating.

And he can also putBeeper, which means he can go to a corner and be like, "Hey, I'm just feeling happy. I'm going to put a Beeper down, say, on that corner right there." Except you can't throw the beepers, right. Karel's kind of limited. He's got, you know, if you look at him he's got no arms. He's got legs. No arms, so basically all he can do is he sort of like drops the little Beeper where he's at. So this puts it on the corner that he's at.

And these things are what we refer to as methods. Okay, methods are basically some instruction that we can call, okay, or use, like move, or turn left, or pickBeeper, or putBeeper. And what we say is Karel responds to this method. What we're doing is we're invoking, or we're calling a particular method on Karel, and he takes some action, which is basically what that method specifies to do.

Okay, so if we kind of think about this program, or if we kind of think about this world, maybe we want Karel to do something. So here's the initial configuration of the world. Maybe what we want Karel to do is pick up this beeper, drop it off at this corner, and end up at this corner facing to the east. And you can think about how we might do that with some of the instructions we have.

So what we want the final configuration of the world to look like, and I'll just put Karel at blinding speed. You can actually control Karel's speed here with this little slider. So I'm going to just show you the end configuration by running Karel so fast you can't even see it. It's just – he's blindingly fast. He's just that good, okay.

And so that's what we want to get to. Notice he's still facing east, but he's picked up that beeper that was on that corner at two, one, and moved it over to the corner at four, two. Okay, and so how might we do this? Right, so one thing we could consider is, what was the initial state of the world again?

So I'm just going to run this program again, and here I have a clip set up, which is your handout number five explains how to get this. But if I want to run a program, there is

these two little running person icons, or as you might notice, we even have our own menu in Eclipse, because Stanford's just that much fun, okay.

So under the Stanford menu, we have these two options for run. Import project is explained in the handout, is what you'll use. We'll give you some initial stuff for Karel that you'll start with, and you'll use import project to get it into Eclipse. But once it's there, we can run it, and so what we're going to do is say run this particular Karel program. So it's get – and we'll go through all this – it's get and go.

And this is our first Karel program that we want to run, and it's sort of, here's the initial world again, all big and happy. So I'm going to shrink this down a little bit, so we can see it over here on the side, while we think about what commands Karel would need to execute to kind of get to that final state we just talked about.

And over here, happily enough, let me resize a little bit, is our first file, which is empty right now, that we're going to write our first Karel program. So what commands might we consider Karel actually doing from the list you have here, to sort of affect what we want to happen in the world?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Yeah, so we want to move. Okay, and then what do we want to do?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**PickBeeper, then move. Notice at that point we could have actually done something else. This is part of the art of programming. There's actually many ways to solve the problem. We're just going to happen to pick one. So we've moved. We've picked the beeper. We've moved again, and now what do we do?

**Student:**Go to the left.

**Instructor (Mehran Sahami):**Turn left. What happens if we turn left?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Right, then what are we going to do?

**Student:**Move.

**Instructor (Mehran Sahami):**Move. And now what do we want?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** Yeah, conceptually what we'd like to do is turn right, right? At this point, Karel's like, "No, man. I'm just, you know, I'm left wing. What can I do? I turn left. That's what I do." And you kind of think about it, and you say, "That's all right, because really the world is just one big spectrum, and if you go around far enough to one side, you end up on the other." So if we make you turn left three times, that's equivalent to essentially turning 90 degrees to the right.

So here, we turn left, turn left, and turn left. All right, now what do we do?

**Student:** Move.

**Instructor (Mehran Sahami):** Move. So we sort of go up the step. PutBeeper and move to the last spot. Now, at this point we'd like to think, oh good times, we can just run this and it's a Karel program, and life is happy. In fact, this is not a valid Karel program. What this is, is an algorithm.

This is a recipe for doing something, and we'll talk in a lot more detail about different algorithms on Friday. But the way you can think of distinguishing between an algorithm and a program is an algorithm is essentially the recipe for doing something. The program is something that is valid syntactically according to the rules of the language.

And so Karel actually has some specific rules to its language that we have to apply to these statements to make them look a little bit different, so they follow the valid syntax of Karel. And that's what we're going to do now. We're just going to kind of go through this step by step, okay.

So one of the things we'd like to be able to do is, first of all, we want these to be valid commands. Right now, they're not valid commands. To turn a command into a valid command, or what we would refer to as a method call in Karel, after the name of the command we put an open paren and a close paren, and then a semicolon.

Okay, so move, open paren, close paren, semicolon is actually the valid move command, or move method invocation for Karel, and we do that for all these things. So to save a little bit of time, I'm just going to copy this and just go through and paste until the cows come home. Let's do a little paste there. Oh, it didn't copy. All right, let's try that again. We do a little paste, a little paste.

So we're going to go and turn all of these into valid Karel commands. Okay, so now you might think, okay they're all valid Karel commands. Are we ready to actually run this Karel program? And it turns out no, we're not yet ready to run – this is not yet a valid Karel program.

What makes it a valid Karel program is we need a few more things. One of those things is, we need to tell Karel where to start running. And you look at this and you're like, what does that mean? Like, doesn't he just start at the top? Does Karel start somewhere in the middle?

No, what it really means is that we're going to do is encapsulate a set of instructions inside something that tells Karel, hey, this is what you're going to execute. And the way we do that is we create something called a run method, and the syntax for that looks a little bit archaic, but basically it says, public, void, run, open paren, close paren, and then we put a brace. And that first brace says everything that you will now see until the closing brace is all part of what we refer to as the body for this thing called run.

So we come down here to the bottom and we put a closing brace, and everything between the open brace and the close brace is referred to the body, or what's actually executed when we do a run. Okay, so get to know where the brace keys on your keyboard are. They will come in very hand in programming, because you'll use them all the time.

One thing we also do with this is we're going to actually insert some tabs to make it easier for the human to read this program. Okay, so when we put in tabs, suddenly everything that's inside the body of run gets tabbed in, and when I look at it visually, it's much more appealing. I can just tell all the things that are actually inside run.

Okay, so we tab those in there and now we have the body of run is inside the braces. What we've actually done is created a new method. Well, run is here, is we defined for Karel a new method called run. And run is a very special method because that's where Karel begins running.

What he says is when he starts up, he comes into the world, he's like a brand new baby. He's like, "Hey, I'm here. I'm in the world. What do I do?" And he's preprogrammed to go find the method called run and start executing from the beginning of it. So he does executed in a top down manner, but he needs to have this run thing.

And so you might see them say, "All right, we're ready to go. Fire up Karel, right. I'm slipping a shrimp on the barbie." I'm sitting here barbequing while Karel's going and moving the beeper. It's like, no, we're not quite there yet. You're like, oh, what else do we need?

Okay, so one thing that we need is we need to take Karel. Karel is actually implemented in Java. If you've done some Java programming before, some of the syntax already looks familiar to you. If you haven't done any Java before, it's perfectly fine. You don't need to know anything about Java. What you do need to know about Karel is that Karel is defined in what we refer to as a class.

Karel is sort of a class of robot. There's kind of like the Karel 1000, kind of comes off the assembly line, and what we'd like to do is take those Karel 1000s and sort of mold them into our own version, which does some set of instructions that we'd like it to do. So in order to sort of say, "I want to create my own version of Karel," what we do is we create a class, and again we say public, and we say class now, instead of void.

So far, all you need to worry about, the stuff that's in purple, public void, and public class, is just kind of the standard syntax of Karel. And later on, all of this stuff will make

sense what it actually means, but right now just think of it as standard syntax, and just put it there, and life is good.

Public class, we now need to give this Karel program, in some sense, a name, and what we're going to name it, for lack of any better name, is we could call it our Karel program, because we did it together. It's kind of a collective process. So our Karel program, and at this point we're not done.

We need to actually say – you might have noticed after I type this why suddenly all this red showed up on the side of the editor. And it's like, oh, bad times, dogs and cats sleeping together, like what's going to happen? Well, what happens is we're creating our Karel program and it says, "Okay, you're creating Karel program," but it doesn't even know anything about the basic Karel 1000.

It needs to somehow know that this Karel program is creating a version of the Karel 1000. We'll call it the Karel 1001, and the way we specify this is we say that our Karel program extends Karel, which is sort of the basic Karel that actually exists and has these predefined instructions in it.

So we say public Karel, our Karel program extends Karel, and now we need to, again, encompass what our Karel, our particular class Karel is going to be running. So we put this inside braces and we say, "Yeah, this whole run thing is part of our Karel program." So we put another brace at the end, and to make it more readable, we go through one more time. Don't want to put a tab there, and tab everything so that it's nicely readable by a person.

Okay, so all this stuff now gets tabbed over one more spot, and even this brace over here gets tabbed one more spot. So now you can see, everything here is part of the run method, because it's nicely tabbed in. We can tell that, and all this stuff here is all part of the body of run, and all of it's encapsulated inside our Karel program.

**Student:**[Inaudible]

**Instructor (Mehran Sahami):**Why is there a space, you mean like down here?]

**Student:**Yeah.

**Instructor (Mehran Sahami):**Just for readability. We don't actually need it. There doesn't need to be a space, but that's a good question. Oh, nice bank. I like that. Feel free to bank. If someone – you're on the rebound and I'll set you up for the rebound, just because you were the first one to do it. All right, so, and you might say, "Okay, now we've got our Karel program. We've got our run method. Are we ready for Karel?"

Okay, and you think we're ready for Karel, and someone's calling, saying, "You're ready for Karel. Do it." What I would actually encourage you to do is take a moment and turn

off your cell phones, unless your doctor. I actually would tell this to my students all the time.

I had one student who was a doctor once, and he says, "Actually, your class is when I'm on call, and if my phone rings, that means, like, someone's dying. So can I keep it on?" And I was like, hmm, yeah, okay. So he got an exception, but other than that, I'd encourage you to turn your cell phones, unless you have a really cool ring. Then you can leave it on.

All right, so at this point, you might say, "Oh, we're ready to run it," but even now, we're not ready to run it. And the reason why we're not ready to run it is, where does this Karel thing come from to begin with, right? Someone, somewhere had to say, "Hey, I'm going to build the original version of Karel that you're now extending."

And that actually comes from a place called Stanford.Karel. So what we do is we add a line called import, Stanford.Karel.\*, and we put a little semicolon at the end of it. And what this tells us is, hey, go and get everything that Stanford's previously defined about Karel.

And when we get to Java, this'll become much more clear, but for right now, all you can think of that import's meaning is, go get me all the standard Karel stuff that Stanford's already done for me. And that defines this thing called Karel, which now we're extending in our Karel program. Uh huh?

**Student:** What is the semicolon for?

**Instructor (Mehran Sahami):** The semicolons always come at the end of the line and Karel's syntax, what they mean is this is the end of one statement. So at the end of the import, for example, that says import the stuff, that's the end of a statement, or for move, or pickBeeper, or turn left, that's the end of a statement. So we put a semicolon. Uh huh?

**Student:** Do the stars mean anything?

**Instructor (Mehran Sahami):** Do stars mean anything? Yeah, that star at the end of Stanford.Karel.\* is actually important, and in a couple weeks you'll understand why. For right now, it just means get everything associated with Stanford.Karel. Uh huh?

**Student:** So then, is the run program [inaudible]?

**Instructor (Mehran Sahami):** It's all part of run. So when Karel starts up, and you'll see that in just a second, we start – we go to run and we just start executing the statements from run. So let's actually run this and see what happens, and I'll take a couple more questions in a second.

So we go up to our little run icon here. Actually, before we do that, let me cancel this, because we already have a running program over here. So let's quit out of the running

program and come here, and say run. And sometimes if you actually happen to write multiple Karel programs, and you hit the run, then this guy sort of looks like he's running with smoke all around him, which means he's running really fast. That means run the last program I was running.

The guy without smoke around him means, run one of the programs that I tell you to run, and it might give you a list if you have multiple programs. Here we're going to pick first Karel program, okay. So yes, we'll go ahead and save the resource because we forgot to save our file, and here is life.

And we'll run it slowly, and so we start the program, and there goes Karel. Woo-hoo! Congratulations. You are now all computer programs. Okay, because you got Karel to do what you wanted to do, and you're like, "Oh, that's so good. Let's do it again." And so you're like, "Yeah, let me start the program again." Wah-wah-wah, thanks for playing.

What happened? It tried to execute the same program again, but the world was in a different state, right? Karel was facing that wall, and the first thing we said is, "Hey, Karel, move." And he's like, "Okay." He's a little happy, scrappy robot. He doesn't know anything about the world, and he just, wham, into the wall. Thanks for playing. Dead Karel.

All right, no, because Karel's blocked, okay, and you get this little bug on top of Karel. It's like Alfred Hitchcock, the birds, he's just being, like, harangued by this little bug. But what that means is Karel tried to do something in the world that the world would not allow him to do, and this is what happens if you try to walk through a wall, or try to pick up a beeper from a corner where it doesn't exist, or you try to put down a beeper and your beeper bag happens to be empty, that's the kind of error you'll get.

So if you want to start this program again, you need to load world – and we happen to have first Karel program, you need to reload your world and then you can start it again from that point. Okay, so any questions about our first Karel program? Uh huh?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):** Yeah, he's moving basically from one grid point to the next grid point. So he just moves one step at a time. Uh huh?

**Student:** Is Karel case sensitive?

**Instructor (Mehran Sahami):** Karel is case sensitive, yes. Everything you type will be case sensitive. My throwing is case sensitive as well, and evidently, that was the wrong case. All right, so something else we'd like to be able to do with Karel, and a few people already pointed out is, "Hey, we created this thing called run. That's kind of cool. Can I create other stuff?" And as a matter of fact, you can, because that's just how snazzy Karel is.

And one thing, you might look at this and be like, "Ooh, ooh, ooh, I know what I want to create." What would you want to create if you could create a new command?

**Student:** Turn right.

**Instructor (Mehran Sahami):** Turn right. Right? That's a social, right? Everyone wants to turn right. All right, except the person who actually said, "Turn right." So what we can do is we can have a similar syntax that allows us to create new methods in Karel, like the run method actually is the place where Karel will always start. But we can create new methods that we could use, and so maybe we want to create a turn right.

So we could say void, sorry, here we'd say private void. Voice – sometimes typing is bad, especially when your hands are a little chalky. So run – you'll notice run is always public void run. For all the other commands that we're going to do as far as Karel's concerned, we're going to use private void, and the name of the command.

So we'll call this turn right, and this command will have a body, and what is the body of the turn right command going to be?

**Student:** Turn left.

**Instructor (Mehran Sahami):** Turn left three times. So I'm just going to copy from here and paste into – and the indenting even works out right. Rock on, right? And what I've now done – I'll go ahead and save this – is I've created a new command for Karel called turn right. So anywhere else in my program, I can now say, "Turn right," like here are these three turn lefts. I can replace them. Let me just get rid of them, and I'll replace them with a turn right, and I'll appropriately indent that.

And what that means is, as you're executing the program, you move, you pickBeeper, you move, you turn left, you move. When you get to turn right, it says, "Let me pause where I saw that turn right. I will go to the turn right method body and run all the instructions that are in that body, and then I will return back to where I left off."

So this program is exactly equivalent to the program we wrote before, now in terms of something called turn right, and just to prove that to you, we'll go ahead and run, and there it is, Karel does the equivalent of turn right and life is good. Okay, so we've created a new command called turn right. Question?

**Student:** [Inaudible]?

**Instructor (Mehran Sahami):** The location's not important. You can put it anywhere. I like to put it at the end, because that's convention in the book, but you can put it anywhere. Uh huh?

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** Why is run always public? Because that's where things start, and for Karel to know where it is, like private's all the stuff you're sort of like, eh, I'm keeping it to myself. But Karel's like – he's, like, looking around. He's like, "Where do I run? What do I run? And so you got to say, "Oh, it's public. It's available to the whole world." And in a couple weeks when we get into Java, you'll see public and private rear their ugly heads again and it'll all be clear.

Uh huh?

**Student:** [Inaudible]?

**Instructor (Mehran Sahami):** Pardon?

**Student:** If you made that a public [inaudible]?

**Instructor (Mehran Sahami):** It would still run. It would be fine. It's just good programming style. See, there's all these things that we kind of will eventually get to, but we got to start somewhere. So we sort of start right in the middle and we're going to build up from there, and eventually we'll go back and fill in some of the pieces underneath.

Uh huh?

**Student:** Do you have to [inaudible]?

**Instructor (Mehran Sahami):** Run is where – has to be called run. That's where Karel always knows where to start, but everything else could be something else. Like, instead of turn right, I don't know, what's the French word for turn right, or the French two words for turn right? Anyone speak French?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Exactly. We could have called it that if we wanted to, and then rather than turn right we would use that, and it would be fine. So all the other things besides run, we can name them whatever we want as long as we're consistent throughout.

All right, so we could also – you might say something else might be kind of fun to do is to have a command called turn around, for example, which turns Karel around 90 degrees. So we'll just do a little copy and paste magic and have turn around here. We might not actually use it in this program, but just to have, and turn around is actually just two turn lefts.

It turns Karel around, and if you really wanted to do it, you could do it as two turn rights, but that's kind of six turn lefts, and that's not really a good idea. Okay, so because turn right and turn around are such common things to do, we actually have a souped up version of Karel. It's sort of like Karel with the training wheels taken off, and he's bad,

and he's buff, and he knows everything Karel knows, all four methods, plus two more, which is turn right and turn around built in. And that's super Karel.

So if we get rid of these and we say, "Hey, our Karel program, instead of extending Karel, is going to extend the super Karel model." Okay, oh, I know, it's dangerous, but it's okay. Super Karel, we've gotten rid of the turn right definition and the turn around definition, but if we run this now, start the program, Karel actually will do the right thing.

All right, let me quit out of this, unless my computer decides to freeze, and sometimes it decides to freeze, and that would be bad. All right, so at this point you're Karel programmers. You're rocking. You know all the Karel's methods. That's all he knows, at least for the time being, and you know how to actually make new commands by creating new definitions that have a body and the syntax around them to create new commands.

All right, so now what we're going to do is I want to ask you a philosophical question. I know, I guess you're like, this is a computer science class, why do we have to do philosophy in here? Because it's good for you, and the philosophical question is what is the downfall of the modern college student? Anyone know? You're like, what does that mean? You're like, I haven't fallen down yet. That's good. Hopefully it won't be that way.

But there is something, probably in your everyday life, that's just sitting there niggling. Uh huh?

**Student:**Procrastination.

**Instructor (Mehran Sahami):**Procrastination? It's related to that. What do you do to procrastinate in the morning?

**Student:**Sleep.

**Instructor (Mehran Sahami):**Sleep. Oh, sorry, yeah, and how do you prolong that?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Snooze. And the thing about snooze is, it's amazing how quickly you can do math in the morning. I don't know about you, but my alarm has a nine-minute snooze on it. Why it's nine, I don't know, but it came out of the factory with a nine-minute snooze. And so you're, like, lying there.

I remember when I was a student, like, the dreaded, like, 10:00 a.m. class, 9:00 a.m. class. Heaven forbid you ever have a 9:00 a.m. class, and you're sitting there, and, like, the alarm goes off, and you're like, yeah, not going to brush my teeth today. Snooze. And then you think about it a little bit and nine minutes later, it goes off, and you're like, okay, it must be, like, 8:09 a.m. now.

And then the math kicks in. You're like, if I don't take a shower, that's like two more whacks on the snooze bar, snooze, snooze. And now you're up to like 8:27 a.m. You're just doing this all while you're, like, have asleep, right. It's like calculus when you're fully awake, yeah, impossible. Snooze bar, you could have, like, your brain removed in a lobotomy and you could still figure it out.

But what the snooze bar actually gives you is a way of doing something a certain number of times when you want to be able to do it a certain number of times. And so you might say, "Hey, Karel, sometimes rather than just, like, telling you turn, left turn, left turn, left three times, can I just tell you to turn left three times? Is there some way of doing that?"

And in fact, in Karel there is, and it's something called a for-loop, and the syntax for this looks like this, for, just the word for, paren, int I equals zero. And so this might look a little bit strange, and if you've done some programming before, it won't look strange, and if you haven't done any programming, this is just the syntax. Just use it and feel good about it.

This is int I equals zero, semicolon, I is less than the number you want to count up to. So if I want to do something three times, I'd have a three here, but in general this could be any number that you want. Then another semicolon, and then an I followed by a plus and a plus with no spaces, and then a close paren.

And then after all that, you have a brace, an open brace, and there's a close brace down here, and everything that goes here is the body of what we refer to as the for-loop, and that body gets executed however many times you say here. It can be one instruction. It can be more than one instruction. So turn left could actually be defined – or turn right could be defined as for I equals zero, I less than three, I plus, plus, turn left.

And what this will do is it will execute this three times. I could have multiple instructions in here if I want to, and it would execute that whole set sequentially three times, but this is the equivalent of turn right. Okay, question?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):**So the spaces are, sorry, there is a space here. Well, you don't actually need a space there, but there are no spaces in here. So you need to have this space for sure. The rest of these spaces are basically optional, but we like to kind of space them out for it to make sense. You definitely should not have any spaces in the I plus, plus. There's no spaces in there. Uh huh?

**Student:**Is all of this syntax in the course reader?

**Instructor (Mehran Sahami):**It's all in the course reader, yeah. Yeah, so you don't need to worry about – if you want to take notes, that's great, but it's all kind of in the course reader if you want to follow along. So you can do something any number of times that

you want, right, and it'll just count up. So this could be, like, if you wanted to, you could say, "Hey, turn left six times," and that's still the equivalent of turn right.

Or we can do the equivalent of snooze, which is that. Right, so basically what does snooze do? Nothing. You just sit there, and you spin around a few times, and you're kind of like in the same spot, facing the same direction when you were done. Okay, but anytime you want to do something some number of times, you can actually do it using this for construct.

And this is the way you should remember it. You're always just going to use this syntax in Karel, and the only thing that's going to change is this number of times that you want to do something. All right, so besides that there's also sometimes, though, where you don't know how many times beforehand you want to do something.

And you might say, "Huh, that's kind of weird. When would I not know how many times I want to do something?" Well, let's say I'm Karel, and I'm standing here, and there's a wall over there. And I know there's a wall somewhere over there, because my world is finite. It has to end at some point, but one told me how many steps it is to that wall.

So if I say, "Hey, I'm going to take ten steps to that wall using a for-loop." One, two, three, four, five, six, and then I'm like, oh, dead Karel, right, and that's a bad time. So sometimes, we just don't know when the program starts, and what we want to say is, "Do something while some condition is true," like, keep going forward as long as there's no wall in front of you. And when there's a wall in front of you, stop going forward.

All right, so the way we do that is something called a while-loop. So at this point, hopefully you're intimately familiar with all of Karel's instructions. And the way the while-loop works, while, because that's what you want to do something.

You want to do something while there is some condition that's true, is the syntax looks like this, while, and then a paren, and inside here we're going to have some condition. Okay, so I'm going to put a squiggly line under it. So you don't put the word condition. We're going to put a condition in there in just a second, a close paren, and then we're going to have some body inside braces, open brace, close brace.

Now, where do these conditions come from? So one thing we can think about is the notion of moving until we get to a wall. And so Karel, it turns out, has some things that he can sense about his world, which are conditions you can check to see if there's something going on in this world.

One of the things you can check is front is clear, and that's an open paren and a close paren, and so there's actually two parens. Lower case F, capital I, capital C, all one word. And so what that asks Karel to do is say, "Where you're standing right now is your front clear," which means is there a wall right in front of the direction you're facing.

If your front is clear, so while your front is clear, what it does, it basically checks if the front is clear, and that's true, then it executes the body of the while loop. So we could say, for example, move here. And so what it's going to do is, while my front is clear I move and it's still clear, and it's still clear, and I just keep doing this.

And at some point, I move along, and my front is no longer clear at this point. Front is clear is no longer true, and so I stop. Okay, and basically at that point I stop executing this loop and I just pick up from right after the close brace. So I'm after I'm done executing a loop, whether it's a for-loop or a while-loop, I just pick up executing after that close brace.

Okay, so I could actually turn this, for example, into a new method, private, void, walk to wall, or we'll call it move to wall like that. And then anytime I basically want to tell Karel, "Just keep moving forward until you hit a wall," I just essentially invoke, or call the move to wall method, and it will just get me up until I get to a wall.

Now you might wonder, hey, where are all these conditions and what are all the conditions? So if we can get the overhead for just a second, I'll show you. Page 18 of your reader. So on page 18, there's the lovely table one. Oh, lovely table one, and these are all the conditions you can check in Karel. There's front is clear, left is clear, right is clear. So you can sort of check to your left and right.

There's no behind is clear. Important for babies, not for Karel. There's beepers present, which means is there a beeper, at least one on the corner that you're at. Are there beepers in your bag? Is your bag not empty? Are you facing north, east, south, or west? So you can check all those conditions.

You can also check the opposite of all the conditions, like front is blocked. So you can actually check, like while your front is blocked you might want to turn left, because you want to keep turning until eventually your front gets cleared up, and then maybe you go do something else. Or your left is blocked, or your right is blocked.

So these are all – I'm not going to go through them all, but they're all sort of obvious. They sort of give you the obvious information that you would expect about Karel's world, and they're all in your course reader on page 18.

All right, so given that you have these conditions you can check. Sometimes you want to do something while some condition is true, and as soon as this condition is no longer true, you stop executing. There's also sometimes that rather than doing something over and over, you just want to say, let me check some condition and if it's true, then do this one thing.

So I'm not going to do it as a loop, but I'm just going to done thing. Okay, and that's something called an if statement, and the way an if statement works is it's if, space, open paren, some condition, again I'll put this condition in squiggles, open brace, close brace, and this is the body. And it says, if some condition is true, then do what it's in the braces.

And if it's not true, just completely skip over this and keep executing from right after the braces.

So you might want to say, "Hey, I only want to pick up a beeper if I know a beeper exists in the world." So I sort of want to have a notion of having a safe pickup of a beeper. So I could say if beeper is present, which means there is a beeper, at least one on the corner that I'm at, then it's safe to pick a beeper.

Because if I didn't necessarily check before, and I didn't know there was a beeper there for sure, I might try to pick up a beeper on some corner, and there isn't a beeper, and Karel blows up, and life is unhappy, and tears well in your eye, and the whole deal. All right, so if statement, that's just what we refer to it as kind of as an if statement. It's all lower case.

Now, sometimes what happens in life is you don't want to just say, "If something is true, then essentially what you want to do is this." You want to say, "If that thing is false, then I want to do something different." Okay, so you might want to say, "Hey, if there's a beeper present, I'm going to pick it up. But if there isn't one there, oh, Karel feels a little sad and he's going to put a beeper down to, like, plant a new tree because he wants to fight global warming."

So else and then we're going to have another body here, putBeeper, assuming that Karel has beepers in his bag. So what this says is, if there's a beeper present, then pick it up. If there is not a beeper present, it doesn't try to pick a beeper, it skips this part and does the part that's referred to by the else. So we'll put a beeper. So it will only execute one of these bodies, and which body it does depends on whether or not there are beepers present. If it's true, it's this one, if it's false, if that's one.

Question?

**Student:** Is that a space or a tab [inaudible]?

**Instructor (Mehran Sahami):** These are spaces here. So these are required to be spaces, and the syntax is also all in book too. So you can see where the spaces are there as well. Uh huh?

**Student:** Can you make multiple [inaudible]?

**Instructor (Mehran Sahami):** For the time being, you should just check one condition at a time. Now, you can actually sort of, what we refer to as nest, these kind of statements. So you could say, "If beeper is present," you might say, "Hey, if there's a beeper present there, that's some marker that I should move forward if there is not a wall in front of me." So what I want to do is, if there is a beeper present, what I now want to do is check to see if I can move forward.

So if front is clear, move. And this is what we refer to as a nested expression. Notice how I sort of tabbed it in a little bit, so that it's kind of nested inside this other guy. What it means is check to see if beepers are present. If they're not present, then it skips this whole body and it just puts a beeper down.

If there is a beeper present, it's going to execute this body. Well, what is the body? It's another statement that says, "Is your front clear?" If your front is clear, you say, "Hey, I was on a beeper, and my front is clear. Now, I'll take a step forward." But if there was a beeper present, and my front was clear, or was not clear, I'm not going to take a step forward, in which case it doesn't do the if part. It finishes up and there's nothing else for it to do here. So it's done with this whole part and it's done with the whole if-else statement.

Okay, any questions about that, what's actually going on there, right? If beepers are present, you know you're only going to execute this part. This part is now a goner, because that's the else part, and beepers present was true. So to do this part, you come in here and you say, "Is front clear?" No, front's not clear. I have some wall in front of me.

Oh, okay, well I'm not going to do the move. Is there an else over here? No. This else is not attached here. This else is attached to this if that says, oh, there's no else. Okay, there's nothing else I should do. Are there any other statements over here that I should execute? No, I'm done with the whole body for this part. So it's done with the entire statement and it continues executing from down there.

Okay, so any questions about that? Uh huh?

**Student:** So is it every brace always comes in pairs?

**Instructor (Mehran Sahami):** Every brace comes in pairs. You always have an open brace and there's going to be some close brace that's going to define the body. Oh, that was the double. All right, so what we're going to do is we're going to put this all together in a big program.

Okay, so we're going to use all these things in a big program, and this program, I'll show you, is basically, Karel is going to run a little steeplechase. You're basically sort of like running hurdles, okay. So we're going to open up another program. Don't worry about all the details right now.

I'm just going to run this so I can show you what it looks like. So you can see what the world looks like and what we want Karel to do. So we're running, we're running, steeplechase. So what Karel's going to do here is they're the world that is always nine avenues long, and he knows that. It's always nine avenues long, which means there are, at most, eight steeples in the world, or eight hurdles.

The reason why we call them steeples as opposed to hurdles is because some of them are big. Some of them are just ginormous, and so what Karel needs to do is he needs to start here and end up over here, and run around every hurdle he encounters. So how are we

going to do that using the stuff that we've learned, and putting it together into a larger program?

Well, the first thing we're going to do is we need to start with our friend, the run method. Right, so in the run method, all, Eclipse doesn't jump out in front of me. Right, so notice we have all the kind of other boilerplate that we want to have. We import Stanford Karel. We're going to have some public class we'll call steeplechase that extends super Karel. Right, all the stuff you've seen before as boilerplate.

We're going to define the run method and what the run method is going to do is, if you have nine avenues you need to go through, that means intervening the avenues, there's at most eight steeples. So I want you to do something eight times. So we're going to have a for-loop that executes eight times. What are you going to do inside there

Well, you have a choice. When you look at the world, either you're facing a steeple or you're not. So if we look over here, right, here Karel starts out facing a steeple, but after he jumps over it, hey it's clear. So he could actually move forward before jumping the next steeple. So that's essentially what we check for in the program. At every step, what we say is, "If your front is clear, there is no steeple for you to jump. So just move ahead.

But if your front is not clear, you got some work to do buddy. So jump hurdle." "And you're like, jump hurdle. I've been sitting here for 45 minutes. You never told me about jump hurdle. Where is jump hurdle?"

And I'm like, "Yeah, I didn't tell you about jump hurdle because we're going to write jump hurdle," and you're like, "Oh, yeah, new instruction that we create." So we're going to break our program down into smaller pieces. So what's jump hurdle about? How do we jump a hurdle?

Well, think about what you want to do to jump a hurdle, and I'll put it in the simplest possible terms. You want to ascend the hurdle. You want to go up. You want to move past the hurdle, and you want to descend the hurdle to come back down. You're like, "Yeah, thanks, Ron, that really helped a lot. How do I ascend and descend the hurdle, right?"

So what we're doing is we're breaking the program down into more and more steps, and notice this is reading like English. And the important thing to realize in programming is programming is not just about writing a program that the computer understands. Programming is about writing programs that people understand. I can't stress that enough. That's huge. That's what programming style is all about. It's what good software engineering is all about.

Writing a program that just works, that someone else can't read and understand, is actually really terrible software engineering. And I've seen software companies do this where some hotshot programmer comes along and they're like, "Oh, I'm great programmer." And they go and write some program to do something, and then it breaks.

And then they come along and they say, "Hey, we need to actually get that program to do something slightly different, or we need to upgrade it. And it's written in terms of the code so badly that they can't do anything with it, and they throw it out completely, fire the programmer, and get a team of good software engineers to actually do it. I've actually seen that happen multiple times."

Okay, so write programs for people to read, not just for computers to read. Both of them need to be able to read it, but it's far more important that a person reads it and understands it, and that the computer still executes it correctly. But that's the first major software engineering principle to think about.

So how do we ascend and descend the hurdle? Move we knew about. Well, how do we ascend the hurdle? If you think about ascending the hurdle. Right, what that means is you're basically facing a wall, some wall that goes up some amount that you don't know.

Well, how am I going to get up that wall? I'm going to turn left, so now I'm looking up the wall. And guess what? If I put my right hand out against that wall and keep moving up along the wall, I can eventually get to a point where I've gotten to the top of that wall. So I turn left to face north, while my right is blocked, which is why that wall is there. I keep moving all the way up until my right is no longer blocked. That means I've extended all the way up the hurdle.

At this point, I'm still facing north. This guy's expecting me to face east, so I can move over the hurdle, right. If I'm facing north, I'm just going to step one more up. So I turn right at the end to now be way above the hurdle and be facing an open place where I can actually do that move.

So now, I do the move. I'm on the other side of the hurdle. I need to descend the hurdle. Well, how do I descend the hurdle? Ah, this is where our friend, move to wall, that we just wrote, comes in handy. So I'll show you both of these at the same time. If I want to descend, I'm at the top of the hurdle, I turn right.

So I'm looking south now, I'm looking down, and I just move all the way down to the wall. It hits the floor, and this is move to wall that we just wrote, right, while front is clear just moves. I move all the way down, and so when I move to wall and I get to the wall, I'm facing south. I want to turn left one last time so I'm facing forward to go again. Okay, so any questions about that?

Let me run Karel, so we can see if he's feeling up to the challenge. So here he is. We'll start the program. There he goes. Notice what he's doing. He goes over every hurdle. If his front is blocked, he goes up and over. Here's one where you can see it.

He goes all the way up until his right is clear. Then he moves and he comes all the way down until he hits the wall, and he keeps doing this. And if he does this eight times, he always ends up at the end because we know it's guaranteed to be nine avenues long. Uh huh?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):** Yeah, good eye. So you might say, "Hey, before when Karel turned right and he was making three right and left turns, what's going on now?" It's super Karel, baby. We're loving it. Super Karel not only knows the three turn lefts, just turn right, but he's like, "Hey, forget turn left, man. I'm just turning right." So he knows, or she knows, or it knows, or whatever, that amorphous form in the sky knows.

Uh huh?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):** Good question, because that's not the specification of the program, right. So one question you could ask is, like, "Hey when I bought my word processor, I bought my word processor to, like, write essay with. So how come when I got my word processor," and this is a valid question to ask, "Why didn't it come with all the essays I needed to write at Stanford?" Right?

I mean, it would be nice if it did, right. I'd go pay the \$400. I'd be like, woo hoo, super Karel, baby. I got super word processor. Right, because that's not the specification of the program. The specification of the program is to go over the hurdles one by one.

Now, just to see if – and you might say, "Oh, that's kind of interesting, but does it just work on this one world?" So let's consider another world, super steeplechase. So we can load a world, say you can just click load world, and there are some worlds that we'll provide to you in your starter project, which is all explained in the handout.

Super steeplechase. So here's super steeplechase. Yeah, that might look familiar. More bars in more places, right. So here's Karel, right. This is like a world that 50 high. That's Karel down there and you're like, "Oh, man, you're really making," yeah, we're making him work. All right, so go little Karel. There he goes. Oh, oh, come on, give him a little encouragement. Go Karel, go Karel, go Karel.

Aw, yeah, because you can just speed him up if your program's running slow, but there he goes, right. Huge steeples, he's done, same program works. So the important lesson that comes from that is to think about the generality of the program, right.

You don't want to just write a program that works in one particular world, unless that's the only world you need to worry about. You want to generalize your program in a way so that it's using, in some sense, general principles that will apply to any world that meets your specifications. Right, it still needs to meet the specification. Our world is still nine avenues long and we know what the specification of the steeples are.

All right, so any quick questions before we go? Uh huh?

**Student:**So if you had the [inaudible]?

**Instructor (Mehran Sahami):**No, because it's still just nine avenues. The for-loop counter stays the same, right. It's that while-loop that's actually going up and down the hurdles, because it doesn't know how big the hurdles are. Any other questions?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):**That it's nine avenues and the hurdles are just pools, basically, of any length.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Yeah, all right, I will see you on Friday. If you have any more questions, come on down.

[End of Audio]

Duration: 48 minutes

## Programming Methodology-Lecture03

**Instructor (Mehran Sahami):** Couple quick announcements before we dive into things. There's one handout, which, hopefully, you should have gotten. It will contain the Karel example we did in class last time, the steeple chase, as well as some more examples that we're gonna go over this time. But I encourage you to actually pay attention to what we do in class rather than sort of looking at it on the handout, because one thing that's always true about programs is once you see the solution to a program, it's easy to lull yourself into thinking oh, I could have done that, it seems so easy.

But when you're actually sitting there staring at sort of a blank page on your monitor, it's a lot harder to generate the program. And we'll actually generate a couple of programs together in class today, so I'd encourage you to sort of take part in that that we do in class, and then you'll have the actual listing for the code to take home with you so you don't need to worry about scribbling everything down quickly.

A couple quick announcements, the class website, one more time – every time I look on Access there's more and more people enrolled in this class, which begins to frighten me after a while but that means there's people who haven't gotten any of the previous handouts.

So if you need to know where to get all the previous handouts if you just joined the class, CS106A.Stanford.edu is the website for the class, you can get all the handouts there, and there's a whole bunch of stuff you should read right away.

Section sign-ups. How many people have already signed up for a section? Oh, yeah, that's a good time. If you haven't at this point, you should have looked around at all those hands that went up and just realized that sections are just filling up to the left and right of you, literally.

Section sign-ups are open, and if you go to CS106A.Stanford.edu/section, you can sign up for a section. The section sign-ups close on Sunday at 5:00 p.m. You need to sign up for a section in addition to enrolling on Access to actually be enrolled in the course – important thing.

Another thing just real quickly is just for your convenience, I mentioned last time sort of the conditions of the [inaudible] about the world. For your convenience, we actually posted them up off a link on the announcements on the CS106A website, so if you're programming somewhere and you forgot your little Karel book and you're like what are all those things that Karel can check in the world, like is front is clear, or is facing north or whatever, you can just go to a link off the website and you'll find them all listed there.

One other thing I would encourage you to do is we haven't been picking up the audio on questions that are asked, so one of the things I'd encourage you all to do right now is you'll notice there should be microphones on the fronts or on the backs of the seat in front of you.

Pick up the microphone right now, just pull it out of the slot and just keep it in your lap. Don't worry, it won't damage anything on you personally. It's just kind of fun. You can keep you there, it'll keep you warm, and that way if you ask a question it'll remind you to use the microphone, all right?

So with that said, I wanna take a quick poll, because last time you got your first assignment and the download instructions and everything. How many people have already downloaded Eclipse? Oh, rock on, good time. How many people have gotten the first assignment filed that you needed to get? And how many people have started on the first assignment? Wow, I love that. Oh, that just warms the cockles of my heart.

How many people are done with the first assignment? Oh, yeah, just [inaudible] and you're like oh, yeah, I got Karel to run around, and it's just a good time. So you've still got a week to do it, but that's a good thing to keep in mind.

So a little bit of additional background on Karel before we dive into the real meat of things, okay? One thing you may have noticed is all the Karel code that you've either started writing, if you're already working on assignment number one or all the code that we write in class is all in file that ends with dot Java. If you haven't noticed that before, you'll – now you know, it ends with dot Java.

Because in fact, Karel is all implemented in Java. So one of the things you might be wondering is hey, I know a little Java; can I use some Java in conjunction with Karel? And the answer is no. For the purpose of the Karel assignments, you should just use the constructs that you've been shown in class and in the Karel reader, just keep it to those constructs. That still gives you plenty of stuff to do with Karel, it's a good time, but keep it to that, and actually starting on Monday we're gonna, like, sort of leave Karel behind and say bye-bye, Karel, and I'll be, like, bye-bye, I love you, call me. But we'll get into Java, so if you know Java now, just use the Karel stuff in the Karel assignments that we actually do.

All right, so with that said, any questions to start off with before we dive into something new? Um-hm?

**Student:**How do you stop the program? Like, I had a problem, it goes up into the corner and just kind of like it says an error message.

**Instructor (Mehran Sahami):**And it kept looping there forever?

**Student:**Yeah, and I tried making, like, an empty program called Stop, but that didn't really work.

**Instructor (Mehran Sahami):**No, just go up to the little icon – or the little thing in the top of the window that allows you to close the window and just close the window. Karel will be okay, he knows how to deal with that.

As a matter of fact, that's – I am so glad you asked that question. It's just a wonderful thing, because it actually leads into the first topic that I want to talk about, which is common errors. And so there are some common errors that may come up, and these are good things to kind of know about. And one of them is the thing you just ran into.

But before we actually talk about that at sort of the level of Karel, I wanna ask you a question: [inaudible] another one of these strange questions. How many people have actually ever read the instructions on a bottle of shampoo? A few folks – oh, wow. Man, that's more than – I was, like, you've really gotta get out more often, right?

But I'm surprised that many. And what do those instructions say? Rinse, right, that's where you rinse your hair, and then you lather, and then you repeat. And if anyone actually followed these instructions, you would still be in the shower now. As a matter of fact, you'd be in the shower for the rest of your life. Why? Because you rinse, you lather, and you repeat, which means you rinse, you later, and you repeat.

And you just keep doing this, and it's like Karel just taking a shower, right? And he's like you keep telling me to rinse – to linse, rinse and lather together. Rinse, lather, and repeat, right? And this is what we refer to in program-speak as an infinite loop. Which as you can guess is a loop that keeps going forever. And this may come up in your Karel programs. Here is an example of an infinite loop. You might be standing somewhere and you might say while front is clear.

Oh, well, if your front is clear, I want you to turn left, because, you know, I have this idea that eventually you'll turn and you'll be facing something and your front won't be clear anymore. And that might be a great idea to have, but what happens is if here's our friend Karel, happens to be standing in the world and guess what? There's no walls around him. What does he do? He rinse, lathers, and repeats, right?

He just keeps sitting there, turning left on this corner, and his front will never become blocked so he'll never get out of that loop, okay? That's one of the places where the syntax of the program is perfectly valid, but in terms of our intention for what we wanted the program to do, it didn't work as intended.

This is the more common thing that you're actually going to need to deal with fixing in your programs as opposed to syntax-type problems. As a matter of fact, I won't mention the name of the company but there was a company years and years ago that put out this advertisement that was, like, our processors are so fast they can execute an infinite loop in 2.5 seconds.

And you kind of looked at that and you were like ah, ha, ha, funny computer scientists. But the thing that was actually funny about it was the company that put out these chips, the chips actually had a flaw in them where if they executed things really fast they would warm up and then they would melt. So in fact, they would execute an infinite loop in 2.5 seconds, and then the processor would catch fire and it would stop.

So sometimes, that happens. Hopefully, that won't happen to your machine. Don't worry, even if you get Karel going real fast, it's okay. If your processor melts, come talk to me, it's never happened. But it'll make for a wonderful story in class, and it'll cost you about \$1,800.00. Fine.

So another kind of error that comes up fairly often also has a little affectionate name to it. And this is something that we refer to in programming speak as an off-by-one bug. Or if you take the first letter of that, it becomes an OBOB. And sometimes you'll hear people refer to OBOB, and if your name is Robert they're not saying, like, oh, Bob. No, that's just referring to the off-by-one bug.

And what that means is you forgot to do something one more time than you really needed to, even though logically it didn't seem like you needed to. So the best way to understand that is with a little concrete example. So let me actually give you a concrete example by going over to the computer.

So here's Karel sitting in his world, and one of the things we might wanna do with Karel – whoa – Karel in his world is to say that we actually want Karel to lay down a line of beepers, right? Just whatever row he's in, just lay down a line of beepers till you hit the wall. And that's a fairly common thing that you might wanna do, okay?

Now one way we might be inclined to actually do this if we were to write some code is to say something like hey, I'm gonna, you know, write a new class, I'll call it fill row that extends super Karel. I import all the Karel stuff, so I'm doing all the crazy stuff that [inaudible] told me to do, and hey, I want to keep putting down beepers until I fill the row. So wouldn't it make sense to say while front is clear, right? There's no wall in front of me. Put down a beeper and move to the next corner.

And as long as my front is clear I'll put down another beeper and keep taking another step and doing this over and over, and this seems like – hey, that seems like reasonable code to lay down a line of beepers. What happens when I do this? Yeah, so let's actually run it. Let's bring up our buddy Karel. Where'd you go, Karel? Here you are. And run that program. And so what Karel does, he puts down a beeper, his front is clear, so he moves, puts down a beeper, front is clear, puts down a beeper, front is clear, puts down a beeper, front is clear.

He moves here, front is no longer clear. Before he puts down the beeper, he says hey, my front isn't clear anymore, I stop. He's off by one because there's one more thing that he needed to do, which was to put down another beeper in this last spot. This is another very common logical error that comes up – you're off by one in terms of what you're thinking, and so lo and behold if we go into the program over here and we look at this comment, and I'll tell you about comments in just a second, it says this is buggy, it's off by one. You need to add a final put beeper.

And so what we do is we say hey, after you got out of your loop, your front is no longer clear, but you haven't put down a beeper on this corner yet. Put down a beeper, right?

And if we save that puppy off and we close this off and just so we know that it's in fact running, doo doo doo, we run our little program and rock on, all right? No more off-by-one bug, life is good. So any questions about the OBOB or the off-by-one bug? That's another common thing.

These are just a couple of common things I wanna show you, so, you if they come up in your programs you don't feel like oh, you're out there adrift, all alone. Like, these have been done millions of times by other very qualified programmers. Don't worry, it happens to everyone.

All right, so one of the things I just mentioned is this thing called the comment. This thing that's up here in green. What's that all about? So as I mentioned to you last time, one of the key software engineering principles is to write programs that are understandable by people, not just by machines.

And so what a comment is is a way of being able to put something in your program that another human being reading your program can actually read and understand, and it actually has no impact on the execution of the program at all.

And so the way the comment works is it starts off with a slash and a star, and now everything you put – it can even span multiple lines – until you put a star and a slash is a comment. It's there just for the person to read, and it doesn't affect the program at all.

And so what I would encourage you to do as part of good programming style is all of your programs up at the top should have a comment that says what the name of the file is and has a little bit of an explanation about what your program actually does.

And if you're wondering how much of an explanation or how in-depth it should be, be it as in-depth as you want, but in the handout you got today you see examples of a bunch of programs, and they're all commented, so you can see examples of comments in there.

There is also shorthand you can do. There is a comment that's just slash, slash, and that just means everything on the remainder of this line is a comment. So it actually has no close, in some sense, for the comment like this does explicitly. Everything else to the rest of the line, when you hit a return, that's how it knows it's done with the comment.

So sometimes if you wanna have a one-liner comment somewhere, you can just put a slash, slash.

**Student:**[Inaudible.]

**Instructor (Mehran Sahami):**Yeah, when you get assigned to a different section leader, you'll actually put your assignments in with your name and your section leader, and that's how it'll get differentiated. So wait until you get your first section, and then you'll see how submission stuff works. Um-hm?

**Student:** What about when you write private programs [inaudible] public ones, do those need comments, too?

**Instructor (Mehran Sahami):** Oh, when you write, like, other methods? Yeah, you should comment those as well. So methods should also be commented as well, and I'll show you an example of that today, actually, so you can see an example of how we do that.

So that's actually – and I love it when everyone just, you know, it's like ringers in the audience. You just lead me into the next topic, and that's a perfect example to say hey, let's actually look in, look at a program that we did last time, which was the steeplechase program, now with comments, okay?

So up at the top – sometimes you need to click this little plus sign to expand the comment, because Eclipse has a tendency to sort of minimize the comments automatically. So just click the plus sign and that'll stand them out.

Steeplechase, so we have a nice little comment here at the top of our program. Notice this is a Java file. As I mentioned before, it ends with dot Java, because Karel's actually implemented in Java. But you should use no features of Java other than what is in the Karel book. And if you're like hey, [inaudible] I don't know Java, hey, nothing for you to unlearn. You're good to go, because you just know Karel, all right?

The other thing that's going on here is you'll notice that inside the program we have comments as well. So this method, the run method, it says to run a race that's nine avenues long. We need to forward or jump eight hurdles eight times. And that makes it explicit to the person, right?

If we didn't have that comment, someone would come along and say hey, the race is nine avenues; why are you only iterating this eight times, right? And that's a natural question to ask. And so you put in comments to clarify things in the program, which are not obvious, okay?

Another thing related to that is what's referred to as pre-conditions and post-conditions. What did you expect to be true before you called a particular method or before a particular method is invoked, and what's going to be true afterwards?

So our friend Jump Hurdle here – remember Jump Hurdle from last time? We ascend the hurdle, which means we go up, we move to cross over the top of the hurdle, and we descend hurdle to come back down the other side?

Well, what we needed for that to be true for this to actually work right is the precondition is you're facing east at the bottom of a hurdle. Which means the hurdle was right in front of you and you were facing east in front of it. That way, we know which way to turn and how to ascend the hurdle and how to get over it, and when we're done you're going to be facing east again.

So you can assume, after this method is done, that you're facing east again and you're at the bottom of the world in the next avenue after the hurdle. So it makes clear, right, because we don't know what-all stuff is going on inside ascend hurdle and descend hurdle, but a programmer now doesn't need to trace through the execution of your program to figure out what's going on.

This tells him what needs to be true before and what needs to be true after, and it helps you with debugging. Because if you're debugging your program and you call this jump hurdle method and you haven't satisfied the precondition, it's probably not gonna work right and that allows you to work backward to figure out what you need to do to fix things before you even call this method, okay?

So any questions about that?

So as we go further down, you'll notice – and this is all in your handout – all of the methods have pre-conditions and post-conditions that are commented in here. This is a good habit to get into, at least for your Karel programs, okay? Even if you don't explicitly put pre- and post-conditions, you should at least have some comment for every method, explaining what that method does. So that's the level of commenting that we actually want you to have.

Notice it's also perfectly fine to have very short, like you know, three-line or in some cases even one-line methods is perfectly fine, because it helps – like here's a two-line method over here – it just helps break the program down, okay?

And that's a process we refer to as decomposition. And so what's decomposition all about, right? We need to think of this in sort of more concrete terms that you may understand from your daily life. So besides reading the bottle of shampoo, another thing that we could talk about is what you did this morning.

So if I were to say hey, what did you do before you got up for your – or before you went to your first class this morning? How did you get yourself ready this morning? What did you do? Anyone wanna volunteer what they did? I know it might be a little scary for some of you. Um-hm? What did you do this morning? Use the microphone, please. Pardon?

**Student:** Basically I basically rolled out of bed, and then –

**Instructor (Mehran Sahami):** All right, so you woke up. Wow, did you even hit the snooze bar at all?

**Student:** Twice.

**Instructor (Mehran Sahami):** Oh, good times. That's always good.

**Student:** That's why I set it early.

**Instructor (Mehran Sahami):**She was good. I had a – my old roommate in college actually would play the snooze bar game, no joke, for two and a half hours. And I would be sitting there, I'd be in the room, like, I'd be awake, staring at the ceiling, and [inaudible] I'd know his snooze bar, and I'd be like Joe, why don't you just not set your alarm? And he'd be like no, man, I gotta set my alarm.

And every day – anyway, I'm glad it's only twice for you. You can tell there's just, like, you know, like I got this little twitch from, like, 15 years ago from the snooze bar. But you woke up, and then what did you do?

**Student:**Okay, and then I brushed my teeth.

**Instructor (Mehran Sahami):**Okay, you brushed teeth. What else?

**Student:**Go to the bathroom.

**Instructor (Mehran Sahami):**Bathroom. I won't ask you for any more detail on that.

**Student:**Yeah.

**Instructor (Mehran Sahami):**What else.

**Student:**That'd be awkward.

**Instructor (Mehran Sahami):**Shower? Was shower in there, maybe?

**Student:**I shower later in the day.

**Instructor (Mehran Sahami):**Oh, okay, that's fine.

**Student:**After [inaudible].

**Instructor (Mehran Sahami):**I'm just gonna stick shower in here.

**Student:**Okay.

**Instructor (Mehran Sahami):**Just for good measure. But it's perfectly fine whenever you wanna take them. What else?

**Student:**And then clothes, [inaudible] clothes.

**Instructor (Mehran Sahami):**Clothes. And then presumably you probably went out of your room or ate breakfast or – did you get up for breakfast?

**Student:**No.

**Instructor (Mehran Sahami):** Yeah, I know, it's brutally early, isn't it?

**Student:** Yeah.

**Instructor (Mehran Sahami):** Like, I think, like, I ate breakfast like once the whole time I was an undergrad, and it's because, like, I just stayed up the night before and I was, like, oh my god, there's a thing called breakfast, and I can actually go to it, so I went.

But so this is an interesting thing, you give us sort of a breakdown of what you do. Thank you very much, what's your name, by the way?

**Student:** Jasmine.

**Instructor (Mehran Sahami):** Thanks, Jasmine. I'll try to see if I can get to you, but – denied every time. So here's, you know, a standard set of things that we might think about doing. Now it turns out this is a very good list of things that you might want to consider doing. Good hygiene for the morning. But in fact, when you think about something like brushing your teeth, brushing your teeth – we all understand what brushing your teeth means.

But if there's someone, like when I, you know, try to get my 1.5-year-old son to brush his teeth, I say hey, William, you need to brush your teeth, he has no idea what I'm talking about. Because brushing teeth for him involves getting the toothbrush – brush – putting toothpaste on it, and sort of moving it on your teeth, right?

So there's a level at which I need to break this down. As a matter of fact, moving it on your teeth, if you've ever talked to a dentist, this is a whole involved process by itself. Like what angle you put it at and how much you move it and how much time [inaudible]. It's like whoa, man, I never thought there was that much to brushing your teeth, but evidently, there is.

And so what happens is we start with a high-level description, and then each one of those steps in the high-level description we break down. And we keep breaking it down until we get to a level of detail, which is sufficient for us to be able to understand minutely. Or if we're doing this on the computer we get to a level of detail at which the computer understands it directly.

What that means is we eventually get to a level of detail where it turns into things like put beeper or move or while something is true, right? So when we get to that level which we call primitives, that's when this process eventually stops. But this is essentially a process that's called stepwise refinement.

Because what we're doing at each step in the process is we're coming up with some steps, and if those steps are not the primitives that we understand we need to refine them into more steps and refine them into more steps until we eventually get to the primitives, okay?

So we start at the high level and we break things down. And the notion of breaking things down into their sub-pieces is something that we refer to as decomposition.

Decomposition. Which just means taking something big and breaking it down into smaller pieces, which is something you should do with your program. You wanna break them down into smaller pieces, okay?

And this whole process, the whole process itself is something we refer to as top-down design. And you'll hear this phrase referred to a lot, and what top-down design is is you start at the highest level, the top, and you go down from there. And that's to be contrasted with something that's referred to as bottom-up design.

And what bottom-up design says is hey, I need to go do something in the world. Well, let me start at the primitives. I need to make Karel, you know, go to the end of the wall and put down beepers. What's the first thing I need to do? I need to put a beeper, and then I need to move. And it starts with the low-level stuff and this is actually the place most programmers, when they start out, this is what they begin doing is bottom-up design.

People have actually done psychological studies. It takes about 100 hours of programming proficiency. In the average case, I think everyone at Stanford's above average so it'll take you less, to go from thinking bottom-up to thinking top-down.

Guess what? It's a 10-week quarter. We might do about 100 hours of programming in here. Hopefully by the end of this class, everyone's doing top-down design. But if you can start from the very beginning doing top-down design, you're golden.

In some cases this makes sense, but a lot of times this just makes your life easier, to think about the most abstract level and break down from there. And so if you think about what we did when we actually did the steeplechase program, this was top-down design, right?

When we actually wrote this program together, when we went through it together, we didn't talk about Karel doing individual moves to begin with. We said hey, we need to run a steeplechase, and that either involves doing the move or jumping a hurdle. At this point, we've done enough step-wise refinement that we've gotten to the level of a primitive. We don't need to go any further because Karel immediately understands move.

Jump hurdle, Karel doesn't understand. So once we've written this, we need to break it down into lower-level steps and say what does jump hurdle involve? And that's where we actually go down and define jump hurdle, and guess what? When we define jump hurdle, there might be some other things like ascending and descending a hurdle, which are the next steps down in the process that we need to define, and we just keep doing this, okay?

So with that said, let's actually do top-down design together. Let's start another program from a blank slate and see if we can do top-down design together to see how this process might actually go for writing code, okay?

So the problem that we wanna try to solve is we wanna – oh, and don't look at the code – we wanna try to get – and this is kind of a funky problem – it's Karel does math. If you didn't think Karel could do math, in fact Karel can do math. Karel's a lot smarter than we sometimes give him credit for.

So this is a little problem called double beepers. So I don't know if you can see the number here, but there was a five here, which means there was five beepers on this corner. And when this program is done running, what we want to have happen is Karel is guaranteed to start in position 1-1 and there's guaranteed to be a pile of beepers in front of him and a space after him, okay?

When he's done, he should return back to the same position and the pile of beepers should have exactly twice as many beepers on it as it had before, which means Karel begins with a beeper bag that's full of as many beepers as he needs, so it has infinite beepers in there so we can double the number of beepers.

So if I speed up the program and just run it, you can see the final state. So let me just crank up the speed here and start the program, and it looks like nothing happened, but in fact this number changed from a five to a 10. It doubled the number of beepers.

So two things should be going off in your head right now – one is thinking hey, that's kind of slick. How do I do that? What's the recipe for doing that? And the recipe for doing that, the sort of general approach you wanna take, is something that we call an algorithm, okay?

So the notion of an algorithm – and this is just a fancy computer science term for an approach to something – actually comes from – anywhere know where the word algorithm comes from? Nineteenth century Persian mathematician named, if I can pronounce it right, Al-Khwarizmi. Al-Khwarizmi, which sounds like algorithm. There you go.

Yeah, add 1100 years to that pronunciation and it sounds like algorithm, all right? Because this is from the 19th century. It's your basic approach. As a matter of fact, there was a band at Stanford years and years ago called the Algorithms, where this was, like, rhythm, like hey, I got rhythm. Yeah. I won't say how good they were. Not very.

All right. So you wanna think about your general approach and then you turn your general approach into thinking in terms of step-wise refinement. How are you gonna solve this problem from the top down?

So let's actually start with a blank slate. Let's close this puppy out and we'll start with a blank slate, which we'll call, oh, Our Double Beepers. So I just gave you the boilerplate stuff to begin with. There's a little comment up at the top that says file double beepers. That should be called Our Double Beepers, actually, and Karel doubles the number of beepers on the corner directly in front of him in the world, he returns to his original position and orientation.

So we have the public class stuff for Our Double Beepers, we have an [inaudible] super Karel. Now we're gonna run the run method, okay? So what do we wanna do to get Karel to double the number of beepers at a high level? What might we think about doing this? Hm. Um-hm?

**Student:**[Inaudible.]

**Instructor (Mehran Sahami):**We need to figure out how many beepers there are. In order to do that, maybe we should move to the pile of beepers, right? It's one avenue in front of us. So we do a move. And now here's the part that almost seems like magic. I'm going to say move, hey, you're on this pile of beepers now. Why don't you, hey, double the beepers in the pile? And you're like, uh, [inaudible] can I do that? Sure, why not, right? You're gonna write the program, you can do whatever you want.

So I would like to say move, double the beepers in the pile, and then actually what I wanna do is not move forward again but I wanna move backward, right? I wanna move – almost like Karel could go in reverse, that'd be kind of cool, to keep the same orientation but goes back.

And you're like but [inaudible] you didn't tell me about move backward. Can super Karel do move backward? And I'm like, no. And you're like oh, okay, well, all right, well, I guess I need to write some of these, right? Let's actually do move backward, because that's the easier one.

If I'm sitting on a particular corner and I'm facing east, what do I need to do to move backward one step and face the same direction? Yeah, I need – so let me define move backward here. Move backward. I need to turn around, which is a primitive that I can use. Then what do I do? Move. And then what do I do? I turn around again so I have the same orientation when I'm done as when I started. And now, guess what? Move backward is something I understand how to do, because I've broken it all down into primitives.

I don't need to go any further in terms of the refinement for move backwards. But now there's the magical part, right? There's double beepers in pile, and that's something that might be a little bit more involved. So I come along and say okay, I need to define what double beepers in pile is. Double beepers in pile. At this point, I need to think about my algorithm. What's going to be my approach for doubling the number of beepers? What's a way that I could possibly think about that problem? Because Karel can't count, right? There is no way for Karel to just say hey, counting how many beepers are on this corner.

What's one way he might approach it? Um-hm?

**Student:**[Inaudible.]

**Instructor (Mehran Sahami):**That would be nice if we had a counter, but we don't have a counter, right? So Karel can't count. How is he potentially gonna do it? Um-hm, all the way in the back. Use your microphone, please.

**Student:** So he needs to pick up one beeper at a time and put it on the empty corner, and when he puts it on an empty corner, put two beepers for that one he puts down?

**Instructor (Mehran Sahami):** Yeah. So why don't we – I know it's not gonna happen. Yeah, I know, everyone's just starting to cower now. I do realize that, I've been teaching for about 15 years and my shot has never improved, so you're just, you know, at the random candy mercy.

Since we can't count, we need to do something beeper by beeper. So wouldn't it be interesting if we said hey, you're on that pile of beepers, pick up one beeper off the pile and take a step somewhere else and put two beepers down, and keep doing that over and over.

Pick up a beeper, put two down. And eventually, you'll exhaust everything on this pile over here but you'll have twice as many on the next pile beside you. And then take that pile of twice as many beepers and move them all back to the place where you started, and then you're done.

That's the algorithm. Let's write the code to do that. And you're like okay, [inaudible] let's write the code to do that. And I'm like yeah, rock on, this is gonna be easy, right? What we're gonna do is we're just gonna say you're on a pile of beepers. As long as there are still beepers for you to pick up – so I'm gonna have a while loop here – while beepers present.

Well, what I want you to do is pick up a beeper, yeah, you know how to do that – life is good. Beeper, I always misspell beeper, I don't know why. I want you to put two beepers next door – put two beepers next door – and you're like oh, this is always the part, right, where you're thinking bottom-up design, you get uncomfortable calling methods that don't exist, right?

You're just like oh, but that doesn't exist, right? And I'm like, it's cool, man, we're gonna write it. And then we hold hands and we sing Kumbaya and everything's good because at the end, Karel's gonna know what to do. So sometimes, you just write methods and they don't exist, and it's fine because you're the one that's gonna be writing them. So eventually, they'll exist, right?

It's not like you're sitting there, like, lighting candles, making some magic incantation, hoping the code just shows up. And you're like, is it there yet? No, it's not there yet, right? You're gonna write it. I had a friend once who actually thought that's how we did computer science. I didn't bother to tell him that that wasn't true.

But, you know, think about it. So we pick a beeper – yeah, I know, it's kind of mean, but that's all right. He does government now. We pick up two beepers and guess what? After we do that, when that's done we will have picked up all the beepers from that pile that we're standing on, and we would have put two next door for every beeper we picked up.

So when we're done with this while loop, we know that the thing that's true is that there's no more beepers on that pile because beepers present is no longer true.

So I have some pile next to me, say on the next corner, that has twice as many beepers on it. And so what do I wanna do? I just wanna move the pile next door back. And that's my function. You're like, oh, really? I can do that? Sure, why not? So what do we need to write now? Well, we need to know how to put two beepers next door.

So let's go ahead and write that. [Inaudible] private void, put two beepers next door, right? Now if I'm standing on a particular corner and I'm facing east, let's say I wanna put two beepers on the space in front of me and come back to where I am still facing east, so I'm left in the same position I was when I started, how do I do that?

I move. I need to put two beepers so I could put beeper, put beeper. And if you really wanted to you could put a four-loop here and say hey, [inaudible] you're doing something twice, ooh, ooh, can I do a four-loop and do four I equal into I equals, zero, I less than two, I++, put beeper? Sure, right? There's multiple ways of writing a program. I'm just going to put two put beepers because it's two lines. The four-loop would have been two lines too.

Now what do I do? Ah, yeah, rock on. I move backward, right? Which allows me to move back one step and keep the same orientation. So I end up, my post-condition is essentially the same as my pre-condition, except I've moved one beeper over to the – or I put two beepers onto the next spot.

And move backward, you're like hey, I already wrote that. So everything now is in terms of instructions that Karel will understand, so at this point I don't need to go any further. So I know how to put two beepers next door, so the only thing left to do in this program is to move the pile next door back to where I am now, okay?

So what that means is how do I move that pile back to where I am? So I'll call this, for lack of – well, I know what the function name is, I'll just – or the method name. I'll just steal it from up here, because sometimes you just get tired of typing.

All right, so I'll move the pile next door back. Well, the place that I call this, right, I call this up here. I'm not actually on the pile next door when I called it, right? My pre-condition is I am on the avenue before the pile. I'm facing the pile, but I'm not on the pile. So the first thing I need to do in terms of moving the pile next door back is I need to move over to where that pile is.

So now I'm standing on that pile of twice as many beepers, and I wanna move all the beepers back one. So what could I do? I could pick beeper, but there's something I wanna do multiple times. I need to have a wild loop, right? And I'm gonna do something – what condition do I wanna test, right? If beeper is present, right?

So as long as there's beepers present there's still more work for me to do because there's still more beepers I need to move back. So while there's beepers present, hey, why don't I move one beeper back? Move one beeper back. All right? Again, I haven't wrote that yet, that's fine.

But when this loop is done, what I know is I've exhausted all of the beepers that are on this pile because every time there was beeper present I picked it up, I moved it back one spot, and hopefully that method brought me back, should bring me back to the place that I'm standing right now, and I just keep doing this until I've transferred all the beepers.

So after I've transferred all the beepers, what do I need to do to satisfy my post-condition, which is I wanna go back to the place I just transferred all the beepers? I need to move backward. Because all the beepers were transferred one avenue back. So I just move backward to go back one avenue, okay?

Now I'm almost done. I'm almost feeling good, but not quite. I need to do the move one beeper back. And at this point, I've gotten to such a simple level that maybe I could just do this. So if I'm standing on a pile of beepers facing east, how do I move one beeper back behind me, put it down, and come back to the spot that I'm at? Well, first let me pick a beeper, right? Because I need to actually move the beeper.

I could turn around first, that's perfectly fine. There's multiple ways I can do things, but I'm gonna pick the beeper first. Then what do I do? Yeah, rather than turning around, let me just use move backward. That'll take me back one spot. Then I put beeper. And I move to move back to the spot that I'm at, right?

Now I'm all in terms of things, what I refer to as primitives, right? Or instructions that I've previously defined. And there's nothing more for me to define, because every time I define some new method and I said hey, I'm just gonna call that, I went and wrote it and I just kept doing that over and over until I'd written everything I needed to do. Um-hm?

**Student:** Wouldn't you have to move backward twice [inaudible]?

**Instructor (Mehran Sahami):** On move pile next door back? No, because move one beeper back, I pick up a beeper, I move backward, I drop it off, and I move forward. So I'm on the pile that has all the beepers.

**Student:** [Inaudible.]

**Instructor (Mehran Sahami):** Yeah, and that's what I did up here, right? So after I doubled the beeper pile, then I moved backwards. So yeah, that's a good point, but you always wanna remember what other things are gonna happen after you execute all that, okay?

And so now the question comes – ah?

**Student:**[Inaudible.]

**Instructor (Mehran Sahami):**That's a good question. The only problem with doing that is Karel starts off with an infinite number of beepers in his bag, right? So if I pick up all five beepers at the corner that I'm at and then I take a step back, or I pick up all five beepers and now I wanna put them all down and double them, I don't know that there were five anymore, because after I pick up the five, I look in my beeper bag and I'm like oh, man, I had infinitely many beepers. I've got infinity plus five, and that looks like infinity, right?

As a matter of fact, if you take CS103, which I encourage you to do because I teach it in the winter eventually, we talk about infinities, and we're like oh, infinities, and it all comes back to Karel. No, it actually doesn't come back to Karel at all. But infinity plus any number is within reason – yeah, if you're a number theorist, you're gonna argue about that – is still infinity. So that's why Karel can't count.

If he had an empty beeper bag we could put all five in it and look inside and then we could put down five. But now we don't have five more beepers to double the number of beepers, which is why Karel had to start with infinitely many beepers, so no matter how many were in the initial pile he would always have enough beepers to double it. But that's a good point.

So let's actually go ahead and run our program now. Let's save, and we'll go ahead and run Our Double Beepers. Doo, doo, doo, doo, doo. Operate just to make sure it actually works. And we'll sort of do it at slow speed. So there's our world, let's start the program. There's Karel, and you can see he's going [inaudible] beeper – whoa.

He just didn't want me to explain that program at all. All right, Karel, I'll remember this. So for every beeper he picked up, he put two here, and now he's just transferring them all back. Go, Karel. And so now he's done with 10, and you'll notice he went back here after 10. Why? Because he needed to check to see if there were any more here, and there weren't, so he moved back and then he was done.

So we just doubled the number of beepers. And this program, actually, notice the initial state of the world is now exactly the same as the ending state of the world, except we doubled the number of beepers. So we could actually run this again if we wanted to – let me speed it up a little bit – and we'll go from 10 beepers to 20 beepers. And there is Karel sort of doing his thing at faster speed, but you can see there's 20 beepers over there, now they all start coming back and saying, you'll be like, hey, [inaudible], how many times can I run it? Can I just run it over and over and I'll get, like, you know, 40 beepers and 80 beepers and 160 beepers?

You just can't. I think eventually the program dies when you reach, like, 100 and some-odd thousand beepers, because it's just like no more, man. No more, all right?

But for all programs you write in this class, if you ever have more than 100,000 beepers on a corner, there's probably something wrong, so you don't need to worry about it, okay?

Now, that's – so congratulations, right? We just all wrote this together, and you just made Karel do math. As a matter of fact, Karel, in terms of raw computational power, can solve any arithmetic problem that any other program can. It just may take him a lot of beepers to do it. But theoretically, you can prove that. We're not gonna do it in this class, but if you're just kind of – you're like can Karel compute, like, square root and logarithm and he can integrate? Yeah, he can. It's just a whole lot of beepers, all right?

So I could show you another program. Here's another little program I happened to write in the office. This is a program that happens to be called Do Your Thing, okay? And I'll show you the full program, I'll scroll it down here so you can see the full body of the program. And all I will ask you, what does this program do? Doo, doo.

This does example the same thing the code you just wrote did, okay? This is really bad software engineering, right? There is no decomposition, there's no comments, there's no indentation, there's no notion of what's going on in this program. If you looked at this program and I said, hey, instead of doubling the number of beepers, I want you to say triple the number of beepers, you'd have to go through this thing, and you might, because you just wrote it, say hey, I know that there's two put beepers in a row, so I put another put beeper in there and it'll probably work, right?

Whereas say you're a software engineer and you just got your first job, and you can look at either that code, or you can come over here and look at the double beepers that's actually in your assignment, that has all the comments, like, hey, this doubles the number of beepers.

And over here, guess what? This is the version that's in your handout. This is all the same code we just wrote except now with comments. Like for every beeper on the current corner, Karel places two beepers on the corner immediately ahead of him, and this says place two beepers on corner one avenue ahead.

And you say hey, if I wanna triple the number of beepers, I change the comment from two to three and put another put beeper here, and I know exactly where it goes and it's easy to understand.

That's why software engineering's important, because that kind of modification to code happens far more often than writing code from scratch – about 10 times more often. So if the code is bad to begin with, it gets 10 times more costly to go and make fixes to it, and that's something that's sort of well documented. Some people actually argue that the number is higher.

But that's why it's important that when you write your program the first time, or subsequent revisions you make to it, you have good style because that's just way more

important than just getting the program working. And I've seen people actually turn in programs called Do Your Thing. That's where I actually got the name, was I actually knew someone once who write a program called Do Your Thing.

And I was, like, you've gotta be kidding, right? And it was just like this, right? It was just a whole bunch of straight-line code, no decomposition. They were like, well, it works. And I'm like yeah, but that's not what software engineering is about. And we got into this – yeah, I won't call it an argument, I'll call it a discussion. And they got really adamant, because they were, like, but it works, so I must get an A.

And I was like, you know, someday if you're teaching this class and you're willing to accept code like this, you can give yourself an A. I'm not, because this is just – anyone who sees this, not only in this class but outside of this class, will be horrendous code, and I don't want any people coming out of this class to go out into the industry and be like, hey, yeah, this is what [inaudible] told me to do, that's why I'm writing it this way, right?

Because not only will they be, like, no, I'm sorry, you're fired, but find that clown [inaudible] and fire him, too, right? So don't write code like this.

Is there any questions about this? Um-hm?

**Student:**[Inaudible.]

**Instructor (Mehran Sahami):**And let me bring up the good code again, just so you don't get, you know, think like that other code is the good thing to do.

**Student:**When you're naming the private [inaudible] methods, do you – should the first letter be capitalized, or not?

**Instructor (Mehran Sahami):**Ah, good question. So in terms of – someone asked before, all the words or methods that we define are all case sensitive. You'll notice that sometimes we use a naming convention. I'm actually bad because I mixed up the naming convention a little bit where the first name of the methods, we can either have a lower-case character or an upper-case character, it's up to you how you want to do it. Just be consistent.

Sometimes, this notation is referred to as camel case, and the reason why it's referred to as camel case, the first letter of every word that actually appears sort of all stuck together is upper cased, and it's kind of like a camel with humps, because every once in a while you get this capital letter with a hump. So that when you – if you hear someone refer to oh, we use capital case or camel case notation, that's what they mean. It's all strung together as one word with the first letter of each word capitalized.

Sometimes what people will do is they'll actually put in underscores. You can put in underscores between words and that's perfectly fine, too. When we get into Java, I'll actually spent probably like half of a lecture talking about the different conventions for

all kinds of things, because when we get into the Java world there's actually a bunch of things besides just method names you need to worry about, and we use all different sorts of conventions for them.

For right now, I'd just say pick one convention and stick with it, and the convention you can use is, you know, the two most common ones are either camel case all the way through or camel case except for the first character, which is lower case. Any other questions? Um-hm?

**Student:**[Inaudible.] Instructor:

Yeah, so it starts at run and every time it encounters some method that it doesn't know is a primitive method, it just says hey, did you define it somewhere? And it goes to that definition. And guess what? If it encounters one that you didn't define, that's an error. And usually it'll tell you that when you try to run the program. It'll tell you that this thing doesn't actually exist.

All righty? Any other questions about – um-hm?

**Student:**[Inaudible] as to what order you should put your [inaudible]?

**Instructor (Mehran Sahami):**No, you can put them in whatever order you want. Usually I just put them in the order as I need to define them, right? So just like we did in class, I just keep going down, which means run starts at the top, and everything comes underneath it.

A couple things related to that. One question that always comes up is how do I know how much to decompose, right? Like aren't there different ways I could decompose? And in fact, yeah, there are multiple different ways you can decompose. So let me just give you some sort of quickie guidelines about how you might think about decomposing, okay?

In terms of decomposition, the thing that you really wanna think about is every one of your methods should basically solve one problem, okay? The only thing is that that problem can be at different levels. So when you think about decomposition, what are some good guidelines?

One guideline is think about each method solving one problem, okay? That problem can be high-level, like double the number of beepers . That's a high-level single problem. When you get into doubling the number of beepers, you don't just wanna say oh, double the number of beepers again, because you're already at that level of decomposition. You need to go one step down and say, what are the things that make sense in terms of solving one problem at a time to break it down?

Now a lot of times, people wonder but [inaudible], what does that mean in terms of lines of code? Because sometimes it's different for me, at least conceptually, to think about one problem. And here's just a rough guideline. It's not a guideline that always holds, but a

rough guideline is most of your methods will probably be somewhere between one and 15 lines long, okay?

And you're like, one line? Yeah, that's perfectly fine. It's perfectly fine for you to have a method that's just one line long that calls some other method, and you might say what's the purpose of that? The purpose of that is giving that thing that you call a different name, right?

So remember when we did ascend hurdle, which basically brought you all the way down until you were facing a wall, and then it did one last turn at the end to get you facing the right direction? But basically all it was doing was move to wall. We could have actually defined ascend – or sorry, descend hurdle to be move to wall and then do the turn afterwards.

That would have been perfectly fine, too, and then descend hurdle would have just been a one-line method. But it would have changed the concept, the way the programmer thinks about it, from being hey, I'm moving to a wall to be hey, I'm descending a hurdle.

So that's another thing that kind of comes up in terms of the methods is you wanna think about them as having good names. The names should describe what the method is actually doing, what problem it's solving, because you want your programs to read like good English, right?

When someone – again, programs are written for people. Computers execute them, but they're really written for people to be able to go and look at and modify, so good names explains what the program actually does, and every one of the methods that you have in terms of decomposition should also have some comments associated with it that explains either at some more detailed level what the method is doing, or if the method is very short and it's fairly self-explanatory, pre-conditions and post-conditions are sometimes a good idea as well.

So think about these things for your assignment. When you're actually writing the code for your assignment, you wanna think about the sort of decomposition using the step-wise refinement process and make sure to comment – both a comment for your whole program and a comment for each method. Um-hm, question?

**Student:** The book mentioned procedural programming and decomposition under that. Is that what we're doing right now, or are we doing the –

**Instructor (Mehran Sahami):** Well, there's a difference between procedural and object-oriented programming. Decomposition actually applies to both of them, but what we are doing is decomposition whether it's procedural or object-oriented. They're just two different – come up to me after class and I'll set you up.

So I did wanna show you one more thing, and the one thing I wanna show you is yet another interesting problem, which is called Clean-Up Karel. And what Clean-Up Karel

was basically gonna do is Karel starts off, it's kind of like after – we should call it After-Fraternity Party Karel. It could be After-Sorority Party Karle, too. I don't wanna make any kind of claims.

But basically what ends up happening is that Karel kind of comes into his world and it's a mess. He's just like what happened? When last I was in this world I just doubled the number of beepers and shown you I can do math. And now you bring me into this world and there's just crap everywhere, right?

And we could call him Angry Karel, too, right? He's the one who's gotta live here. And there's, like, cups strewn all around and they're just on random corners, and the things that you know about Karel in this world is he starts at location 1-1 and every corner of the world can have at most one beeper on it. It means it can either be empty, like some of these corners, or have one beeper on them.

And what Karel needs to do is he needs to go through his whole world and clean it up, which means he needs to gather up all the beepers that are out there. And he can end up wherever he wants to end up, okay?

Now when we kind of think about a problem like this, one thing we might be inclined to do is you say hey, if there's a whole world that I wanna clean up, I might wanna break this problem down into a series of steps. One step – and something that is, like, oh, I can go and write it right now because I know how to write it is maybe to clean up one row at a time, right?

Because I probably know how to clean up one row, I probably just wanna keep moving until I get to a wall, and if every time I move there happens to be a beeper there, just pick it up. And so you get all excited and you go and you write the code to do that, and so you start off by having something like this – let me just focus on the top code up here – right, which is hey, I wanna clean a row.

This thing just does not wanna stay on me. It's like no, you will not talk into the microphone. So cleaning up a row I could say hey, you know, [inaudible] told me about the OBOB too, he told me about the off-by-one bug. So before I even do my first move, I'm gonna see if there's a beeper where I'm standing now, and if there is, I'm gonna pick it up.

And then I'll check if my front is clear, and if it is I know I can move and I should check to see if there's a beeper there, and I'll just keep doing this over and over until I've picked up all the beepers.

And that's a perfectly fine thing to do, right? So now you've written some method called clean row that can clean up a whole row, and you know you've sort of dealt with the off-by-one bug because you sort of do one extra thing outside of your loop, either before or after. In this case, you happened to do it before just so you can see a different kind of formulation of it than doing it after, and that's great.

The only thing is, now you need to figure out where clean row is gonna go in your program, okay? This is an example of sort of doing more, the notion of bottom-up as opposed to top-down design. It's easy to get excited, to say oh, there's something I know how to do, and just go write it. And if you happen to write the right thing that fits cleanly into the rest of your program, that's great. But sometimes it doesn't, and you might need to go back and modify it, so keep that in mind.

But if you do this, that's perfectly okay. It turns out that in this case, that's a fine thing to do, to clean one row at a time. And so if we wanna think about how that fits into the larger program, here's kind of a funky thing. And let me just explain the algorithm to you, because the algorithm here is somewhat related to an algorithm, for example, on one of your programming assignments.

I'm gonna clean up the first row, because I know that first row I always need to clean. So I'll just go clean up the first row, and now I wanna check to see am I done. Well, if I've just cleaned up a row, the only way I know I'm done is if the ceiling's above me, right? I've reached the end of my world.

And so I'm facing east, so I check my left to see if there's a ceiling up there. And if there's a ceiling up there, then I'm done. But if there isn't a ceiling up there, then I need to do more work. So while my left is clear, right, there's no ceiling up there, I'm gonna reposition for it to clean up a row to the west, because I just cleaned up a row sort of moving to the east – am I facing your east? Yeah.

I just cleaned up a row to the east, and I wanna say hey, no ceiling here, that means there's another row to clean up here. So what I wanna do is move up one row to be able to now head west and clean up the next row, because I'm gonna go back and forth, that's my algorithm.

So I'm just gonna write some function that says reposition for a row to the west, and go ahead and clean that row. Now what do I need to do again? I need to check my roof. But now my roof, because I'm not facing that way, isn't on my left anymore. I need to check my right to actually check my roof, so I say if my right is clear, then reposition for a row to the east. You're gonna go back the other way and clean that row, okay?

Now there's this else here, and we'll get to the else in just a second, but just to figure out what reposition to west and east are, they're actually very simple. I'll just show you those down here. Reposition for row to west means I just finished a row to the east, how do I get up to harvest the next row or to clean up the next row?

I do one turn left that gets me to face north, I move up one step so now imagine I've floated up one row, and then I do another turn left and I'm facing like this, and now I'm ready to plow over the other direction. And I do sort of the opposite for repositioning for a row to the east. So I'm standing here, I do a turn right, I look up, I move up one row, I do another turn right, and now I'm ready to go back that way.

So these are very simple, right? They're in terms of primitives. So this is the whole program. You've just seen the whole program now. The only question that comes up is what happens with this little turn around here. What's going on there?

So you said hey, [inaudible] you just finished cleaning a row and you just checked to see with your right hand, actually, if my right is clear, if there was a ceiling above you. And if there was a ceiling above you, right? Or if there wasn't a ceiling above you, you went ahead and got ready to clean another row. But if there was a ceiling above you, what do you wanna do?

And you say hey, if there was a ceiling above me, I wanna stop – just stop. Is there some way to get Karel to stop? For the love of god, Karel, stop. No, Karel's just like no, man, I'm going. I'm going. You're like, there's no way to just say stop, Karel? How do you get Karel to stop? You need him to finish executing what he's executing. What is he executing, right? If we just got here and we cleaned up a row and I checked my right and there's a ceiling there, if I don't do anything, what's Karel gonna do?

He's gonna go back around and say hey, is your left clear? Yeah, because my left is below me, it's clear. And it's gonna go and try to clean more, even though I just finished the last row.

So what I actually do is when my right is blocked over here I say hey, you know what, Karel? To get you to think that you're done, I'm gonna have you turn around. So now when you check your left, the ceiling is gonna be there and you're gonna break out of that loop.

So you wanna think about the conditions on your loop to actually get this to happen. So in our final 10 seconds together, I'll just run this to show you that it actually works. Doo, doo, doo. And you're like actually, [inaudible] you're in, like, our, you know, final minus four minutes together. I know, I'm sorry. Education's one of those odd things that the more – you get more of it for paying the same amount, and somehow you feel like you were cheated.

So let's start Karel. Look, if I let you out 10 minutes early, everyone would be like oh, yeah, rock on, but if I gave you a car that only had three doors on it, you'd be like what's up with that? Strange, how that works.

And there's Karel, just cleaned up his world. All righty, so I will see you on Monday. Have a good weekend.

[End of Audio]

Duration: 53 minutes

## Programming Methodology-Lecture04

**Instructor (Mehran Sahami):** So welcome back. Now we're officially started with the class. I hope you had a good weekend. I was just asking people before class what kind of stuff they did this weekend. So if anyone wants to, come early. We'll just engage in random conversation.

So a couple of quick announcements: There's one handout, which is your section handout for this week. Make sure you get that. What I'll do every week that we have a section, is that week, you'll get a handout on Monday, which is a problem that you're going to go over in section.

This gives you a chance to think about it before section. It's not required that you do it. It's not extra homework; it's just something and if you want to read through it to get familiar with what you want to do with section or possibly do it yourself if you want more practice to be prepared for section, you can do that and then you'll go to section; you'll cover the problem and actually in section, you'll get solutions to the problem, as well, so you'll get examples of code for all the section problems.

Section assignments – hopefully you should have signed up. The deadline for signing up was Sunday at 5:00 p.m. Section assignments will be mailed out to you tomorrow night so you'll know which section you've been assigned to and you can go to sections this week, get your section leader's email address, complete Assignment 1, at least the email part by sending email, and I don't need to mention the sections actually start this week, so you should go this week once you get your section assignment.

If for whatever reason you didn't manage to sign up for a section, you should remedy the problem by emailing cs198@cs and let them know you forgot sign-up for a section for CS106a and they'll let you know what times are available and you can sign up.

Don't email me because I have nothing to do with the sections, other than actually assigning the problems for the section. Don't email Ben because he's not involved in administration for the sections either. He also worries about the content of the sections. But the 198 folks handle all the administration. Uh-huh?

**Student:**[Inaudible]

**Instructor (Mehran Sahami):** Yeah, you should submit it again after you've been assigned to a section just to clarify. All right, but if you've already submitted it, we can sort figure it out eventually, but it's in your best interest to re-submit it after you've been assigned to a section. So any questions about anything else before we start? Uh-huh.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** Yeah, the most submission is the one that's counted if you submit multiple times. It just lets you kind of submit more if you'd like, if you submit

something and you're like, "Oh, I made a mistake." Then you can go back and submit again. We don't feel like, "Oh, every time I like save, I need to submit." Don't worry; just submit when you feel like you're done and if you need to go back and make changes, that's fine. Uh-huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**If it's really necessary and the main thing is if there's space in other sections. Yeah, you would email cs198@cs and let them know. All righty.

So let's go ahead and get started with the actual content. And so today is the day where we make a little bit of a transition. It's like one of our transition points in life. It's kind of like coming to college. And the transition point we make is you now come into the Java world, and you're like, "But I'm still doing Karel." Yeah, in lectures, we're going to start to wave bye bye to Karel and Karel's like, "Yeah, I'll be there in your dreams when you're actually programming, right."

So Karel's not really gone until you're done with the Assignment No. 1. But now, we take – we sort of graduate into the world of Java and you'll actually see that a lot of things that exists in Karel's world will come back because we give them to in a gentle form in Karel's world and then the Java world, they come full steam.

So before we dive into that, I want you to get a little notion of the history of computing. Okay, and we'll go over this very briefly, but computing actually has a very long history and it dates back about 4,000 years, okay. So roughly 4,000 years ago, the first one we think of is computing device was made available, but then you want to know what that was? Advocates.

The first computing device allowed you to do some fairly rudimentary arithmetic on it, but it actually was something that allowed people to compute a lot faster than they could by keeping track of stuff in their head. The real sort of advances in what we think of as computing actually came around in the 1800's. That was someone by the name Charles Babbage. Anyone heard of Charles Babbage, by the way, a just quick show of hands. We can just call him Chuck.

He was actually a very well learned and well-known person in his day. He had the Lucasian chair, which is the same chair at university that actually Steven Hawking has now that Sir Isaac Newton had at one point. So, pretty smart dude, and he came up with this notion of something called the Difference Engine, which is a way of being able to automatically.

And in those days, right, they didn't have silicon. Well, they had silicon, but it was in the form of sand, and they didn't have computers in the way we think of computers, so he wanted to build a mechanical device like with real, actual, you know, mechanics and pieces of woods that's been around, and in his time, he thought is would be powered by

steam, right, because this is sort of pre thinking about electronic computers that would do automated calculation.

And he designed something called the Difference Engine and then he designed something called the Analytic Engine, which was supposed to be even more powerful, and interestingly enough neither one of them was actually built during his lifetime.

It turns out that years later, the difference engine was actually built from a bunch of wheels and rotors and stuff and now the things, you know, that he would have liked to think of as the Analytic Engine is actually these puppies. It actually turned in the 1800's; he had a lot of the same ideas that are in our laptops or in our computers today strangely enough.

But, so that's kind of where our computing we think of the notion of computing is really coming from, and it turns out the first programmer was a woman by the name of Ada Byron, actually Augusta Ada Byron and if the Byron looks familiar, too, is because she was the daughter of Lord Byron, the English poet and she was actually very taken by the designs of Charles Babbage's machines and actually wrote programs.

The machines were actually written to have sort of cards, sort of like the Jacard Loom if you've ever heard of the Jacard Loom, if you haven't, it's not important, that actually would have programs for the kind of computations that it would do and see, she actually devised various programs for Charles Babbage's Analytical Engine.

And you might be sitting there and you might think, "But Marion, you've just told me that the Analytical Engine was never actually built in their lifetime, so what's she doing writing programs for a computer that doesn't exist," and that's actually something that happens today. People write programs for computers that don't exist and you might wonder why. That's not a very good use of your time.

Well, guess what, if the next generation of computers is going to come out while it's still being designed before it's actually manufactured and built, someone needs to figure out what kind of programs you want to run on that machine, so there are actually programmers today who actually write programs for machines that aren't built. And they simulate them by hand and they go through and try to figure out what they would do and it's perfectly reasonable thing to do.

But as a result, Ada Byron is in some sense, the first programmer, right, because she was actually writing programs for in some sense, a general purpose computational device over 100 years ago, which is kind of an astounding thing if you think about it.

Now it turns out computing actually got its first real, you know, what we think more closely of as computing devices in the 1930's. I should put 19 here just so we're clear – 1930's and 40's, they were sort of the first prototypes of electro mechanical computers that were actually built at the time.

There was a computer that was built at Iowa State by Atanasoff and Berry. Their names are actually not – well, they’re important if you’re a lawyer because it turned out there were actually lawsuits at the time by who patented it in the computer, but we won’t talk about that.

Started sort of in this era with sort of prototypes and then the one machine you may have actually heard of by 1946, there was a machine called ENIAC, which was actually built by Elkhart and McCauley at U Penn which was standard for, if I can remember this, for electronic, numerical integrator and calculator.

And basically with one of these big things, you know, sat in a big warehouse, a few tons, but it actually did computational, you know, the kinds of things we would think of as computation and it was really in some sense the first large-scale electro magnetic computer that we think about, again, you know, modular lawyers.

And then really it was in 1971, that the first microprocessor came around, so we come to a fairly modern time, right, so it’s not all that long ago, like we’re talking 36 years, the microprocessors have actually been around.

And so the first microprocessor and one of the folks who was on that team who built the first microprocessor, Ted Hawk, is actually Stanford alum, interestingly enough, it was built at Intel. The Intel 4004 was the name of that microprocessor and at the time, no one actually thought that having a single chip microprocessor was going to be that interesting.

He sort of designed this thing and it was originally going to be going over for use in some machine in Japan and they were just going to give them some design and the patent for it and the folks were getting like, “Well, we’re not really interested. Yeah, sure it’s an interesting chip, but we’re not really interested in, you know, owning the patent. We don’t think there’s anything interesting necessarily here,” and so they kind of ran with it and the rest is history, as they say.

And as you can imagine in the last 35 years or so, there’s just been an astounding amount of development in computing and computer science, right, so really this field is like hundreds or thousands years old, if you think about it, but really it’s kind of in your lifetime that all the interesting stuff is really happening, which is also why it’s exciting to be alive now and be a computer scientist because you’re in this acceleration phase, right.

Think about what’s going to happen, like all the stuff that happened in the last 36 years, you’re gonna be around to see the next 36 years and you’re gonna be doing the next 36 years, so that’s just an exciting thing that I think about.

But another thing that comes up when you think about this, what is computer science, right, like why do we – we’ve been talking about Karel programming and a lot of this class we’re gonna learn about software development and Java programming and so a lot of people tend to equate computer science with programming and they think, “Well, why

isn't a major just called, you know, computer programming and why is there really a science to it in some sense."

That's something that people actually would use to debate. They don't really debate it so much anymore. Is there a real science to computing or is it just programming and in fact, the difference between the two is that computer science or CS as we'll just affectionately refer to it, really is the science or in some sense, the study of problem solving with computers, and you don't need to write his all down; it's just for your edification, with computers or I should say, computational devices.

Computational – I'll even say methods because some people actually look at, you know, theoretically what's possible without even necessarily thinking that it's real life in hardware, right.

Turns out it actually surprises people there are some functions that you can prove are not computable. I'm like, "Really, you can?" Yeah, and it's good to know about them before you go and try to write a program to actually compute them because in theory they're not computable and you can prove that, and if you're actually interested in that, take CS 103 or CS 154 and we'll talk about it all the time.

But that's really what the science is, is thinking about problem-solving and the approaches to problem-solving, how you analyze how efficient a solution is to a problem, how solvable it actually is, the different approaches you take to it.

So, notice the word programming doesn't show up here. Programming is an artifact. It's something we do to realize that particular problem-solving technique in a computer or some computational method. It's just something we do as part of the process, right. It's not what the whole thing is really about.

In some sense, there is actually this famous quote that computer science is just as much about programming as astronomy is about building telescopes. Right, astronomy is about far more than building telescopes. It's really sort of a science of celestial bodies.

Telescopes is the mechanism by which you actually look at them. Uh-huh, question in the back?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Well, if you think about computer engineering, part of it is just kind of a definition that a particular school wants to have for it, right. And some places actually – our computer science program is in the School of Engineering. Some places it's actually in humanities and sciences. It depends on the view and it also depends on the particular program that's there, so some places may actually –there's no way I'm gonna hit you in the back – may just be training programmers, right.

And the science of computing is not what they're actually interested in, and so it depends on the program, but we really think of the computer science as a science, and it's an

interesting philosophical argument. We can certainly talk about it more in class, but I just wanted to kind of push on.

All right, so with that said, that's kind of the quick tour of a couple hundred years of computer science history, it turns out when we get into modern day, when we actually think about computing, what does a computer really understand and what are the programs that you write on it and how do they turn into something that the computer understands.

So it turns out that the computer actually understands zeros and ones. And so we like to think of that as binary, right. Binary is just a number system that only has zeros and ones. It's base two numbers basically and that's all that computers understands, zeros and ones on and off.

So how do we take these things like move or turn left and turn them into ones and zeros, and so there's something called machine language that is what a computer understands or in some sense what the microprocessor understands, right. The machine languages is defined by what chip you have inside your computer. That's eventually going to boil down into a bunch of ones and zeros.

People are real bad at writing ones and zeros in a form that computers can understand. So we program what's known as a high-level language, okay. And there's all kinds of high-level languages, so one of the ones that you're gonna learn in this class is Java and there's some other ones you may have heard like C or C++ or Basic or Fortran or whatever the case may be.

These are all high-level languages, because they're at a higher level than what the machine understands, and so there's this question that's actually part of the science of computing is how do you go from this high-level language into something the machine understands, right. What is this translation process because there actually is some translation that needs to happen and this process going from the high-level language to the machine language is something we refer to as compilation, right.

So compilation is something, strangely enough, that's done by a compiler and Eclipse which you've been using this whole time is in fact a compiler. It takes the instructions that you write in some high-level language and converts them into some form that the machine understands.

Now it turns out that the process for doing this in some languages is actually slightly different than other languages, so I want to show you a quick little overview of how that looks in Java just so you can get a real sense of what's going on between the time you write a program and the time the machine executes it so you sort of understand it at a low level.

As programmers, you write what's called source code. Know it, learn it, live it, love it. Source code – that's what you're gonna write. When you were writing Karel programs,

you were writing source code. When you write Java, you're going to write source code. What the machine understands, these zeros and ones here, it's something that's called objects code.

An object code is essentially just the low-level instructions, the machine instructions that the computer understands and so going from source code to object code is basically what the compiler does. It's the translation from this to this.

Now in regular languages, that might look, if we got the overhead. Is shouldn't say regular language. In some classical languages, it might look something like this, you write some source code in what we refer to as the source file.

The source file is just a file that contains source code. So you write your program that goes into some compiler, say Eclipse, which is the compiler you're using and what might come out of it is an object file because what is contained in that file is just essentially object code or in this case a bunch of zeros and ones that the computer understands.

And there might have been other programs that someone wrote along the way like some libraries you might make use of like you did with Karel, right, all the basic Karel functionality might be in some other library and so there's other files that contain object code and all of this stuff gets linked together into one big set of object code which we refer to as the executable or the application.

That's the thing that, say, when you double click on your word processor on your computer and it runs, you're running some executable file, which is just basically a file of a bunch of ones and zeros that eventually or as some time ago, someone wrote and source code and it got compiled down in this executable file.

And for a lot of languages like C or C++, this is the process that actually happens. Now the people who did Java thought of things slightly differently and here is kind of where things get funky in Java, okay?

In Java's world, part of what's going on is actually run on a virtual machine. What does that mean; it means you write some source file in Java, okay, that goes through some compiler and what comes out of it is not an object file, but something called a class file.

It takes all of the high-level stuff that you write in Java or in this case, you could say Karel because it's the same thing, and turns it into some set of numerical instructions that are not yet ones and zeros that the computer understands but are some intermediate language, which is just a numerical language.

And sometimes you refer to this as Java bite code, but the name is actually not important. It's some intermediate language, and guess what, there's other classes just like there were before that contain instructions in this intermediate language and those all get linked together in some big file that we call a jar archive which just stands for jar just means Java archive, strangely enough.

So it's actually redundant to say jar archive. People just do, but it means Java archive. All right, and then this whole thing is now instead of this intermediate language, it's something that neither the human really understands – okay, there's a few humans in the world that might and they're a little weird and we won't talk about them. Most people don't understand this and the computer doesn't understand it directly either.

So you say, "Well, that's the most useless thing ever. Why would I ever do that?" These instructions go to something called the Java Virtual Machine, the JVM, and what the Java Virtual Machine says, it says, "Hey, guess what? I'm going to be pretending like I'm a machine that understands this as my object code. I understand this stuff as the basic language I speak, and I take that and when I run it, I do something on your computer." And you might say, "Why would you want to have this extra process?"

And the reason why you want to have this extra process is the fact that guess what, in the world there's things like Macs and there's PCs and there's Linux computers and there's all kinds of machines out there, which means if you sort of do things in either the good old way or the bad old way, the compiler needs to understand what are the ones and zeros that your computer speaks and the ones and zeros that a Macintosh speaks are different than the ones and zeros that a PC speaks.

And they're different than the ones and zeros that the Linux machine speaks. So the compiler needs to know all that if it's actually gonna generate this kind of code for all of those different sorts of machines and most compilers don't. Most of the time you've got a compiler that says, "Hey I'm a PC compiler. I'm just gonna generate stuff for the PC and that's all I do."

In Java's world, things are a little bit more interesting because the compiler doesn't need to know what kind of computer you're running on. It says, "Hey, I'm producing this intermediate language and the intermediate language is the same for all computers." The only thing that has to be different on your computer is you need to have this little thing called the JVM that knows how to translate this intermediate language into what's going on on your computer.

So if you have a JVM for your Mac or a JVM for your PC or a JVM for your Linux machine, they can all run this same low-level intermediate code and do the right thing on those computers.

Okay, and so remember when you set up Eclipse or if you haven't set up Eclipse yet, you'll get there soon enough, but when you're setting up Eclipse, we asked you to download and install something called the Java run-time environment and how many people remember that? Raise your hand if you remember that? You folks – guess what the Java runtime environment provided? It provided this thing for your computer, okay.

So you write the program once; you compile it once and now that class information that you have, that class file can be run on any computer that has this Java Virtual Machine

and Java Virtual Machines are sort of ubiquitous. They exist for a lot of different computing platforms, okay.

So that's why they actually do it that way, and it's kind of funky, but it's a little bit different. Now, with that kind of said, that's the low-level stuff. That's kind of what's happening at the low level and you don't really need to know that low level intimately. I just want you to understand what's happening from the programs you write to what happens when they get executed on the computer.

Now what you do need to know, is how do we begin to write programs in Java. In Java, you'll see some examples of it today, is what we refer to as an object-oriented language, okay. So Java is object oriented and not all languages are object oriented. What is this whole object oriented thing about, so you should know what this whole notion is about before you start programming in the language that's kind of seeped in the idea.

The idea basically is that a program is written as a bunch of classes, right, so if you think about, let's back up for a second, if you think about Karel, right. When you are writing Karel programs, of you saw Karel programs, you would actually create some Karel program by saying, "Public, class Karel program extends Super Karel." Let's just say we are Karel model.

And then you put some stuff in here and you have like your run method in here, maybe some other methods and what you did was you were creating a class that was basically the information for your program, what instructions you were actually going to execute as part of that program.

So what a Java program is is just a set of classes. You may have more than one class. Like in Karel right now you always just have one class and inside that class you may have a bunch of different methods, but you always have one class. A Java program can actually have multiple classes associated with it.

And when we start in this class, we'll start with some simple ones that are all just one class. What you have to think about is that you can have more than one class, okay? And what these classes are, what you can think of a class as, okay, a class is just basically an encapsulation of some behavior that the program does, just like in Karel's world, you had some behavior of a bunch of different methods you define for Karel to actually to be able to execute, right, those are just different behaviors.

But you also have some data, so it's behavior and data, because in the Java world you are now actually gonna keep around a track of information that are not just beepers in some world, but they're actually just numbers that you're actually gonna store somewhere and that's your data.

And so what we do is we think of having the behaviors, all the kind of manipulations you might want to do with data, along with the data and put that all into this thing called the class.

So just like in Karel where you had a class and you had behaviors, you can just imagine, yeah, we're gonna have some behaviors, which means in Java we're gonna have some class and define a bunch of methods in that class, but we're also gonna have some data associated with it and that whole think together is gonna be a class.

Now, it turns out, the thing that's interesting about classes. Is there any questions about the notion of a class? It's just kind of this abstract concept right now. The thing that makes classes really powerful is the classes actually get organized into hierarchies, I mean, just draw it here, okay.

And you've actually already seen this in a very small way with Karel, so all the things that you've seen in Karel are gonna translate over. Remember we talked about the basic Karel model and when you are writing a program you could say extends Karel and when you did the basic Karel model, you got like move and turn left and pick beeper and put beeper and that's all you got.

And then we said, "Hey, you really want to turn right and turn around, well there's this thing called Super Karel and Super Karel is just a souped up version of Karel." Well, guess what, Karel was just a class, and Super Karel was also a class, but Super Karel is a class that extends the functionality of the basic Karel class, okay.

So we say that Super Karel, the way we would draw things, if we were gonna put boxes around them and draw like the book is say that Super Karel is a sub class of Karel and Karel is the super class, this is where it's gonna get confusing, but super class are the things in some sense that are more general and the sub class are the things that take the basic idea of some class and extend it.

So Super Karel does everything that Karel does, but it also does turn right and turn around, okay. So Super Karel is a sub class of Karel or alternatively Karel is a super class of Super Karel, which sounds a little bit backward, but the super here and the super here mean two different things. Super here means sort of above, and super here means cooler than, okay.

So don't want to think of them as they mean the same thing, it's just an overloaded term, all right? So you might say, "Okay, Marion, this is getting a whole little abstract for me." Well, let me put it in terms that you understand. You're a human being, right? At this point, you should say right. And if you're not, come talk to me. All right, it wouldn't be the first time.

So we have humans, okay. Humans are just a class, okay, and all humans are primates. Primates are a super class of humans, because all humans are primates, all primates are mammals, and all mammals are animals. So there's actually a much deeper and richer hierarchy say in biology than there is with Karel and when you actually see Java, Java will eventually have sort of a richer hierarchy associated with it.

Now, the interesting thing about this, this is not only humans primates, but monkeys are also primates, okay. Humans are not monkeys and monkeys are not humans, so there's not a strict – you might say, "Well, we don't know about that." But there isn't a strict relationship between humans and monkeys other than they're both primates, and that makes them mammals, and that makes them animals, and the whole notion of having this hierarchy of classes, right, so you could think of being animal as just a class and it has some behaviors. What does it mean to be an animal?

Well, to be an animal means you digest food, right, it's not something like something like [Inaudible] actually, and one of the technical terms – I'm not a biologist, but just what my friends tell me, embryos pass through the blastula stage. Any biologists in here? Is that right?

**Student:** Yeah.

**Instructor (Mehran Sahami):** Okay, good, good – because I was like asking a friend of mine, like, "What differentiates animals from plants," and he was like, "Oh, of course, the embryo was passed through the blastula stage and I was like, "Oh, right on." I had no idea, and I was like, "How do you spell blastula?"

So I'll just put blastula down here, right. So that means, guess what, all mammals inherit those same properties. Mammals also have some internal digestion of food and also pass through this blastula stage, but mammals additionally we know are warm blooded in general and they have mammary glands which is where the name mammal comes from.

So mammary glands or just kind of a generalization of sweat glands, but that's what mammals have, and then all primates are mammals, which means all primates have all of these same properties plus they have more and so primates interestingly enough have five fingers. I didn't know all primates have all five fingers, but evidently I'm told that they do. They have opposable thumbs. And they also have fingernails, but fingernails we just won't put up there. It's not a big deal, okay?

And then we get down to humans and monkey, right, and humans it turns out in theory anyway were supposed to have highly developed brains. I'll just put brains in quotes here because other animals have brains, too, but we're supposed to have highly developed brains and we have an erect body carriage.

That's just what we are. We're wrecked brains. And monkeys have mother things that are going on, so evidently their brains are not as highly developed and they're sort of like their knuckles drag on the ground. Sometimes I do that in the morning, but that's not important.

But that's the whole point. This is the class hierarchy. These are all classes and things that are sub classes of some other class means they inherit all of the behaviors of the things above them, all the way up the chain plus they may have their own additional

behaviors and that's the key concept here of organizing things in classes and you'll see that in more specific instance when you get into Java.

You saw it already with Karel in just really simple example, right. There was four commands for Karel. Super Karel gave you those four plus an additional two; that was it; okay? Now, besides this idea, there's another key idea that comes up besides this notion of classes, okay? And that's the notion of the instance of a class, okay.

And what the instance of a class is, is humans are a class. There is no one person that is, "Oh, you are humanity, right?" You are a person, right, so [Inaudible] what's your name?

**Student:**Eduardo.

**Instructor (Mehran Sahami):**Huh?

**Student:**Eduardo.

**Instructor (Mehran Sahami):**Eduardo, all right. So I hope I'm spelling this right, Eduardo; is that right? Is an instance of human, I would hope, right. So what that means is you're going to have things in your Java programs that you create which are objects and what an object is to differentiate it from a class, so Eduardo is an object. I'm sorry; sorry to break it to you. That's just – you're an object.

An object is an instance of a class; it is a particular example of a class. There will be multiple objects or multiple instances potentially of some class, and that instance of a class has all of the characteristics of the class and all of the other classes above it in the class hierarchy.

So by looking at this, I could say, "Hey, Eduardo, guess what, you went through the blastula phase when you were an embryo." Good times; because you're an animal, right and I know that and this is inherited all the way up.

So that's the important thing to keep in mind is you're going to be writing classes; you're going to be writing things that define some set of behavior and along with that, you'll be creating instances so that this is an instance of the class and the instances are what we refer to as objects so all of your instances in the world are just things that we think of as objects and your classes are in some sense the templates for those objects, okay.

So any questions about the general concepts; I know they're a little bit high level, but it's important for you to kind of understand. Okay. So with that said, let's actually begin to look at this notion in Java, okay? And so the way that this is gonna work – uh-huh?

**Student:**[Inaudible] in Karel, what was the object?

**Instructor (Mehran Sahami):**So in Karel, it turned out what you're actually just creating was a single Karel object was being created that contained your program and

that's what was actually run. So you created the class Karel, and when that puppy actually sort of fired up and you saw little Karel running around in the world, that was a Karel object; that was an instance of your class.

Now, you actually didn't see multiple objects or multiple instances there. There was only one Karel instance. So there it's very – it's a good question because it's easy to get confused between the instance and the class in that case because it was only one.

But in here you'll actually see we'll actually create classes and we'll have multiple instances of them, which are running Karel with basically the instance of that object. Okay, question?

**Student:** So, basically like you can have, like several Krels running around in like one program?

**Instructor (Mehran Sahami):** You could if we sort of set it up that way. In the Karel, the way it's kind of set up is it only allows you, it only sort of behind the scenes creates for you this one instance of Karel, but in some, you know, alternative Karel universe, we could have actually have just taken her class and said, "Hey, guess what, we're gonna just create multiple Karel instances from your class and have them all run around in the same room."

We could have, and guess what, that's what seemed to have happened in this room, right. Somewhere along the way, there was this class human and someone came along and created multiple instances of you. Well, I wouldn't just say just one person, but now there's multiple instances of you, right, and you're all running around in the world doing your thing and you interact with each other and guess what objects actually do; they interact with each other, okay?

So with that said, there is gonna be some functionality we're gonna use that was written by someone else. So just like Karel, the basic version of Karel was written by someone else. When we start with Java, we're gonna start with some set of scaffolding or libraries that were written by someone else that are gonna allow us to do a bunch of powerful things in Java very early on, okay? And this is something that is called the ACM program hierarchy. So it sounds all complicated.

But all this is, right, and much the same way that I drew that picture over there with animals, mammals, primates, humans and monkeys, this is a hierarchy of classes that exist in Java's world or at least are provided by the ACM and you might say, "Who is the ACM?" Anybody know who the ACM is? They're big; they're bad; they're nationwide.

They're the Association for Computing Machinery. It's the oldest computing society. They've been around for actually for about 60 years or so. And you're like, "But Marion, I thought you said 100 years. Yeah, before then, people called it math."

So Association for Computing Machinery has this program hierarchy and what that means is the kind of programs that we write in this class, just like when you wrote your Karel programs you extend Karel or you extend Super Karel, you're going to be extending different kinds of programs, either a console program which is something that produces textural output, a dialog program which brings up little dialogue boxes that ask for information or a graphics program that actually draws some pretty funky stuff on the screen.

All of those classes are classes that are inheriting from a class, a super class called program. So all of the things that you write as a dialogue program or a graphics program are all something that are programs. And all programs are something uptight J-Applet, which just means Java Applet and all Java Applets, are of some type called Applet.

Anyone ever heard of an Applet? An Applet is something you can run on your web page, interestingly enough. That's kind of where the term comes from. It's like a lightweight application, and applet – application, but small.

And so it turns out since all of the programs that you're writing are actually inherit the properties of being an applet, they actually will be applets, which means you can put them on your web page and run your programs on your web page if you want and later on in the class we'll talk about how to do that, okay?

But in this class, as least, you're not gonna write anything that directly extends like J-Applet or Applet or program. Everything you write is gonna extend down to this level, but it's important to know that there's different kinds of programs that you can actually write at that level.

So let's look at an example of a Java program, and I'll show you your first Java program today. So we can feel good about Java – no it's not that one. It's this one. It's small; it's tiny; it's fun; it's Java. All right, so let me expand this whole thing out so you can see the whole program still fits on one screen, okay.

And you might suddenly notice there's a bunch of things in this program that look very similar to Karel. That's because Karel was implemented in Java. So the first thing you want to look at here is we have a file called `helloworld.java`, just like your Karel programs are written with a `.Java` file, you are creating a `.java` file here. This is a source file because it has source code in it.

Up at the top, you have a comment, "Gee, it actually uses the exactly same structure as Karel comments." Yeah, because Karel comments were actually Java comments. Comments in Karel and Java are exactly the same. Just like you imported `stanford.karel.star`, now you're gonna be using the stuff that the lovely folks at the ACM have provided for you, so you're going to import `acm.graphics.star` and `acm.program.star`.

What are these things? These are just – well, I won't back up because we're not on the slide. These are just – remember I said you write your classes and someone else may have written some classes and they all get linked together before they're executed. These are just some other classes that someone else wrote that are going to get linked into what you do.

Right, they provide you the definition for things like what a graphics program actually is. So now, what you're gonna do, as we talked about, all Java programs are just the collection of classes, so just like in Karel, we have public class and some name for your program. Here we'll call it Hello program.

And a Hello program is a particular kind of program. It doesn't extend Super Karel because it's not a Super Karel program anymore. It's gonna be a graphics program. It's actually gonna draw some stuff on the screen, so it's gonna extend the graphics program.

But all the boilerplates should look the same, right, that's what we had you do Karel because all the stuff from Karel just translates directly over to Java, except now we're sort of cooking with gas; now we're doing the real thing, okay.

So it extends graphics program and guess what, inside here we have a run method just like Karel and that's where the program begins executing. Now, where things get funky is you're like, "Oh, may what happened to pick beeper and plug beeper and like life was so good; it was easy and you know, turn right and it was like the extent of it. What is it like add nugi label, hello world, like what is this all about, okay?"

We'll go through this step by step. All this is saying is when you create a graphics program, we'll go through the details in just a second, you're gonna get a blank screen, you're going to get an empty canvas in some sense.

You're gonna be an artist and you're gonna draw stuff on that canvas, and the thing that you're gonna draw stuff on that canvas and the thing that you're gonna draw on that canvas is some label and all that label is just basically words and the words are gonna be Hello World, because you wanna write Hello World on the screen and you wanna write it at a particular location on the screen.

The location on the screen you wanna write it at is 100,75 and I'll tell you where that is on the screen in just a second. And then once you create this little label, you're gonna add it to your canvas. You're gonna say, "I have some canvas, plop that puppy onto my canvas," okay?

So let's run the program and I'll just plop it onto the canvas or you'll see what's going on. Life will be good. So we run this program, we drop the microphone. Dropped the microphone again, all right. So we want to do Hello program. We're getting excited. We're running; we're running. The disc is just turning away and there's your first Java program.

You're all Java programmers now. What did you do; you created a graphics program, which brought up this big window and said, "I'm blank canvas. Draw on me." And you said, "All right, Hello World – woohoo and I'm done," because that's all you did. You said here's Hello World, put it up on the canvas for me at a location 100,75 and thanks for playing; that's all I'm gonna do.

But now you've actually gone through the whole process of compiling your Java program and turning it into this intermediate code and this intermediate code gets executed and you actually did something and that's like half the battle right there. The next nine weeks is the other half of the battle, but half the battle is getting this up. I kid you not, really. If you can make that happen, you're just most of the way there.

So let's do something a little more exciting than this. Let's actually do some interaction with the user. Let me show you another program. Remember this one is a graphics program. I told you there was a bunch of different kinds of programs you can have when I showed you the picture. I said, "We have a graphics program." Let's look at something called the console program over here. Uh-huh.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**No, not in Java. You're always extending one kind of program and a graphics program will actually, as you saw you can put up text. A console program is just meant to be a textural program, so you're not gonna do any drawing.

But here's another program, right, again, a lot of the same things is before. You have some comment up at the top. You have some libraries that you're gonna import. In this case, you're not doing anything with graphics, so you don't need to import acm.graphics.star, you just need o import acm.program.star which just says, "Get me all the standard stuff for a program."

If you're doing graphics, you also need to have a second line for acm.graphics.star. If you're not doing graphics, you don't. So again, public class. This one called add-to integers because guess what, it's gonna ask the user [Inaudible] programs, put tow integers and add them together, but in fact, it's gonna do something interactive with the user which is pretty exciting in itself. This is gonna extend the console program because it's not gonna have any graphics.

Aw, yeah I know, it's sad times, but sometimes text is very powerful. What's it gonna do when it runs? It's gonna write out to the screen and this time we're gonna use something called print lin, we just like to drop some vowels here and there, which stands for print line, and if print a line, this program adds two numbers, then it's gonna read an integer from the user and we'll go through all this stuff in more detail. Don't worry; this is jus a high-level overview so you get some basic idea of what Java looks like.

It asks the user for N1, right, because you have to be very scientific, so rather than give me the first number, you say enter N1, right, and suddenly we're much more formal.

Enter N2, right, and so what this is actually doing it's asking the user for a number and whatever the number the user gives us, we stick in this location called N1.

And whatever the number the user give us here we're gonna stick in a location called N2 and we'll actually talk about what these locations are and how they get set up and everything next time, but I just want you to see something before we can kind of go into all the details because we gotta start somewhere.

Then we're gonna add N1 and N2 together and store them in a place called total and then we're gonna write out the total is, whatever the total is, and then a period. Right, so this is in some sense the world's most expensive calculator that adds two numbers.

And we'll go ahead and run it, add two integers. We're compiling; we're feeling good and that's the program we want to run, and now this comes along and notice, now we have some window that rather than putting graphics all over it and telling it where to display some words like Hello World, we just said, "Write some line," and so what it's gonna do is just sequentially write lines on the screen because this is what we think of as a console.

All a console really is, the way you can think about it is it's a window that contains texts and can potentially interact with the user, so there's places where we might ask the user for input. So we said if this program adds two numbers, I hope you can see this in the back. It's tiny, tiny font. Enter N1 and it's sitting there with a blinking cursor, and any time you see the blinking cursor, that means, "Hey, I'm expecting some value." So give me something of value. What's some integer?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Twelve; someone said 12; someone said one, so I'll take 12 as N1 and then it asks for N2. I'll give it one and low and behold, it says the total is 13 period, end of program, okay? At this point, there's a blinking cursor, but the program is done. But now you've just seen a real Java program that actually takes in values from the user, computes some, does some computation on them and displays output, okay?

And this is a console program as opposed to a graphics program because all the stuff we're doing here is textural. Okay, so any questions about the basic idea of classes or objects or these different programs we're writing? The programs that are running, when they run, they are being, they're objects. They're instances that are being run for you. What they're doing as instances, they're objects that are inheriting all the behavior from the class template and then when we run, an instance is actually created in this run. Uh-huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Because if you say acm.star, strangely enough, you won't necessarily get everything underneath ACM as well. You'll get everything at that level.

So there's actually a package for ACM and there's a package ACM graphics, and there's a package ACM program and what you really want is everything inside ACM program and ACM graphics, not just the things that are in ACM graphics by itself.

Yeah, it's an interesting technical question, and you don't need to worry about the details. Just add both of them and you'll be fine.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**No, the console program you just print out. That's one of the differences with the graphics [Inaudible]. With the graphics program, you have to tell it where stuff goes. Console program is just writing out line by line. Uh-huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Sorry.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Yeah, every time you run a Karel program, you can think of what it's doing is creating an instance of Karel which is that little guy you see on the screen and he's going around and doing stuff, so in Karel there was only one instance of the object.

Here you'll actually see we'll create some objects where we can have multiple incidents of an object in the same roll and I'll show you some examples of that in just a second, but we just haven't gotten there yet, okay?

So what we're gonna do now, is we're gonna think a little bit about graphics that we can actually draw. Because the graphics world it's easiest to see multiple objects. We're gonna put multiple stuff up inside our little graphics canvas.

So what's this whole graphics thing about? Let me tell you a little bit about the graphics window and then we'll put a couple different objects up in the graphics window and you can see what I'm talking about. So in terms of graphics, all graphics that we're gonna do are modeled called a collage model and if you remember in the days of yore when you were a week tyke, did anyone have like a little felt board with little felt animals that you put on it?

Yeah, that was a collage, or if you had a big piece of paper and you had some like crazy, not crazy glue, what was that, glue stick. You don't smell that stuff too much – like, but you would take some piece of paper and you're like, "Oh, here's a picture of a cat," and you would like put some glue on the back of it and put it up on there, and then you'd say, "Oh, here's a picture of a dog," and put it up on there. That's the model for graphics in Java.

We're gonna have these little pictures or little objects and we're gonna say, "Show up here and show up over there." So what you have is a collage which starts off as a blank canvas and you're going to put objects onto that canvas, okay?

And so what we can do is create various kinds of shapes or pieces of texts and put them up in various places on the canvas and you just saw an example of that with Hello World, but I'll show you some of the other things that you can create. Two different kinds of objects that exist in the graphics world, one you just saw. It's called a GLabel for a graphic label. This is just text in some sense.

It's text that you can put up in the world. There's also a GRect, which is a rectangle, oddly enough, a GOval, which is an oval or if you actually make the height and width the same, it becomes a circle, right. So that's why we don't have circle; we have oval and same thing with rect versus square because all squares are rectangle, and GLine, which just means a graphic line.

So these are the different kinds of objects we can have. These are classes. They are templates for objects. We are going to create particular instances of these objects and put them up in the collage, but these just tell us the kinds of things we can have in the general class.

So if we come back, if we close this guy, and we come back over here to fund the below program, what we've done is we've said, "I want a GLabel. GLabel is the class. When I say new GLabel, what that actually does is says, "Give me a new instance of that class. Give me an object that is a GLabel," and the things that you're going to specify about that particular instance or it has some text associated with it, which is Hello World.

We've put all the texts inside the double quotes, okay. So Hello World without the double quotes is actually the text associated with this label and we give it some location, 100,75.

So now this little object, okay, which is an instance of a label says, "Hey, I'm one particular label in the world and the kind of label I am, I say Hello World, and I'm at location 100,75," and it's like, "Okay you have this label, what are you gonna do with it."

And I'm like, "Put it on the canvas," and the way you put it on the canvas is you say, add. So all graphics programs, when you say add, there is something that you will specify to add, and the things that you were specifying to add are instances. They are individual objects that you're going to put on the canvas and the way you get those individual objects is you create a new version of some particular class, okay?

So this creates that new label, location 175 and when it gets added, it gets put up there. Now, there is other kinds of things we could do with this, so if we're kind of return over here, this is just the add-to integers program, let's do add-to integers real quickly – 17, 25, 42, writes up same thing you just saw except now in excruciating detail.

Let's say we want to do something that makes a GLabel more funky than we originally had, okay? So again, we're gonna have our Hello Program that's gonna extend the graphics program. Then we're gonna do things slightly differently. What we're gonna do here is we're going to create a new instance of a GLabel object.

So if we say new GLabel and again, we're gonna name it Hello World and 100,75, so I have some label that's Hello World and location 100,75, and I say, "Hey, you know what label, I want you to look different than you actually are, okay?"

So now what do I have this object, I have this little label, you can think of it as a little object that sits somewhere in the world that's not being displayed yet because I haven't added it yet to my canvas. It's being stored somewhere which I have named label, okay, and we'll talk about this notion of GLabel, label next time, but it's something I've created called Label which is just a new instance of this Hello World label.

The first thing I want to say is, "Hey, you know, I want to change what the font looks like." Anyone know what font is. All right, you've probably used different fonts in your word processors. It's just the way characters actually appear on the screen. It's a thing that comes from type setting.

I want the font to be san serif 36, so what happens when I start off, is I say, "Create for me a new label that's name is Hello World," and so it says, "Okay, I have some label object, the label with the words associated with that label Hello World."

Notice there's still nothing on the screen. Here's my screen. Here's my canvas. This is all in the computer's memory somewhere. It's saying, "Yeah, you've created some label. You haven't displayed it yet. You've just created it," and you say, "Hey I want to tell that label to change its font to sans serif 36."

The way I do that is I have an object that I've created. The object is named label. I use the name of the object followed by a dot, followed by a method that I'm going to call on that object, remember in Karel's world, we had methods. The way we called Karel's methods was we just gave the name of the commands to go execute that method.

Now, all methods are associated with objects, at least for the time being. So when I have label and I say, "Set font," what that actually means is set the font for this particular label object to be sans serif 36, and so when I execute that, what happens is it becomes 36 point font in sans serif which means serifs are just little things of if you can see the font up there has you know, swishes at the end of letters, those are serifs, so a font like this doesn't have any of the squishes at the end of the letter. San serif, sans means not, Okay?

Then we say, "You know what, I'd really like you to be red because black is just so blasé and red is really the new black, so set your color to be the red color." Like, okay, and so I execute that and it turns itself red. There's still nothing on the screen.

This is all like of with the label and now that I've taken this label and I've taken this text and made it big and turned it red I finally say, "Hey, graphics program, add this label, add this object to your collage," and it says, "Okay," and it adds it at the location at which this label was created which was 100,75.

So then it finally shows up on the screen. It only shows up on the screen when I actually do the add. Up to that point, all I've done is modify some of the properties of this thing, okay.

So starting next time, what we're gonna actually get into is thinking about what is this GLabel thing about, what does it mean when we actually specify a method for an object and go into all that in a little bit more detail, but hopefully seeing an example of what your first Java program looks like.

All right, I'll see you all on Wednesday.

[End of Audio]

Duration: 51 minutes

## Programming Methodology-Lecture05

**Instructor (Mehran Sahami):**All righty. Let's go ahead and get started. A couple of quick announcements before we start today – so hopefully you're all busy working away on Karel and life is good. Just quick poll – how many people have actually finished Karel already? Oh, yeah. I won't ask how many people have not yet downloaded Eclipse. There are no handouts today. Getting' a little breather – no handouts. Don't worry; you'll get some more of that next time.

Sections start this week, so hopefully you all should have gotten an email about your section assignment and who your section leader is, so you can actually do Assignment No. 1, the email portion. You should have been able to do the programming portion the whole time. But make sure to go to section this week.

And the other thing is the Tresidder Layer, which every once in awhile you've heard me refer to. This is a computer cluster up in Tresidder. Is staffed by one of six helpers like almost continuously around the clock or most of the times at reasonable times when people are working.

So Sunday through Thursday, every day except Friday and Saturday because contrary to popular opinion, computer science people actually do have lives or we actually like to pretend we have lives, but every day from Sunday through Thursday 6:00 p.m. to midnight there will be a staff of helpers on there and actually some of the times there is like two or three or four people there.

And they're there just dedicated for the 106 classes. They're not like general consultants. They're just there to help you work out problems in this class, and they know like what assignments you're working on, the whole deal. They're all like, your section leaders and they're all be exceptionally trained to do this.

The other thing that's going on, hopefully you should be doing Assignment No. 1. I've actually gotten a bunch of Assignment No. 1 emails that have already come in. In the early days, when I got the first few, I actually tried to respond to them all, but then at some point, I just woke up and I like, you know, went to my computer and was like, "Oh, you've got mail."

And it just [inaudible]. So I couldn't respond to everyone individually. I apologize if I don't respond to you individually, but I do read them all. I guarantee you that I actually read them all and I look at backgrounds. And just to prove to you that I do, here's some interesting ones that have come in so far – just to share three.

So there's someone actually spent their time in Taiwan living in a Buddhist monastery, which I thought was interesting, except for the fact they were actually living there as a monk whom I thought was pretty interesting.

Someone else used a – I wasn't quite sure on this concept, but maybe I can provide a clarification. There was a vegetarian who only eats low-quality meat, and so he mentioned that as things like burgers and not steaks. And I would qualify that by saying that's not a vegetarian; that's called being a grad student.

And last, but not least, there was actually someone in here who's on the Colbert Report, which I thought was actually pretty interesting. I don't know in what context, but come talk to me afterwards.

So with that said, any questions about anything before we start? Today we're actually gonna go over some of the graphic stuff you saw, talk a little bit more about objects and classes and get into variables and values and all kinds of goodies. Any questions?

All righty, then let's just dive right in. So one of the main topics for today is this thing called a variable. And a variable, you know, like variables come up in mathematics and it's like, oh X and Y are variables, right and there are these things and we do all these manipulations on variables.

In the computer science world, they're really friendly, right, and you don't have to worry about integration or differentiation or you know, those kind of variables. Variables are kind of your friend, and basically all the variable is in the computer science worlds is it's a box. It's a box where we stick stuff and the stuff we stick into that box can change.

That's why we call it a variable because it's a box that has a variable contents, and you think back you know, in the days of yore in math, and you know, oh yeah, it's kind of like X can have different values, yeah, it's basically just like that.

So in computer science, what we think of as a variable, is each variable has three things associated with it. It's got some name, and that's just how we refer to that particular box. It has a type, which is something a little bit different in mathematics, but the type basically says what kind of thing does this box store? Some boxes store numbers; some boxes store letters; some boxes will store other things, like little objects in the world. But a type is just what's stored in that box.

And then there's a value, and the value, as you can imagine is just what's in the box. What is the actual thing that's in there, right? If it stores the number, then it might store, for example, the Value 3 and that's just the value, and it may have some name associated with it.

And how do we actually name these? There's actually a rule and it's not a very complicated rule, but a very simple rule you need to remember for what are valid names for variables in Java. So a valid name, so this is how you actually name these puppies, has to start with a letter or an underscore. So it starts with a letter like one of the alphabetic letters and can be upper or lower case or the underscore character.

Okay, and that's kind of you know, underscore. It's down at the bottom of the line, okay? And then after you have that initial letter or underscore, then after that, you can have any number of letters, numbers, that's like the number digits, like 1, 2, 3, 4, you know, 0, etc. or underscores, okay?

So you can't start with a number. You have to start with a letter or underscore, but for most purposes in this class, just think about them as letters. When not actually using underscores, you might occasionally use numbers. You can actually have numbers after the first letter.

There is one slight caveat to this rule which you can't have any variables that's name is the same as some which known as a reserved word in Java, which means its name can't be the same as some special word in Java, like the word class is a special name in Java, and there's actually a page in your book, I think in Chapter 2, that lists all the special names. It's like out of the English language it has about 127,000 words. I think there is like 40 in Java that you can't make a variable name.

Okay, you have lots of other choices. As a matter of fact, there's lots of things that don't have to be valid words in English. They can just be any name that follows this rule. The important thing to think about in terms of a name and this is one of the good software engineering principles, is make your name descriptive. If you have a program that's maintaining for example, the balance in the bank account, a real good name for the place where you store the value of the balance would be something like "Balance."

A real bad name would be something like "A," because no one knows what A is. It's like hey it's A – yeah, I know what A is. And someone says, "Yeah, A is balance," and someone says, "No, no, no, in my program, A is actually how many miles I bicycle today," and you're like, "No, no, no, man, A was balanced." Well, if it's called Balance, there's just no ambiguity, so give them descriptive names.

So that's kind of a name part of this. The next thing is what is this type all about. What are the different types that you can actually have. And there are some things that we refer to as primitive types. These are the types that have the smaller developed brains and use knuckles drag on the ground. No, there just the types that are built into Java, okay and some of the basic types we have is something we call an INT, which is short for an integer, but we actually write INT, so INT, I-N-T, is the name of the type, okay.

And this is just an integer of value. It's just gonna store some whole number basically. It stores a number between minus 2 billion and plus 2 billion, but for all intents and purposes, you can just imagine you can store any integer in there, okay?

There's also besides integers what other kinds of numeric data do we have? People already know, it's like I would think it would be like real values, but a bunch of people are already saying double because you've read ahead and you've done the assignment the way you should. And that was kind of a mini social, but both went to the same person.

There's this thing called a double and a double is actually some real valued numeric value, right. It's something like 2.3 is a double or even 2.0 can be a double, okay? Why is this thing called a double – anyone know, as opposed to like a real? Uh-huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** Yeah, there's this wonderful verb people call the I triple E, which is like the Institute for Electrical and Electronic Engineers and they come up with all these standards for things. Is anyone a member of I Triple E here? No one? Oh, man, join and pay your dues. It's a good time. I'm not actually a member myself.

But what those folks actually do is they come up with standards for things and one of the standards they came up with is how you represent numbers that are real valued numbers inside a computer, right, because remember a computer only understands ones and zeros, so how do you actually represent a real valued number.

And so there's a standard, and part of that standard has to do with a precision of the number, how many digits in some sense and a rough approximation you store and double stands for a double precision real number.

And so for the purpose of this class, all real valued numbers that we're gonna use are just of type double. Okay, there's a couple of other types that I'll just mention now and we'll actually go into them in much more detail in a couple of future classes.

One is called Boolean. And if you've ever heard of Boolean logic, this is a logical value, so this is just essentially true or false, and we'll talk about that in excruciating detail next time, but I'll just let you know that there's a type called Boolean.

There's also a type called care or Car and as you can imagine it's because we like sort of the first syllable of most things and this is the first syllable of character, okay, and so we'll also talk about this in a couple of weeks time when we actually get into a some things with characters, but that's just a character.

It's a variable; it's a box, but still is a character. It's a box that stores an integer, box that stores a real value, box that stores true or false and a box that stores a character. Those would be the different types of them.

So one thing people – so that's kind of types, at least some of the basic types. And then when you think about value that we actually store in this box, people always get uncomfortable when they see INTs and they see double, right, and they sort of say, "But Meron, like 2 is an integer, right?" And I'm like, "Yeah." And they're like, "But you just told me 2.0 is a double, right?" And I'm like, "Yeah, 2.0 is a double."

And so the natural question is why? Why do we have both these things? How come like all integers aren't subsumed by the real values, right, you're kind of the mathematician type and you're like, "Yeah, there's like strictly more of these than there are of these. So

why are all these like subsumed in here. Why do we have this integer type,” okay? And the reason we do boils down to a simple question and the simple question you want to ask yourself is how much versus how many, okay?

So if you ask someone, let’s say you, ask me, just so I won’t embarrass you, “How much do you weigh, Meron?”

**Student:** How much do you weigh, Meron?”

**Instructor (Mehran Sahami):** I weigh about, you know, 155.632 pounds, okay? That makes perfect sense, right. If I can put a decimal point, I can put as many numbers after it, or I could just say 156, right and those are both valid kinds of things.

Now you could ask me, “Hey Meron, how much children do you have?” I have 2.3 children. Does that make any sense to you? Yeah, it’s like we had three until that grisly accident. No, it’s just that there’s sometimes – I know that’s horrible to say. We actually have one, and he’s just fine.

There’s sometimes in the world when you care about counting, and when you care about counting, it doesn’t make sense to have fractional values. Those things are integers. They’re a how many kind of value. When you’re thinking about how much, that’s a double value, and you actually want to keep these distinct because if I ask you what’s the next number after 1, you say –

**Student:** 2.

**Instructor (Mehran Sahami):** 2 because you’re thinking integer, and that’s perfectly right and if I say what’s the next number after 1.0, you say –

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah, you start mumbling and you’re like, “Well, I know it’s not 2.0. That would have been the obvious answer and that’s probably not it. How about 1.0000001, and I’m like, “No you missed a couple of zeros in there. Just keep going. Wait until the end of the quarter and put that 1 and come back and talk to me.”

In real values, there is no next value, okay, so when you care about counting, it doesn’t make sense to use a double. That’s when you want to use an integer, so having these things be a distinct type actually make sense and you’ll kind of see that as we go along, okay.

So the value is basically just what you’re actually gonna put into this box over here and it’s gonna be some of the values that we actually talked about, so let me show you some examples of how we might use variables, what the syntax for variables actually is in Java.

So what we need to do with variables before we use them in Java is we need to declare a variable. It's kind of like declaring your love. If you're gonna have some box somewhere that's got something stored in it, you need to come tell the world, "Hey world, I've got this thing called X. It's cool. Check it out." Here's how you do that in Java.

You say INT X, okay? What that does is you specify a type for your variable, what's the box gonna hold. You specify a name for that variable. In this case, the name is X and because all statements end with a semi colon, just like you saw in Karel's world, we put a semi colon.

What does this do? It creates a box somewhere in the computer's memory that's name is X that's gonna store an integer in it. Right now, what does it store?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Nothing. We haven't told it anything to store in particular. And so in Java's world, unless you initialize you always want to think about initializing a variable, giving it some initial value to start off with and there's some rules that we'll talk about as we go along regarding when Java will automatically sort of give you a zero value in there and there's time when it doesn't and sometimes it can get a little bit confusing.

So the general rule you want to remember and it's also good software engineering is give variables an initial value when it makes sense, okay? So how might we give it an initial value? Well, on that same line, rather than just putting the semi colon there, we could say X equals 3 and what that does is it gives it an initial value of 3 to x.

Okay, so you could actually have the declaration of the variable without the initial value, but you always need to give it an initial value before you use it, so a lot of times we just give it the initial value when we actually declare it, okay.

If we want it to have something of type double, maybe something called Y, we could have something called Double Y equals 5.2 and what that means is somewhere in memory we have this thing called Y, which has value 5.2 in it, and it's just a real value.

So the general form for how you do these declarations is you say type and I'll put little squiggles underneath these just to let you know that this is the general form, the name and then potentially an equal and some initial value, followed by the semi colon, and that's how you do the declaration.

And so the question that comes up where do you actually make these declarations and the place you generally at least for the time being make the declarations of value, for declarations of variables is inside methods.

So when you create some method like you have public, void, run and you're gonna put everything that's gonna go inside run in there, you could, for example, have this line in

there and this would create X with a value of 3 and Y with a value of 5.2 and you wouldn't have the general form there, but for example, you could have those declarations inside run and what that means is you have those variables created with some values available to you inside this method called run.

They're only available to you inside that method called run. They're not available to you in some other method and we'll talk about how you get values passed around between different methods in a few days, but for the time being when you have your variables they're only available in that method. Okay, so any questions about that – variables or values? Uh-huh?

**Student:** If you have them inside the public run method [inaudible], will that method be able to take that variable?

**Instructor (Mehran Sahami):** Well, you won't be defining another method inside run, right; you'll be defining another method that's separate from run. It's just a separate method. It won't be available in that separate method. It doesn't matter if it's public or private. It just not available there. So this public or private thing that you have here doesn't affect the visibility of your variables. Your variables are always only visible inside the method in which they're declared and set up, okay?

Now, one other thing that we did when we sort of, you know, did this equals thing over here, if we wanted to, we could have actually done this in two lines rather than setting up X initial value there, we could have said into X and then we could have set X equals 3 over here and gotten the same effect.

And when we say X equals 3, this equals is not equals in mathematics; it's actually what's known as an assignment statement in Java or in most programming languages. And the idea is we're taking some value over here. That's what's on the right-hand side of the equals and assigning it to whatever variable is listed over on the left-hand side of the equals, so we're assigning that value, okay?

And so the general form for an assignment, let's just do that over here, is pretty straightforward, you just saw it, is variable and this is the name of the variable. So I'll squiggle – equals some value or some expression for a value and we'll talk about expressions in just a second with a semi colon at the end of it, and that's how you assign a value.

You can assign values multiple times to a variable, to a variable. It's a variable, the contents change. So you say, "Hey, X equals ,," and then down here somewhere, "Hey X equals 4," and when you get to this line up until you get to that line from here to here, X will have had the value 3 and when it gets to this line, it'll stick a 4 in the box. And that's perfectly fine because X is a variable. As long as everything you stick in there is of the type of which X is declared, you're okay, okay?

So what that also means is when you're doing Assignment you can do things that are perfectly fine in programming, but to mathematicians, they just go nuts; they go crazy like veins just burst in your head and you hear these popping noises and people dying in the streets, which is you can say, "Hey, I have some variable called total. Let me create some variable in its total," which I'll give some initial value like 10 and that's a good time.

And then I say, "Hey total equals total plus 1." And if you're a mathematician, you look at that and say, "No, man. Total can't equal total plus 1. That's just not right." And then you got this whole philosophical question about, you know, if this is like total is equal to infinity and you're just like, "No, no, no, that's not what we're talking about."

We're talking about assignment. This is into equals, this is an assignment. So what it says is evaluate the right-hand side, okay? Total – what was total's previous value. Go and look it up in the box over here you have some box for total. Its value was 10. It looks it up. It says, "Oh, that was 10." I'm gonna add 1 to it, that gives me 11. What do I do with that 11? Stick it back in the box named Total. So it just says okay, and it puts an 11 over here and that's perfectly fine.

So it's perfectly fine to do something like then where you take the old value for a variable, you do some manipulation on it, you stick it back in the same variable. Okay? So with that said, now that you know about variables, we can go back to a couple of the programs that you saw last time and things will now begin to make a little bit more sense, so if you come to the computer, remember our little friend, add two integers that you saw last time.

Suddenly the world opens up a little bit and we say, yeah, add two integers is a console program, right. So it's gonna write some text out to a console somewhere and we have our run method. We print out a line; we know that the method print [inaudible] when we say that, whatever we put inside the double quotes gets printed to the screen so it prints out this program adds two numbers, so we execute that line and after we execute it, it writes it out and it comes to the next line.

Low and behold, what have we done here? We've declared a variable named N1. What's the value for that variable that we're gonna assign to it? We're going to call some method. The method we're gonna call is something called Read INT and this is a method that's provided for you in console programs. All console programs can do this. The way it works is you give it some text between two double quotes. It writes that text on the screen just like you saw last time and asks for input from the user.

Whatever value the user types in and hits enter, that number is the value that gets what we call return by read INT. It's the value that [inaudible]. It's kind of giving it back to you. Do what you want. What are you gonna do with it? You're gonna assign it to the Box N1, so here's the box for N1.

We execute this statement. It's coming; it's gonna print to enter N1 on the screen. It's gonna ask us for a value. Let's say we type in the value 17 and it sticks that 17 so this expression over here evaluates the 17. It sticks 17 in the box for N1, goes to the next line, now it declares N2. We have some box. Its initial value also is gonna come from the user via this read INT method.

So we call read INT. It asks us for another value; we type in 25, and it sticks that in the box. Now, we're gonna declare another; we just declare until the cows come home, right? Declaring is fun; it's cheap. It's easy; do it, do it often, right, so we're gonna declare another variable total, so we have another box over here for total and the value of total that 's gonna get assigned there is just whatever value is in N1 plus N2.

So to evaluate this side, it says I need to look up the box N1, so at the point where it reaches this line, it evaluates whatever is in N1. This isn't some truth that holds for all time. It's just an assignment that it does when it reaches that line. So it says, find out what's in N1, add it to what's in N2.

That'll give you the value 42; stick that in Total, which is what you get. And the last line says, I'm gonna print something to the screen where the total is and you see these funky plusses and you're like, "Whoa, whoa, whoa Meron, I thought plus was light lading things together. What's going on? You're like trying to add – this thing's got the value of 42. I know that because it's in the box. You're trying to add 42 to some text? That's not right."

Well, in most worlds, it would not be right, but it turns out in the Java world it's perfectly fine and when it sees something that some words with a plus sign, this plus sign here is not numerical addition anymore. This is essentially you can think of as a concatenation operator. It says, "Take whatever text is here and add to that text, the textural form of this value."

Well, the textural form of 42 is just the Characters 4 and 2, so it sort of adds that 42 here and then it adds to the end of that a little period, so it concatenates all those things together and then prints them out and says the totals 42.

So any questions about the program? If you're feeling good, like you understand what just went on in the program, nod your head. Rock on. That's what I like to see. All right. So with that said, we can go back to something else we did last time, right, so now you know how all this variables and assignments and types and you're like, "Hey, Meron, yeah, that's all good and well, but last time you were telling me about like classes and objects and drawing stuff on the screen like, what's up with all that."

And this is the time when we actually do sort of the, you know, Reese's Peanut Butter Cup. Like it's two great tastes that taste great together. We take some objects over here and some variables over here and we just slam them together, and now you're going to have variables that contain objects. They taste great, all right?

So here are some of the objects that you considered last time, right? We had things called, like some classes. You have the G label class and the G –rect class and the G-oval class. These are all these graphics classes like a label [inaudible] or rectangle or an oval. I'll show you examples of all these things this time.

And these guys were actually all classes that are part of a hierarchy of other classes and the other classes, they're all part of is G object? So what G-object is a G-object is a class. Don't let the name fool you. Some people see G-object and they think, "Oh, that must be an object." No, G-object is a class. It just means a graphics object. And guess what, a rectangle or an oval or a line or a label or all different graphics objects. So what that actually looks like in terms of the hierarchy just like you saw last time, is the classes are represented, kind of like this in the hierarchy, all of these classes are actually G-objects. Okay, they're all sub classes of G-objects. It means they're specializations. So any behavior that G-object has, all of these things have, but they might have their own specialized behavior, as well.

Now, the interesting thing about all this is that when you have classes and you create objects, you can store those objects in a variable. You can say, "Give me a variable that is a box," and the type that that box holds is an object that's the type of some class, so all classes can actually be used as type means, okay?

Now, that's kind of funky. Let me show you an example, okay? Here is another program you saw last time. Now, we can just kind of reveal the mystery about variables. We have this program, Hello program that extends the graphics program, right, so it's gonna draw us some pictures.

Well what are we actually doing on this first line? Hey, we're doing a declaration and assignments on the first line. What's the type of our variable? It's a G-label. So we use the class name as the type. The name is label, so we have a box called label and what that box is gonna hold is any object of the class G-label. Okay, so that's the type. It's gonna hold any object.

And the question comes up, how do we get an object of type G label. And here's the funky part. The way we get an object of type G label is we have to ask for a new G-label.

So we give the name of the class here and class is depending on the class, you know, see some examples of this will take in or give in some initial values for what we refer to as parameters, and these parameters are all separated by commas. But at some initial values, that this object uses to sort of set itself up. It initializes the objects.

So objects rather than just taking one value, can potentially hold multiple values in them. In this case, the G-label is gonna have Hello World as some text that it's gonna display in some location that's 100,75.

So we say, "Hey G-label, you're the class. I don't have an object of you yet. You're the class." When I say, "Give me a new G label, it's sort of like going to the factory. You sort

of show up at the G-label factory and you say, “Yo, G-label factory. Give me a new object,” and G-label factory is sort of sitting up there and says, “Well, what kind of new object do you want? Yeah, I can give you a new G-label, but how is that initial G label going to look,” and you’re providing these parameters that sort of specify what the initial G-label looks like.

So this line is now created a G-label, okay? And it has some initial, so when we execute that line, it has some initial text that it’s gonna display and somewhere it’s also storing the value is 175 because it knows where that’s gonna be on the screen.

Now, what we’re gonna do is we’re going to tell that object to do some additional stuff. The way we tell an object to do something is we specify the name of the variable for the object, okay. So the object name is label. That’s the name of the variable. And so that’s what we say here. We don’t say G-label. We say the name of the variable label.

Then we have a dot; then we have the name of the method that we’re going to call for that particular object. So it’s a little bit different than Karel’s world, all right. In Karel’s world, every time you call the method, Karel was the one executing that particular method.

Here when you have multiple objects, you need to specify which object you want to execute this method. So you say, “Hey, label, yeah, I’m referring to you, buddy. Set your fonts to be this particular font, san serif which you saw last time.” It just makes it big. It makes it 36-point font. And then you say, “Hey label, yeah, you, same guy, or gal or it or whatever it may be. Set your color to be red.”

And it says, “Okay, I’ll set my color to be red, and then finally if I want to take this label and stick it up onto my graphics canvas which is actually the program I’m looking at. Right, I’ve done all this stuff and my program’s still bringing out a blank screen. It’s like here you go, nothing going on here. I say add and the name of the object that I want to add, right because now I’m holding this little object called label.”

This is my little felt Hello World, and I want to say, “Hey take the felt hello world that’s so cute and just stick it in the world,” so we add it to the graphics program, basically and it shows up in the screen where it knows it’s supposed to show up at a 100, 75.

So the general form for this, the people will refer to when you see something like this, you have name of the variable and the method name and the Java-esque names that we give to these, just so we hear other programmers refer to them, you know what they’re talking about, is they say the object is the receiver. So the object here is the one who’s receiving or being told to do something, okay?

So the name of the variable is the receiver and the method that’s being called is called the message and so there’s this whole notion of sending messages, kind of like you could think of programming as I am, right, and in your little I am you have like your friend, Bob, and your friend, Sally and your friend, Dave, and your friend, Label, right. And

Label is just sitting there in I am and Label is not very talkative and once in awhile you can send a message to I am which is a form of message that label understands and then Label will do something.

So this is called the message that we send to the receiver, is kind of the nomenclature. But a lot of times, you'll just hear people say, I made a method call on the object and it means exactly the same thing. Again, we like to give funky names to simple concepts. Okay, so any questions about that, uh-huh?

**Student:** If you add the label first, then you start [inaudible].

**Instructor (Mehran Sahami):** Yeah, you can still, after you sort of put the little object up there that says, "Oh, the object's in the world, and say, "Object change your color to red," and that's where it's a lot cooler than the felt animals you had when you were a kid, because it'll just change to red right there on the screen because once it's been added, it's up on your canvas, and now any messages that you send to it will actually change its appearance on the canvas. Okay.

So with that said, let's kind of go through and understand a little bit more about what are some of these messages or methods that we can send to particular kinds of objects. So to do that, first let's talk about the graphics world that we live in. There's a couple of things you want to think about the graphics world.

How do we specify all these things like locations and how is it laid out. Well the origins, the upper left, so the upper left corner of your graphics window is location 00. Every thing in your graphics window is measured in pixels.

Pixels are the little dots on your screen, so if you get out a magnifying glass and you go up to your monitor and you look real close and you get like eye strain and then you want to sue, don't worry. It's just fine. At least you saw a pixel and it brought you al little closer to Nirvana. So a pixel is just a little squares essentially on your screen that could actually get displayed.

All the numbers that we refer to are in terms of pixels. So when we think about X and Y coordinates, X is how far you go across, sort of across the screen and Y is how far you go down. The X coordinates just increase to the right from 00 and the Y coordinates, different than what you're used to in mathematics, Y coordinates go down. So normally, we think of this as a negative Y direction. Not so in Java's world. This is the positive Y direction going down.

Hey that's just the way life is. It's been like that since I was a wee tike, or actually, since Java came around. Okay, and so you might wonder, "Hey, if I have some label, like Hello World, and I specify some location, where is that really on this Hello World thing and so the G-label coordinates, the coordinates that 100,75 that we gave it were the X and Y location of the base line of the first character. So that first pixel right there on the lower

left hand corner of the H, okay? And so that's 175. That's how things were laid out. Uh-huh

**Student:** Is it possible to specify [inaudible] using ratios?

**Instructor (Mehran Sahami):** Well, everything's in terms of pixels. There's some stuff you'll see later on in the class where we actually specify some things like in polar coordinates if you're familiar with polar coordinates, you'll get to that later on.

But you want to think of these numbers as just being absolute pixels and you can do some math to figure out where those pixels should actually be. But they're still just pixels, just way off.

All right, so remember all of these classes are organized in some hierarchy and everything that we talked about, right, G-label, G rect, G-oval and G-line are all G-objects, which means any behavior that a G-object has, all of these puppies have. So let's start off by thinking about what are the behaviors or methods that are available on G-objects.

So one is set color; you just saw an example of that, right, you specified the object, you say set color and you give it some color and I'll show you where the colors come from and it sets the color of the specified object to the color you just specified as this thing called a parameter. So the thing that's inside the parentheses in our formal speak, we refer to as parameter. So methods have parameters. They're the values that you provide inside the parentheses when you invoke a particular method, okay?

So there's a single parameter there, color. There's also something there called set location, which takes an X and Y coordinates. Again X and Y are in pixels and it sets the location of that object to that XY, so this method has two parameters, X and Y. And you might say, "But Meron, when I created the G-label before, didn't I specify the X and Y location when I told the factory to give me one of these G-label objects?"

Yeah, you did, but you didn't have to, and it turns out there's times you don't want to; you want to just say, "Hey give me a label that's got the words Hello World in it and later on I'll figure out where it's actually gonna go on the screen, so I'll specify what the X and Y are," but until you specify the X and Y, it doesn't know, but you can specify after you've created the object, is the important thing.

And move, and here's the funky thing. You can actually tell an object, like you have your little, you know, Hello World somewhere on the screen and it's kind of like the furniture idea of objects. You're like, "Yeah, I don't like how it looks there. Move it a little to the right. And so there's a move method and its coordinates if you're sort of a Calculus person or DX and DY, and if you're not a Calculus person, don't worry, you don't need to be a Calculus. This is the closest we get to Calculus in this class. Rock on.

DX and DY just means how many pixels in the X direction and the Y direction should you move this object from its previous location and these can be positive or negative, because you can actually have something negative say in the Y direction and it'll move it up on the screen.

So it's just the offset or how much you want to, so think of it as the difference in X and the difference in Y, how much do you want to change where it moved. Uh-huh, question?

**Student:**Do you use a set location XY and then [inaudible]?

**Instructor (Mehran Sahami):**Can you do set location and so you –

**Student:**[Inaudible] later define X and Y.

**Instructor (Mehran Sahami):**No, if you want to say set location XY, you're referring to two variables X and Y, so there's variables X and Y and each have already been declared and have values.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Yeah, so anything that's a parameter here, you can actually use a variable rather than an actual value, like an actual number, and we'll see some examples of that as we go along. [Inaudible]. So where do all these colors come from? It turns out these colors come from a standard library that someone in the Java world wrote that's called the Java AWT package, so if you want to use these colors at the top of the program, and I'll show you an example, that should say, importJava.awt.star and these are all the colors, so they're all the name color, dot and then what the color actually is.

So you just saw an example where we used colored dot red to make Hello World red, but there's all these different colors and they're all in your book, so you don't need to scribble them down hurriedly, but you know, different shades of black or grey or white and you know, magenta and sienna if you're sort of a color photography sort of person, but there's a whole set of colors and you can just go to town on them. Okay.

So all G objects, all of these things respond to these methods because these guys are all objects, so any of those three methods will work on any of these objects, of objects of any of these types. But there are some additional behaviors, there are some additional methods that some of the specialized classes, the sub classes actually implement.

So G-label has some additional things that it does beyond just what a G-Object in general would do. And the big one, well first of all, it's got what we refer to as the constructor. You can think of the constructor and I'm gonna do a little violence to the term, but you can think of the constructor as a factory, okay? What the constructor says is, "Hey, I want a new one of these guys," so I use the word new and then I specify the constructor, okay?

The constructor is the name of the class of which you're going to get a new object and then some initial parameters. So the initial perimeters for a G-label for example, could be what you just saw, the text that you wanted to display and its initial X and Y coordinate, okay.

There's some other ones. For example, you can change the fonts, right, it makes sense for a label. It doesn't make sense for a square or rectangle, to say, "Hey, square, you're font's gonna be Helvetica." And the square is like, "I'm four lines, man, I don't have a font. What are you talking about?" That's why the notion of setting a font isn't something that all G-objects have. It's only something that G-labels have which means the G-rect and G-oval and G-line don't have a set font.

This is just something that you can do with a G-label and what it does it says specify the font that you want that label to show up in on the screen, and the general form for specifying the font – this is all in your book and it shows you examples of the fonts, but the general form is inside double quotes, because you're gonna give it as a piece of text, you specify the family of the font.

That would be something like times or Helvetica. The style of the font, which is, you know, plain or bold or italic or bold or italic, so yeah, now in fact you can do all those funky things your word processor does and then the size of your font, how big or small you actually want it to be and you specify that as some texts and that would be what you send over to this sent font method to set the text.

Okay, so any questions about that? Uh-huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**No, you get them all through graphics program. Rock on. All right, so how do we draw some geometrical objects? Have another question?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**If you're doing a graphics program without any colors, you don't need Java.awt.

**Student:**But if we're doing one with color, then we need Java.awt in addition?

**Instructor (Mehran Sahami):**Yeah, in addition to a graphics program, and I'll give you an example of that in just a second. I have to show you a program that has it.

So drawing geometrical objects – it turns out some of these other guys have funky things that you can do specifically with them. So first of all, there's the constructors, the factories from which you can build new things. So how do I make a new rectangle? I say, "Hey, I want a new G-rect." I specify the X and Y location. It's the upper left-hand corner of that rectangle and then I specify the width and height of that rectangle in pixels.

Okay. So that's the basic idea is upper left-hand corner and then width and height of the rectangle.

Similarly, for oval – ovals are kind of funky because you look at this and you're like, "Hey I have an X and Y and I have a height. I didn't like –width and height like ovals. I thought an oval was defined by like two [inaudible] and that whole thing where you have a string. Did you ever do that – you stick the two like nails in the board and you put the string around it and you draw anyone? There's like two people – yeah, like sorry."

Do it; go get a piece of wood, stick two nails in it, put some string around it and go get a pencil, something you can draw ovals and you can draw a 1 and if you're like 6 years old, you'll draw like 1,000, and if you're 18, and draw one, you're like, "Yeah, that was just stupid."

But the basic idea behind an oval is the reason why we specify this way is you specify sort of an imaginary rectangle for the oval and the oval shows up as an oval that just happens to touch the sides of the rectangle, which is how we sort of specify how wide and broad it is.

So think imaginary rectangle and sort of the four ends of the oval would sort of be touching it, and I'll show you that momentarily. G line is very straightforward. G-line, the way we specify line just like you know, the days of yore with Euclid, line is defined by two points, right, that's an infinite line, but we can think of where it ends at the two points. So we specify an initial XY location and an ending XY location and it will just draw a line between those two points, pretty straightforward. Uh-huh?

**Student:** [Inaudible] is the set location defined as the lower left corner of your object, and the G-rectangle [inaudible] left corner?

**Instructor (Mehran Sahami):** Well, for set location is the lower left corner of for textural objects, and then it becomes different for rectangles and ovals and stuff like that, yeah. So it's a slight variation because that's just because we're trying to deal with, you know, texts and rectangles symmetrically and it's tough to do, okay.

There's also some additional methods that are just shared by G rect and G-oval. They don't actually apply to lines or to labels for that sense, rectangles and ovals can be filled, so they can either be an outline. You can either have a rectangle that looks just like this which is an outline or it can be filled in which means it's just a big solid chunk of stuff. So you said if it's filled, you say this is either true or false with the actual word true or false.

So if you've set it to true, then when you draw it on there, you say, "Hey, put this thing up on my canvas." By adding it to the canvas, it will show up filled in; it will show up solid. And if this is set to false, it just shows up as the outline of the object, whether or not that's an oval or a rectangle.

And almost done here. There's also a notion of a set fill color, and you might be wondering, you're like, "But Meron, you told me that set color. Here's where things get a little funky, you told me that set color is something that was defined for a G-object, so if set color is defined over here, doesn't that mean that a G-rect and a G oval already can set that color?"

Yes, that's true. "So what's the set fill color all about?" And this is a slight variation. When you set color for something, you set the color for the whole thing if that's solid or if it's an outline, you're setting the color for the whole thing.

When you set fill color, you are just setting the color for this internal portion. So imagine like this was your rectangle, what color used to fill it in can actually be different than what color the line is. So if you want the line, say, to be black and you want the fill to be red, you can actually do that by setting the fill color to be red and then the outline is still whatever color the actual outline was if you want to do that.

Okay, so if the color – if the interior is fill, so the set fill has to be true, then you can set the fill color to be different. That's a minor thing, but it's kind of fun if you're drawing stuff, okay? So any questions about that? Oh, wait – we have a question over there, uh-huh.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Oh, in the place of this word fill?

**Student:**Yeah.

**Instructor (Mehran Sahami):**It's either the word true or the word false. So if you say true, set filled is true which means it will be a solid thing; it will be filled in. Uh-huh?

**Student:**If your when you're using the [inaudible], you specify a [inaudible], what is the why would you use the set location [inaudible].

**Instructor (Mehran Sahami):**Maybe you wanna do some animation. Say like you have like your bunny in the world and he's gonna want to move along.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Well, it's set somewhere to begin with and then later on, guess what you're actually gonna be doing a program that involves animation. Rock on. And you will move things around. So that's where it comes from. All righty.

So one last thing that might be useful to you, sometimes you want to say, center something in the graphics window or draw something in the bottom, how do you know how big the graphics window actually is.

There's two methods from the graphics program class, so these methods you might notice they don't have any objects, that is their receiver, and the reason for that is that both of these methods, their receiver is the graphics program, so if you don't specify the receiver for a method, the thing that is actually receiving the method is the encapsulating class.

Okay, just like in Karel, when you had Karel and you said move and you didn't specify an object for the move, the move is going into Karel, so get width and get height. We'll return to you the width and the height of the graphics window in pixels.

So these actually give you back a numeric value that you can, for example, assign to a variable and then do manipulations on it. Uh huh.

**Student:**If you don't fill in [inaudible] something like set color to change the entire background of the graphics [inaudible]?

**Instructor (Mehran Sahami):**Well, you can make the huge rectangle the size of the graphics window and set it to be filled and set it to be a color and then you can change the whole background that way. Uh-huh?

**Student:**Can you then change the size of the graphics window?

**Instructor (Mehran Sahami):**There is some stuff for later on, but for right now, just think of a fixed size graphics window. Yeah, so let me push on just a little bit if we can because what I want to actually show you now is a graphics window that makes, or a graphics program that makes use or a bunch of this stuff so over here, here's a little program called fun graphics, okay.

And so we have a comment up at the top and here are all the imports you need. Notice I [inaudible] for you. We need `acm.graphics.star` because we're gonna write a graphics program. We need `acm.program.star` because all programs that you write in this class are gonna need that. And because we're gonna use colors we additionally need `Java.awt.star`. If we weren't gonna use colors, we wouldn't need that, but it's needed for the colors. Uh-huh.

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**It did; it might have just been hidden because sometimes it just appears hidden until you expand it out, but it actually had it in there. So here's what we're gonna do. Let me just run this program, show you what it displays on the screen and then show you how we actually get that display, so it's just like, "Hey Meron, you're just drawing garbage on my screen," no, there's really like a method to this madness. Not really, there are multiple methods to this madness [inaudible].

I know, sometimes computer scientists should not be – that was actually not a canned joke. That just came to me – how horrible is that? So here is what's going on. We drew a bunch of stuff on that screen, how did it get there. Well, hey Hello World; we probably

knew how that got there. This should all be familiar to you now. We got a new G-label. We had some variable called label that we assigned it in. We set its font. We set its color to red. Because we set its color to red, that's why we need www.Java.awt because we're using colors.

We set its color to red and we said, "Hey, I got Hello World, throw it up on the screen," so we add label. You give it the name of the variable and there it appears.

Over here we say, "Hey G-rect. I want a new rectangle that's upper left-hand coordinate is 10,10 and it's size is gonna be 50,50, which means it is a square. It's not filled in. I haven't specified its color right," so if you don't say set fill, it's not automatically filled in. If you don't specify the color to automatically black and then I add it and then you get that little box all the way up in the upper left-hand corner.

Then, I say, "Hey, you know what, I want some other G-rect because it's cooler." It's gonna have colors and it's gonna be bigger and better and it's gonna be nationwide, and so what it's gonna do is I'm gonna have rect 2 which is another variable of Type G-rect.

It's perfectly fine for me to have multiple objects of the same type which they knew G-rect that's upper left-hand coordinates is 300,75 and whose size is honking – it's 200 by 100, so it's big long thing, and it's gonna be filled in and I'm gonna set its fill color to be red, so the whole thing is red, throw it up there and so what I get is that big, rectangle. Just ignore the big oval one in front of it for now; I get this big red rectangle up there.

And I say, "Well, that's kind of cool, but I want to see what this oval thing is all about. So Hey, Meron, make me an oval." And I'm like, "All right, well, G-oval will make you an oval," and the dimensions of the oval, its upper left-hand coordinate and its size are exactly the same as the rectangle, and the reason for doing that is to show you the oval in relation to the rectangle that you specified.

It's set filled is true and it's set fill color, not its set color, but its set fill color is set to be green, which means the internal dividend will be green, the outline of the oval will still be black, because we did not set its color and then we added and there you get that green oval and you can notice its four ends touch the same four ends as this rectangle and if you look real closely there's actually a green line that demarcates the oval and then the middle fill is actually green.

It might be a little hard to see, but that's the case. Then we also have a line, so I will just call this my funky line, and my funky line is a line that starts at 100,150 and it extends to the location 200,175 and there is my funky line. Yeah, it's a funky line.

All right, and then I add another thing and you're like, "Dude where's my line." And so Dude where is my line is a line that should cut across the entire graphics window because it starts at 00, well not the entire graphics window. It starts at 00 in the upper left-hand corner and goes to 100,100, and this is a common error which is why I how it to you.

You look up there and you're like, "But Meron, there's only one line," and that's your funky line or my funky line as the case may be. Where is Dude where is my line? And it's not up there. Why is it not up there?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):** Yeah. You did not add it to the graphics content, okay. Common error – people will go ahead and create something and add all the colors and set all the sides and be like, "Ah, I'm rocking," and then they'll run their program and nothing shows up and they start tearing their hair out and beating their section leader and it's a bad time, and they're sitting there contemplating life in jail and then they realize, "I just should have added it to the canvas, all right."

But you can't give that as an excuse to the judge. All right. So any questions about that? All right. Let me very quickly – I need to push on real quickly. So I want to tell you a little bit about expressions before we finish up and we'll do more expressions, but you need to know some basic stuff on expressions.

So all an expression is is you already saw one. And expression is something like INT total, equals N1 plus N2. This is an expression. It's just some mathematical expression that tells us how things kind of relate to each other.

And basically all an expression is it's a bunch of terms and these terms can be either individual variables or they can be constant values, like they can be like the Number 2 or 3 or they could be some method called like Read INT that actually gets a value from the user, for example, and there's some operators between them, like the addition operator, and so when you take some terms and you put operators between them to combine the terms that's called an expression, all right.

Fairly straightforward stuff – you've probably seen these. There's a bunch of operators you should know about. The operators are things like plus, minus, times, which is actually a star or asterisk, which is how we refer to times. It's not as X – division and then strangely enough, this thing's that looks like percent and it's not the percentage operator, although you might think that. This is what's known as the remainder operator, okay.

So these puppies generally work like you would think, add two numbers, subtract two numbers. There's also a unary minus. The unary minus is the funky mathematical way of saying negative sign. So you could actually say like equals negative N1 plus N2. That's like a negative minus. That's perfectly fine. Okay, or you can use that same thing, you know, as subtraction if it's between two terms.

Multiplication, division, the thing about division that is funky is it depends on how division is used, okay and I'll tell you a little bit more about that next time, but I want to talk about remainder right now.

And remember back in second grade, when you did like short division and you said, “Oh, I had 5 and 2 – how many times does 2 go into 5 – 2 and you were like remainder 1. Anyone remember that? Wasn’t that such good time? I loved remainder – like that’s the online like math. I’m like – and when the remainder was 0, I got pissed.

But that’s what remainder was all about. It’s just what’s left after you do the division. So that’s the way you should think about percentage. It’s the remainder operator, and you’re like, “So how does that work?” So what that means is if I say 7 remainder 2, the value of this is 1 because it’s after I divide 7 by 2 and I see how many whole times 2 goes into 7, what’s the remainder, it’s 1, okay? You could say something funky, like, hey what’s 7 remainder 20? You’re like well, 20 doesn’t go into 7, Meron. So that would be 0 remainder – rock on.

So that’s the way it works. It’s not the modulus operator. If you’re a mathematician, or if you’ve worked with other languages and you’ve seen clusters and you’re like, “Oh, that’s the modular operator.”

It’s not in Java; it’s remainder and if you try to apply it to negative numbers, unexpected things, what’s not unexpected; it’s just not what you would have expected in the mathematical sense. So as far as you see it in the book and everything we do in this class, just apply it to positive integers and you can only apply it to integers so there’s no remainder with valued doubles, there’s no remainder.

And so when we come back next time we’ll talk a little bit about some of these expressions. Any questions before we break? All right, and I’ll see you – oh, there’s a question. Why don’t you just come up and ask it. Thank you.

[End of Audio]

Duration: 52 minutes

## Programming Methodology-Lecture06

**Instructor (Mehran Sahami):**All righty, welcome back. If you haven't turned in the assignment yet and you, at some point want to, turn it into that box up there.

So a couple quick announcements before we start. First of which, first of which is the quarter is already 1/5 of the way over, right? After today it's like two weeks of ten week are over, so that's hard to believe.

But there's three handouts in the back including your next assignment because the fun never stops; when one assignment's due the next assignment goes out, and a couple other handouts. Assignment one, as you know, is due today so please drop it off in the box in front and, congratulations, you'll all – well, assuming you didn't take a late day you're all programmers, right? Because hopefully you all did, Karel, you got him to run around the world and do stuff. Hopefully figured out some interesting out rhythms and now you can turn it in. Question?

**Student:**I didn't know if you needed a hard copy [inaudible].

**Instructor (Mehran Sahami):**Un huh, you want to get a hard copy in as soon as you can, like right after class. But hard copies are important because we want you to turn in both because we use the electric submission to be able to actually run it and your section leader makes comments on the hard copy and so it's important to have both.

But just because you asked...

All righty. So I want to take a quick pain pole before we start. So let's actually dive into the real sort of meaningful things. What the pain pole really is – remember I asked you to think about how much time it actually took you to do the assignment? So total it up over all the Karel problems; how many total hours; think about it, it took you to actually do the assignment. Right? And we're just going to go through and do a quick show of hands.

How many people actually got through the assignment in zero to two hours? All right. Maybe like a couple people. I'll make a small bar. How about two to four? A fair number. Four to six? That's good to see. Six to eight; eight to ten; ten to 12; 12 to 14; 14 to 16; 16 plus? Rock on. Thanks for admitting it. It's a good time. Hopefully it was a good time.

But that's – first of all, one thing to note is the world is surprisingly normal. Right? Like everything in the world is just normally distributed, that's just the way it is.

The second thing is that computer programming, right or software engineering is a pretty high variance event, right that you can go from less than two hours to 16 plus. I don't joke when I say it's actually very high variance. But hopefully what this give you is again, what we shoot for, right, is about ten hours per week outside of class for, you know, work. We shoot for here and as I mentioned, you know, it looks like you're all

doing real well because you're sort of below the average point. The truth of the matter is kind of as the quarter goes on, the assignments tend to get a little bit harder which means you'll actually see this curve kind of move down a little bit more into that range. This is good times to actually see it here right now.

It also hopefully gives you a chance to gage for yourself how you're doing sort of relative to expectations. Right? If you're sort of doing real – you know, hopefully you put in your comments and it was good software engineering and everything and I'm totally willing to believe it was and you just wrote your code and it just all worked and it was beautiful and, sort of, if you're done in this end, as long as you're still feeling like you're understanding the concepts and you're plugging away, that's important and just keep plugging and you will do just fine, trust me.

There have been times when I have been down here myself and it wasn't fun when I was there, but you just keep plugging away and it works out.

And but the important thing is if you were sort of in a particular range, even if you're in this range and things just worked but you didn't understand why, that's more dangerous than being in this range and understanding why because all the concepts in this class will build on top of each other. So make sure you understand the concepts not just that, oh Karel happened to do the right thing. Yes, he got to the right spot in the middle of the world, but now he's just spinning around. That's fine if he's just spinning around, he just got that middle spot, right. We're kind of a lock, like you just throw in enough instructions until I kind of did the right thing; that's not real good understanding and you want to talk to me, talk to your section leader, talk to the TA to try to clear that up.

All right, so with that said we're just going to dive in because there's a question in the back.

**Student:** Is it an honorable violation if you look at someone else's code once you already both handed you assignments in and gotten it back?

**Instructor (Mehran Sahami):** Once you've gotten it back and it's already graded, it's fine to actually be able to look at someone else's code because at that point you can just kind of, you know, share ideas.

All right, any other questions?

All righty, so a couple things to cover real quickly. Last time we talked all about methods and some more about objects. There's two things you should know in the programs that you're going to be doing, is we talked a little bit about one of them last time in terms of how to get input from the user. There're these functions that you should know about.

One is called READ INT and there's some prompt inside double quotes that you give and what that does is ask the user basically for an integer and gives you back some value that you can say, assign to an integer. There's also a version of this to get doubles, which

surprisingly enough is called RE DOUBLE and has exactly sort of the same properties. So it's called RE DOUBLE; it has some string here as it's parameter or some text here in its parameter inside double quotes which it displays to the screen and then gets you back a value which is a double one you can assign to a double. Those are just two things off the bat that you should know about because that's how you're going to get input, at least for the time being, from the user in a lot of cases.

Now, one thing you want to do once you actually get some input from the user is, you want to do some manipulation on it like some expressions that we talked about last time. We talked about some of the different operators like addition, subtraction or unary minus, it's the same symbol, multiplication, division and my favorite, the remainder. And so we talked about all those except for this little guy last time. All of the operators kind of work the way you would expect them to, okay. And we'll talk a little bit more about division in just a second. The interesting thing about division – so all of these things work with both – or I should say – all of these work with both integers and doubles. The remainder, as we talked about, only works with integers, right because it doesn't make sense to have a remainder when you have real values.

These three guys work exactly the same for integers and double, just the way you would expect addition, multiplication, all that happy stuff, to work. Division kind of rears its ugly head because it actually works slightly differently if you're doing division for integers versus doubles. Okay? The whole point of that is, if you're doing a division and the two arguments that you're dividing, right if both of these things are integers; in this case I have integer constant which is what I mean, the values, right. If both of these integers, what it does is integer division which means it does the division and throws away any remainder. So what you get back is an integer. So 5 divided by 2 when these are integers gives you back the Value 2. That little remainder thing is just gone. If you want to get the remainder you use this guy. Okay?

If either one of these particular values happens to be a real value, like a double, then it will do real-value division and give you back a real value. So if you happen to divide 5, even if 5 is an integer, by the Value 2.0 and so it knows it's a real value because it's got a decimal point in it, this will give you back 2.5 as a double and so you can assign that to a double. Okay?

So if either one of the arguments is a double, you get real-value division; if they're both integers, you get back the integer portion. Un huh?

**Student:**I'm a little confused about the double; the double is just a real number?

**Instructor (Mehran Sahami):**It's just a real number. Yes.

So another thing that kind of comes up when you do expressions – yeah, sometimes you're taking notes and you just don't know; it's like candy raining from the sky.

The other thing to keep in mind is just like arithmetic, sometimes you want operators to evaluate in different order. There's an order precedent for how these things actually evaluate in case you have to have some big honking expression. The order of precedent is you can have parentheses. Parentheses are the highest precedent. That means you evaluate everything in parentheses first, then multiplication, division and the remainder operator have the same level of precedents. And so if you have multiple of them; they're evaluated from left to right and then addition and subtraction. Again, if you have multiple, evaluate left to right.

So it's just like regular rules of precedent in algebra, which hopefully you're familiar with, but to make that concrete let's say you have some integer X and we say X equals 1 plus 3 times 5 divided by 2. How does that actually evaluate?

Well first of all, we say do we have any parens? No we don't have any parens. That would be the highest level of precedence. You can always force something to evaluate more highly by putting it in parens. So these guys are all at the same level, so we evaluate left to right. So we come across and we say here, here's multiplication, we evaluate this thing as 15, right? We don't do this addition first; this 3 times 5 becomes 15. Then we divide it by 2. And, you remember what I just said? Hey, this is an integer, this is an integer so this is integer division, so 15 divided by 2 in integer division is?

**Student:**Seven.

**Instructor (Mehran Sahami):**Seven. Rock on.

So this whole thing is 7 and then we add the 1 to it because addition has the lowest level of precedence of all the operators here and what we get at the end of the day is that X is equal to 8.

Okay, hopefully you can see that and if you can't, I we'll just tell you X is equal to 8. So those are the rules of precedent. They're exactly the same as, hopefully you know from algebra. Un huh?

**Student:**Isn't it true [inaudible] should be in parentheses?

**Instructor (Mehran Sahami):**It's nice to clarify; it's nice to always put in parentheses. Sometimes you have to put in parentheses to get the right thing to compute and I'll show you that in just a second, but it's nice now to just to fully parenthesize everything. Uh huh?

**Student:**Where do exponents fall in there?

**Instructor (Mehran Sahami):**There is no built-in exponent operator. Okay? So if you're sort of like a mat-lab person or something like that there're functions that we'll get to later on that compute exponents but there's no build-in primitive operation for exponents. Okay?

So with that said, sometimes there's times in life when you say hey, but Marilyn, I have these two integers but I really want to get some value back which is some real value. Right? So what you can do is – let's say I have INT X, okay? And let me give X some initial value; so I can say X equals 5. Then I want to take like half of that and assign it to some double Y. So if I say double Y equals X divided by 2, you might think, hey, this is a double right, isn't it going to do the right thing and give me back 2.5? No. It evaluates the right hand side first and then assigns it for left hand side. So you have 5 here. That's an integer. This 2 is also an integer. It does integer division which means you get 2. Once you get that 2 it says hey, but that two into a double and says, okay, it's 2.0. So you're like, huh? That was totally weird, but that's because it was doing integer division. So what you need to do is, you need to tell it – you need to say, I want you to do real-value division by temporarily treating one of these things as a double. That's something that we refer to at a cast. It's kind of like you were making a movie and you need to come up with a cast and you know, you get someone who's going to like, play Harry Potter for you, but he doesn't really wear glasses so you say, for the purpose of being in the cast, I'm going to put some glasses on you. For this one thing, you're going to wear glasses and then we're going to take them off. It doesn't change intrinsically who you are, it just makes you appear different for this one operation.

And the way we do that is we specify the type that we want to cast to in front of the thing that we want to cast. So we would say DOUBLE X divided by 2. All this means is, take this thing, which is not normally a double, and for the purpose of this operation, treat it as though it were a double. It does not change X from being an integer intrinsically; X remains an integer after this operation, but now it becomes 5.0 and when we do the division, we do real-value division, we get 2.5 and that's what goes into Y.

Okay? So that's what's called a cast. You can also do that, interestingly enough – let's say you did that and now Y is some box somewhere that has the value 2.5 in it and you say, you know what, I really like the integer part of Y. And so I'm going to have some integer Z and I'm just going to set that to be equal to what I would get if I cast Y to be an integer.

Well, if I cast Y to be an integer, you might say, oh Marilyn, does it round? Like do I get 3 because I remember if it's a .5 we round up, that was always the way it was when life was good. No. You don't round up. As a matter of fact, it could be 2.9; it could be 2.9999, you still don't round up. This is computer science. It's not necessarily forgiving. We always round down. You take the integer of a real value, it truncates it; it drops anything after the decimal point; it doesn't matter how big it is, sorry. It's just like the dollar; we just devalued your currency, right. It just – drop everything after the decimal point. That's life in the city. Okay?

So, if we go to the computer for a second, we can put this all together in a little program and actually see what's going on. So I wrote a little program that averages two numbers and it's running right now. I'll show you the text for that program over here.

So here's some program average two integers. I say this program averages two numbers; I read in one number from the user; I read in another integer from the user and then I come here, and notice I commented for you that it's buggy, I say hey, the average is just adding the two together and diving by 2 right? That's what I remember with a mathematical expression for average of two numbers when I looked it up in my book of mathematical expressions for averaging two numbers. And then I write out the average. So what's the problem here? Uh huh?

**Student:** The [inaudible].

**Instructor (Mehran Sahami):** Yeah, so there's two problems here. Da da daa. And then we reveal the comment and there they are.

First, we need to parenthesize the expression because in terms of precedent, the division has precedent over the addition. So if we were to say take the average of 5 and 6, we don't get 5.5, we get 8. Why? Because it took half of 6; it took that 6 and divided it by 2 and then added it to the 5. So it's actually at – doing this operation over here first and we want to say, hey, add some parentheses to force the addition to happen first.

Now ever after we force the addition, this whole expression here is still going to be an integer because it adds two integers, that's just going to give us an integer back and then we divide that by an integer. There's two ways we can fix this. One is, we can change the 2 to a 2.0, so we can explicitly say that that constant in there that we're dividing by is real value and that will force real-value division.

Another way we can do it is to say treat this whole thing as a double for this operation. So after you add those numbers together you get something which is the total of those two numbers, treat that total as though it were a double and then do the division. So if we do that, then we're good to go. Uh huh?

**Student:** If you put a double in there, do you still need that other double before you add it?

**Instructor (Mehran Sahami):** If I put a double in where?

**Student:** Where it is now is now, [inaudible].

**Instructor (Mehran Sahami):** Uh huh, yeah because this is the type of average. Right? So if average is not – I need to declare average first of all; I need to give it a type. If I don't give it a type double it can't store a real number anyway. So that's what that double's about. Uh huh?

**Student:** If you cast a double can you expect everything after to double on the line?

**Instructor (Mehran Sahami):** Yeah, it affects the most immediate thing after it. If it's parenthesized, it's a parenthesized expression; it's just one variable, it's just that variable. Uh huh?

**Student:** If the user enters a double into the computer, what difference would it make?

**Instructor (Mehran Sahami):** Ah good questions because someone wrote this little read integer for us so let's run it and see what happens because it's just that smart.

So here's – and one, we're like hey, here's 5.5, I want 5.5 and it says illegal numerical format. Right? It has a big cow; it's in red. Right? All that means is, I want an integer; you didn't give me an integer. Give it to me again and notice it prompts you again and you're like, okay, sorry. I meant the letter A. It's still illegal in numeric format. I would do that with like my friend's computer; like you know, they write their programs and they didn't do like, nice error checking or whatever that all the READ INT stuff gives you and they're like, give me an integer and I'd be like, how about A. And they'd be like, we'll that's not an integer and I'm like, to me it is. I'd put it in and watch their program crash and, you know, I was mean. Yeah, there's 5.5, it's working. You laugh now, wait until I do it to you. Uh huh?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Pardon:

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** There's something you can do in java having to do with formatting. We're not going to get into that now. If you're really interested, you could come talk to me afterward.

Oh, , that's going to be the goal for this class; to see if I can get one to stay up on the camera.

All right, so with that said, hopefully you kind of understand the notices of castings for an operation and the rules of precedence that are actually involved. There's a couple shorthands you should also know. Uh huh?

**Student:** [Inaudible] can I skip the top [inaudible]?

**Instructor (Mehran Sahami):** In that particular case with Z, you can skip the cast because if it says, hey if you have Y here and you want to assign it to an INT, the only way it can do that is to cast it automatically to be an INT, so it will do it for you automatically.

**Student:** The site, the part that I want it to return it contains much information [inaudible]?

**Instructor (Mehran Sahami):** Yeah, and so over here if you add a double and you try to assign it to an INT it just – you can't because it doesn't have space; it doesn't store what's after the decimal so it just truncates it for you. Yeah. I just put INT to explicit about the truncation.

There's a couple other things we want to do which are shorthands for arithmetical expressions because there are some arithmetical expressions – arithmetical, is that even a word? Arithmetic expressions – you've gotta keep me honest, sometimes I just make up words. I'll tell you stores about that at some point, but not right now.

Let's say we have INT X, our old friend, after I told you all this stuff – good variable name, everything's X. So let's say we say X equals 3. Now there's a lot of things sometimes you want to do with X. Something that we want to do a lot is to add 1 to X. Right, so if we want it to add one to X, how do we do that?

Some people already know. We'll do it the long way. X equals X plus 1. That adds 1 to X and stores it back in X. Turns out, this notion of taking a variable, adding something to it and storing it back to itself is very common and so there's shorthand for doing that which is X plus equals and then some value. So if I change this, say to a 5, and I said X equals X plus 5, that's the same thing as saying X plus equals 5. It just means take that value that's over here and add it to X and store it back into X. In the case of adding 1, it's not adding 1 because often times you want to count. It's something we do so often there's even a special shorthand just for adding 1. Which, is X plus, plus. This may look a little familiar to you from the Karel world. You're like, oh, I remember I plus plus, is that the same thing? Yeah, you were adding 1 to I in Karel's world. Here you're adding 1 to X.

So there's no spaces here or anything, X plus plus just means add 1 to X. If you've ever heard of language C and C plus plus, that's where C plus plus gets its name from. They sort of started with C and made it a little bit better by doing the plus plus. Okay?

You can also do sort of subtraction in a similar way. So if you want to subtract 1 from X, that's the longhand form of doing it, you can say X minus equals 1, which is a slightly shorter hand and there's a complete shorthand for it which is X minus minus, which means subtract 1 from X and store it back in X. Okay?

There's a couple other shorthands, you can also think about multiplication. So if you want to say X equals X times 2, like you want a double 2, you could say X times equals 2. Same sort of effect, sort of a times equal. There is no super shorthand for, you know, times equals 2, it just stops there. There is also divide equal, as you can imagine; so X equals X divided by 2, X divide equal 2. You know, the value here is basically the value there, so if you want to multiply it by 5, you would just put a 5 there, same kind of thing. But these little shorthands – you can use these with integers, you can use them with doubles, they work for both, you're fine. Okay?

So besides the shorthands, these are just some little syntactic things you should know, right. Hopefully all these things are fairly clear and they make sense.

There's also a notion, kind of time for a new concept; the notion of a constant. I'll write it in big letters. And the idea behind the constant is it differentiates it for a variable and that it does not vary, right. Variables vary, constants remain the same. And so you might wonder why do I care about what is a constant, right? Like isn't the value like 3 here a constant? Yeah but some constants are meaningful and sometimes you want to give them a name. So for example, PI. Right? PI is some constant that has a value like – let's just say for right now, 3.14. So in your program, you could have some double called a PI, all upper case that's the conventional use for constant, separate words have underscores to differentiate them, equals 3.4. And you might say, oh that's great, now anywhere in my program, like if I'm computing like, you know, the area of a circle and it's PI R squared, right, I could just say something like, take the radius, multiply it by PI, multiple it by PI again and this would be, you know, my – assuming I had R somewhere and I have area, I could compute it like that. And oh, isn't that so good. My program's readable now, it's so good, I know – yeah, that's all wrong. .

No one says anything, like sometimes I've gotta throw things at you to be like, is anyone paying attention? Yeah, we just squared PI. PI R squared, remember that? Yeah, there's R times R times PI. I can just do this for computing the area and everything's just fine, right? And it makes it a little bit more readable. But the problem is PI is a variable here right? Nothing prevents someone from coming along and saying, yeah, you know what, I don't really like your PI. I like my own PI and my PI is 6 because it's just bigger. Right? Then you go compute the area and your area got a whole lot bigger and you're like, why's it so much bigger? I don't understand. Right? Because someone changed your PI. Get your hands off my PI, all right.

The way you want to do that is, you want to force PI to not change. You want to tell the machine this is a constant. I will give you a value, it will remain unchanged. And the way we do that just happens to have some sort of bulky syntax associated with it, but I'll show you what that looks like. Okay.

So the way we say that – sort of follow along. First, we say private because we want to keep our constants to ourselves. So what private means is this constant I'm going to define, I'm going to define the constant in a class. It's not defined in a – you can define one in a particular method, but generally we define our constants in our entire class – I'll so you where they go in just a second – and we say private because we don't want anyone to be able to see our constants outside of our class. So first, we say private. Then we say static, which is just kind of a funky word which means that this constant sort of lives for the class and there's only one of these for the WHILE class. So it's not like – you, if you're an object for the class, you don't have your own version of PI and some other object has their own version of PI. The way you can think it is, remember I told you a couple days ago you're all objects? You're all objects and we have one constant, which is the amount of mass in the world, or universe, right. And so you don't have a separate mass and I'm like, hey, I have my different amount of mass for the universe and it's different from your amount of mass for the universe. We share the same mass for the universe. So all of humanity, or you could think of all objects that exists share the same

mass for the universe. That's what static means. Everything of a class shares the same thing. You don't have – each object doesn't have its own version. Okay?

Then, we say final, which means, get your hands off my PI. I'm going to give you a value and it's the final value. No one else is going to be able to give you a value. No one else is going to be able to give you a value to this because the value I give you know is the final value.

Then, we specify the type. Right? So we'd have DOUBLE for PI and then if we can zoom out a little bit this is going to be a two-board extravaganza because it's just that big. Public, or private, static, final, double, name of the constant PI and then its value 3.14. So all of that means you're going to get something who's of type double, it's value's not going to change after I give its initial value, there's only one of these for the entire class and it only lives inside the class, but we just put all those things there just so you know. Okay?

Now the important thing about doing this is you want to give your constants good names and – well, good names is one reason, right, so when someone reads your program they can say oh, he's multiplying by PI, that's a good time. The other thing is that it allows the program to change easily. Right? So I could have an area computed somewhere; I could have could have circumference computed somewhere else that also uses PI. I could go somewhere else and do something somewhere else funky with PI and then someone comes along and like some physicist comes alone and sees my program and says, oh, you must be an engineer because you're like 3.14. Like in physics, if we say 3.14, like particles miss each other when we accelerate them to the speed of light. You're like, well, don't accelerate particles to the speed of light. But they come along and they say, no, no, no. PI is 3.14159622, there's always a 2 before the 6. I'll stop there.

I have a friend who actually has PI memorized the 400 digits. I was like, that was just a tremendous waste of your time. . But that's okay, I'm sure it's probably useful somewhere. If I really need to know, I'll go ask him or I'll write a program to compute it. So we can get more exact values of PI, right.

The other thing that kind of comes up is that now, if I use PI all throughout my program, I'll I need to do is change it in that one place and I get more precision all throughout my program. So that's one thing in terms of nice software engineering, right? It changes everywhere so I don't need to go through and hunt down, oh, where did I put 3.14 in my program and I have no inconsistencies because if everyone's referring to that same constant value, they're all referring to the same thing.

The other thing you want to keep in mind, in terms of good names is think about names that are readable. No joke, I actually worked on a program for a large corporation once that will remain nameless, where this constant was in the program. I don't know, he had some extra stuff there and I looked at it and I was like, yeah, that's very descriptive. 72 equals 72 and I was like, I was really tempted, I just wanted to come along and be like, la, la, la, la, la. Like, does the universe end? Like what would happen? I have no idea where

this is getting used. And it turned out, what this – anyone want to venture a guess what this was used for? It's the number of pixels in an inch on most monitor resolutions.

That's what it was. It was like this gooey program, or user-interface program and that's what this was because someone somewhere in some class had told them, hey, if you're going to have any values in your program other than 0 and 1, which are very common and occasionally some other value will come up, like you want to divide by 2, it doesn't make sense to say this is the thing I use to divide by 2 and its value is 2, right. There's some values that come in your program that it's fine to just have the actual number in there. But other than these things, most numbers that appear in a program actually have some meaning and you want to have them assigned to a constant. So this person probably heard some rule somewhere that says oh, you should have constant for every value in your program and they didn't know what to name it so they just called it 72. That's bad. You want to give it real names for what it actually stands for.

All right, so any questions about any of this stuff? Uh huh?

**Student:** If the rationale like behind name constant and using that whole privacy thing that so nobody can change it but like, to change, if you have like just double PI equals 3.14 you'd still have to go to like the guts of the program. So how's it any better?

**Instructor (Mehran Sahami):** Well, the difference between this is that programmatically no one changed it. Right? So if I just said double PI equals 3.14, somewhere in my program someone could have come along and written this line and I can't stop them. Right? So I – it's bad software engineering because I'm not preventing them from doing this. This actually prevents them from doing this, so if they go into the program and say PI equals 5, when they try to compile, the compiler's going to uh huh, you told me PI was final now someone's trying to change it. Bad times. So that's the thing. You always want to force other people to also have good practice by doing stuff like this. Uh huh, question?

**Student:** Is there a library that we can import that has standard constants?

**Instructor (Mehran Sahami):** There is a math library but for the purpose of this class – if you're really interested in math kind of stuff, come talk to me, but for the purpose of this class, there's some basic functions that are in the book and so you can use those, kind of basic mathy functions but if you are really interested in other kinds of stuff we can talk about it separately.

I need to push on a little bit so I'm just going to hold questions for just one second because we need to go to an entirely different topic.

And the entirely different topic we're going to go to is something called Booleans, and there was a time when I told you about this type called Booleans, which is just the type for variables whose value is true or false. So we can say BOOLEAN P and P will take the

values either true or false. Anyone know where the word Boolean comes from? Yeah, George Boole. So know it and learn it.

I should ask if anyone knows how George Boole died? I asked this once, I'm going to say it, he got sets of false. I thought that was kind of funny. He actually died of pneumonia and the reason why he died of pneumonia – this is a true story – was he was out walking one time from his house to the college where he was a professor and he was in the rain and he got wet and he caught pneumonia and so when he came home, his wife actually, who happened to be the niece of the person who Mount Everest is named for – just a totally random side note – history's just fun – so it turns out she had this belief that the way you get over a particular illness is you experience the same conditions under which you got the illness. So she's like, oh you were in freezing rain, that's why you got a cold, so while he was lying in bed, she would get buckets of ice water and just douse him and he died. So medicine's come a long way; true story.

All right, so returning to our friend, the Boolean; Boolean has the value of true-false which means what we want to do on the – when we assign something a – assign a Boolean value to some Boolean variable, we need to figure out some expression that evaluates to true or false. Right? It doesn't evaluate to 5 or 10 or something, it evaluates to true or false. Like, 3 greater than 5 is a true-false expression, right. You can look at this and say Marilyn, 3's not greater than 5, yeah, and the value of this is false and that's what gets assigned to P.

So there's certain operations we can use called relational operations who, when we evaluate them on two values, give us back something that its value extensively is true or false. So let me show you some examples of that by going to the overhead. Let's see my little – oh, the remote mouse, I love the remote mouse.

So Boolean expressions is just a test, basically, for condition and it evaluates to true or false. Right? You can actually use the words true and false in your program and say P equals true, for example, those are actually parts of language.

So there's a bunch of different value comparisons you can do, like equal equal, two equals in a row, no space, is equals. It means like, does A equal equal B. It is not a single equal. What does single equal mean?

**Student:**Assignment.

**Instructor (Mehran Sahami):**Assignment, right? When you say X equals 5, that's an assignment. When you say X equal equal Y, that is a Boolean expression that evaluates the true if X and Y are the same. Not equal is exclamation point equal, or sometimes, now that you're all programmers, after Karel, this is called a bang. Because it's kind of like loud. It's like bang, exclamation point. Right? Caught your attention, hopefully you weren't sleeping. If you were, you're not. Bang equal means not equal and if you're familiar with some other language where you say like less than or greater than means no equal, nuh, uh, not in Java baby, doesn't happen.

All right, so then there's greater than, less than, greater than or equal to and less than or equal to. And the order of the symbols for these two actually makes a difference. But as you can imagine, those are some common things you might want to consider for two values to see whether or not, or two variables to see whether or not, for example, they have equal value. Okay?

Now, Boolean comparisons, order of precedents, it turns out there's a bunch of operations you can do on Booleans. There's an operation which is the bang, or the exclamation point, which means not. That's the logical not. So if you're a logical person, this is familiar to you. If you're not a logical person, I'll tell you what it means. If you put not in front of some Boolean expression, or some Boolean variable like P, if P is true then NOT P is false and vice versa. It just means give me the inverse of the logical value that P has or whatever that expression would have been. Okay?

There's AND, which is ampersand, ampersand. Get to know where your ampersand is on your keys on your keyboard are. This is logical and. What that means, if I have two Boolean expressions, P and Q, they can either be variables or expressions. P and Q is only true if both of the sub-expressions P and Q are both true. If either one is false or both are false, it is false. Okay?

And last but not least, there's OR. If you've never used the vertical bar keys on your keyboard, find them. You're like, I may have never used them in my life. You will probably use them for the first time in this class. Or, if I have two expressions P or Q, it's true if P or Q or both of them are true. So if either P or Q or both of them are true, then what you get when you take the OR of them is also true. These are in order of precedents so you're NOT's get evaluated first, then your ANDs, then your ORs. Much in the same way the parentheses get evaluated first, then multiplication, division, then addition and subtraction; same kind of thing.

So we can look at an expression, for example, Boolean P equals something like this. Notice I've fully parenthesized it to make it a little bit more clear. X is not equal to 1 or X is not equal to 2, so this might be a common thing you might think you want to do if you want to say, well I want to figure out if X is not equal to 1 and X is not equal to 2. So if it's not equal to 1 or it's not equal to two, isn't that the right thing? And in fact, this is not the right thing. This is buggy. P will always be true in this case. Why? Because if P's equal to 1, well the not equal 1 part is false but it's not equal to 2 and that's become true, so yeah, false or true and it becomes true and vice versa if X equals 2.

So what you really want to say is, X is not equal to 1 and X is not equal to 2. That will make sure if you want to make sure that X is not equal to 1 and X is not equal to 2. That's how you would write it although English people tend to say X is not equal to 1 or X is not equal to 2 and so it's confusing if you try to write that out in logic. This is what you really want in that case. This is buggy. Okay?

Hopefully this gave you an example of a Boolean expression. Is there any questions about this?

All righty, then let's move on.

Short circuit – question?

**Student:** Where do you get the [inaudible]?

**Instructor (Mehran Sahami):** Just fine it on your keyboard, it depends on your keyboard, but it's a vertical bar, it's somewhere on your keyboard. It will be and if it's not on your keyboard, throw your laptop away and get a new one and get angry at someone, whoever sold it to you. It's gotta be on there. Trust me.

Short-circuit evaluation. We actually stop evaluating Boolean expressions as soon as we know the answer. What does that mean? Okay, so let's consider something like this. P, which is a Boolean, equals 5 is greater than 3 or 4 is less than 2. Well you know what, 5 is greater than 3 so as soon as we evaluate that part, that's true. True or anything is true, right? It doesn't matter what the value of this guy is because we already got a true and true or if anything is going to be true. So in fact, Java does an optimization and this second test is not evaluated at all. That's why it's called short-circuit evaluation. As soon as we know the answer, we short circuit out and we don't evaluate the rest of it. You might say, oh, yeah Marilyn, that's interesting. Why should I care? And the reason why you care is there are sometimes making use of this actually makes a difference. So here's a useful example. That's a not-so useful example. Where's a useful example?

Let's say you want to avoid dividing by 0 because dividing by 0 is an error. You can say, well P is equal to; X is not equal to zero; if X is equal to 0, this is false. False and anything is false so you want to evaluate the second part over here and you'll never actually divide by the 0. If X is not equal to 0, this is true. So to know the value of true and something else, it needs to actually come over here and evaluate the second part because it got a true for the first part it needs to know that this is true. But if it evaluated the first part and got past it, you know that X is not equal to 0 so you won't divide it by 0. Okay? So these kinds of little tricks are actually used sometimes in code and it's called – sometimes it's called a guard to prevent this from happening. But that's why short-circuit evaluation is something you should actually know about because you can actually use it for useful things. Okay?

Any questions about that? Any questions about Booleans or the operations or that kind of happy news? Uh huh?

**Student:** What happens if you divide by 0?

**Instructor (Mehran Sahami):** If you divide by 0 you – usually you'll get – well, you will get an error, you'll get an exception but we won't talk about exceptions just think of it as an error.

So let's actually move on. We're just going to cruise through tons of stuff today because life is just good.

So it's time to talk about statements. Okay? Just like in Karel when you had statements like, move and turn left and all that happy news, now we're gonna do all that happy stuff in Java as well. Okay?

So one thing we first need to know about is this thing called a statement block, or a compound statement. They are referred to as the same thing. Usually I say block, the book likes to say compound statement, it's just a set of statements enclosed in braces. So you have some opening brace, a bunch of statements and then a closing brace. This is what we would refer to as a block. Why do we care about blocks? The reason why we care about blocks is, remember when we declare variables, a variable has something called a scope. All a scope is – the way you can think about it, it's not a mouthwash – the way you can think about scope is that it's the lifetime of the variable because variables come into the world when they're declared and one thing I didn't tell you is that at some point variables die. It's very sad. But where a variable dies and lives is the block in which it is declared. So when we say X equals 5 up here, X is alive until we get to the end of the block in which X was declared. Which means when we get to a closing brace, that X goes away. That's its scope, or its lifetime. Okay?

If we declared X outside of these, the scope, it would not die when it came here. But if X was declared out here, this scope is some which is referred to as an interscope. X is still alive inside the interscope and it does not die when we get it out. It just dies when we get to the closing brace of the scope in which X was declared. Uh huh?

**Student:** Can you declare a final end [inaudible]?

**Instructor (Mehran Sahami):** It – well it doesn't die because of where we declare it. So if we declare it inside a method, it could die when we get to the end of the method. We're going to declare it in the class so it's never actually going to be at the end of a method so it won't die, it will be alive for the whole time that class is alive. Uh huh?

**Student:** When you declare [inaudible]

**Instructor (Mehran Sahami):** Uh huh.

**Student:** Will it actually pull out of the sub block?

**Instructor (Mehran Sahami):** Yeah, as long as the variable's declared outside of the sub block, that's fine. And well – this is kind of technical point and sort of, in most intensive purposes you don't need to really worry about the nuances here but it's just something important to remember and we'll look at some examples as we go along.

So other statements you should know about; the IF statement. You're like, oh, IF, it's just like Karel, and yeah, it's just like Karel, right? We say if some condition, and that condition is something that evaluates to true or false. It can be a Boolean variable or it can be some Boolean expression, anything that evaluates to true or false. So you can use, and, or's, not's, all that happy news, now in a condition and then you have some opening

brace and close brace just like Karel and if the condition is true the statements get executed. So we want to check if something is even, we can say if that number, when it's divided by 2, if it's remainder equal the 0? Its remainder is equal to 0 and it's divided by 2, it's even so we're going to write out NUM is even.

Now notice here, I don't have the braces and it turns out there's this one special case that says if you have an IF statement, and it actually applies to a couple of other things, and it's only – the body is only one statement, you don't have to have it inside braces. If it's more than one statement, like this, then you have to have the braces. So this is just a special case that exists. But I like to think of use braces with IF; you have to if there is more than one statement inside the braces. But I like to think of something that I refer to as the orthodontist rule. Right? What do the orthodontist say when you go to an orthodontist? You need braces. Right? It doesn't matter if your teeth are straight or you only have one statement or whatever, you need braces. It's always a good idea to use braces, which defines a block, right, so this now is a block, even if there's only one statement in the IF. There's only one special case that I'll show you in just a second, but in general always use braces. Right, think orthodontist. Always use braces, it's just a good idea. Okay? Any questions about IF? Hopefully not because you hopefully were using them a lot with Karel.

We have the IF ELSE's just like Karel. Same kind of thing going on here. We have some condition, either we do the statements or we do the ELSEs part if the condition is false. So if we – the remainder of NUM divided by 2 is 0, we say NUM is even; or if the remainder is 0 and if it's not then we know it's odd and we write out NUM is ODD and so are you. All right? So not a whole lot of excitement going on there. You saw this in Karel; same thing exists in Java, now it's just a little bit different because you're not checking a front that's clear anymore, all right. We're not in Kansas anymore. You're actually doing some Boolean expression for that condition.

Let me just give you a couple more and then we'll take some questions. There's something known as the cascading IF. This is kind of funky. A cascading IF says, what I want is I'm going to check for at least just one condition to be true. I'm going to start off with something that looks like a regular IF. This is like high school grading, right? If your score is greater than 90, then print out A. Otherwise ELSE and what are we going to do with the ELSE? We're going to do another IF. ELSE IF your score is greater than 80, print out B; ELSE IF your score is greater than 70, print out C and otherwise it's kind of – bad times. All right?

So if you think about this, only one of these IFs gets executed. Why does that? Because if we come up there and we say score's greater than 90, we do this. We skip the ELSE part. Well, what's the ELSE part. The ELSE part is everything else. Why is it everything else? Because it's an ELSE and then it's one statement, right. There's no brace here. Remember I told you, if you have no braces it's one statement. Well what is that one statement? It's an IF followed by an ELSE and that ELSE has one statement contained within it so it's part of this bigger IF and that has an ELSE which is contained as part of that IF. So all of this stuff, in some sense, all cascades down. So if I do the IF part up

there and I don't do the ELSE, I skip over this whole thing. If I don't do the IF part, then I come and I evaluate the ELSE which means I evaluate this first condition and it's true then I print out B and I skip the rest. If it's false, then I evaluate the next condition. So this is just sort of the idiomatic form – or a pattern form that you see, it's called a cascaded IF for when you want to check a bunch of conditions in sequence and the first one's that true does its IF part and the it skips the remainder of it. It's so useful for things like grades where you're like, is it greater than 90? No. Is it greater than 80? No. Is it greater than 70? As soon as I find one I'm done. Otherwise, I have some catch all at the end if it says things are bad. All right?

So that's the cascading IF. There's something called a SWITCH statement. I'll touch on this very briefly. The SWITCH statement you can get it in excoriating detail in the book. It's just kind of nice for you to have. You can actually do anything with the SWITCH statement – anything you could do with a SWITCH statement you could also do with IF statements so it's just kind of a nicety.

The way the SWITCH statement works is, it says IF you have some integer value, so here we're going to ask user for day of the week, right, and the first day of the week in Day 0, which is Sunday. So they enter some number hopefully between 0 and 6 and we say depending on what number they entered – so that thing that goes inside the parens for a SWITCH statement has to be an integer value. Okay? Has to be an integer value.

There're some other things later on in the class where we look at the boil down to integer values but it cannot be a double, interestingly enough. And we specify the syntax goes like this, case, then what value matches that case. So this is Case 0. And you have sequence of statement until you hit something called a break. So notice, this is funky because the only braces here are this brace down here and that brace up there which encloses this whole SWITCH statement. There's no braces in the middle. The way it knows where to stop inside a SWITCH, so if someone enters 0 it says let me find Case 0. Oh, here's Case 0. I start executing from this line and keep going until I hit a break. When I hit a break, I jump out here to the end of the closing brace. So if the user types in, let's say 6, it says it is Case 0, no, so it skips that. Is it Case 6? Yeah, so it prints out Saturday and then it breaks which means it skips to default. If they enter any other value other than 0 and 6 it says does it match 0,no, so it skips that; does it match 6, no, so it skips that; it comes to the default with is kind of the be-all if it doesn't match any other cases and it rights out hey, it's a weekday. Okay?

So you can do this with an IF, right? But this is just a nice way – there's a lot of times – the idea is to think like you have some mechanical switch and you've got to pick one of the options and that's why it's called a SWITCH statement. Uh huh?

**Student:**Do you have to do in that order, or...

**Instructor (Mehran Sahami):**No, your order can be anything that it wants. You can have – you know, the things can even be out of order. So they don't even need to be sorted. Was there a question in the back? Uh huh?

**Student:**[Inaudible].

**Instructor (Mehran Sahami):**Pardon?

**Student:**Can you break out of the method?

**Instructor (Mehran Sahami):**You can't break out of the method all it does is break out of the switch, so it just takes you – you start executing it at that bottom brace again. You don't actually leave the method that you're in.

**Student:**But if you have like a...

**Instructor (Mehran Sahami):**Let's take it off line. Let's take it off line. Uh huh? Was there another question over here? All right. Let's press on then.

So in terms of four loops, you've also seen before hopefully in the context of Karel, so we're just zooming through. The way the four loops looks like, here's the general form of the four loop. It has four, just like you saw before in Karel and now it gets a little bit more complicated but not much.

We have something called the INIT. What the INIT is, is when you come to a four loop, it does whatever statements are in that INIT once and only once at the beginning of the loop. Okay?

Then you have something called the condition. The condition is checked every time before we go through the loop. So the first time we come to a four loop, we do the INIT, we check the condition. If the condition, which is a Boolean or it can be any Boolean expression and it's true, we execute what's in the loop. If it's not true, we skip over the loop and just execute immediately after that closing brace. Okay? So similar with what you may have done with Karel, for example, except in Karel you probably executed some number of time. And the step – so we execute the statements inside the loop only if the condition is true. And the step is done every time after the loop. Okay? So what does that mean? This is kind of a whole bunch of stuff to keep in mind. Let me just give you a simple example.

Here's a loop that you may have written sort of in the Karel world except now we're going to print out the value VI instead of making Karel do something, right? So this was a form of syntax used for Karel to do something five times. Remember that? If you remember that nod your head. Yeah. Hopefully.

So what is this really doing? Right now we can pull back the covers. Well the INIT was to say create some new variable named I and set its initial value to be 0. The test you were doing was to say as long as I was less than 5 execute what's inside the loop. And the step says every time you go through this loop I plus, plus. Add 1 to I and store it back to I. So when we execute this, what happens? The first time it comes to execute it says I equals 0, it checks to see if 0's less than 5. It is and it says okay, I execute the loop so it

prints out a 0, adds 1 to I and goes back and checks the test. So you can think of the I plus plus as happening after it's executed the statements in the loop, but before it does the test again. So now it adds 1 to I, I has the Value 1, it's still less than 5, it prints it out, it adds 1 to I, it's now 2; less than 5, it prints it out, adds 1 again, it's 3; it prints it out, adds 1 again, it's 4. Now after it prints out 4, it adds 1 to I again. Now I has the Value 5. It checks the test again. Is 5 less than 5? No. Five is not strictly less than 5 so it's done. Okay, so you get the values from 0 to 4 but notice it still went through the loop five times. It did the body; we just started counting at 0.

And as computer scientists, that's a very common thing to do is, really when you want to count something 10 times, you count from 0 to 9 instead of from 1 to 10. That's just what we do because zero's a real number and we love it, we care about it. Uh huh?

**Student:** Is the scope of I is the four loop after we're done?

**Instructor (Mehran Sahami):** Good question. Exactly, the scope of I is the four loop. So when that four loop – oh that was real bad – when this four loop is done, I goes away. Okay? So the lifetime of I is until we get that next closing brace in the scope of which I's the clairton. Sort of, we think of the scope of I's being the four loop so when we get to the end of the, when the four loop is finished executing and we sort of move on down here, I is gone away.

But we can create another one in some other loop, that's fine.

There's other funkier things we can do with the four loop rather than just counting from 0 up to some value. We can actually start with a value like 6 and count down. So we can say I's initial value is 6, as long as it's greater than 0, subtract 2. So we're going to use sort of that minus-equal-to funkiness. And so when we start off, I starts with the Value 6. Six is greater than 0 so it writes out 6, subtracts 2, 4 is great than zero, it writes it out, subtracts 2 again, 2 is great than zero so it writes it out again, and it subtracts 2 form I and so now I have the Value 0. Zero is not greater than zero so, it's done. Okay? So that 0 is not displayed because after we do the step, we always check the test again before we execute the loop one more time. Okay? Any questions about that?

So let's do the WHILE loop super quickly because you've already seen the WHILE loop in Karel's world. Same kind of thing. While condition, if that condition is true we execute some statements. The condition's checked before every iteration of the loop, just like in Karel. Right? So that's why we did all this stuff in Karel because it carries all over directly in Java and we execute the statements only if the condition's true.

Let me show you an example. X starts with a Value 15 while X is greater than 1, every time through we're going to divide X by 2 and write out the value. So first time X has 15, 15 is greater than 1, we divide by 2, we do integer division so we get 7, we write that out and go back up there. Seven's greater than 1, we get 3, we do it again, we get 1, when we do 1 divided equal 2 what do we get?

**Student:**Zero.

**Instructor (Mehran Sahami):**Zero, which is not greater than 1 so we're done. Okay? Nope, let me go back. Nope, let me just end the show. All right, so any questions about the WHILE loops and we'll review our friend the loop and a half next time. All right. So I will see you next week.

[End of Audio]

Duration 50 minutes

## Programming Methodology-Lecture07

**Instructor (Mehran Sahami):** Alrighty. Welcome back to yet another fun-filled exciting day of CS106a. So a couple quick announcements before we start – first announcement, there are two handouts in the back. Hopefully, you already got them. If you didn't already get them, you can get them later.

Second announcement, just a little reminder, microphones, if you go ahead and pick up those microphones right now and keep them in your lap. They're warm. They're cozy. They're fun. Use them to ask questions. Just because, looking at the videos, got a, kind of, general reminder that the questions aren't actually coming through. Whoa, or I can just breath real heavily into my microphone.

All right. So one real quick point before we dive into the main meat of the lecture, it's just a clarification on something we did last time called The Cast. So remember last time where we had a cast, which was this thing that allowed us to say treat this one data point, or this one data item, this one variable as a different type for one particular operation.

And so last time I told you you could actually do something like int X equals – let's say we had some double up here, like YY, and maybe Y got some value like 3.5, and I said you could do a cast like this, and we put Y here, right? And so that'll take Y. It'll truncate it, right? It drops everything out of the decimal point and assigns it to an int.

And I, sort of, said, misspeaking, actually, that you could drop this part. It turns out that in Java, any time you do a cast of something, that loses information. Like, when you go from a double to an integer because it truncates, it drops everything after the decimal point, you have to put in the cast, so you have to make this explicit.

If you go the other way around where you're not losing information, like, if you were to have Y equals X at some point where you know that X is an int and Y is a double, you're not gonna lose information by assigning an int to a double. So there you're okay.

You can do the explicit cast if you want, and actually I would encourage you, actually, to use the explicit cast, but in this case, you don't need it. In this case over here you actually do. So when you lose information, you need to make an explicit cast. For most of you, it probably won't affect your day-to-day life, but that's just something you should know. Alrighty. So any questions about anything we've done up to now?

We're just gonna, kind of, press ahead with something that we've talked about last time. Remember last time we're talking about our friend the while loop? And the while loop was something that you used when you don't know how many times you're gonna do something. This is what we refer to as an indefinite loop – not an infinite loop, right? It's important that you have "de" in there. Infinite loop keeps going forever, but an indefinite loop is one that when it starts out, you don't know how many times you're gonna do something. You're just gonna do it some number of times.

Well, it turns out there's times in life where you're gonna do something some number of times. You don't know how many, but you figure you probably want to do it at least once, okay? This was, kind of, the case of my brother going to college. Like, he figured, "I'm gonna go to college." But when he got there, he didn't know how many degrees he was gonna get, and he just keep going and going, and five degrees later, he was, kind of, like, "Okay. I think I'm done now." So that was his while loop, right? He didn't know how many times, but he knew he wanted to do it at least once.

Now, the interesting thing is that in programming, this actually happens fairly commonly. So I'll show you a quick example of this. If we go over to the computer, there's something we call the loop and a half because in some sense, you want to half of the loop at least once, okay? Think about it this way. Let's say we want to take some program that reads in a bunch of numbers from the user until the user enters zero to stop, and it, kind of, computes the sum of all those numbers, so it adds them all together.

So we might have, basically, some sentinel value, right? We're gonna use our constant, our little friend last time that we talked about, the constant, to define what is that value that the user enters in which point they want to stop entering values. This, is in programming speak, is often referred to as a sentinel. It's, kind of, the last thing I'm gonna give you that tells you I'm done; I'm not gonna enter any more values. But until I give you that sentinel, I'm gonna give you a bunch of values that I want you to add together, for example.

So that sentinel we might define as a constant, and so we could have some loop that says well, if I'm gonna keep track of some total or some sum of all the values you enter, that home's gonna start at zero. I'm gonna ask you for some value, so I'm gonna read in some integer and store it in the little box named Val, right? So I just declare it a variable, and then I'm gonna have a loop, and what my loop says is if the value you gave me is not the sentinel, that means you don't want to stop yet, right?

So I have a logical condition in here that's checking against the constant, and that's a perfectly fine thing to do. So we're taking a lot of those concepts you saw last time and just, kind of, sticking them all together into one bigger program. If I haven't seen that sentinel yet, then take the value you just gave me, and add it to my total, and store it back in the total using the little plus equals shorthand you saw last time, okay?

Then what I need to do is after I've done this, I need to ask you for a value again. So I say value equals read it. Notice I already declared the value up here, right? I already declared this variable up here, and this guy's lifetime is until it gets to the closing brace at the level at which it's declared. It's not declared inside the while loop, so this closing brace doesn't make it go away. This closing brace makes it go away, which means it's actually alive the whole time in the while loop, and it's perfectly fine.

So val, I read in some other value from the user, and, again, I go back through this loop. Did they give me zero? No, add it in. Did they give me, zero? No, add it in. And when they give me zero, then I say, oh, the value is equal to the sentinel, so this test becomes

false, and I come down here, and I write out the total as total, and you might say, oh, okay, Mehran, that's fine. What's wrong with that?

Well, in computer science or actually in computer programming, we really hate duplicated code. If we can avoid duplicating code, even if it's one line, we generally try to do it, and you can see there's a duplication here, right? What's happening is there's something we want it to do at least once, which was to ask the user for some value, but we need to keep doing that over and over in the loop, even after that first time. So we, kind of, run into this conundrum where we say are we gonna have to repeat code?

And there's a way around this. So the way around this is we pop out that piece of code, and we're gonna pop in our friend the loop and a half, okay? And so what the loop and a half says, the first thing that looks funky about it is we say while true, and you see that and you go, "Oh, my God." Right? Any time you see 'while true' you're thinking bad times, right? "I'm never gonna break out of the loop. Every time I come here and evaluate the condition, it's always true. Isn't this an infinite loop, Mehran?"

And it would be except for one caveat, and here's the caveat. We're gonna ask the user for a value, right? Notice we declared the int val inside here. We could've declared it outside if we wanted to, as long as we didn't do the read into outside. We could've just said int val and declared it out here, but we're only gonna use it inside the loop, so we're just gonna declare it here.

We read an integer from the user. We ask if the value is the sentinel. If it is, we break. What a break statement does is it will break you out of your closest encompassing loop. What does that mean? It means it finds whatever loop is the loop you're currently in and jumps you out of just that loop. So if you hit a break statement, it will jump out to, essentially, the place right after the closing brace for that loop and keep executing.

So what it allows you to do, essentially, is to check the condition to break out of the loop in the middle of a loop, which is, kind of, a funky thing, rather than at the very beginning or every time you iterate through. So we get the value from the user. If the value is the sentinel, we break, which means we never execute this line or the rest of the loop. We jump down here. If it is not the sentinel, then we add it to the total. While is true, so we execute the loop again and go and read another value from the user, right?

So notice that this read int line is no longer repeated. We only do it once, but because of the structure of this, we're always guaranteed that this portion of the loop up here is always executed at least once because we had a while true, and we're checking the condition to break out of the loop in the middle, okay? Now, one caveat with that is in terms of programming style, it's generally a bad time to have multiple of these kind of breaks inside a loop because it makes it very difficult for a programmer to figure out what condition has to be true to break out of the loop.

If there's only place you see a break, then there's only one condition you need to check to break out of the loop. If you have multiple statements, the multiple if, you know, blah-

blah-blah break in your loop, that means the programmer has to keep track of multiple possible places you could've broken out of the loop in their head, and that gets pretty dicey. So you generally want to avoid that when you're doing it, okay? So any questions about the loop and a half? Uh huh.

**Student:** Is it okay to redeclare a variable in a loop like that over and over again?

**Instructor (Mehran Sahami):** Yeah, it's fine to actually declare that variable inside the loop. You don't need to worry about, like, efficiency issues or anything like that.

So let me show this to you in action in a larger program, right? So here's that loop and a half in the context of an actual run method. It prints something on the screen. It says total to be equal to zero, and then it comes here, while true will always execute, right, because the condition's true.

So it asks the user for some value. We enter one; one is not equal to the sentinel. So we add that to the total, and you can see here it's keep track of value and total. Here's just my two declared variables and the little boxes for them, and they get updated as I go along.

So while it's still true, I read another value. Let's say the user gives me two. I add it to the total, which is now three, and I come back up here. I'll go through this a couple more times, okay? User gives me zero.

Now, notice when the user gave me zero here, I hit the break because my value is zero, and that's equal to the sentinel. So it says, hey, the if part is true, so do the break, and it immediately jumps out to the end of a loop. It did not do – even though it [inaudible].

So I feel a little like Roger Daltrey when I'm doing this. Anyone know Roger Daltrey, lead singer of The Who? Go look it up on Wikipedia, and watch the video. Man, I'm feeling old. All right. And then it says the total is, so this total plus equals value. The value just doesn't get added that last time, even though it would've been a zero and wouldn't affect the things.

It would be different, right, if we said the sentinel equaled a negative one. Then you would actually see that the total didn't get one subtracted from it, and we could've set the sentinel to whatever we wanted, but in this case, we used zero, okay? So that's our little funkiness, and it writes out the total of zero for actually doing the loop and a half.

Now, in terms of doing all this stuff, okay, you might think, "Well, hey, Mehran, you showed me about the for loop last time. You showed me about the while loop. It turns out I can write the same loop in equivalent ways using both of them." So you remember the for has the initialization, the test, and the step initialization happens once. Then the test happens every time through the loop, and then the step happens, kind of, every time at the end of the loop?

That would be exactly equivalent if I wrote it like this. The int happens once before the loop. I have some while loop that checks the same test. I have the same set of statements that would be in the for loop body, and I do the equivalent of the step in the for loop at the very end of the loop. So these two forms are exactly equivalent to each other, and you might say, “Hey, Mehran, if they’re exactly equivalent to each other, why do I have these two loops, and when do I use one versus the other?”

Well, for loops, you want to think about what we refer to as Definite Iteration. That means we generally know how many times we want to do something, and that number of times we want to do something is how we, sort of, count in our test. When we don’t know how many times we actually want to do something, that’s an indefinite loop, or indefinite iteration as I just talked about, that’s when we use a while loop, when we generally don’t know how many times we’re gonna do something, okay?

So that, kind of, why we give you both forms, and most programming languages actually have both forms, but to a programmer, it’s more clear between seeing the difference between for and a while, it’s, kind of, the meaning of it, right? Are you counting up to something some number of times, or are you just gonna do something until some condition is true? That’s, kind of, the differentiator there. So any questions about loops?

So let’s put this all together in the context of a bigger program, okay? So I will show you a bigger program here. So we’ll come over here. Here’s a little program; let me just run it for you real quickly. Come on. So we run along, and what this program’s gonna do is draw a checkerboard, right? Just like you’re used to in the game of checkers or chess, a little checkerboard on the screen.

So this is gonna be a graphics program. It’s gonna draw a bunch of G rects, which are just rectangles – in this case, they’re gonna be squares, to draw a little checkerboard, and you should think, “Huh, this might have some similarities to, maybe, some of the stuff you were doing in your current assignment. Yeah, so let’s go through this code and see what’s actually going on, and we can put a bunch of things we learned together like constants, and loops, and blah-blah-blah.

So what’s going on over here is, first of all, we start off with a couple constants, right? And the constants we’re gonna have tell us how many rows and columns are gonna be in our checkerboard. We want to draw a standard checkerboard, which is an 8x8 checkerboard. So we have private static final int, end rows is eight, and private static final int, end columns is also eight.

Notice that these two constants are not defined inside a method. They’re defined inside the class but not inside a particular method, which is generally where your constants are gonna be defined, right, in a class but not inside a particular method, and then for the run part of the program, first thing we need to know is we need to figure out – because we want this thing to be general, right? No matter how big our window is, we want to draw a nice little checkerboard.

So we need to figure out how big do those squares on the checkerboard need to actually be so they appropriately fill up the screen. How do we do that? We get the height of the graphics window – remember our little friend, the method Get Height, which tells us how many pixels high the graphics window is, and we divide that by the number of rows.

Since we divide that by the number of rows, this is gonna give us integer division, right? This is gonna give us back an integer, and we're gonna assign that to square size, which is an integer. So everything's perfectly fine because this is some number of pixels. This is an integer, and so this is an integer value divided by an integer value gives you an integer division.

So that tells you basically how many squares can you fit in, sort of, the height of the screen, and that's gonna be the size of one of the sides of our squares, and, as you know, squares have the same size on both sides, same length on both sides, so we're perfectly fine.

Now, what we're gonna do here is, right, we want to get the structure that's like a grid. In order to get a grid, we're gonna have what we refer to as a pair of Nested Loops. All that means is one loop is inside the other loop, okay? So we have two for loops here, and you can see with the indenting, one is, kind of, nested inside the other one.

So one is gonna count through the number of rows. So we're gonna have one loop that's gonna go through the number of rows we want to display and another loop that, within each row, is going to, essentially, count the number of columns in that row because we're gonna put one box for every little square grid on there, which means for every row, we need to do something for every column in that row.

So we're gonna have one for loop that's going to have I as what we refer to as its index variable. So remember when we talked about for loops, and we talked about counting, we said, "Oh, yeah, you say something like int I equals zero I less the num rows." And that'll count it from zero up to num rows minus one, which that iterates num rows times or N rows times.

That's great. The only problem is this variable I, remember, is alive until it gets to the closing brace that matches this brace, which means that I is alive all the way to down here. Well, if that I is alive all the way to down here, if we have some other for loop in the middle, if it's also using I, those I's are gonna clash with each other, and that's bad times.

So what do we do? We just use a different variable name. What we're gonna do is what's the next letter after I? J, right, so I, J, K you'll see are very popular names for loops, and those are the one place where descriptive names – you don't need to have some big name like, "Oh, this is the Counter Through Rows variable," or something like that. I and J are perfectly fine because most programmers are used to I, J, K as the names for control variables, or index variables, and loops is how we refer to it because this is what the loop is using to count through, so we think of I as the loop index.

So we have one here for J, and notice all the places I would've had I are now just replaced by J. So J gets initialized to zero. We count up to N columns, and J plus plus is how we increment J. So a common error with nested loops is you have the same variable up here as you had down here, and that's generally bad times, okay?

Now, once we're counting through all the columns, the thing we need to figure out is where is this square that we're gonna draw gonna go? What's its XY location? Well, it's XY location says, well, first of all, if you want to figure out its X, that's, sort of, going horizontally across the screen, right? Going horizontally across the screen depends on which column you're currently in.

So I take J, which is the index variable for this loop over the number of columns here, and I multiply it by my square size. What I'm essentially saying is move over in the X direction J many squares. That will tell you where the X location is for the next square of the tracks you're gonna write out.

Similarly, Y, which is the vertical direction, I need to figure out which row I'm in. Well, I is what's indexing my rows, right? So I'm just gonna take I and multiply it by the square size. That tells me how far down the screen I need to go to get the Y location for this square. So now I have the X and Y location. Any questions about how I computed the X and Y location? Don't be shy.

All right. Are you feeling good about X and Y? If you are, nod your head. All right. I'm seeing some nods in there, some blank stares. If you have no idea what I'm talking about, nod your head. All right. That's better. There's always this indefinite ground where it's, sort of, like, no one nods, and they don't give you any feedback. So I'm begging you for feedback, all right?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**We could've put this outside of the J loop and just computed – or we could've put the Y location outside of here and just computed it once because that Y location will remain the same for the whole row. So that would've been an interesting little optimization we could've made, but we didn't make it here because I just wanted to compute X and Y together so you would see them computed together, and because our forms are pretty similar to each other, okay?

So once I know the X and Y location for that square, I need to say, hey, get me a new square at that location. So I say new G rec. That gets me one of these new square objects. Where is the upper left-hand corner for that square? It's at the XY location I just computed. How big is the square? Square size width and square size height, right, because it's a square; it's got the same width and height. So that gives me a new square.

Now, I gotta square at some location that I want to display that square out, right, because I computed the right X and Y. The only problem is at the checkerboard. Some of the squares are filled; some are not filled. How do I figure out which squares should be

filled? This is where I do a little bit of math and a little bit of Boolean logic, and it just makes your life, kind of, beautiful.

So rather than having a whole bunch of ifs in there and trying to figure out, oh, am I on an odd square, an even square, which row am I in, blah-blah-blah? You do a little bit of math, and you figure out your math, and you say, hey, you know what, that very first square that I draw in the upper-right-hand corner, okay, let me see if I can bring that little square back up again. So I'll run this one more time.

This guy up here, I can think of that square all the way up there as the 00 square. That guy's not filled, but if I were to move one over that is filled, or if I were to move one down that is filled – so if I think about the index location that I'm currently at, and I take my X and Y coordinates, or I should say my I and J variables because that's what indexing my rows, and my columns and add them together, if that puppy's even, then it's not a filled square.

If it's odd, then it is a filled square because the very first one is 00. That one's not filled, so even's not filled, but if I move one in either direction, I get something that should be filled. If I move two in either direction, I get something that should not be filled, et cetera.

So the way that translates into logic for the code is I come over here, and it looks a little hairier than it actually is, but I say take the I and J, and add them together, and mod it by – or I should say – remainder it by two, right? If I remainder by two, if that remainder is zero, that means it's even.

If the remainder is one, that means it has to be odd.

The remainder can't be anything other than zero or one when I'm taking the remainder and dividing by two. So if that remainder is not equal to zero, that means it's an odd square, right, because my remainder divided by two is not equal to zero; it means it was equal to one. What I'm gonna do in that case is I'm gonna set filled.

So this looks very funky. You might be inclined to think, hey, shouldn't you have an if statement here, and say if this thing is true, then set filled? Well, think about, right, if this thing is true, set filled will become true, and I'll get a filled square. If this thing is false, set filled will become false, and I'll get an unfilled square.

So it works for me the way I want it to work in both cases, so rather than worrying about all this controlled logic for an if statement or all this other stuff, I just take my condition here, which is this big honking Boolean expression that evaluates the true or false and just stick it in for the value for – that set filled is actually expected, okay?

And this will give me the alternating effect for squares, and when I finally do this, I need to say, hey, you know, I got that square. It's filled the way I want it to be filled. I know its location. Throw it up on the canvas for me, so I add it. And then I just keep doing this

over and over, and I get all of my – for every row, I'll get a line of squares across and all the columns. Then I'll come down to the next row, and I'll just keep doing this until I get eight rows of squares. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**Use a mic, please. That'd be great. Thanks.

**Student:**How do you designate is as a Boolean expression? Does it just automatically assume that is going to be true or false, or – Instructor:

Right. So that's a good question. The key that's going on here is this little operator right here, the not equal operator, right? The not equal operator is a logical operator, which means it's gonna have two arguments that it works on, and what it's gonna give you back is true or false.

So the way it knows that it's Boolean is that this particular operator, the not equal operator, always gives you back a Boolean value, right? Just like greater than or less than, all those operators are Boolean operators; they give you back a Boolean value. Whereas, the stuff I'm doing over here, right, is just math. This is giving me back an integer, and I'm comparing that integer with another integer using a Boolean operation. That's why I get a Boolean. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**Does this only – well, this will work in any case as long as the window is wider than it is tall because the way we compute the squares is based on the height as opposed to the width. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**No, that's different, and we'll get into that now. Well, the private part's the same, but I'll show you what all of those things mean in just a second. So we're gonna do methods in just a second, and then it'll become clear. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**Because everything's in terms of pixels, and the pixels are always integers. So we just want to compute in terms of integers because all of our values mean our integer values. So let me push on just a little bit. Uh huh, question right there?

**Student:**Why are we doing it this way if we're supposed to be doing top-down programming; is it the opposite?

**Instructor (Mehran Sahami):**Well, in this case, top-down programming is how you break things up into smaller functions, right? Here we only have one function. So we

could actually think about, “Oh, should I do one row at a time?” And that would actually be an interesting decomposition is to have a function or some method that does one row for you, and I iterate that some number of times, and the reason why we didn’t do it that way is we haven’t done methods in Java yet, which is the next topic we’re gonna get into. So thank you for the segue.

So the next topic we’re actually gonna deal with is our friend the method, but in the Java world. So hopefully you’ve already seen in Carroll, right? You saw how to break things down in Carroll, and we did decomposition in Carroll, and now it’s time to bring that into the Java world, okay?

So when we think about methods in Java – you already saw a couple of these things, right? Like, read ints, for example, is some method that you’ve seen, and so we might say, you know, int X equals read int, or print lin is some other method you’ve seen, right, that just prints out a line.

Now, one way we can take these notions and make them a little bit more familiar is, first of all, we can say the idea of a method is just like in Carroll. You want to use methods to be able to break down programs into smaller pieces that you can reuse; that’s critical. But, in Java, they get a little bit more complicated, and here’s where the complexity comes in, and it’s easiest to draw the notion of how they differ in the Java world by thinking about mathematical functions, right?

So somewhere, someday, which you’re never gonna have to worry about again as long this class is concerned, is you learned about this thing called the sign function. You’re like, “Oh, Mehran, you told me there was gonna be, like, no, like, calculus in here.” Yeah, don’t worry. This is just for this example, and then the sign function goes away. You never have to worry about it again, at least for this class, right?

But the way the sign function worked is it took some value that you computed the sign of, right? So there was something here like X that you took the sign of, and what you got back from the sign function was some value that was essentially the sign of X. So you not only had something that we called a parameter that was what that function was expecting in terms of information going in, you also got some information coming out, which is what the value of that function returned to you or gave back to you, okay?

That’s the one bit of additional complexity that comes up with methods in the Java world as opposed to the Carroll world, and the Carroll world, you didn’t have any parameters that went in, and no values came out. In Java’s world, you do, okay?

So, first of all, just to wrap up on the little math example, a couple people asked last time about some mathy functions, and it turns out that there’s this thing you import called `Java.lang.math`. They’re a bunch of mathematical functions you get in the Java world for free, and some of those are, for example, oh, a function you might be using on this next assignment like square root.

So if we have double, let's say Y – or I'll say XX, XX equals 9.5, and then I might have some other YY here, and I want Y to be the square root of 9.5. I would say `math.sqrt`, which is the name of the method for square root, and then I give it the value, or I give it a variable whose value I want to compute the square root of.

So this guy takes in something of type double, computes the square root, and gives you back that value that you can now just assign to some other double, for example, okay?

And there are some other ones in there. There's a power function. Someone asked about power last time. That's where you also get the function; it's also in there, but power gives you a nice example of you can have multiple parameters. So power you actually give it an X and a Y, and what it does is computes X to the Y power. So it's essentially computing something that would look like that if we were to write it out in math-ese, and it gives you back that as a double. So we could assign that somewhere else.

But notice we have two parameters here, and they're separated by a comma, okay? And that's, generally, what we're gonna use to separate parameters is commas, just like you saw when we created, for example, objects over here, right? We created a new G rect, and we had to give it four parameters. They were just separated by commas, same kind of idea going on over here.

Now, the question comes up, what's the whole point of having these kind of functions exist in the Java world or methods in general? And the critical part about methods in Java, there's the whole notion of decomposition and top-down design. That's part of it. That's not the most critical part.

The most critical part has to do with a notion we think of as information hiding, okay? So what is information hiding all about? The real idea – the way you can think about this is that everything in the world is a function or some method. So, for example, anyone know what this is?

**Student:**CD player.

**Instructor (Mehran Sahami):**CD player, you're like, "Mehran, a CD player, come on. You've gotta be kidding. Like, what, were you, like, on some 1990s archeological dig or something?" It's like, oh, okay. I think I've found the CD player. This is a CD player, right? It's also a method. What does that mean that it's a method?

Well, guess what? It takes something in, and the thing it takes in happens to be CDs, and we have a little CD, Alice in Chains, always a good call, Apocalyptica. Any Apocalyptica fans in here? Yeah, a few, it's Metallica done on the cello, and Tracy Chapman just to date myself.

But what do you do? You take a CD. You put inside here. You hit close, and you hit play, and out comes music, and the music that comes out of this thing depends on which CD you put in, but the interesting thing about it, if you think about it, is all of the electronics

in here are completely reusable, right? I can use this CD player on, virtually, any CD, right?

Someday we might get to the day where you go to the store, and you buy a CD, and all the electronics to actually play the CD are inside the box, and you just plug your headphones into this, and you just listen to your CDs, and then we toss away those electronics when we get rid of that CD, and we get a copy of all the electronics again. That would be a huge waste, right, if you kind of think about landfill space and all that stuff.

So why don't they do that? Why don't all the electronics to listen to a CD come with the CD? Because I can create them once and generalize them in a way so that if I pass in the right parameter, which is the CD, I can use the same thing over and over for all different kinds of values that go in, and, as a result, produce all kinds of values coming out, okay?

So that's the thing you want to think about in terms of methods is you want to think about methods and defining what's gonna go into them and what's gonna come out of them in a way that's general enough that they can be used over and over again, and that's what the whole beauty of software engineering about is thinking about that generality, okay?

So with that said, how do we actually define a method, okay? So we need a little bit of terminology to think about how we define methods. Yeah, it's time for that dreaded terminology again in the object-oriented world, okay? So when we actually call a method, right, you've seen this a little bit, just like we did over there. We have what we refer to as the receiver, the name of the method, and then some arguments, which are the parameters, right?

So here the receiver happens to be math. The method name is power, and then there's some arguments X and Y that get passed in, okay? The way we think about this thing is we say that we are calling or invoking a particular method; here is the name of that method. So a lot of times you'll hear me say, "Call the method." That's what we're referring to is invoking that method.

What we passed in terms of these arguments is the parameters, what that function is actually or that method is actually expecting, and you'll hear me sometimes use the terms function and method interchangeably, and they basically mean the same thing. And this guy can, potentially, give me back some value, just like square root and power did over there.

Now, sometimes we also refer to this, and you may have heard me say this in the Carroll world, right, as sending a message to an object, right? So if we have our friend the G label, and I declare, let's say, I'll just call this lab to make it short, which is short for label, and I ask for a new G label, and that G label, I'm just gonna give it the words high at location 10, 10 just to keep it short and fit it on the board, and then I set its color by saying label – or in this case just lab.set color is color.red, okay?

So when I've done that, lab is the receiver. The method name is called Set Color, and, alternatively, I would say that I'm sending the set color message to lab, okay? That's just the terminology that we use. You should just be aware of it. So set color is the message that we're sending, okay? That's the receiver of the message, same kind of thing as the name of the method and the receiver of that method, or making a method called or a method invocation; those names just get used interchangeably.

So how do I actually create a method in Java, okay? Let me show you the syntax for creating a method. It's a little bit different, just slightly different than Carroll, but actually very similar, and so the way this looks is I first specify the visibility. You're like, "Huh? What is that?" You'll see in just a second.

So we have the visibility, the type that the method may potentially return, what value it may return, the name of the method, all these have squiggly lines under them because these are things that you will fill in momentarily, and some parameters that get sent to that method, and inside here is the body, which is just a set of statements that would get executed as part of that method.

Now, what do all these things mean? First of all, let's start with the visibility, which I'll just abbreviate as vis. The visibility, at least as far as you need to know right now, is either private or public, and you're like, "Hey, yeah, I remember those from the Carroll world."

Yeah, and they're exactly the same. The way you want to think about it is right now all you need to worry about is your run method is public, and, pretty much, all the other methods you write, as long as that method is only used in a single class, which is generally the class that you are writing that method in, they will be private. So, for all intensive purposes right now, all the methods that you're gonna write, except for run, are all gonna be private, and run has to be public. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**The way things are set up, you just need to make run public because it needs to have a larger visibility the way we've, kind of, defined the CS106 stuff right now. At the very end of class, like, when we get to the end of the quarter, I'll, sort of, lift the covers on everything, and you'll see why, okay?

So that's the visibility, private or public. Run is public, most other things are private. All private means in terms of visibility is the only people who get to see this method are other methods inside the class that you're defining, okay? So that's why it's visibility. It's who else can see it, okay?

The other thing you want to think about is the type. What's the type? The type here is specifying what gets returned by this method. So remember we talked about some methods, for example, that give you back something, okay? It is the type of the return value. What does that mean?

If I have some function that's going to compute the sign of something, and you give me a double, and it's gonna give you back a double, its return type is double, and you might say, "Hey, but, Mehran, when I was doing methods with Carroll, none of my Carroll methods actually returned any values." Right? Yeah, that was perfectly fine, and guess what word you used when they didn't return a value?

**Student:** Void.

**Instructor (Mehran Sahami):** Void, yeah, that's a social. Good times all around, right? So void is a special type, which basically just means I'm not gonna return anything, right? So its return type is void. It's like you're sinking into the void. You're getting nothing from me. You know, it's, kind of, standoffish, but that's what it means is I'm not giving you back a value. You need to know that I'm not gonna give you back a value, so I still specify a return type as void.

Then the name, you know about the name, right? That can be any valid name for a method just like we talked about in Carroll, and there's some parameters. The parameters are information we pass into this method that it may potentially do some computation on. So I'll show you an example of that in just a second, and what parameters actually are, and how we set them up, okay?

Some functions may actually have no parameters, like the functions you wrote in Carroll, right? There was no information that got passed into those functions, in which case, the parameter list was empty, and all you had was open parens, close parens, which means there's no parameters for this function. When you call it, you just say the name of the function and open paren, close paren, okay?

So, with that said, let me show you one more thing and then some examples. So you've seen a whole bunch of syntax so far of while loops, for loops, variables, declarations, all this happy stuff, and now I tell you, "Hey, there are these methods, and this is, kind of, how you write a method." And you're like, "Yeah, that has a lot of similarities to Carroll." But there's this value that you're returning. How do I return the value?

Well, surprisingly enough, you use something called the return statement, which just says return, and then after return what's over here is some expression which just could be some mathematical logical expression, and when you get to a return statement somewhere inside a method, that method stops immediately. It does not complete the end of the method, and immediately returns whatever the value of expression is.

A lot of times, the return will come at the end of a method, but it need not come at the end. Your method, at the point that it's currently executing, will stop and return with that value that is expression as soon as you hit it, okay? So that's, kind of, a whole mouthful of stuff. What does this actually mean? Let me just show you a bunch of examples, and hopefully this will make it a little more clear.

We want some methods; give me methods. So here's an example of a simple method. This is a method that converts feet to inches, right? It's, kind of, like, "Oh, Mehran, yeah, this is, you know, yet again, an entirely useless thing for you \$2,000.00 computer to do." But it shows you a simple example of all the pieces you need to know to write a method.

What's the visibility of this method? It's private. It's only gonna be used inside this class. What type is this feet to inches thing gonna give back to me? It's gonna give me back something that is of type double. The name of the function is Feet to Inches. What parameters does this function take? It takes in one parameter. The type of that parameter is a double, so what I'm expecting from you is something to double. It could actually be an explicit value that's a double.

It could be a variable that holds a double U inside it, but that's what I'm expecting is a double, and the way I'm going to refer to the double that you give me in this function is with the name Feet. So all parameters have a type and a name, and the name is the name you're gonna use inside this method to refer to that parameter.

So what do I do? You give me some number of feet as a double; I multiply it by 12, and that's what I'm gonna return to you. So when I get to this return statement, the method is immediately done, and it returns the value over here in the expression. Often times, you're not required to, but I like to parenthesize my expressions to make sure it's clear what's actually being returned. So let me show you another example. We'll just go through a bunch of them. Uh huh?

**Student:** Is it common to not put a comma as a separator for the type and then the name of the [inaudible] –

**Instructor (Mehran Sahami):** Well, the type and the name come together, so there is no comma there. Let me show you another example that actually has more than one parameter, okay? We'll get to that right now. So here is a function that's the maximum function, right? Again, it's private. It's gonna return an integer because you're gonna give it two integers, and it's gonna return to you whichever one has the larger value.

So its name is Max. It's going to return an integer, and what it's gonna take is two arguments here, two parameters. Well, the first parameter, val one is an integer, and val two is also an integer. So the way I specify them is always they come in pairs, type and the name that I used to specify – type and the name I used to specify it, right?

So val one and val two are both integers. What do I do inside here? I just have an if-val statement. If val one is greater than val two, then val one is the max. So right here I'm gonna return val one. As soon as I hit that return, I'm done with the method. I don't go do anything else, even though there was only an else that I wouldn't have done anyway, I return val one. If val one is not greater than val two, then I return val two because val two needs to be at least greater than or equal to val one, so it's the maximal value, okay?

So, again, if you have multiple – you can have as many parameters as you want, sort of, within reason, right? And within reason I mean, like, sort of, a few thousand. You can have as many parameters as you want, and that's a lot of typing. They get separated by commas and they come in pairs, value and the name that you use to refer to it. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**No, it will not print anything on the screen. All this will do is return that value to whichever place invoked this method. So if you think about something like read ints, right? Read ints gives you back a value, but it doesn't actually, I mean, that value happens to show up because these are typed it on the screen, but it wouldn't have printed it out otherwise, right? This doesn't print anything out; it just returns it back to where you, sort of, invoked this method from. Uh huh?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**There is, and we'll get to it in about three weeks.

**Student:**[Off mic].

**Instructor (Mehran Sahami):**Yeah. So here's another one. This is something we refer to as a predicate method, and a predicate method is just our fancy, computer sciency term for a method that returns a Boolean, right? A method can return any type you want. This one happens to return a Boolean. Private Boolean is odd. What does this do? You give it an integer; it tells you back true or false. Is it odd, right? The name is pretty self-explanatory.

So it takes that variable, computes the remainder after we divide by two, and if that remainder is equal equal to one, this whole thing is gonna be true, and it'll return true. If it's not equal equal to one, that means it's equal equal to zero. In that case, this is false, and it'll return false.

So this whole expression here again because we have an equal equal, which is a logical operation, right, that's gonna give us something back that's true or false, and that's what we're gonna directly return to the person who called this function, okay? So any questions about that?

Alrighty. So let's look at something in a slightly larger context, right? Let's put this all together in a full program so you can get a notion of defining the class, and defining the methods in it, and you can, sort of, see the whole thing as we build it up. So here is something called the Factorial Example Program, and it just extends the console program, and all this puppy's gonna do is basically compute some factorials from zero up to max num.

So we have some constant here, which is the maximum number we want to compute factorials up to, and if you remember – let's take a slight, little digression to talk about

factorials. So if you remember in your math class in the days of yore, there was this thing that looked like this, all right? And you saw it and you went, "Oh, It's  $n!$ , right?" I hopefully caught your attention.

This was actually  $n$  factorial, right? Yeah, the explanation point – totally different now because a couple people were, like, sleeping, and "What? What was he talking about?"  $N$  factorial, and all  $n$  factorial is is multiplying all the numbers from one up to  $n!$ , okay?

So five factorials, strangely enough, is just  $1 \times 2 \times 3 \times 4 \times 5$ , and one factorial is just one, and here is the bonus question; what's zero factorial?

**Student:**One.

**Instructor (Mehran Sahami):**Oh, I love it. It just warms the [inaudible] of my heart. Yeah, it's one, and there are some people who ask, "But zero, you know, like, I, uh – " and they get all wound up tight, and they're like, "No, man, mathematicians just said it was." Because if you think about it, it's multiplying – it's starting with one and multiplying zero additional terms, right? So I'm left with one. It's a good time.

So how do I actually compute that? Well, in my program I'm gonna have some run method that's going to count from zero up to max num and write out all the factorials. It'll say zero factorials this, one factorials this, two factorials this, by calling a method called Factorial passing in the value of the thing I want to compute the factorial of.

How do I compute the factorial? Well, I just add some method. This probably is gonna return an int. It's name is factorial. It's gonna take an  $n!$  of the thing I want to compute the factorial of, and how do I do that? Well, I start off with some result that's value is one, and I'm gonna have a for loop that's gonna count – here's one of the few times as computer scientists we don't count starting from zero; we actually count starting from one because it makes a difference.

We're gonna count from one up to and including  $n!$ . That's why this is less than or equal rather than just less than. So from one up to and including  $n!$ , so there's still  $n!$  terms I'm gonna count through. What am I gonna do? I'm gonna take my results as I'm going along and times equals it by I, which means the first time I multiply by one, store it back to result, then by two, then three, then four, all the way up whatever this  $n!$  value was, and when I'm done doing this loop, I return to my result, which is my factorial, okay? Any questions about the factorial method by itself?

Now, one thing that comes up, freaks people out a little bit, don't get freaked out; it's okay. You see an I here, and you see an I there, and you're like, "Oh, we just did that thing with nested loops, Mehran, where you told me that if one thing is inside another loop, like, I shouldn't use the same name. So why are you using the same name here?" And the reason why I'm using the same name is to make a little bit of an example, which is that the I here and the I here are different I's, okay?

When you think about a particular method, a method can declare its own variable. So this result, its lifetime or its scope, is until we reach the end of the method down here. This guy lives inside this method. This I is an I that only lives inside the factorial method. It doesn't interact at all with the I outside here.

So every method, and the interesting thing here is run is just another method. It just happens to be the special method that we always start executing from, but every method can have its own declared variables, and this I here is just its own declarative version of I. It has nothing to do with the I up there, okay? Question?

**Student:** Okay, so is it possible to, like, give a variable from one method to the other method?

**Instructor (Mehran Sahami):** We'll talk about that in just a bit. So someone was wondering is there a way I can, sort of, give some variable from over here down to over here? And I'll show you how you have to, kind of, think about that in just a little bit, okay?

But any questions about this notion of the same I, that this I is actually a different I than that I. When you declare a variable in the method, you are getting your own version of that variable in the method. It has nothing to do with the same named variable in any other method, okay? Kind of, a funky thing. Uh huh?

**Student:** [Off mic].

**Instructor (Mehran Sahami):** That's just starting off the factorial, right? So if someone gives this five, we need to start off by saying we have some initial result, and then we're gonna multiply that by one, and then we're gonna multiply that by two. If we didn't have some initial thing we started multiplying, there would've been some unknown value there, and actually Java would've given us an error because it doesn't allow you to use uninitialized variables, okay?

So this is, kind of, a funky thing, but important to, sort of, think about. There is a method in the context of a larger program. Variables within methods are not the same as the same named variable in another method. That's the key take home thing, like, know it, learn it, love it, tattoo it backward on your forehead so every day when you look in the mirror when you take a shower, you see it, and you just know because that's one thing that trips people up. Uh huh? Was there a question up here?

**Student:** So when you return result, does it get mapped into I? Is that why [inaudible] –

**Instructor (Mehran Sahami):** Ah, good question, so when I return here, what's actually going on? When I actually could say I'm gonna compute the factorial of five, okay? This guy comes here – let's make it easy. Let's have him computing a factorial of two. So it multiplies one by one, then it multiplies it by two. I get two. It returns this two. What happens to the two?

That two goes back to the place at which this function was called. Where was that? That was up here. So this place up here where I said, hey, two plus your explanation point equals is what I would print on the screen. If I was equal to two here, right, two exclamation equals, and then what's the value that I get back when I call factorial of two?

Well, it computes factorial of two. It happens to be the value of two. It just sticks in a two for whatever you had as the calling method, okay? So you can think of when you call a particular method, the name of that method, think of it as getting replaced by the result that gets returned from that method, okay?

So if I run this program, just to make it clear, let me show it to you in its full glory over here. Here's a factorial example. I'll go ahead and run it, and you can see what its output is. Factorial example, we'll run it, and there it is, right?

So it says – if you can see up there, a zero factorial equals one because it writes out zero factorial, right, in the code over here. It would write out I has the value zero the first time through. So it writes out zero factorial equals, and then it calls factorial on zero, and what it gets back from factorial of zero is one.

So that one is just what gets displayed in the place of factorial zero because that value I get back from factorial zero is what's actually used here, and it keeps doing that all the way through the loop, right?

So down here it calls a factorial of nine, gets the value for factorial of nine, and returns that to the place where it's actually gonna print it out, okay?

So let me show you one more quick thing. Okay. Which is returning an object from a function, okay? So it turns out, interestingly enough, you can not only return, like, ints, and doubles, and Booleans from functions, you can also return whole objects.

So here is a method called Filled Circle, and what I'm gonna give to this method is some X and Y location for the center of the circle. Notice that's different than what G Oval Expecting to the upper left-hand corner. I'm gonna specify an X, Y location which is the center of the circle, a radius for that circle, and a color, and what I want this method to give back to me is an actual object that is a circle whose center is that XY. It has the given radius R, and it's filled in of that particular color.

Well, how do I do that? Inside here, I say, hey, I'm gonna create a new object that's of type G Oval, and I'm gonna call it Circle, and I'm going to get a new oval, right, and remember oval expecting the upper left-hand coordinate for the oval. I want the circle centered at XY, so what do I do?

I take that XY, and I subtract the radius from the X direction and the Y direction, which gives me the upper left-hand corner for the bounding box for that circle, and then I say give me something that has a width of two times the radius and a height of two times the

radius because that's what defines that circle, right? The diameter is two times the radius in both the height and the width direction.

So I get that oval. I've now called this Circle because it's my new object. I tell that circle to be filled because that's what I want to return is a filled circle. I tell that circle to set its color to be whatever color was passed in here. Notice the K sensitivity. Color with a capital "C" is a type. Color with a lowercase "c" is actually the name of the parameter that I'm gonna refer to, and that's what I use here is lower case "c."

So I set the color of the circle to be whatever color was passed in, and then I return this object. So that whole big box that contains this object that is some filled-in circle at that location, I say, hey, you, this is what you wanted because I'm giving you back a G oval. Here you go, and I give you back the whole thing, which means when you invoke this method, you better take the result and assign it to someplace that's a G oval, otherwise – or just add it to your canvas. You need to do something with it that you would normally do with a G oval, and you can't assign it to an int because an int and a G oval aren't compatible. Uh huh, question?

**Student:**[Off mic].

**Instructor (Mehran Sahami):**Okay, yeah, you can't assign it to an int; you have to assign it to an object of the same type. Uh huh, question back there?

**Student:**Why are X and Y set to doubles if the pixels are all whole numbers?

**Instructor (Mehran Sahami):**Ah, good question, so I could've actually made this with int and I probably should have made it with int, so it was my bad. Uh huh?

**Student:**In the red method, how would you add that oval? Like, what's this, like, syntax?

**Instructor (Mehran Sahami):**So the way you could think of it – let me just write it for you up here. So if I actually had the run method, so I'll blank, void, run, and then let's say I had this function filled circle, right? So I called the method Filled Circle, and let's just say I want to put this at location 10, 10, and its radius is two and the color that I want to give it is red, okay?

Now, the thing that this gives me back is a G oval object. There's a couple things I could do with it. I could declare a G oval out here, call this O, and assigned filled circle to it, and now I have the object O out here, which is actually that circle that this method gave back to me, and I can do whatever I want with it. Like, I could say add O, and it would add it to my canvas.

Alternatively, if I really wanted to, I wouldn't even need to set filled circle to some object that I want to keep track of out here. I'm just gonna say, hey, you're gonna give me a circle, I'm gonna add it directly to my canvas. So anything that I would be able to do in my run method with a regular G oval is the same thing I can do with this G oval that's

coming back from a method because it's just giving me back a G oval, and I can use it in the same way. Alrighty? I will see you on Wednesday then. We'll talk a little bit more about functions and methods then.

[End of Audio]

Duration: 52 minutes

## Programming Methodology-Lecture08

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

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

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

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

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

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

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

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

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

**Student:** A toaster.

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

You don't care how the toast is produced. One way the toast is produced is there's coils in here that heat up, that, sort of, bake the bread, and bread comes out, that's great. Another way the toast is produced is there's Keebler Elves inside here, and they're like, "Hey, we got bread." And they eat the bread, and they excrement toast, right? And that's

just what comes out. Elf excrement, but you don't care because it's toast, right? You're like, "That's great. Something's being hidden from me." And in the case of elves, you're glad it's being hidden from you, but you get back what you cared about, right?

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

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

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

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

So when we talk about calling these methods and what's going on parameter passing, a little bit of complexity comes in, and today we're gonna go through that complexity in

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

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

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

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

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

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

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

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

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

One more time, just for good measure, so – whoa. That was bad measure. Sometimes these things just never work right. You have to hate technology. All right. So we add one to I. I now becomes one, right? I was zero over here. We add one. I becomes one. I is still less than four, so we execute this line once again. Now we're calling factorial passing at the value one. It gets a copy of that one. Where does that one go? That one goes into the parameter that it's expecting, the N. So it gets a one because that's the value we passed

in. We passed a copy. We didn't pass I itself. I is still safe and sound over there, the actual variable I. We just got a copy of its value. That's what got passed.

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

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

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

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

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

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

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

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

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

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

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

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

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

**Student:** Yeah.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

So this guy is only visible in terms of its visibility, which we'll just call Vis in the method, and, as a matter of fact, it's lifetime – it only lives in the method, and when the method is done, the scope of this variable is basically gone, and it goes away. These puppies are visible in entire object. So in the class in which they are declared, when you create an object of that class, you get an Instance Variable; for every Instance Variable declared, you get one, and it's visible in the entire object. That means its lifetime is it

lives as long as the object lives, which means once you call some particular method, if that method has local variables, they come into being, and when the method is done, they go away. When you create an object, its Instance Variables come into being, and they stay around until that object goes away, you're done using the object. That's how long they live.

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

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

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

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

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

that state what you need to store in between calls to methods that is actually part of the state of the object that you care about, okay? Any questions about Instance Variables versus Local Variables? Uh huh?

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

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

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

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

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

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

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

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

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

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

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

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

Let me show you a slightly more involved program. So we can get rid of this craziness, and here's a little program that rolls some dice. So what this does – notice now I have the imports and everything in here. I need to import acm.util.star if I actually want to use the

random number generator, and now just to be a little bit more complicated, I'm gonna show you an example of having Constants and Instance Variables together because just where we put them and what the convention is in the book is just a little bit funky.

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

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

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

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

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

whatever that Max Roll was after some number of rolls. So it tells the user how many times I had to roll before I got that Max Roll, okay?

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

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

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

**Student:** [Off mic].

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

**Student:** [Off mic].

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

**Student:** Okay.

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

**Student:** [Off mic].

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

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

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

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

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

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

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

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

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

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

**Student:** [Off mic].

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

**Student:** [Off mic].

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

**Student:** [Off mic].

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

**Student:** [Off mic].

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

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

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

[End of Audio]

Duration: 51 minutes

## Programming Methodology-Lecture09

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

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

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

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

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

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

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

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

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

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

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

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

Is there any questions about String? All right.

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

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

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

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

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

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

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

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

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

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

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

Um hm.

**Student:** [Inaudible.]

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

Um hm.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Student:** [Inaudible.]

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

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

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

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

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

Any questions about that?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Any other questions?

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

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

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

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

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

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

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

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

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

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

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

Any questions about that?

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

**Student:** [Inaudible.]

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

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

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

**Student:** [Inaudible.]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Student:** Aki.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[End of Audio]

Duration: 52 minutes

## Programming Methodology-Lecture10

**Instructor (Mehran Sahami):** Okay, I would just, even at this point, just email text. Might be easier. I think we need to get started.

Let's go ahead and get started. Couple quick announcements before we start. So one of them is that there are three handouts in the back, including your next assignment. And your next assignment's a little game called Breakout. How many people have ever heard of a game called Breakout?

Yeah, when I was a wee tyke, just [inaudible] to the game, actually. It was my favorite game ever. And it was just fun. And you're actually gonna be implementing it. How cool is that?

So that's one of your handouts. There's a couple other handouts. There's a section handout and a handout of some code examples.

One thing, just so you know, in the handout on Breakout, it actually will refer to a Web demo. So if you wanna sort of see how the game plays – because it's hard to capture an actual game in just screenshots. There's a few screenshots in your handout.

But if you wanna actually get an idea for how the game plays, there's a demo of the actual working game on the Web page. So if you go to the CS106a Web page, both on the Announcements page, as well as on the Assignments page for Assignments 3, there's a link you can click on for demo, and that will actually take you to an executable demo of the game so you can play it.

Assignment No. 2, as you all know, is due today. So quick, painful – hopefully it wasn't too painful for you. But we'll just see how long it took.

Anyone in the zero- to two-hour category? A couple folks, actually. Two to four? Good to see. Four to six? Ooh. Six to eight? Eight to ten? Ten to 12? Twelve to fourteen?

**Student:** [Inaudible] a lot.

**Instructor (Mehran Sahami):** Few folks on 12 to 14. Fourteen to 16? And 16-plus? Oh, wait, do I have any 14 to 16? No, there's a couple section leaders in the back row – 14 to 16. They're just – they're pulling for you because they're like, "Marilyn, you're making the assignments too hard." Hopefully I'm not. Anyone in 16-plus?

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** All righty. Thanks for letting me know. Once again, the world is normal, and the world is where you would expect it to be, which is hopefully, most folks are below kind of this 10-hour mark because it's what we're shooting for. Good times if you are. If you're not, if there was some kind of – I'd ask you to please be

quiet if you're just coming in right now. In terms of if you're in one of these higher sections, if there was some bug that just caused a problem for you, and you figured it out, and it wasn't a big issue, that's fine.

Sometimes, that will cause you a couple of extra hours. If it was really a conceptual issue, that there was something where you just didn't know what was going on – maybe you just wrote a bunch of code, and eventually, it worked, or you needed to get a lot of help for it to work, please come talk to me. Talk to the TA, Ben, or talk to your section leader to make sure those issues are clarified because we're just gonna keep building on the stuff that you've done. You'll actually see a lot more graphics today.

So with that said, I actually want to kind of ask you a sort of a “where we’re at” kind of question. So last time, we talked about perhaps the most critical concept in the class, which is classes, oddly enough for the name. So I wanted to ask, actually, a quick question. So far, with where we’re at and what we’ve done in the class, how many people are just generally feeling okay? If you’re feeling okay, raise your hand.

Okay. How many people think we’re going too slow? Couple folks.

How many people are just – are feeling like we’re going too fast, or you’re feeling overwhelmed? Okay.

So first of all, if you are feeling overwhelmed, if you’re out of the category of just fast and into the category of overwhelmed, that’s kinda why I asked those together. If you’re feeling overwhelmed, please come talk to me, or talk to your section leader, or talk to Ben.

We’re happy to spend as much time as you need to make sure you don’t feel overwhelmed, and you feel like you understand everything that’s going on in the class. Have no qualms about coming to talk to us. That’s why we’re here. It’s more fun when we actually get a chance to talk to you if you’re having any problems.

And in that sense, I think we’re kinda going at a reasonable pace, perhaps a little bit quickly. But that’s kinda how we have to be at to actually cover all the material in the class.

So with that said, are there any questions about classes right now, just [inaudible] anything we’ve done?

I wanna spend a little bit of time touching on classes before we dive into our next great topic today, which is graphics. So if we actually have the computer for a second, I just wanna briefly review the classes that we wrote last time.

So we wrote a class that hopefully is near and dear to many of you because many of you are instances of this class, which is a student. And we went through, and we talked about all the things that you actually wanna have in a class. And hopefully this is just review.

You have some constructors in the class. You can have multiple different constructors in the class. But here, we just have one where we take [inaudible] a name and ID, and that name and ID gets set to some instance variables or ibars that all students have, namely student name and student ID.

So I'll scroll down to the bottom, and here we have our private instance variables, student names, student ID, and units earned. These are the variables that every student's object has their own copy of. That's why they're instance variables as opposed to, say, a class variable.

This guy over here, because it's got a static, is called a class variable because it's got a final that's actually a constant, as hopefully, you've seen many times by now. And so all units – all students share units to graduate as the same constant value, 180.

So it's both a constant and a class variable. Most things that are constants make sense to be class variables because all elements of that class sort of share the same constant value. All students require 180 units to graduate.

So we did a bunch of things in here, and we kinda went through the constructor. We went through a few places where we said, "Hey, if you wanna be able to access portions of this class which are private, you can't directly access them from outside." That's why they're private. So no one can touch your private parts. And so what you needed to have was you needed to have these functions if you wanted to allow people to access them, that we refer to as getters.

And the reason why we refer to them as getters is because they start with the name "get." And what they do is they're public methods. So someone can actually call these methods from outside of your class. And they just return the value for some appropriate thing that the person would want. GetID would return student ID.

Now, you might say that's kinda weird. Why do I have these getters when student ID, I could just make public and let people access it directly? Why would I do this?

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** So you can't change it. If student ID was public, that's a public variable. People cannot only read what the value is; they can set what the value is. If I make it private, I control access to it, and here, I could let you read what the idea is by giving you a copy of the ID. I don't let you set it. The only way the ID actually gets set is when a student first gets created, and you get a student ID up here, which you have for life. So when you get created, you have the ID for life. This is actually true at Stanford.

As a faculty member, now, I have the same ID number, no joke, that I had when I was a student. It's kinda funky, and it's just like, "You are in our universal system, and you will be here for the rest of your life. And it doesn't matter if you go away, and then when you

come back, you will still be that same ID number because you can never change it once you're created." Same kinda idea going on over here with this class.

Now, sometimes we do allow someone to change something. And we'll have what we call setters as well. So for units, we not only have GetUnits, which returns the number of units earned. We also have what we refer to as a setter, which sets the number of units.

And you might say, "Okay, that's kinda odd, Marilyn. If you're allowing someone to set the units and get the units, why don't you just make that variable public?" because at this point, you're allowing someone to both set its value and get its value. What else are you gonna do with that variable?

And the reason why we still don't make it public is because of information encapsulation. The person who is getting the number of units you have and setting the number of units you have doesn't need to know how we keep track of that information. It could actually turn out that the way we get your units is we go through your whole transcript and total up all your units.

And we never actually store it as one value, and that's how we get it back for the person. And when they try to set it, if they try to set more units than you already have, we just create some dummy classes. Or if they try to reduce the number of units, we drop some of your classes, and that happens occasionally.

So it doesn't matter how it's implemented underneath the hood. It's information hiding. All the person needs to know is that they can get it and set it. In this case, it's simple because it is just referring to one variable. But – and you'll see that oftentimes is the case, but not always, which is why we still have this encapsulation. And so for some things, we have getters and setters.

And then there was a few other things here where we could increment the number of units someone has, or we could check to see if they have enough units to graduate by just checking if their unit count is greater than graduating – the number of units they need to graduate.

And last thing, which I said all classes need – and again, I will repeat this because all classes you write that are not programs. And any classes you write that does not extend, say, console program or graphics program, should always have something called ToString.

String is just text, and what this does is it just returns, basically, some piece of text that tells the person using this class what this object actually encapsulated. It doesn't need to contain all the information in the object. But here, we're gonna return the student's name and then the student ID number inside parens.

So that's a quick review of the class. And part of the reason I want to do a quick review is we're actually gonna extend this class, which means remember when we talked about classes in the days of yore, and we had classes and subclasses and superclasses?

Here's a student. All of you are students. Now, there's some specializations of students, as well. As a matter of fact, some of you are, for example, frosh, which is a kind of student. And some of you are sophomores. And some of you are juniors. And some of you are seniors.

The size of the boxes don't actually mean anything other than it's smaller. Some of you are grad students. Some of you are the dreaded other student. We won't talk about that. We'll just call you the other student because occasionally, you may not be any of those things, and we still need a way of keeping track of you.

But you're – all of these things are classes, which are specializations of Student, which means you have all the properties that a student does, and perhaps some special properties that you may have by being in one of these particular classes.

So let's actually write one of these – oh, let's say – let's write the frosh subclass. How many people in here are freshmen, frosh? Woo-hoo. So hopefully, I picked the majority class.

So we're just gonna go ahead and pick a – create a new class by extending the student class. So what we're gonna do is we're gonna build a class that extends the functionality of our existing student class. And in this case, what we wanna do is create a Frosh subclass of students. So we wanna create the subclass here called Frosh that is a Student, which means it's gonna extend Student.

So Frosh will have all the same properties that regular Students do, except for the fact that they start with 0 units. And I'm sure like many of you are sitting out there going, "Oh, no, no, no, Marilyn. There's a wonderful thing called the AP exam."

Yeah, let's just pretend the AP exam didn't exist, for the time being because there actually are some people, as sad as that may be, that start with 0 units. And so we're just gonna say that all Frosh start with 0 units.

Think of this as the number of units you've earned at Stanford. And I know there's even a few of you out there who've earned a non-0 number of units at Stanford. But let's just say it's 0. Be thankful it's not negative. You're like, "Oh, you're a freshman. You get negative 5 units. Thanks for playing."

Now, the one thing we're gonna do is we're gonna specially designate Frosh in the strings that display an object. So that two-string method we talked about – it has the name of a person and their ID number. For the case of Frosh in particular, we're going to actually designate them as Frosh just because your RAs love you, and they wanna know who are the Frosh?

So we're gonna have to find a way of saying, "Hey, if you want the string version of this object, we're gonna actually have the string contain the word "Frosh" in it.

So here's how we might do that. First thing we're gonna do is we need to define the Frosh class. So it's a public class, Frosh, that extends Student. So every Frosh is a Student. And then inside here, we need to have a constructor for the Frosh class or, well, for – to create an actual Frosh. It's not gonna bring an entire class into being at once. It's gonna bring each individual freshman into being. And it's gonna take in a name and an ID, just like a regular Student does.

Now, here's the funky thing. What's the super thing all about? If we're gonna initialize a freshman, what we need to say is, "Hey, as a freshman, we're gonna initialize you. You're gonna – we're gonna set your number of units to be 0 because that's one of the properties the freshman have – is their number of units start at 0. But you're also going to have the same initialization done to you that all regular students do."

So how do I refer to the initialization process, or the constructor, that regular Students go through because I'm writing a special constructor for Frosh. And unless I specifically say, "Go and do the same things that you would do for a regular student, they won't happen."

So the way I specify that is I say, "Super," which doesn't necessarily mean, "You're super," although it could. It could because freshmen, you're pretty super. I don't know if when you got your acceptance letter, there was that little handwritten note on the bottom. I remember when I got my acceptance letter, it was a different dean than there is now. And on the bottom, it said, "Super, Marilyn," and I think that was the last handwritten thing I ever got from Stanford.

So "super" means call the constructor of the Super class. So by calling Super a name and ID, what we're doing is saying, "What's the Super class of Frosh?" It's Student. So this will call the Super class of Frosh, which is Student, with name and ID, which means it's gonna go through that initialization process of setting name to be equal or setting student name – that instance variable to be equal name, and student ID to be ID, the stuff that we did back over here.

So let's just hop back over here for a second. The things that we do in the constructor of a Student – so all of this stuff we still wanna happen for freshmen, which is when we say Super name and ID, it's actually calling, essentially, this function, to initialize all of the fields of a regular student. And then it's gonna do some more stuff specifically for freshmen.

So when we come back over here and do our little Super dance, what we get is, essentially, the constructor of a Student being called to set up a Student, which happens to be a freshman, which is the particular case of student. So we're gonna do some additional work by setting the number of units equal to 0.

So that's how we set up the constructor. Now, the other thing I mentioned was we need to set a `ToString` method.

Now, here's the funky thing. You might say, "But Marilyn, student's already had a `ToString` method. So if I didn't do anything here, wouldn't all Frosh be able to give me a string version of what they are?" Yeah, they would, but we wanna create a specialization of it.

So what we do is we do something called overriding. And overriding just means even though your Super class already had a method called `ToString`, in the subclass I'm going to define a new version of that method. If the method has the same name and the same set of parameters, which in this case happens to be none, it does what's called overriding, which means for all objects that are of type Frosh, when you call `ToString`, you're gonna call this version of the method.

The version that exists in the Student is no longer germane for Frosh. It's still germane, say, for sophomores or juniors who may not implement their own version of `ToString`, but for Frosh, it's gonna call this version of it.

So overriding just means forget about the old version. I'm sort of overriding it with my new, funky version that's specific to this particular subclass.

Any question about overriding? Has to have the same name for the method and same parameter list.

So what we do here is remember, student ID and student name are private. And now, the funky thing about private is even when I extend the class, when I create a subclass, that subclass does not have access to the private elements.

And you're like, "Whoa, Marilyn, that's kinda weird. I have a Student, and inside a Student, I can play around with Student name and Student ID. But as soon as I create this thing called Frosh, which is a subclass of Student, it doesn't have access to those private elements anymore, which means if I wanna be able to get the name and the ID of the student, I can't just access the variables directly. I need to access the corresponding getters `GetName` and `GetID`."

So now if I call `ToString`, what I'm going to return is a string that says Frosh: and then your name and ID number. So that's how it differentiates it from the standard version of `ToString` is `ToString`. The regular one just provided name and ID number. This one actually sort of designates you as a Frosh because we know you're a member of the Frosh class.

Any questions about this notion of public versus private and why in a subclass, you still can't access the private portions?

Uh-huh? Is there a question over there?

**Student:** Yeah, so if the [inaudible] class [inaudible] Super class?

**Instructor (Mehran Sahami):** Yeah, so the way you can think of the subclass is the subclass is just another class. So the visibility that it has into its super class is the same visibility that any other class would have. So it can't access the private portions directly, even though it's a subclass. It needs to call public methods to be able to access those elements. It can still access any of the public elements. It just can't access the private elements directly, which is kinda funky.

That sort of freaks some people out, and there's this thing called protected, which eventually we'll get into, but we won't talk about it right now. All you need to worry about is private and public.

But if you sort of extend the class, you create a subclass. You need to understand that the subclass does not have access to the private elements.

Any questions about that? Are you feeling okay with that? If you're feeling okay with that, nod your head. Well, we're mostly heading on in, so that's a good thing.

So what I wanna do now is now it's time for us to draw pictures. It's just a lovely thing. This is really a day that's about graphics. And graphics is all about drawing pictures. And a matter of fact, for your last assignment, you drew a bunch of pictures, right? You – in Assignment No. 1, you drew some stuff that used the graphics library, and life was all good.

So we're gonna revisit that drawing that you just did for your last assignment and just soup it up. We're gonna bump it up so you can do all kinds of funky things like animation and games. And eventually, we'll get into where you can read a mouse clicks and the whole deal. And you will just be good to go. But we gotta build up from where we sorta started before.

So where we started before was the ACM graphics model, which is when you were writing graphics programs, you were using the ACM graphics model. And we talked about this before. It's a collage. You basically start with this empty screen, and you basically put little felt pieces onto the screen. So you're saying, "Give me a square," and, "Give me an oval, and throw it somewhere else." And you just kinda add these things to a little canvas, which is your empty screen to begin with.

Now, a couple things that you sort of may have noticed as you were doing the last assignment, or you'll certainly notice here. Newer objects that we add, so when we add things to our canvas, if it happens to have obscured something else, that's fine. It'll just obscure it. The newer things are sort of laid on top of the old things.

And this is something that we refer to as the stacking order, or sometimes we refer to it as the Z-order because if you're sort of like that Cartesian coordinate system person, the Z-axis comes out toward you. Here's the X-axis; here's the Y-axis. The Z-axis comes

toward you, which imagine if you stack these things on top of each other, if you think in 3D, that's kind of the Z-axis.

But if you don't think about that, think of these as just pancakes you're laying on top of each other. And you're stacking them. And the new stuff occludes the stuff behind it. And that's what we refer to as the stacking order.

So hopefully, this is all review for you. And as a matter of fact, this stuff should all be review. Many moons ago, oh, say two weeks ago, a week ago, we talked about ACM graphics, and we said there's this thing called a GObject, and a bunch of things that inherit from GObject are GLabel, GRect, GOval, and GLine – are all subclasses of GObject. And everyone was like, "Oh, yeah, I remember that. It was a good time. I was creating these objects. I was adding them to my canvas. It was just fun."

And then you look at this diagram, and you say, "Hey, Marilyn, why do you draw it so funky?"

And the reason why I drew it so funky is there's a whole bunch of other stuff that now it's time for you to learn about. So here's kind of we think about the ACM graphics package in the [inaudible], and you're like, "Oh, man, there's points and dimensions and compounds and these 3D rectangles and all this stuff going on."

You don't need to worry about all that stuff. There's just a whole bunch of stuff that comes along with Java and the ACM libraries. And it's kinda like you're out there, and you wanted to buy the basic car. And you got the car, and they put this jet engine on top of it, and you're like, "That's really bad for the environment. I just really wanna be able to drive my car without the jet engine." And they tell you, "Okay, well, the jet engine is there, but you never have to turn it on." And you're like, "All right. That's cool." And that's what we're gonna do.

So all of the stuff you see in yellow is stuff we're gonna talk about. And the rest of the stuff isn't really the jet engine. It's not that cool. Really, the rest of the stuff is '70s racing stripes and a big hood scoop and stuff like that. It's like, yeah, you could have it on your car if you really wanted to, but probably not in the 21st century.

So we're gonna focus on all the stuff that really is kinda important for what we're doing. And there's a few other things. You can read about them in the book, but we're not gonna spend class time on them. You're not gonna worry about them in this class. If you really wanna know, you'll know.

But we're gonna do all kinds of cool stuff and images and polygons and just things that will be interesting, hopefully.

So with that said, let's first talk about this thing called GCanvas. And GCanvas is this thing that kinda sits up there, and you're like, "Yeah, GCanvas is not one of these objects that I sort of put – that I kinda create and put up on my collage. What's that all about?"

So let's talk about that and get that out of the way, and then we can focus on all these other things that we can kinda draw and put up on our board or our little canvas.

So what a GCanvas is, is it represents, in some sense, the background canvas of the collage. It's the big piece of felt that we're gonna put all the other little shapes on top of. And you might say, "But Marilyn, didn't we already have one of these? When I create a graphics program, don't I already get sort of my empty canvas that I put stuff on?"

Yeah, in fact, you do. What a graphics program does for you automatically, just because it loves you that much, is it automatically creates one of these GCanvas objects and makes it large enough to fill the entire program window.

So actually, when you're adding your objects to a graphics program, you're actually adding them to a canvas or a GCanvas object. The graphics program has just created one for you seamlessly, so up until now, you never had to worry about it. As a matter of fact, for about the next five weeks, you're not gonna have to worry about it – oh, two weeks you're not gonna have to worry about it.

And so you might say, "But if that's the case, when I was calling Add, won't my Add call methods? When I was calling those, weren't they going to a graphics program because I never worried about this thing called GCanvas?"

Yeah, in fact, they were going to the graphics program. The graphics program had a method called Add. And when it got it, it said, "Hey, you wanna add an object? I'll pass them over to the GCanvas I created."

So what it was really doing was forwarding, just like you think of call forwarding – get a call from your friend, you're like, "Oh, yeah, you wanna talk to Bill? This isn't Bill. Let me forward you over to Bill, and you can talk to him."

We call the Add method on graphics program. Graphics program said, "Oh, yeah, you wanna add that? Well, the person who really takes care of the adding is the canvas, so I'll just call Canvas passing in the same arguments to Canvas that you've passed to me." That's called forwarding. It's basically just some method that sits there that actually passes all the information that goes [inaudible] someone else to actually do the work. We also refer to that in the working world as management. So you basically forward on to something else that does the real work.

And so forwarding, to put it in the speak of object-oriented programming, is when the receiver of a message, so before, graphics program was the receiver of the message, then call some other object with that same message. So that's how we'd say it to sound real funky and get paid more money. Same kind of idea.

So it turns out that GCanvas and GProgram – or sorry, graphics program, which is really just forwarding a bunch of your calls over to GCanvas, support a whole bunch of methods, some of which you've seen. And I wanna give you the quick tour.

So there's add, and you've certainly seen that. It just takes some object, some GObject like a GRect or a GLabel, and adds it to the screen. And you can add some object at a particular x-y location if that object doesn't already have some existing x-y location. So there's two versions of that method.

Up until now, we've told you had to add things. Now you're sort of old enough to actually say – it's like training wheels. Before, you could add training wheels. Now you can remove the training wheels. So for an object, you can actually say, "Remove," and it will take that object off of the canvas – just rips it right out.

And if you wanna get really funky – if you're having a bad day, and you just come in there, and there's this nice picture of bunnies and flowers and stuff, and you just say, "No, man, that's not my world. RemoveAll." And it's just all gone. All of the objects you put on that canvas is – it takes the canvas outside, shakes it out, and all the little, fluffy bunny pieces go away. And it's RemoveAll. It just clears it.

There's GetElementAt, and this is a very funky thing. You would use this in Breakout. Pay close attention. What GetElementAt does is it – you give it a particular x-y location, a pixel location on the screen. What it will return to you is the frontmost object at that pixel location if one exists on the canvas. If one does not exist on the canvas, it will return to you something called null. And null just basically means no object.

You can assign null to something of type GObject, but that just basically means that GObject is trying to deal with – is just there's nothing there.

Otherwise, it will actually give you back a GObject, which could be a specific thing. It could be a GRect, a GOval, a GLine, whatever it is. But all of those GRect, GOval, GLines are all of type GObject. So in fact, even if it gives you a GRect, it's still giving you a GObject back because the GRect is just a specialization of a GObject. So we'll actually give you back an object that's at that x-y location.

That might be real useful, for example, if you're playing a game that has a bunch of bricks across the screen, and when your ball is supposed to be hitting one of the bricks, you wanna check to see is there actually a brick there? You could call GetElementAt, and if there is some little brick, which is actually, say, a GRect, it will give you back that GRect object, which then, for example, you might wanna remove.

Just a few hints – uh-huh?

**Student:** On the x-y coordinate, is it – it's not the x-y coordinate that's the top left-hand corner, right? It's just [inaudible] –

**Instructor (Mehran Sahami):** This is an x-y coordinate of the whole screen of the canvas.

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** And if there is some object at that x-y coordinate anywhere – if that point is in – has an object that is – that point is encompassed by that object and will return the object – and just the foremost object. That's why you care about the stacking order because if there's multiple objects there, you get the frontmost one.

GetWith and GetHeight you've probably knew about when you were centering stuff. I guess the width in pixels of the entire canvas and the height in pixels of the entire canvas, which is for the entire screen if you're talking about a graphics program.

And SetBackgroundColor – that's kinda new, kinda funky. You wanna set a different background color than white, you just pass in a color, and it will set the entire color for that canvas or the entire window in the case of a graphics program to that color, which is kinda fun.

Now, there's a couple other things that only graphics programs do. GCanvases do not do these, just graphics programs. And they're very useful for games and animation.

One is called PauseInMilliseconds. What it does is [inaudible] computer these days run real fast. If you ran a game without any pauses in it, people would just not be able to play the game. So you can pause for some number of milliseconds, which is thousandths of a second. So what it does when it gets here is basically just sort of stops execution for some number of thousandths of a second and keeps going. So it allows your programs to kind of slow down a little bit to make them playable.

And sometimes, you wanna wait for a click, and there's, strangely enough, a method called WaitForClick. And it suspends the program. Basically, your program just stops executing until someone hits the mouse button. And then once they click on the mouse, it keeps executing again.

So those are some useful things for games or even for debugging sometimes, if you're doing a graphics program, and at some point you wanna stop and say, "Hey, let me stop the program here and see how far it's gotten," and I don't want it to go any further. So I click. You could just stick a WaitForClick call inside your graphics program.

So here's the class hierarchy for just the GObject. This is the stuff we're gonna focus on here. You've already seen GLabel, GRect, GOval, and GLine. We're gonna spend a little bit more time on GLabel to get into the specific nuances of it. And then we'll talk about some of the other stuff as well.

So here are some methods that are common to all GObjects. So everything that is a GObject, which includes all of this stuff, is gonna inherit all of these methods. So a lot of these you've seen before. SetLocationForAnObject – you've set its x and y location; Move, which you've set its offset by some dx and dy, which is just how much in x direction and how much in the y direction your object should move. That's real useful for animation, and I'll show you an example of that in just a second.

GetXAndY – this just returns the x coordinate of the object and the y coordinate of the object, which, for example, would be the upper left-hand corner for something that's large.

GetWidthAndHeight – this is actually pretty useful. Turns out if you have some rectangle, you know what its width and height are. Later on, you might wanna, if you forget – you forget the [inaudible], you could actually ask it what's your width and height?

This is actually really useful for text because if you wanna draw some piece of text centered in the screen, you actually don't know how big those characters are until you actually figure out what its font is and all that stuff. So you – a lot of times you wanna create your text. I'll show you an example of this momentarily. And then after you've created the GLabel, say, "Hey, what's your height and width so I can center you appropriately on the screen?"

Contains, which is also kind of similar to GetElementAt, but a little bit different – it actually returns true or false. It basically returns true if an object contains the specified point. So this same method you call on a particular object. So you can tell some rectangle, "Hey, do you contain this point, rectangle?" And it'll say "yes" or "no" or "true" or "false" if it actually contains that point.

SetColor and GetColor – hey, it's a pair of setters and getters, just like you saw with students. Uh-huh, question?

**Student:** [Inaudible] rectangle, or [inaudible]?

**Instructor (Mehran Sahami):** Pardon?

**Student:** [Inaudible] contains the area of the shape, or the [inaudible]?

**Instructor (Mehran Sahami):** It has to do with the – what the particular shape is yourself. So this is one of those things where I would ask you to actually do it experimentally. So there are a lot of things you can just try out. Just try the experiment. If you want, you can read it in the book, but it's actually more fun to try the experiment because you'll get somewhat funky behavior.

GetColor and SetColor, as you would imagine, sets the color of the object or gets the color of the object.

Here's one that we haven't talked about so far. It's kinda funky – SetVisible and IsVisible. So SetVisible – you set it to be either false or true. If it's false, the thing becomes invisible. If it's true, it becomes visible. You might say, "Hey, Marilyn, how is that different than Remove? I thought Remove takes an object off of the canvas."

This is not taking an object off of the canvas. It's just making that object invisible, which means if you wanna have some object on the canvas and flash it, for example, to be there or not be there, all you need to do is set its visibility to alternate between true and false.

You don't need to keep removing and adding it back on because removing it and adding it back on potentially changes the Z-order because that object now gets tacked onto the front. If this object was in the middle somewhere, you don't wanna change the Z-order by removing it and tightening it back onto the front. You just wanna make it invisible where it is. And you can ask an object for its visibility.

Last, but not least, if you wanna change that Z-order – if you're actually a big fan of drawing programs, a lot of these methods will look familiar to you. `SendToFront` and `SendToBack` brings an object either the front or the back of the Z-order. `SendForward` or `MoveForward` moves it back one level in the Z-order if you wanna actually just re-order stuff in the Z-order of objects.

Uh-huh?

**Student:** If you set an object to be invisible, and then you [inaudible]?

**Instructor (Mehran Sahami):** I knew you were gonna ask that. Try it. Try it, and you'll find out because a lot of these things are actually interesting, and then you realize that you would probably never do this in real life. But if you wanna try it out, it's more fun to be experimental than just giving you the answers for a lot of these things because then you'll never try it. So I want you to at least try a few of them.

Methods defined by interfaces – what does that mean? What is an interface? So there's this funky notion we talk about in computer science – or actually, it has a specific meaning in Java. We talk about it in computer science in general as well. But it's the notion of an interface. And an interface – sometimes people think of, "Oh, is that like a graphical interface? Is that like using my mouse and little things that appear on my screen?"

That's one kind of interface. That's the interface that humans work with. There are interfaces that programs work with. And basically, all an interface is – the way you can think about this is it's a set of methods. That's what, in some sense, defines the interface. And why do you care about defining some particular set of methods? Because what you wanna be able to say is there's a set of object or set of classes that all have these methods.

So you have some set of classes that have that set of methods. That seems like kind of a funky thing. Why would you wanna do that? You might say, "Well, hey, Marilyn, there's kinda a similar concept. Isn't that whole concept of inheritance you talked about sort of like this because if you have over here your `GObject`, and you have something that's a `GLabel`, and you have something else that's a `GRect`, you told me a `GObject` has some set of methods, so doesn't the fact that `GLabel` and `GRect` are both `GObject` – aren't they some set of classes that have the same set of methods?"

Yes, that would be true. So then you might ask yourself, “So what’s different about this thing called an interface than just inheriting from some class?” And the difference is that sometimes you want this set of methods to be shared by a set of classes, which don’t have this kind of hierarchical relationship.

So some – an example of that might be, for example, a class called an Employee. And students, for example, can be employees, but there are gonna be people who are not students who are employees as well. And there might be some entirely different class of people who are employees. So if I had something called an Employee here, and I might say, “Hey, well, at Stanford, I have a bunch of different specializations. I have my Frosh Employee, and I have my other Employee over here that’s a Senior Employee.”

And then Stanford comes along and says, “Yeah, but there’s also this thing called a Professor, and not all Professors are Employees.” And you’re like, “Really?” Yeah, sometimes it happens. It’s weird, trust me.

But it turns out that sometimes you cannot only have some person who’s a Senior Employee who’s an Employee but some person who’s a Senior Employee who’s also potentially a Professor.

And you’re like, “But Professors have some methods associated with them – not many, but they have some.”

Employees have some methods associated with them. So there’s these different sets of methods that I want, for example, to have one of these guys to be able to share. But there isn’t a direct hierarchical relationship. That’s what you specify in an interface. You basically just say, “If Interface is the set of stuff,” and I’ll tell you which classes actually provide that set of stuff. And they don’t have to have any kind of hierarchical relationship, but just the way of decoupling the hierarchy from having some set of common things that you’d like to do.

So let me show you some examples of that in the graphics world. So there’s something called GFillable. That’s an interface. So GFillable is a set of methods that a certain set of classes have, and the certain classes that have it are a GArch, a GOval, a GPolygon, and a GRect. So notice you might say, “Hey, Marilyn, yeah, those are all GObjects. Why isn’t everything that’s GObject GFillable?” Because there are some things like a GLabel that’s a GObject that’s not fillable. So that direct hierarchical relationship doesn’t hold. I have some subset of the classes that actually have this.

And Fillable are just the things that can be filled. So SetFill I set to be either true or false, just like you’ve done with GRects or GOvals in the past that fills it – that either sets it to be just an outline or filled. I can ask it if it’s filled by saying IsFilled. That returns a Boolean true or false if it’s filled. I can set the fill color. I can get the fill color – getters and setters once again rearing their ugly heads.

So Fillable is just a certain set – it's an interface, and there's a certain set of classes that – what we refer to as implement that interface. That means they provide all the methods in that interface.

And there's some other interfaces. There's Resizable. GImages, GOvals, and GRects are resizable, which is kinda funky. And Resizable just means you can set their size, so you can set the dimensions of the object to be different after you've created the initial version of the object. Or you can set the bounds of the object, which is both its location and its width and height. You can change those after you've created the object. A GLabel I can't do that with.

And there's one more set of interfaces, which is called Scalable or GScalable. And a whole bunch of things actually implement GScalable or provide for you the set of methods in the GScalable interface. GArcs, GCompounds – some of these we haven't even seen before, and you're like, "Marilyn, what's a GArc?" Don't worry, we'll get there. You've seen GLine. You've seen GOval. You've seen GRect. You'll get all the other Gs in there.

It's all about the G today. That's just what's going on. And so you can scale these things, which you give it some scale factor, which is a double – a value of 1 point [inaudible] means leave it unchanged. That's basically 100 percent.

Now, you can potentially scale it. If you give it a value like .5, it means shrink it by 50 percent in the x and y direction if you give it this version. If you give it this version, you can actually set the scaling in the x and y direction to be different, and I'll show you an example of that momentarily. It's kinda funky. It's fun. You can destroy your pictures. It's easy. It's one method.

So let me give you a little animation just to put a few of these things together – a little bouncing ball. So if we come over here – and you should have the code for this in one of your handouts – I'm gonna create a little bouncing ball. And I'll go through some of the constants pretty quickly. The ball has a diameter and some number of pixels. There's a gravity, which is at every time step, how much more quickly is the ball going downward? How much is it affected by gravity? So every cycle, its speed is increased by 3.

[Inaudible] some delay in number of milliseconds for the bouncing ball? Otherwise, it'll just go way too quick, and you won't see it, so it'll have a 50-second millisecond delay between every update of the bouncing ball.

The starting x and y location of the ball is basically just at – y is at 100; x is near the left-hand side of the screen because it's basically the ball's diameter divided by 2. The ball has a constant velocity in the x direction, which is 5. And every time the ball bounces, it loses some speed, so how much of its speed does it keep? It keeps 90 percent of its speed, basically. That's what the ball bounces. It's just – what fraction of its speed does it keep as it goes along?

And so our starting velocities for x and y – or the x velocity’s never gonna change. It’s just gonna be set to be whatever the starting x velocity is. We’re never actually gonna change it. The y velocity, which is how much the ball is moving in this direction, starts at 0, and over time it’s gonna increase until it hits the ground under the bottom of our screen, and it’ll bounce up. And we’ll see how to implement that.

And our ball is just a little GOval, so I’m gonna have some private instance variable that’s a GOval that’s the ball I’m gonna create. And then I’m gonna move it around.

So what am I gonna do? First of all, in my program, I’m gonna do this thing called Setup. What does Setup actually do? So when I do the Run, I call Setup. What does Setup do? Setup says, “Create a new ball at the starting x and y location with a given diameter in both the height and the width.” So it’s basically just creating a circle at that location. It sets the ball to be filled, and it says, “Add the ball to the canvas.”

So basically, all Setup does is it adds the ball somewhere – an oval, a GOval, that’s filled somewhere onto the canvas. That’s great. That’s all Setup does. It created the ball, added to the canvas. Now, how are we gonna animate it?

The way we’re gonna animate it is as long as the ball’s x coordinate is not yet the width of the screen, which means the ball’s gonna sort of bounce across the screen like this – until it’s gone to the end of the screen, I’m just gonna keep running the animation. Once it sort of bounces off the right-hand side of the screen, game over. It’s gone past the width of the screen.

So what I’m gonna do on every iteration is I’m gonna move the ball. I’m gonna check for a collision to see if the ball has hit the bottom of the screen. And after I check for collision, which is if it hits the bottom of the screen, I need to send it back up, I’m gonna pause before I move the ball again.

So that’s my cycle. Move the ball, check for collision, wait. Move the ball, check for collision, wait, kinda like standard animation.

So what is MoveBall and CheckForCollision do? MoveBall’s extremely simple. It just says on every time step, I increase how fast the ball is going down, which is its y velocity, by gravity. So I plus-equal its current y velocity with whatever gravity is.

Uh-huh? Question?

**Student:** Why do [inaudible] oval at the top and then [inaudible]?

**Instructor (Mehran Sahami):** Well, when I declare it, all it said is set aside the space for that object. When I say NewGOval, it actually creates the object. So I need to create the object before I can use it. By just declaring the variable, I haven’t actually created the object.

So up here, when I declare this GOval up here, this private GOval, that just says, “Give me the box that’s gonna store that object that’s a GOval.” I don’t actually fill it up with a GOval until I do the New. So I have to do that and do that to actually create the object.

So MoveBall, I just showed you. We accelerate by gravity, and we move the ball in the x direction and the y direction. The x direction’s always gonna remain constant, so the ball’s just slowly gonna move across the screen, but its y velocity’s gonna change because it’s gonna bounce.

CheckForCollision looks way more funky than it is. Here’s all that’s going on in CheckForCollision. I check to see if the y coordinate of the ball is greater than the height of the screen minus the ball’s diameter. If it is, that means the ball has not hit the bottom of the screen yet because it’s y – sorry, if it’s greater, it means the ball has hit the bottom of the screen because it would be below the bottom of the screen because y coordinates go down.

So if the ball’s y is greater than the height of the screen minus its diameter – because we have to give the ball some room to be able to hit the bottom of the screen, which is the diameter, then if it hasn’t fallen below the floor, there’s nothing to do.

If it would’ve fallen under the floor, then what I do is say, “Hey, you’re gonna bounce off the floor,” which means when you hit that floor, change your y velocity to be the negative of what it was before, which means if you were going down before – your y was increasing – now you’re gonna be going up, or your y is gonna be decreasing.

So change your velocity to be the negative of what it was before multiplied by the percentage of your bounce that you’re keeping, which is 90 percent. So it’s gonna slow down every time it bounces in terms of how far up it’s gonna go, but just by whatever this percentage is. But the important thing is you need to reverse its direction because it’s bouncing off the ground.

Last but not least, and this is just kind of a minor technical point, but I’ll show you anyway. It’s all in the comments. If the ball would’ve fallen through the floor, like it was actually moving so fast that it actually was here in one time step, and next time step, it would’ve been below the floor, we just say, “Take the difference from the floor, and pretend you already bounced off the bottom.”

So we’ll just kinda add that over to the topside, and you’ll just essentially take that amount that you would’ve bounced through the floor and say that bounce already happened and go back out. Slight technical point, but this is the math that does it – not a big deal.

What does this actually look like? It’s a bouncing ball. Ah, animation. You can play games, and then the simulation ends when it goes off that end of the screen. It’s just so cool, you just wanna run it again, but I won’t because we have more stuff I wanna cover.

But now you, too, can bounce the ball, and you can make it different colors. And every time it hits the bottom, you can set RandomColor, and your ball changes every time it [inaudible]. And you're just like, "Oh, my God, there's so much I can do. It's so cool." Yes, there is. And we won't explore it all right now.

But what we will do is explore a couple more things. So our friend, the GLabel class – you're like, "Oh, but Marilyn, I already saw the GLabel class. I've been doing GLabel since the cows came home. I did GLabel on this last assignment. I know how to create a new label in a particular location, to change its font, to change its color, and to add it. So what new were you gonna tell me?" And on that slide, I'm not gonna tell you anything new.

What I'm going to tell you new is on the next slide, which is what's the actual geometry of this GLabel class if you wanna center stuff on the screen, for example?

So there's a bunch of typesetting concepts that are related to GLine – to GLabel. First of them is the baseline. The baseline is basically the line on which the text appears. But notice there are some things that actually go down below the baseline. That's the way typesetting works. You actually have a baseline – some characters, like a j or a p, go below the baseline.

So when you're setting up a particular GLabel, the value that you set for where that GLabel should actually be placed is the origin of the baseline. It's the far left-hand corner of the baseline. It's not down where you get these descending characters. It's the baseline that most characters are actually sitting on.

So then you say, "Well, how do I get the descent and the ascent and all this other stuff?" Well, before you do that, the height of a font is its distance between two successive baselines.

But more importantly, if you're just trying to center one line in the screen, you care about two things called the ascent and the descent. The ascent is the amount above the baseline by which the tallest character can actually reach. The descent is the distance that any character will drop below the baseline. So for ys and gs and js and stuff, you actually have some non-zero descent, which is the amount below the baseline.

So if you wanna figure out how big something is, usually when we center stuff, we just care about the ascent. We don't actually care about the descent. So if you wanna center something, here's an example of centering something. We again have our little text, "Hello, world." We set it to be big. We set it to be red. We haven't figured out where on the screen to put it yet because we're gonna compute that after we set the size of the font so we know how big it is. Now we can ask this label, "How big are you? How wide are you? How tall are you?"

So to get in – to get it to display right in the center of the screen, its x coordinate is gonna be the width of the whole screen minus the width of the label. So we're gonna take the

width of the whole screen, subtract off the width of the label, and divide the whole thing by 2. That will give us this location right here.

The other thing we need to do is figure out how to center relative to the height of the characters. What we do is we get the ascent. We don't care about the descent of the characters. Standard typesetting kinda concept, but now you know. You get the ascent of your font, which is how high it goes above the baseline. You subtract that off from the width of your window, and you divide by 2. That will tell you what the point is right there in terms of the y direction to be able to figure out how to center this text. And then you add it at that particular y – x-y location.

So now that you know how to actually make your font bigger or whatever, you can figure out the size of your text so you can appropriately space it on the screen.

Couple more things before we wrap up for today. Last thing I wanna cover is this thing called the GArc class. What is a GArc? Actually, there are a couple things I wanna cover. The GArc class is basically just drawing an arc somewhere. It's a little bit more complicated than it looks. Friends of mine that are artists tell me that you can draw everything in the world using just straight lines and arcs. Is that true? Yes? No? Maybe? Let's just pretend it is.

You already know how to draw a line. You have GLine. So now that you know how to draw it – GArc after a couple more slides, you can draw anything in the world. It just might take you a whole bunch of little objects to draw it.

So an arc is formed by taking a section from the perimeter of an oval. What does that mean? Basically, it looks something like this. The steps that are necessary to define that arc is we specify the coordinates and size of some bounding rectangle, just like you did with the GOval. A GOval sits inside some rectangle, and that's what tells you what the dimensions of the oval are. Here, I've drawn it as a circle.

So here's a bounding rectangle, and it's got some upper left-hand location. So we have some x-y upper left-hand location. We have a width and a height. That's one of – what's gonna define for us what essentially the oval is gonna look like.

But we're not gonna draw the whole oval. We're just gonna draw an arc from the oval. How do we specify the arc to draw? We need to specify something called the start angle and the sweep angle, so [inaudible] the start and sweep angle. Both of these angles are things that are measured in degrees starting at this line, so starting at the X-axis.

So if I say for my start is 45 degrees, it goes up 45 degrees and says your start would be here. Your sweep is what portion of the oval you actually draw out in degrees. So if I set a sweep of 180, I will draw out a semicircle because I'll draw 180 degrees starting at the 45-degree mark. And it always draws in the counterclockwise direction if you give it positive values. If you give it a start value or a sweep angle in negative values, it will start on the other side and go in the clockwise direction.

So let me show you some examples of that, because that's kind of a big mouthful. What does that really mean? Here's just a few examples so you know. We're gonna draw some arcs centered in the middle of the screen, so we're gonna figure out dx and dy. Well, let's just assume they're the middle of the window. We already computed them using GetWidth and GetHeight. And d, which is the diameter of the oval we're gonna be using, or the circle in this case, is just preconfigured to be .8 times the height of the screen.

So if we do something like we say, "Create a new GArc that has its bounding box being d and d, that basically means we're gonna have some oval that's height and width are both the same, being d, which means it's just gonna be a circle with diameter d."

We're gonna start at the 0-degree arc. Remember, 3:00 is where we start – 0 degrees. And we're gonna sweep out an arc of 90 degrees, and that goes in the counterclockwise direction because it's positive. So we get an arc that's essentially this little quarter of that circle because it's 90-degree arc starting at 0 degrees.

There are some other things we could do. We could, for example, say, "Hey, you know what I wanna do is start that same circle with the same size. Its height and width are both d. Start at the 45-degree mark, so coming up 45 degrees from sort of this 3:00 mark is right there. Have a sweep of 270 degrees, so we get this big portion of the circle that looks like a giant C. But that's 270 degrees of a circle."

We can do some slightly more funky things using negative numbers, for example, where we say, "Draw a new arc. Your starting point is negative 90 degrees."

What does that mean? It means 90 degrees going clockwise. So I start at 3:00. I go 90 degrees clockwise. That's right here. And then I sweep out an arc of 45 degrees, which is counterclockwise because it's positive, so it sweeps out 45 degrees.

One more for the road – this one says, "Start at 0 degrees, and set your sweep to be negative 180 degrees. So if it's negative 180 degrees, it goes in the clockwise direction," 180 degrees starting from 0, and we get this thing that looks like half of a circle or a smiley face.

So any questions about GArc? There's all kinds of stuff you can do with GArc. But at least if you understand the basic geometry, hopefully it'll make sense.

Uh-huh?

**Student:** Why [inaudible] fill a GArc?

**Instructor (Mehran Sahami):** You can fill a GArc.

So last slide before we go is a filled GArc. Good question. Basically if you write a GArc, whatever that GArc is, and you set its fill to be true, what you get instead of just the arc is you get the pie wedge that would've got associated with that.

So just imagine you have some [inaudible] line that sweeps the arc. That's what you get with the filled version of it.

Any questions?

All right, then I will see you on Wednesday.

[End of Audio]

Duration: 51 minutes

## Programming Methodology-Lecture11

**Instructor (Mehran Sahami):**Hi, all. Welcome to CS106A. Let's go ahead and get started. If you're on your way in, just feel free to come on in. You can always get the handouts at the end.

A couple quick announcements before we dive into things. There's two handouts today. There's a whole bunch of coding examples. There's coding examples up the wazoo. They're sort of like – I gave you a bunch of code that might just be useful for you to look out for Breakout, just because it's good times.

So also related to Breakout – no, I think we just change the way we say this. It's just Breakout, and so any time you see people on campus – every quarter I come up with some random thing that I wanted to try to get people on campus to say. And I've never gotten one to take off seriously. Well, let's see if we can actually get Breakout to work. So just practice that.

On the count of three: One, two, three. Students:

Breakout.

**Instructor (Mehran Sahami):**Yeah, good times. All right. See, sometimes you gotta find these little [inaudible] of humor yourself.

Anyway, the note on Breakout is that first of all, there's actually a version of it that's available on the Web site, so you can actually play around with. The other one is we've talked a lot about classes so far. And with the initial project that we give you for Breakout, there's one class in there, you don't actually need to write a whole bunch of separate classes for Breakout. It's perfectly reasonable to write all your code in that one class.

If you want to add additional classes and have additional files in the project, and do it like we did in class, that's perfectly fine, and if it makes sense for you. But you shouldn't necessarily feel obligated that, "Oh, we talked about classes, so we have to do that for Breakout." For Breakout, you're actually fine. Classes can potentially provide some [inaudible], but they're not a requirement for the assignment.

So just couple other quick questions. In class recently – so now that we've had a few classes, you can sort of get the feel for – we do some stuff on slides. Occasionally, we do some stuff on the board. And so I wanna get a preference. If you have a – if you get a notion – if you have a preference for slide versus board.

How many people prefer slides?

How many people prefer the board?

Interesting. All right. How many people like some combination of the two? You can vote again if you want.

All right. So we'll try to do that. All righty. So today's gonna be a little slide-heavy day because we didn't have the vote so far. But we'll try to do some more board stuff as well.

So here are the class hierarchy you saw before. And we covered a lot of the class hierarchy last time. But there were still a few bits that we have left to cover, which is GImage, GPolygon, and GCompound. So you'll see those. And then you'll be ready to do all kinds of cool stuff today.

As a matter of fact, today we're gonna break out things you can do with a mouse and graphic – and more graphics kind of stuff with the mouse. And it's just a whole bunch fun [inaudible] doing Breakout.

So GImage: All the GImage is, and this is a funky way that you can put pictures inside your program, like pictures that you take with a camera or whatever or your phone. You can just stick them in your program. And the way you do this is there's a class called GImage. And so you can create an object of type GImage by saying, "new GImage." You give it the name of the file. That's the full file name that with a dot and whatever ending it has, and then the xy location for where you wanna display that image. The xy is the upper left-hand corner of the image.

So the file – image file is the name of the file containing the image, and x and y, as I just mentioned, are those coordinates. Now, where you have to stick this file – that's something to remember. So when it looks for the this file, what it will do is it will look in your current project folder, and if it does not find that file name in your current project folder, then it will look in a subdirectory named "Images."

So you can have the subfolder if you have a whole bunch of pictures called "Images" inside the folder for your project. Put all your images there. It will also look there as well.

If it's not in either one of these places, bad things happen. And if you wanna know what bad things happen, you can actually try it. But I wouldn't encourage you to try to do bad things. So just stick in the folder with your project, and you'll be fine. And you'll get the image.

So let me show you an example of what that actually looks like. Just so you know, both .gif or "jif" formats. Some people say "jif," even though the "g" stands for "graphics" over here, which is kind of a hard "g," and then .jpg, which is another graphics standard, and you'll see the files either in .jpg or .jpeg, which just another format. Both of them are supported, and so most cameras or phones or whatever actually end up taking pictures in one of these formats.

So here's a simple example. So if we had some program run, we can say I wanna have some image that's a GImage. So I create one, and let's say I have to have the Stanford

seal, which hopefully we have rights to, I would hope. It's probably some guy in Kansas actually has rights to the Stanford seal and charges every time we use it. It would not be the first time something's like that has happened. But that's not important right now.

What is important is that Stanfordseal.gif [inaudible] – it looks for that file inside the folder for our project. And then it's gonna – we're gonna add that particular image and that location 00. So here, I did it the way that I just specify the file name without giving the x and y coordinates. You can give it the x and y coordinates if you wanna begin with.

But if you don't, it's kind of funky to say, "Hey, load that image." And now I can get the size of that image so I can center it in the screen or whatever, if you wanna do that.

You can also – what's also sorta fun is you can remember GImage [inaudible] implements this particular interface called resizable. So we can actually say, "Hey, image, I want you to scale one and a half times in the x-direction and .5 times in the y-direction." It's kind of like Stanford comes to school and checks into the dorm and gets on the meal plan and puts on the freshman 15 and is just like, "Oh, oh, oh, I'm Fat Stanford now."

But you can take pictures of your friends and just distort them in funky ways. It's just a good time. And so you can do all kinds of – a bunch of different options on things like images. That's just one of them.

There's also this class called Polygon. And Polygon's kinda a fun, kinda a interesting, cool class. And what it lets you do is it lets you represent graphical objects bound by line segments. So we need to get a little bit more formal. But here's a little diamond. Here's a hexagon. They're just polygons. And all a polygon basically is is just something that has multiple sides on it. That's where "poly" comes from. It just means many.

So the basic idea that's interesting about polygons is a polygon has a reference point. So when I create a polygon, I'm actually gonna tell the computer sort of all the points on the polygon. All of those points that I specify, like let's say the four corners of a diamond, are all in some relation to some particular reference point that I pick.

And so in this case, I might, for example, pick the center of the diamond. And in fact, in most cases, all of the vertices that I lay out, since they're gonna be relative to some reference point, that reference point often is convenient to pick the center. It just turns out, and it doesn't have to be. You could actually have your reference point be upper-left corner of the bounding box or whatever you want.

You just gotta make sure that when you sort of tell it what the vertices are that you lay them out in the right order. But oftentimes, the center of the polygon, if it's a regular polygon, which means it's sort of symmetric all around, then the center is often easiest.

So I'll show you some examples of this, how you actually construct one of these GPolygon objects.

So the first thing you do is you say, "Create an empty polygon." So you create empty polygon object. And then what you're gonna do is you're gonna specify the vertices of that polygon one at a time using a method called addVertex. And addVertex can take an x and y coordinate. These x and y coordinates are relative to your reference point. So if you think of your reference point as being the center of the object, and you say, "Oh, my xy is 1,1," that's one pixel over and one pixel down from wherever you think of your reference point as being.

You never actually specify the reference point to the GPolygon. The reference point's just in your mind. All of the vertices are just relative to whatever reference point you happen to pick. And I'll show you an example of this to make it concrete in just a second.

After you set an initial vertex – so you need to set the first one with something called addVertex. After you set the initial vertex, you can set all the remaining vertices in one of two ways because we just give you options. We like you.

One way is just to keep adding vertices. And what – when you call a addVertex again, what it does is it adds a new vertex, again, relative to the reference point. And it essentially creates an imaginary line between the last vertex you just added and the first vertex you added. That's how you're getting edges of your polygon. You're sort of specifying the corner points.

Another way, instead of specifying corner points, is after you specify the first one, you can explicitly addEdge. And addEdge adds a new vertex relative to the preceding one. And here, it's going to do it with offsets of dx and dy.

So relative to where we were before, we actually specify sort of an offset in the x-direction, offset in the y-direction. It's kinda like you were almost creating a line, and it's gonna add an edge. This one here is using absolute coordinates relative to the reference point. That's kind of a – the key difference between these two things. And I'll show you an example of both of them.

One final thing you should know is the polygon is, in some sense, closed for you. So if you were drawing a diamond, and – let's back up so I can show you the diamond. If you're drawing a diamond, you might specify that as the first vertex, then this is the second, then this is the third, and then this is the fourth.

How did that know what the fourth and first are what should be attached to each other? It just closes it for you. So after you've added the last vertex to the polygon, basically what it does is it just automatically sort of links up the first and last segments, or first and last vertices, to sort of close the polygon for you. That's how you actually specify. So you never actually need to go back to the first vertex again. It just does it.

So that's kinda a whole bunch of stuff in theory. What does it look like in practice.

So we're gonna create a diamond. We're actually gonna create it using some method that we'll see called `createDiamond`. And so when we call `createDiamond`, we're expected to give us back a `GPolygon`.

Here's where all the interesting stuff's going on. So here's the `create-diamond` method. `CreateDiamond`'s gonna return a `GPolygon`. So it needs to create one because it's gonna return it back to us. And what it's getting is the width and height of what that diamond should look like. So the first thing it does is it says, "Create a new polygon." It creates a new, empty polygon, just like you sort of saw before.

Then what does it need to do? It needs to add the first vertex. So it adds the first vertex. And it's gonna do this relative to some imaginary reference point that I pick. The reference point that I'm gonna pick is going to be the center of the diamond, which means my first vertex in the x-direction is going to be minus width over two. So let me just show you where that vertex would be.

So if I think of the imaginary center, and I've just executed that line to add a vertex, I'll say relative to my imaginary center, my first vertex is at half the width of the diamond over in the negative x-direction, and on that same line, so 0 in the y-direction.

Then where do I go from there? Well, I'm gonna add the next vertex. The next vertex I'm gonna add, relative to my center, is going to be at the same x coordinate as the center, but it's going to be minus height divided by two upward. So what I get, essentially, is this is an x vertex, and it just kinda creates the segment in between them.

Where's my next vertex? It's gonna be over here. It's width divided by two for my imaginary center point and on the same line as the imaginary center point. So the y offset is 0.

And then last, but not least, the last vertex I'm gonna add is on the same x coordinate as the imaginary center. But the height is half the height of the diamond, downward. So I get that.

And now if I say, "Return diamond," it matches up the first and last vertex. So what I get back is a closed polygon. I actually get that diamond. And that's what's returned back to me in my "Run" function over here. So diamond is now this little polygon.

Question?

**Student:** Do you add [inaudible] you add the bottom to the top [inaudible]?

**Instructor (Mehran Sahami):** It depends on if you do something like `addEdges` versus `addVertex`. So again, I would try it. And especially if you try things like where your lines would've crossed, you'll see sort of funky things between `addEdge` and `addVertex`.

Uh-huh?

**Student:**Do you ever actually [inaudible]?

**Instructor (Mehran Sahami):**You never specify the reference point. All of the computation we did here are all relative to some imaginary reference point. But we never specified a reference point exactly. It's always a good idea, if you're thinking about creating a GPolygon, to identify for yourself what that reference point is before you draw the line segments.

Uh-huh?

**Student:**[Inaudible] is there a way to try to [inaudible]?

**Instructor (Mehran Sahami):**Yeah, you just figure out what the size of the image is by asking its height and width. And then you need to specify scaling factor to make it large enough to do the whole screen.

Uh-huh?

**Student:**[Inaudible] addEdge [inaudible]

**Instructor (Mehran Sahami):**Yeah, so let me actually show that to you right now. Here's the – so addVertex. We just returned the polygon. We set its fill to be true. We set its color to be magenta. We write it out in the middle of the screen – not a whole lot of excitement going on. The one thing that's important to note is when we add the diamond to our canvas, the location that we give for adding it to the canvas is the location of the reference point. So every time that we do something with the polygon once we've created it, the xy location that we specify for the canvas is the imaginary reference point.

So here are – middle point was get width divided by two – or sorry, get width divided by two is this way. Get height divided by two is the center of the screen. And because we use the center as the imaginary reference point for the polygon, it actually shows up in the center of the screen.

So actually, using GPolygons a lot of times is just convenient even if you wanna circle because you wanna specify the reference point of the circle to just be in the center of the screen as opposed to its corner.

Uh-huh?

**Student:**[Inaudible]

**Instructor (Mehran Sahami):**If you really wanna do – we should talk offline. We should talk offline.

**Student:**[Inaudible]

**Instructor (Mehran Sahami):** Yeah. There's a bunch of things we could do. Essentially, there's a bunch of stuff with polar coordinates that I should mention. We're not gonna deal – so if you read in the book about polar coordinates and fade and all that, we're actually not gonna do it in this class.

So if you're like, "Oh, polar coordinates." You're like, "Yeah, polar scares me," it scares me, too. Polar coordinates, you only need to worry about if you're a bear, or you're in the Antarctic.

We just not gonna worry about polar coordinates. We're just gonna do Cartesian coordinates. But there are – it's easier to do rotations if you think polar.

So in terms of adding an edge, we're gonna do the same thing for `createDiamond` using edges instead of vertices. So if we wanna create a diamond adding edges, we have to still specify the first vertex, just like we did before.

And now the edges – what I'm specifying for the values for the edge is an offset in the x and y direction. And my offset in the x-direction is width over two, so that would move here. And my height is negative height over two, so that would essentially move over here because it's relative to my last point. It's not relative to the imaginary reference point.

And so basically, if I sort of follow this all the way around, what I do is I `addEdges`. Again, after I add that last edge, the polygon is closed for me automatically. I get back the same diamond. I'm just now doing it with edges instead of with vertices, same sort of deal. I make it magenta because magenta's a fun color, and I write it out on the screen.

So that's `GPolygon`. Now, things get a little more funky. And the thing that gets the funkiest of all – it's kind of like the George Clinton of graphics class, is `GCompound`.

And so what `GCompound`'s gonna do – anyone know George Clinton? All right, a couple, but at least it's not totally lost. `GCompound` basically is as you'd imagine. It's a compound shape. It allows you to take multiple other shapes that you've seen and put them all together and treat that one thing as one object, which is very convenient sometimes if you wanna draw something complicated and then move it all around the screen, for example, or rescale it.

So adding objects – the way a `GCompound` works is you add objects to a `GCompound` just like it was a canvas. So just like you've done before where you say, "I have some canvas, and I put these little objects on it, and it draws pictures," if I have some `GCompound`, and I add a bunch of objects to it, I now have this compound thing that encompasses all of those objects.

So you can now treat the whole compound as one object, though, which is the whole reason for having this, and I'll show you the example of this in just a second.

So similar to a GPolygon, a GCompound also has a reference point that when you add objects to the GCompound, you add all the objects relative to some imaginary reference point that the GCompound has.

And finally, how do you display this thing? So when you add things to a GCompound, they're not displaying on the screen. You need to take the GCompound and add it to the canvas just like all the other [inaudible] or ovals or whatever you did before. And when you place it, you place it relative to its reference points, and it will draw all the objects that it encompasses.

So let's actually just see what that looks like. So let's draw a little face. Oh, little face, and so you can see we're gonna use some different things here. We're gonna have an oval. We're gonna have a couple of other ovals as the eyes. Hey, triangle – what should we use for the triangle?

**Student:**[Inaudible]

**Instructor (Mehran Sahami):**Polygon, right. Some people are like, "Three lines." No, GPolygon. It's your friend, really. It's a fun time. And here's a little rectangle for the mouth.

So what we're gonna do with this is we're just gonna go ahead and take a look at a class GFace. So the GFace class that I'm creating is just a whole separate class. I'm going to create this class GFace as extending GCompound. So a GFace is going to be a GCompound. And what I'm gonna do is I'm gonna have a whole bunch of constants here that just specify the width of the eyes and height of the eyes and all the sort of features of the face. And I'm gonna have some objects that comprise my little GCompound.

So I'm gonna say, "Well, the head's an oval, and the left eye and the right eye are both oval, and the nose is gonna be a polygon. And the mouth is gonna be rectangle." These are all my private instance variables. They're just the variables I'm gonna use to keep track of the things I create and add to my GCompound.

So the compound has a constructor, which is just the same name of the class. It's GFace, and what it's gonna take is the width and the height because I wanna allow the face to automatically be resizable, which means all the stuff that I create that's part of the face needs to be in terms of the width and height that are past, and so it will resize itself depending on whatever width and height the user gives me.

So the first thing I do is I say, "Hey, the head's gonna be an oval." And the oval is basically just a circle that's off-size width, height. It's gonna be the size of the whole head, so whatever size you give me is the size of the head.

The left eye is gonna have the eye width times width, so it's gonna scale by whatever width you give me for the face. I'll scale it by eye width, and same thing for the height. I'll scale it by eye height. And I'm gonna create two eyes, left eye and right eye. Notice

that at this point, I haven't placed where my left eye and right eye are. I've just created them.

The nose is going to have a function, or a method, associated with it called `createNose` that's going to return for me a `GPolygon`. And so we'll take a brief digression down here to take a look at `createNose`. So what's `createNose` gonna do? It's gonna have some width and some height for the nose. It's gonna create a new polygon and add vertices to construct a nose.

So the first vertex it's gonna add, and we're gonna think of the center of the nose as being the reference point, the imaginary reference point. So the first vertex is sort of the height or the bridge of the nose. It has the same x coordinate as the center of the nose, if we think of some imaginary triangle here.

Actually, let me just draw a little triangle so we can use the board. Here's the nose. Here's our imaginary reference point. Where's our first vertex gonna be? It's gonna be up here. That's the same x coordinate as our imaginary reference point, and half the height. You can just imagine these two are now equal – waving of hands. It's gonna be half the height upwards. So that's gonna set the first vertex here.

And then where am I gonna set the next vertex? The next vertex, relative to the center, is gonna be width divided by two over, so width divided by two over.

And then in terms of height, if we can see it over here, it's gonna be height divided by two downward. So this is the next vertex, and then the last vertex is over here. And that's gonna give you your little triangle for the nose.

So when we create the nose – losing my chalk. We're gonna return that polygon that's created here because this method is returning a `GPolygon`. And the place that gets assigned is back up here where we actually have a nose.

**Student:**[Inaudible]

**Instructor (Mehran Sahami):**So the nose is whatever I get back from `createNose`. It's going to be the polygon for my nose. And the nose is, of course, scaled by the size of the face. So we have nose with the nose height to scale the size of the face.

And the mouth, last but not least, is just a rectangle – again, is scaled by the height and width of the face given the mouth width and the mouth height.

So now I've created all my little pieces. How do I put them together to create the face?

Well, when I call add methods inside of here, what I'm creating, I'm creating a new `GCompound`. What I'm creating here is `GFace`, which extends `GCompound`. Since it extends `GCompound`, that means all the methods that exist for `GCompound` extend here. This is not a graphics program. This is a `GCompound`.

So when I call add here, I'm not adding it to the graphics window. I'm adding it to the GCompound. So addHead and 0,0 – my imaginary reference point for this face is going to be this upper-left-hand corner of the bounding box.

So if I say add, head is 0,0. If I think of this as 0,0, and this is an oval, it's going to add the oval here because what I'm specifying is the upper-left-hand corner of the oval.

Now where am I gonna add other stuff? I'm gonna add the left eye to this kinda funky equation over here. But basically, all it's saying is I take a quarter of the width of the face and subtract from it the eye width scaled by the size of the face divided by two. So basically, what it's gonna do for the x coordinate is do something on sort of this part of the face. It's gonna bring it over a quarter from where it would've been relative to the center of the screen.

And then I'm also gonna do something relative to the height wherefore the left eye and the right eye – they're both gonna be at the same height, where I basically just look at a quarter of the height down the face is where those eyes are gonna show up. And I have to do some accounting for the eye height. So if my eyes – if my face were really big, my eyes still show up at the right place.

So it's just a little bit of math. You can sort of trace through it by hand if you're interested. And the exact coordinates for where these things are going is really not that important as long as we get a light – right layout that looks like a face. The important thing is all these things are relative to our little reference point at 0,0, which is the reference point for the face. We haven't put anything on the canvas yet.

The nose is just gonna be at the center, so nose is gonna be halfway over on the width and halfway over on the height. So if this is our reference point over here, we go halfway over on the width, halfway down on the height, that's the very center of the circle. And because when we created our GPolygon, the imaginary reference point of our triangle was the center, it places it relative to the imaginary center point of the triangle. So we get the triangle right in the middle of the face.

And last, but not least, we're gonna put in – we're gonna add the mouth. And the mouth is basically gonna go near the bottom of the face. It's actually gonna go in the center of the face. This is just basically finding the center point for the x shifted over by the sides of the rectangle, and then it's gonna be three quarters of the way down the circle, which is kinda how we get this.

So we add all these things relative to this reference point. And now, if we actually want to display this, we need to write some program that displays it. And so we'll have a program called DrawFace. And all DrawFace does is pretty simple. It says, "I'm gonna pop up a little pop-up in the middle of your screen."

I'm gonna have a graphics program that's gonna have some face width and face height. And what I'm gonna do is I'm gonna create a new face, giving it the width and height.

And where do I wanna place this face on the screen? I wanna center it on the screen. The reference point for the face, though, is not the center of the face. It's this upper-left-hand corner. So I need to figure out, relative to that upper-left-hand corner – I'm losing another piece of chalk – how to center the face.

So what do I do? I get the width of the screen. I subtract from it the face width, and I divide by two. That's kinda the classic thing you did when you were trying to center, say, a rectangle. You would take the width of the screen, subtract off the width of the rectangle, divide by two. What it will essentially do is figure out the coordinate to place this guy so this whole thing is centered in the screen. And we do essentially the same thing in the y-direction as well.

So we look at the height of the screen. We subtract off the height of the face and divide by two, and that gives us how much we should go down.

So again, when we finally place this on the canvas, we're placing it – when we specify the point on the canvas, that point is the reference point of the whole GCompound.

**Instructor (Mehran Sahami):**

Are there any questions about that?

Uh-huh?

**Student:**[Inaudible] the face [inaudible]?

**Instructor (Mehran Sahami):**I could've actually done it as just creating a GPolygon object and adding everything inside of another program. But I wanted you to actually see it as extending a class just because that's a common way of doing the decomposition. You say, "This thing that I'm gonna create really is a GCompound, so I'm gonna create it as a derived class of GCompound." It's the more common way of doing it.

Uh-huh?

**Student:**[Inaudible] –

**Instructor (Mehran Sahami):**Oh, sorry.

Uh-huh?

**Student:**[Inaudible] compound [inaudible] in the compound [inaudible]?

**Instructor (Mehran Sahami):**Yeah, so you can think of there's a z-ordering of the compound that's just like a canvas. So you actually have it – you can have layering of objects. And objects will include other objects in the compound. And it's the order in which they're added.

So the funky thing also about doing this is remember our friend, the bouncing ball? Let me just refresh your memory with the bouncing ball. I like the bouncing ball. Doo, doo, doo, doo, doo, so I'm running. I'm running. I'm running. Here's our friend, the bouncing ball. Come on, bouncing ball. We're bouncing. We're bouncing. And you're like, "Ooh, the bouncing ball got bigger since last time." Yeah, I changed one constant. It got a little bigger. That's okay. You wouldn't have remembered if I didn't tell you. How many people were like, "Oh, I measured the number of pixels."

Well, here's bouncing ball. And what's the whole beauty of having GCompound and creating a new class out of GCompound because I don't want bouncing ball anymore. Ball is boring. I want a bouncing face. How do I create a bouncing face? Hey, I got this class over here, GFace. That's a good time. Instead of a ball, I'm gonna have this thing be a GFace.

You're like, "Don't do it, Marilyn." All right, well, now that I do that, I'm still gonna call that ball just so I don't have to change the ball everywhere in my program. There's one other thing that happens, though. When I create the ball, I'm no longer creating a G-oval. I'm creating a GFace. Notice GFace still takes a width and a height, so my parameters are unchanged. The only other problem is a compound, a GCompound, does not satisfy the criteria or the interface for being fillable.

So this little thing that make the ball fillable and set the fill of the ball to be true, you'll see, "Oh, little air condition over here," because GFace or GCompound in general doesn't have a notion of filling the whole compound, so we get rid of that.

And now we save this puppy off, and we're gonna run. And let's see what happens.

First thing, with just modifying 20 seconds worth of code, we didn't get an error, and we'll run bouncing ball. Dun, dun, dun, and I have bouncing face. Yeah, scaled to be just the right size. Life is good. Three lines of code to modify. That's the beauty of object-oriented composition is when I have something that's a ball, and I have – and a ball, a GObject or g-oval's a GObject, and I have something else that's a GObject, I can just slam one in for the other.

Bouncing Stanford logo? Yeah. Bouncing face of your roommate? Yeah. You can do it, all right? We're just gonna not bounce anything else for the time being.

But one thing that's even more interesting than just being able to bounce stuff around is what we refer to as event-driven programs. And all an event is is when you do something in the world, that's an event. When you went to Stanford, that was a big event. You got your big packet in the mail, and you opened it up, and you're like, "Oh, I got into Stanford. I'm going to Stanford, or maybe you gnashed your teeth and tried to figure out where you were gonna go or whatever." Some of you were just like, "Oh, yeah, I'm going to Stanford for sure."

And that was an event, and there was cake and celebrating and the whole maybe – I don't know – maybe you were like, "Oh, I gotta go to Stanford. Bummer."

But it was an event. And in some sense, your life is a program, and it's driven by events. These little things happen, and it changes the course of your life. Hopefully coming to Stanford changed the course of your life in a good way.

In terms of actual programs you write, there will also be events that will change the course of your program. But these events are a little bit more minor than, say, getting into Stanford. There are things like when the user interacts with a computer, like clicking a mouse or pressing a key. Those are events that happen on a computer. And your program should be able to detect certain events that are going on and respond to them by doing various kinds of things, say when you're writing your Breakout program.

The way we be able to determine if an event has taken place is kinda a clandestine operation. It's kinda like we're in the CIA. And we gotta say, "Oh, did an event take place? Well, I don't know [inaudible] if an event took place. I need to send out my minions."

And your minions are called your listeners. Your listeners are something you put out there that says, "I'm here. I'm listening for events. Is anything going on?"

And unless you put out your listeners, you're deaf to what's going on in the world. It's kind of like you weren't listening. You weren't watching the mail for that Stanford packet, and the Stanford packet came, and you just never checked the mail, and you never knew. And no one was gonna tell you because you didn't put out your listener for the Stanford packet.

Well, on the computer, there are two different kinds of listeners we think about. There's a mouse listener and a key listener, which gets events from the mouse and events from the keyboard. And so the way we create our listeners is in our program very early on, we either say addMouseListeners or addKeyListeners. We can actually say "both" – listen for both things. And these are methods that are supported by graphics programs, for example.

Now, in order to use these listeners, you need to have a library added in your program that deals with events, and that's the Java awt.event.\*. So on your programs that use events like listeners, and we'll look at events in just a second, you'll need to have this little import line in there. And this is all in the book. You don't have to hurriedly copy it down.

A bunch of people have asked for slides. And I was hesitant to post slides on the Web because I think if slides are up there, people will not come to class.

So first of all, how many people would like me to post the slides on the Web?

How many people would continue to come to class if I post the slides on the Web?

Yeah, it should be all those same hands. Good times. All right. I'll probably post them on the Web, then.

All right. So there are certain methods of a listener that get called when an event happens, and this is the funky thing. Up until this time, all of your programs have been what we refer to as synchronous. There are some series of events that happen, and you know the next event is gonna happen, or the next method's gonna get called after the last method got called. You're now entering the world called the asynchronous world, which means you don't know when things are gonna happen.

Events happen. You might've been waiting for weeks for that Stanford little acceptance letter to come. You didn't know when it was gonna come. Eventually, it came, but you didn't know. And there were some sad people who we won't talk about right now, but if you're out there watching on video, it's fine. It's perfectly okay. You were waiting for the Stanford acceptance letter, and it never came. That happens. It's not a bad thing. You're still listening. You were active. You took a participatory role in life. That's what you should do. It's a good thing. But you didn't know when or if it was gonna happen.

And that's what an asynchronous event is. It happens, and you don't know when it's gonna happen. You just gotta be prepared when it happens. You're like, "I'm gung ho. When it happens, it's gonna happen, and I'm gonna be there because I'm listening for it." That's what you wanna think about.

So let's look at a simple example of a event-driven program called ClickForFace. You're like, "What does that mean?" But it probably involves the face. Yes, in fact, it does.

So click can mean anything. ClickForFace. Here's an example of an event-driven program. What this is gonna do is it's gonna bring up a blank screen, and every place we click on the screen, it's gonna put a little face there. That's why it's ClickForFace, all right?

So how does this work? Well, first of all, ClickForFace is a graphics program. It's gonna drawFace wherever the user clicks the mouse. Now, here's the funky thing. There's a couple funky things that happen in this program. This program has no run method because there is nothing we need to do in terms of a bunch of sequential steps in this program. All we're doing is we're waiting for events. We're waiting for someone to click the mouse. Until they click the mouse, we got nothing to do except for we gotta be listening for that mouse to get clicked.

So there's a method called a init(), and init() automatically gets called when a program starts. And you might say, "Well, that sounds a lot like run. What's the difference between run and init()?" The way you wanna think about the difference between these is run is when you're actually doing some real work.

Init() is generally when you're just saying, "There's a few things I need to initialize. I need to put my little ear out there to be listening for things. But there's not any real work

I'm gonna be doing in the program." And [inaudible] distinction is not a big deal if you mix the two up. But there is a difference, just so you know.

And we won't ding you for it heavily or anything like that, just so you know. Most of your programs will probably have the run method. In Breakout, you'll have a run method. You won't have init(). It's not a big deal.

So all init() does is it says, "Hey, I gotta be listening for those mouse events." So it adds the mouseListeners. It just says that. And that means now I'm listening for the mouse.

What happens when the mouse gets clicked? Here's the funky part. There is a method called mouseClicked(). And mouseClicked() takes any parameter of type mouse events. Now, if you're paying very careful attention to this program, you will realize that nowhere in your program do you actually call the mouse click method. You're like, "Huh. If I never call the mouse click method, why am I writing it?"

This is what asynchronous events are all about. This particular method has a special name that is understood by the listener. When it hears a mouse click, it will call your mouse click method for you. That's why it's asynchronous. You don't know when it's gonna get called. All you know is if there's a click, it will get called, and you will get this parameter called mouse events.

And what a mouse event has is information about that mouse click. Namely, you can ask the mouse event, which we're just calling e for events, GetX() and GetY(), and that gets the xy location of where the mouse was clicked on the screen.

If the mouse was not clicked in your graphics window, you don't hear that because your listener can only listen in your graphics program. It's not listening to your whole computer. So don't worry if you're like, "Oh, and I'm reading email. Is it listening?" No. It's okay. It's just looking for little clicks in your graphics window.

And so what we're to do when we get to mouse event is, "Hey, our friend the GFace." We're gonna create a new GFace. That sounds like it should be a rap group, don't you – GFace. Maybe that should be – all right, all right, we won't get into that.

And it's gonna be a circle with a face diameter of just some constant I specify – not a big deal. Thirty – I'm gonna create some new round GFace, and I'm gonna place it at the xy location where the mouse was clicked. Note that the xy location that I set the face is this relative location of the face. The face will not show up exactly in the middle where I clicked the mouse. It will show up slightly to the right and down of where I clicked the mouse.

But just to show you that this works, let's click for face. It's like bowling for dollars. We're clicking for face. Doo, we're running. We're feeling good. Any questions about asynchronous events, by the way? Let me just run ClickForFace.

Question?

**Student:**Do you have to [inaudible] click to [inaudible]?

**Instructor (Mehran Sahami):**If you have two programs running simultaneously?

**Student:**[Inaudible] if you have [inaudible].

**Instructor (Mehran Sahami):**Uh-huh?

**Student:**[Inaudible]?

**Instructor (Mehran Sahami):**Yeah, so while your program's running, if you get these asynchronous mouse clicks, your – the mouse-click method will get called for you, yeah.

**Student:**[Inaudible] will it stop [inaudible] method [inaudible]?

**Instructor (Mehran Sahami):**It will stop momentarily for you to deal with that method called mouseClicked(), and then it will continue execution.

**Student:**[Inaudible]

**Instructor (Mehran Sahami):**Yeah. So here we have – there's nothing in the program yet because we haven't clicked yet. Ah, ClickForFace. So you can hit it, and you can spend weeks. You're like, "Yeah, there's a click here. I'm gonna put some face on top of each other. It's like a party over here." Yeah, ClickForFace. That's it.

If I click out here, I don't get any faces. I [inaudible] into some other application because it only listens for events inside here. So that's ClickForFace. There we go.

Uh-huh?

**Student:**[Inaudible] a [inaudible]?

**Instructor (Mehran Sahami):**No, because you don't know where it would return it to. You're not the one that called it. Someone else called you, so you're like, "Hey, you called me. Here's a good time. Have a candy bar." And it's like, "I don't wanna candy bar." And it gets really upset, so – but I hope you wanted the candy bar.

So yeah, you don't return anything. But that's a good thing to think about.

So that's ClickForFace. Now, there's a whole bunch of things you can actually listen for. It's not just clicks. The general idea is first, you add your mouseListeners. That's critical. Common thing people do is they forget their listeners. And they're like, "Hey, I'm clicking for face. And I'm not getting any face."

And it's not because the face doesn't love you. The face loves you. I know. I've been writing the face at 3:00 a.m. last night. It loves you. And then what you need to do after you have your mouse listener in there, and then you add definitions for any listeners you wanna have.

So here are the common set of listeners for mouses. There's mouse-click, which you just saw. That's called whenever the user clicks the mouse. There's mouse-pressed. What's the difference between a click and a press? The press is that the mouse button has been pressed down. It has not yet been released. So if you wanna do something like a drag where you hit a button, and you move the mouse around, and then you release it, you can check for press and released. So a press and a release together is a click.

So you can actually kinda do multiple things. You can do something when the mouse is pressed. You can do something when the mouse is released, and that'll also count as a click. So mouseClicked(), mousePressed(), mouse released. MouseMoved() – every single pixel that the mouse moves, it's like, "Oh, mouse move, mouse move, mouse move." Yeah, I can't even do it fast enough. That's how fast it goes. It's how quickly the mouse moved.

And mouseDragged() is when someone actually clicks on something and then moves the mouse while holding down the button. In fact, we'll say that the mouse has been dragged, which is like mouseMoved() except that this is only happening when the button's down, and someone's moving the mouse.

But all these take the same kind of mouse events, and so you can get the xy location for where that click happened, or where the mouse recently moved to. And that's gonna provide you the data about the event.

So let's look at another one. Let's actually track the mouse. And you're like, "Oh, that wily mouse. It has gotten away from me. How do I track the mouse?" MouseTracker might be real useful for Breakout, just in case it wasn't clear.

What the MouseTracker's gonna do is basically, I'm gonna have a run method here because there's actually some work I wanna do other than just having listeners. What am I gonna do? I'm gonna create some empty label. I'm gonna make a font for that label real big so that you can see it. I'm gonna add that label at a particular location on the screen because it's an empty label. When I start out, nothing will show up. And then I listen. I set out my mouseListeners, and I'm like, "Woo, mouse? Mouse?" And what I'm checking for is mouseMoved(). That's the only listener that I'm setting up here.

If mouse moves, I get some events. I'm going to change the text for that label to be a mouse with the xy location it moved to. And I need to keep track of my label, both here and here, so I need to keep track of the label in between method calls so I actually have my label be a private-instance variable.

So if I run this puppy, it's ten lines of code, but check out just how cool it is – well, after it runs. You're like, "How cool is it, Marilyn?" We're gonna track the mouse. Doo, doo, doo. We're running. Oh, yeah, see, it moves off the screen, stops tracking, moves back into the tracking to start tracking. One thing you can do if you're just totally bored is can you get  $y(0,0)$ .

[Inaudible] last night. It was 3:00. I'm like, "I can be preparing a lecture, but I wanted to see, can I get to  $0,0$ ?" I'm like – an hour and a half later, I was like, "There it is." And then I wept.

So besides the mouse, there's also things you can do with the keyboard. So the keyboard is also your friend. So, returning to our friend, the slide, besides tracking the mouse, we can also do things with the keyboard. And you're like, "Hey, Marilyn, for Breakout, do I need to do anything with the keyboard?"

Not in the basic version of Breakout, but if you wanna do the cooler version of Breakout, where your paddle [inaudible] can shoot at the bricks and take them down? Some people have done that in the past. It makes the game much easier. But it's fun. You can listen to keyboard events.

So keyboard events work just like mouse events. You add a listener. But the listener you're gonna add is keyListeners. It's perfectly fine for program to add both mouseListeners and keyListeners. Why do you have to add these to begin with? And the reason why you have to add these is because your program really needs to know, "Do I need to pay attention to these things?"

Because it actually requires some work on the part of the computer to pay attention to these things. Because one thing you might just think [inaudible], "This whole listener thing is just dumb, Marilyn." And I'm like, "Any program I run, yeah, if I'm pressing the keyboard, it should just know about it, right?"

No, not necessarily. If you're writing a piece of software that needs to run really fast and doesn't care where the mouse is, why should you have it worry about where the mouse is or worry about what keys are being pressed?

So that's why this thing that's important to remember that you need to put in there, but doesn't come in by default because it's actually programs where we care about efficiency, and we don't care about what the user's doing, which, sadly enough, is many programs. We don't have listeners.

So for the key listener, there's three things I check for. These are the most common ones. There's actually a few more that are in the book, but these are the ones you really need to worry about for most things. There's keyPressed(), which is called when the user presses a key. There's keyReleased(), which is when they let go of the key. So keyPressed() is when they push it down. They have not necessarily let go of the key. keyReleased() is they've let up on the key.

So you can actually do funky things what – when you press something, like the screen goes black, and then your roommate leaves and doesn't know what you're doing, and then you lift up the key, and it comes back. You can actually do that.

KeyTyped() is basically a click. It's press and release together. So when you press and release a key, that generates a keyTyped() event. What you get in this E that you're getting is an event. It's not a mouse event. It's a key event. And key event is an object which has information about, for example, which key was pressed. And the book goes into details of which key was pressed.

But I wanna show you some simple example of this where we don't even care what key was pressed. All we care is that a key was pressed. So yet another example. Woo-hoo. And I can actually just close PowerPoint.

So what we're gonna do is we're going to DragObjects around. What does DragObjects do? So DragObjects is a graphics program, and what I'm gonna do is create a rectangle on the screen and add it, create an oval on the screen and add it, and add mouseListeners and keyListeners.

And you might see this and go, "Marilyn, why is it a init() method as opposed to a run method? Shouldn't that really be a run method because you're doing some work?" Yeah, probably. You can – it's [inaudible] the other. The book actually has a similar program for this. It's not the same program but a similar program that they call it init() over there. So I just made it init() just kinda to match the book so you're not like, "Oh, Marilyn, slides in the book are just completely wrong," and again, this death struggle.

No, it just – it doesn't really matter that much. But we create the rectangle. We create the oval. We add them both to the screen. We add the listeners, both the mouse listener and the key listener. And now here's what we're gonna do that's funky. We're going to allow objects to be dragged on the screen.

What does that mean? That means when you click on a rectangle on the screen, and you move your mouse holding down the button – that's called a drag – we're gonna have the rectangle move with you. Woo-hoo. Rock on. So mouse drag. Mouse drag is gonna get some mouse events.

Now, there's a little bit of funkiness in here with this little GObject thing. What does that mean? So I'll get back to that in just a second by first showing you what this GObject actually is. So what I'm gonna keep track of in my program is a generic GObject. And you might say, "But Marilyn, we don't create generic GObjects." Yeah, that's because its variable GObject is either gonna be the rectangle or the oval. It can be either one. They're both GObjects. So the common denominator is that they're GObjects.

So this object – this variable is gonna keep track of what object is currently being dragged on the screen – what object I clicked on so I know what I'm actually dragging, whether it's the oval or rectangle or that I haven't clicked on any object at all.

GPoint is basically just something that holds the x and y location. It's a very simple object that we haven't talked about till right now. But basically, all the GPoint is is it's a little tiny object, and it has an x and a y location in it. And so I can set the x and y location in the point, and I can say, "Give me both getters and setters." Get the x and y location. It's just a little convenient encapsulation. I could've actually had this be a separate variable x and a separate variable y. I just created the GPoint so you could see a GPoint.

And I'm gonna have little randomness in my program, too. So the random generator will once again rear its ugly head. And so I have – I get an instance of the random generator. You'll see we're gonna generate some random colors in just a second.

So with that said, how did this mouse press and mouse drag thing work? MousePressed() is when someone clicked the button. They have not yet released the button. It is not a full click. It's just the mouse was pressed. The mouse was pushed down. I get this event. And that event, E, basically has an x and y location associated with it.

As you saw in the previous example, I can get the x and y location separately. There is a way I can just say e.get point(), and it gets the x and y location together in a little point object and gives that back to you, which is why I created this thing called the GPoint, so you could see it. It's just a nice [inaudible] that gives you x and y at the same time.

And when I say, "Hey, get that point, and what I wanna do is get the elements at that point." So wherever you clicked, call getElement() at. That will return to me the topmost object at the point you clicked. If it's the rectangle, it will give me back the rectangle. If it's the oval, it will give me back the oval.

If there is no object at the point I clicked, I get back something called null, N-U-L-L, and that gets assigned to GObject, which means your GObject – there is no thing. There is no object there. Null is just a way of referring essentially to no object. But you can assign any object the value null to say, "There's not really an object here."

And so getElement() will either give you the oval, the rectangle, or null. Get element – that's something you're gonna use for Breakout. Pay close attention.

So what happens after the user clicks? If I get an object at the place they clicked, I now see are they moving the mouse? If they're dragging the mouse – after they click, if they move the mouse at all, that generates events that are mouse-dragged events. So gets called the position of the mouse that the mouse has been dragged to.

First thing I checked, are you dragging a natural object? When you click the mouse, did you click it on top of an object? If you did, then the object is not null. If you didn't click it on an object, then it is null, so hey, nothing to see here. You didn't click on an object, and now you're trying to drag around. You've got nothing to drag, buddy. Sorry. Thanks for playing. I'm not gonna do anything.

But if you did click on an object, then what I'm gonna do is move that object. And remember, move is a relative coordinates. It says, "Move this object relative to where it was before." So I say, "Get the x location and the y location that the mouse has moved to from this event, and subtract off essentially the last place that the mouse was, which is where the mouse actually clicked on this object." So I get some relative amount of movement.

And now after you've moved the mouse, I need to say the mouse is actually at a new point now. So I update the last points that the mouse was at to be equal to wherever the mouse was actually moved to.

So last is always the last location of the mouse was, either dragged to or got clicked on.

Last but not least is I wanna change the color of the object that you last clicked on or that you're dragging to a random color if you type any key. So I don't care what key you type. I'm not actually gonna look at what the key event was. But if you typed any key, keyTyped() will get called, and if your object is not null, I'm gonna set it to object color to be – or the color of that object to be some random color that I'm gonna get my random color generator or my random number generator gonna give me random colors.

So let me run this so you get a sense of what's actually going on – how these pieces fit together. Doo, doo, doo, doo, doo. It's like you can create a whole drawing program now, all right?

So we're gonna drag some objects around. So when this puppy starts off, we get a oval and a square, which both start off black because I didn't set their color. And they happen to be slightly on top of each other. If I click on the rectangle and move it, oh, yeah. Notice how it tracks the mouse when I move it. When I now click off the rectangle and try to move around, nothing happens.

Or if I click here and try to drag, I'm holding the mouse button down, nothing happens. If I click on the oval and move it around, oh, it gets moved around. Now I wanna be funky. I press a key. The last object I dragged gets assigned a random color. Oh, that's kinda fun for about three seconds, all right?

If I click on the other object and move it around, you can actually see the oval was put in front of the rectangle, and the z-ordering is not changing by me moving the object around. I never say, "Send to front," or "Send to back," or anything like that. I could do that if I wanted to, to bring an object up to the front, which is kind of the behavior you're used to from other applications. When you click on something, it comes to the front. Not here because I'm not changing the z-order, so it's just back over here.

But the last thing I clicked on, I can also get random colors and change its colors. Yeah, that's – it's just – yeah, oh, so much fun, all right?

Any questions about that? We're dragging. We're clicking. We're changing colors.

**Student:**[Inaudible] object [inaudible].

**Instructor (Mehran Sahami):** Yeah, I could've done set location. I just need to do a little bit more math for set location because I need to figure out to set the xy of the object relative to where the mouse was clicked inside of the object and its coordinates sort of off in this corner.

So one final thing I wanna show you before you go, because now you've seen all the codes to actually be able to create a simple game like "Breakout." But here's another simple game, just to show you another example of stuff going on.

I call it the UFO game. I gave you all the code for the UFO game. And you may recognize this game. Oh, yeah, it's sort of like Space Invaders Lite. You remember – anyone remember Space Invaders? Two people. Yeah, this thing is coming down the screen toward you, and you're in the middle. You get to shoot these little shots out at every time you click on the mouse. And if it ever reaches the bottom of the screen, you die.

But if you hit it – oh, I'm coming so close – it's gone. Thank you. Hours, hours of work. Let me just show you a little bit of the code so you can understand. This will be important for Breakout. When the code runs, how are you gonna create effective animation and also other stuff going on at the same time?

I'm going to give each object in the world a chance to do something. I'm gonna call moveUFO() to allow it to move, move bullet to allow it to move if there's a bullet in the air. Check to see if there's a collision between these two things, and wait before I do this cycle again. So I just continue to do this cycle over and over until my game is over.

And the way I check my game is over is basically that little square got onto the bottom, or I actually shot it and got a collision. I'll leave it to you to look at the code because if I look – go look at the code in excruciating detail, I'll probably get Breakout questions, which you have all the code so you can understand how hopefully it works. And you can leverage it for Breakout. So I'll see you on Friday.

[End of Audio]

Duration: 52 minutes

## Programming Methodology-Lecture12

**Instructor (Mehran Sahami):** So welcome back to yet another fun filled, exciting day of CS106A. A couple quick announcements before we start. There's actually no handouts for today, and you're like if there's no handouts for today, why are there two handouts in the back? If you already picked up the two handouts from last class, you might want to double check to make sure you don't already have them, but if you don't have them, feel free to pick them up. We just don't want to cut down any more trees than we need to, so if you accidentally picked them up, you can put them back at the end of the class, pass them to a friend, whatever you'd like to do.

As per sort of the general consensus last time, everyone wanted to have class slides on the web, so now all the class slides are posted on the web and after every class when we have some slides in class, they'll be posted on the web as well. There's a little link called, strangely enough, class slides that is now on sort of the left hand navigation bar on the 106A website, and so you can get all the class slides from there. In some cases, what I actually did was if we covered some slides over multiple days but they were all about the same topic, I might have aggregated them all into one set of slides or one deck, so you'll see them in there as just one topical kind of thing.

Just wondering – how many people have started break out? Good. How many people are done with break out? You folks get to see good times. You're well ahead of the curve. I won't ask how many people have not started. I assume it's just the compliment of that. It is a fairly beefy assignment. It would be in your best interest to start soon. It's due on Wednesday. With that said, it's time to launch into an entirely new topic, and the entirely new topic is something that we refer to as enumeration. Enumeration is a pretty basic idea, and it comes from the word enumerate, as you can kind of imagine.

When you enumerate something, you basically just have some way of referring to something through numbers. So if we wanted to enumerate, for example, the year that someone is in college, we might have freshman and sophomores and juniors and seniors – that's the enumeration of the year you might be. And so the basic idea is we just want to have some set of things that we enumerate or give a set of numbers to, essentially, and you don't necessarily want to think of it as having to be a set of numbers, but it's basically just some set of items that go together.

We generally give them some numbers or some listing as a way of keeping them all – a way we might keep track of them. One way we might do this in java, for example, is just have a series of constants that are integers and so just to save myself a little bit of time in writing, constants. Yeah, a beautiful thing. So we might have some constant public static final [inaudible], and so if we're going to do enumeration, oftentimes we just use integers to refer to each of the individual items and we just count them up. So frosh would be one, sophomores two, juniors three, seniors four, and grad is five.

That's just what year you might be in school. Oftentimes, computer scientists actually start counting from zero, but sometimes it actually makes sense to have these things be

numbers that start at one. For example, if you want to know which year someone is in school or if you're doing months of the year, January is generally number one as opposed to number zero, so just to keep with common [inaudible], we might actually number it this way.

Now, there's something that was introduced in one of the later versions of java, java 5.0, which is kind of the second to latest version, which is something called enumerated types, and the book talks about them briefly. I'm not going to talk about them here sort of for the same reasons the book doesn't talk about them. The book actually talks about them and then says the advantages versus disadvantages of doing enumerations using this thing called enumerated type versus just listing them out as integers. This way, at least for the time being in sort of the development of java's world, seems to win out.

We're just going to teach you this way. As a matter of fact, in the old school, anything before java 5.0 had to do it this way, so it's just best that you see it this way, because most code these days is written this way and it probably will continue to be until at some point this new enumerated type thing takes off. As you see in the book, it talks about enum type. Don't worry about it. We're just gonna do it this way.

The only problem that comes up with doing something like this, though, is that you want to figure out, well, how do I read in these things and display them? Well, these are all just integers, so if I actually want to ask someone their year in school, I would have to keep track of that with some ints that I might call year, and so I would read in an int, and I might ask the person for their year, for example. And when I read that in, that's all good and well. The only problem is this thing is just an integer. The user gives me some number, hopefully between one and five.

I might want to actually have some logic in here that checks to make sure they gave me a number between one and five, 'cause if they gave me a six, I don't know what that corresponds to. Maybe that's the dreaded other student category. I need to do something to guarantee that it's in this list one through five. The other problem is if I actually want to print out someone's year, there's no way for the computer to know that year one should print out the word frosh, so if I do something like print lin here and I just write out year, that's gonna write out an integer.

It's gonna write out a value, whatever the value the user gave me, presumably between one and five. So if I actually want to have some nicety in there and say, oh, if the year is one, I actually want to write out frosh as opposed to writing out a one, I need to do that manually. So somewhere in my program, I need to have some function that I do a switch statement or I can do cascaded ifs and I switch, for example, on year.

I might say, well, in the case where year happens to be frosh, then what I'm going to do is actually print out print lin frosh in quotes, because that's the only way the computer knows about it, and then I would have a break, the funky syntax from our fun, the switch statement, and then I might have some case over here for sophomore. I need to do this

manually, and that's just the way life is in the city. These things are just integers. We have some enumeration.

So in the program, the program can read a little bit more text because we can refer to frosh, sophomore, junior, senior and grads as constants in the program, but when it comes to displaying them on the screen, we need to actually write it out manually because the computer has no other way of knowing that a one means frosh.

Well, case one is the same thing as case frosh, 'cause if these are in the same program, frosh is just the constant one, and so in fact that's why we want to refer to it that way because it's more clear for someone reading it. They see oh, what do you do in the case where you're a frosh? Well, I'm gonna write out frosh. It's fairly straightforward enough. But I'm just using these constants. If someone else wants to come along and say you know what? I love frosh and they're all great, but I'm a computer scientist. I start counting from zero.

It'll just change everywhere in your program as long as you've used the constants if you're referring to zero. They might just say, well, actually frosh are the coolest. They're a six. That's fine. You can do whatever you want. The computer really doesn't care. Most people probably won't care, either, so we just start counting from zero or one most of the time. Any questions about the general idea of enumeration?

Well, that's the thing. In your program, you want to set up the expectation that they're entering a number. If they were to enter the string frosh, because read in does error checking, it's going to say that's not the right format. So one thing you could actually do is rather than reading an int, you can read in a line, which would read in a string, and then you'd need to have some logic to convert that over to one. So you'd sort of do this process but backwards. That's why enumerations are something that are useful to have when you're writing real programs, but they can get a little bit bulky to use because you have to do the conversions.

Right now, I'm just making the constants public because I might want to refer to them in some other class. If I have some other class, they can also refer to these constants. If I was only going to refer to these constants within this class, I'd make them private. With that said, it's time for something completely different. I know the sun isn't out much right now, but I sort of had this thrill where I wanted to barbecue one last time, and I figure if I can't barbecue outside, I'm going to barbecue inside. Now I wanted to light a fire in here, but as you can imagine for various reasons, that was frowned upon by the university.

So I can't actually light the fire, but I can basically do everything else that would be involved with grilling, which doesn't really turn out to be that exciting when you don't have a fire. But the basic idea is if I leave some things on the grill for a while, they'll get a little hot. Just pretend they were on the fire for a while, okay? If you leave something on the grill too long, what happens to it? It gets kind of burned. It accumulates something. It accumulates something we refer to as char. It turns out as a computer

scientist, you get a different kind of char than other people get on their burgers when they're grilling them, or, as happens to be the case, on their ding dongs.

Turns out they don't make ding dongs. How many people know what a ding dong is? Good times. If you don't, come and look at the box. It turns out they don't actually make ding dongs. I'm sure this is copyright Hostess 2007. I went so many places last night trying to find ding dongs, but you can eventually find them. The basic idea, though, is we want to think about something that we want to refer to in a program that isn't always just a number. So far, we've been dealing with a whole bunch of numbers. We had doubles. We had integers. Life was good.

I told you about this thing called string, but we didn't really explore it, and now it's time to explore it a little bit further to figure out what these little puppies, this little char or little characters are all about. So what we're gonna do is we're gonna explore the world of char. There's a type called CHAR, which actually is a character, and so the way we refer to this type, even though I just called it char, is we don't call it a char. We call it either a char like the first syllable in character. Some people call it a car, even though it's not called a character because they just look at it and they're like oh, if it was one syllable, it would just be car.

And some people look at it and say no, Mehran, if it was just one syllable, it would be char. Don't call it char. I will beat you. That was a joke, by the way. I'll be trying to explain that to the judge when I'm in jail and he's like, well, the video I saw, it didn't appear to be a joke. CHAR is just a character. We can declare variables of this type by just saying CHAR and I'll call it CH. I have these little CHAR CH, and what do I assign to these puppies? What I assign to them is something that's inside single quotes. That's the key.

I can say something like CH equals little a, and I put the a inside single quotes. Not to be confused with double quotes. If I put it in double quotes, that's a string. That's not the same thing as a CHAR. So I put it inside single quotes. That indicates that it's a character. The other thing that differentiates a character from a string is that a character always is just a single character. You can't put more than one character there. It can't be like CH equals AB. That's not allowed. It's just a single character. The funky thing, and you might say, so Mehran, why are you telling me about this thing called characters right after you talk about enumeration?

Because it turns out in the computer, the computer really doesn't know about characters in some deep way that you and I know about characters. All characters inside the computer are actually just numbers. At the end of the day, we can refer to them as these characters in their niceties for us, but to the computer, they're actually an enumeration. A happens to be some number and B happens to be some number and C happens to be some number, and so now it's time for you to know what those numbers are. So if we just look at the slides real briefly, there's this thing called ASCII.

Anyone know what ASCII stands for? It's American standard code for information interchange, and we'll just do it as a social. The basic idea for this is that somewhere, some time ago, someone came along and said we're gonna create a standard enumeration of all the characters that exist, and so here's the first 127, as it turns out, character enumeration, which is the part that's mostly been standardized. All the rest of the stuff, there's still some debate about, but this one, everyone's standardized on. Now this little diagram that's up here is actually in octal numbers, and you're like octal numbers? It's base eight.

Computer sciences think that way. We think in base two. We think in base eight and we think in base 16. Most people think in base ten. Yeah, that's why most people aren't computer scientists. Here is base eight. I had this roommate at Stanford who thought everyone should count in base 12, because base 12 was divisible not only by two but by also three and four, and ten wasn't, and he would just try to convince people of this all the time. I was like that is so wrong. Now he works for the Senate, which I wouldn't be surprised if we have some resolution someday. The United States will now count in base 12.

But anyway, the basic idea here, as you can see in this, is that first of all, the character A is not the number one, and there's actually a distinction between the uppercase A and the lowercase a. Another thing that's also kind of funky is you might notice the numbers up here, like zero, one, two, three – the digits. The number zero is not the same thing as the character zero. The character zero just has some enumeration that's some funky value. Who knows what that is. It would actually be 60 in octal notation, which turns out to be 48.

That's not the same thing as the number zero, so it's important to distinguish that the number zero that we think of as an integer is not the same thing as the character zero which we would put in single quotes, and that's true for all digits. You don't need to care about what the actual numbers are. If you really do care about memorizing it, the way you can read the chart is the row on the chart is like the first two digits, and then the third digit is given by the column.

So A is actually 101 in octal notation, if you really care about reading this, which makes it the value 65, but we don't really care. We just think of it as A inside single quotes. There's a couple things that that gives you, though, that are useful, and the couple things that it gives you that are actually useful is it guarantees that the letters little a through little z are sequential. It guarantees that uppercase A through uppercase Z are also sequential, and it guarantees that the digit zero through the digits nine are also sequential. That's all you really need to care about.

Other than the fact that these guarantees are given to you, you don't care about what the actual numbers are up there. There's a couple other things that look a little weird up here that I'll just kind of point out. There are some characters that are like these backslash characters that are \c and \n. The backslashes are just special characters that are treated by the computer as a single character, so \n is new line. It's like a return. \c is the tab

character, and if you're really interested in all these things, you can look them all up in the book, but those are the only ones you need to know.

You might wonder hey, Mehran, how do I get a single quote, because if I'm trying to put characters inside quotes – if I want little ch over here to be a single quote, do I write it like three quotes in a row? No. That will confuse the computer to no end. The way you actually do it is you put in a backslash and then a single quote and then another quote, and this is what's referred to as an escape character. When it sees the backslash, it says treat the next character as a little character and not as any special symbol that, for example, closes the definition of a character or something.

So this whole thing is just the single character apostrophe or single quote, and it's inside single quotes, so that's how it knows that this is supposed to be the character single quote as opposed to the closing of the character. A couple other things that are sort of funky. What happens if you want to actually do a backslash? Backslash is actually just backslash backslash. If we put that inside quotes, that would be the single character backslash. There's a couple others that are worthwhile. Double quote – same thing. We would put a backslash and then a double quote if we wanted to have the single double quote character.

Not a huge deal, but you should just know that if you want to put apostrophes inside some text that you're writing or something like that. How do we actually get these characters? Rather than getting single characters, so before we talked about over here our friend, read int, which reads in a single integer, you might say hey, do we have a read char? Can I read a single character at a time? Not really. What I end up doing is I read a whole line at a time from the user and then I can break up that line.

I'm going to have some string, and that string S, I would read a line from the user, and that's going to be a whole bunch of characters that are stored inside that string S, and I can pull out individual characters using something called char at. So I can say CH equals S dot, and so the string class or the string objects have a method called char at, and you give it a number and it gives you the character at that position. So I could say char at and as computer scientists, we always start counting from zero, so I could say char at zero, and that's the very first character in the line the user actually entered.

I can do a print lin on that character directly, and it'll just write out that first character. The way to think about this – let's say it read a line, S, and the user gave me a line and they typed in hello. Then they hit enter. Hello gets broken up into a series of characters where this is character zero, one, two, three, four, and that return that the user types is thrown away. It's cleaned up for me so it gets rid of it automatically. If the length of the string that the user typed in is five, it's actually indexed from character zero to four. So char at zero would give me the H character out of here. That's a critical thing to remember.

We start counting from zero as computer scientists. Any questions about that? We have our friend over here. Let's see if I can actually get this all the way out of the way. We have our friend over here that tells us, hey, the letters are guaranteed to be sequential in

the lowercase alphabet, in the uppercase alphabet and the characters of the digits zero through nine, so how can I use that? It turns out you can actually do math on characters, and you're like oh, Mehran, the whole point of having characters was that I wouldn't have to do math on them. Well, you can actually do math on characters. It's kind of fun.

Let's say we want to convert a character to a lowercase character. We might have some method. It's going to return a character, which is a lowercase version. I'll call it two lower of whatever character is passed into it. So it's passed in some character CH, and what it's going to return is the lowercase version of that character. So the first thing I need to check is hey, did you give me an uppercase character? If you give me an exclamation point, what's the lowercase version of an exclamation point? A period. No. We can't lowercase punctuation. It just doesn't work.

It's like what's the lowercase version of a period? A comma. What's the lowercase version of a comma? A space. What's the lowercase version of a space? Yeah, somewhere, it stops, and it stops here. How do we check that this thing is uppercase? Well, these are really numbers, and we're guaranteed that they're sequential. So we can treat them just like numbers. We can do operations on them like they were integers. We can say if CH is greater than or equal to uppercase A and CH is less than or equal to uppercase Z, if it falls into that case, then we know that it's an uppercase character 'cause they're guaranteed to be sequential.

If it doesn't fall into that range, then we know that it's not an uppercase character because it's outside of the sequential range of uppercase characters. So what do we do? We're going to return something, which is a lowercase version of that character if it's an uppercase character. How do we convert it to lowercase? Anyone want to venture a guess? We could. How do we know how much to add? It's on the chart, but we don't want to use the chart, because we don't want to have to remember what's in the chart. Yes. For that, A. It's the difference between the uppercase and the lowercase character.

So think about it this way. First thing we want to do is figure out – I'm going to explain this a slightly different way, which is first we want to figure out which character of the uppercase alphabet you typed. So we take the CH you gave us and subtract from an uppercase A. If we do that, if CH was uppercase A, we get zero, which is the first letter of the alphabet. If it's B, we get one. If it's C, we get two. This will give us in some sense the number of the letter in the uppercase alphabet. Once we get that number, we need to figure out what the corresponding letter is in the lowercase alphabet.

So translate according to that chart into the lowercase alphabet. I don't want to memorize that chart. How do I know the starting position of the lowercase alphabet? It's lowercase a, so I just add lowercase a, which is the same thing, basically, as taking the difference between the lowercase alphabet and the uppercase alphabet. But if I do that, basically this portion tells me figure out which letter, in terms of the index of that letter, and then offset it by the appropriate amount to get the corresponding letter in the lowercase alphabet, and that's what I return.

Otherwise, what happens if I'm not in the uppercase alphabet? I just say hey, I'm going to give you back – you wanted lowercase version of exclamation point. Not happening. You get exclamation point back. I have to still give back a character. I can't say I'm gonna give you back this big giant goose egg, and it's like sorry, thanks for playing. I can't do that because goose egg is not a character. I just return CH unchanged. We can do a little bit of math.

It doesn't matter. I just get the offset, but I'll still give you a B for that. The other thing we also want to think about is we can not only do this kind of math on characters, we can even count through characters. You remember in the days of Yore when you learned your little alphabet, like the little alphabet song? I thought for five years of my life L M N O P was one letter. Totally screwed me up. That's just the American educational system. If we have some character like CH, I can actually have a four loop counting through the characters.

I can say start at uppercase A as long as the character is less than or equal to uppercase Z CH++. I treat it just like an integer, and now what I have is a four loop that's index is a letter that counts from uppercase A through uppercase Z. I can do this lowercase. I can do it with the digits from zero to nine, but I can treat these things basically just the same way I treated integers, because underneath the hood, it's an enumeration. They really are integers. Other things to keep in mind is characters are a primitive type. This type CHAR is like an integer or like a double. It's referred to as a primitive type. It's not a class.

You don't get objects of type CHAR. You actually get these low level things called [inaudible] the same way you got integers, which means when you pass characters, you get a copy of the character. It also means that a character variable like CH is not an object, so it doesn't receive messages. It can't get messages. But it turns out there's a bunch of operations we'd actually like to be able to do on characters, and so java gives us this funky class called character, and character actually has a bunch of static methods. They're methods that you can apply to characters but you don't call them in the traditional sense of sending a message.

This is how they work. If I have CHAR CH, I can say CH equals – I give it the name of the class instead of the name of the object. So I say character got and there is, for example, something called two uppercase that gets passed in some character and returns to me the uppercase version of that character. Just like we wrote two lower here, you can imagine you could do a very similar kind of thing with slightly different math to create an uppercase version. It does that for you. This method is part of a class called character, but it is what we refer to as a static method. It's not a method that every object in that class has.

It's a method that just the class has, so the way we refer to it is we give it the class name and then the method name, because this CHAR thing is not an object. It turns out there actually is a class called character that you can create objects of, but we're not gonna get into that. We're just gonna use these little things called CHARs, which are our friend. There's a bunch of useful methods in this character class, and I'll just go over a few of

them real briefly. Real quickly, you can check to see if a character is a digit, is a letter or is letter or digit.

That's good for validating some inputs if you want the user to type in something in particular. These taken characters are Booleans. Question? We're not gonna worry about letters from different alphabets for now, but in fact, they're all supported. The numbers just get bigger from what they could be. Though I only showed you the first 127 letters, it turns out that the standard that java uses actually supports over one million characters, and so you can have all kinds of stuff like Chinese and Arabic and all that. In terms of other things you can do with characters, you can check to see if a character is lowercase or uppercase, and all these at this point are trivial.

I know how to write these myself. In fact, you do, and you could, but they're so easy to write that they're just kind of given to you for free as well. A couple others like [inaudible] white space. That was actually convenient. It checks to see if a character is either a tab or a space or a new line. Question? It returns a Boolean. It just says the thing that you gave me is a letter or a digit. For example, here is the digit 2. So it would return true for that, except it might hit someone else along the way. Yeah, if it's either a letter or a digit. It doesn't let you know which one. It's just if it was a letter or a digit.

It's not punctuation is kind of the idea. And then finally, two lowercase and two uppercase, you actually just wrote two lower yourself, but you also get those. These are all in the book, so you don't need to worry about copying them down, and the slides will be posted on the website. So characters are kind of fun because we can do math on characters, and it's kind of like oh, that's sort of interesting, but it gets more fun when you can put a whole sequence of characters together into our friend, the string. It's time to bring the string back out.

Time to polish off the string. I actually wanted to see if I can do this just for laughs. I want to see how far it will go. All right. Strings – I'm gonna try that again by the end of class. Our friend, the string class. Strings, in fact, are a class and there are objects associated with strings, as you sort of saw before. So we could have, for example, a string that we declare called STR, and we might read that in from the user using our friend read line that you just saw an example of previously, and we pulled out individual characters.

Here, we're going to read a whole line. It turns out there's a bunch of things that we would like to be able to do on strings. The key concept with strings is a string is what we refer to as immutable, which means that a string cannot be changed in place. If the user types in hello and I say, yeah, hello is kind of fun, but I really like Jell-O, and so I want to get rid of the H and replace it by a J, I cannot do that in a string. So if you worked with other languages where you can directly change the context of the string, not allowed in java.

They are immutable, which means if I want to go from hello to Jell-O, what I need to do is somehow create a new string that contains Jell-O. I might take some portion of this string and add some new character to it, and I'll show you some examples of how we

might do that, but the key concept is strings are immutable. They cannot be changed in place. When we do operations on strings, what we're actually gonna do is create new strings that are modifications of the previous strings we had, but we're still creating new strings.

I'll show you some examples of methods for strings in just a second, but I want to contrast between strings and characters just real briefly before we jettison our friend the character and deal all with strings. So CHAR as we talked about is a primitive type. It's not a class versus string is a class, so we have objects of the string type. If I were to have CHAR CH and I were to have string STR and I want to do something like converting to uppercase, for CH I have to call character dot two uppercase CH. I don't actually pass this message to CH, 'cause CH is not a class. It's a primitive type.

I need to call this funky class and say hey, class character, let me use your two uppercase method to make this character. In string, there actually is a string operation two uppercase, and the way that works is I could say string equals STR, so the receiver of the message is actually an object. It's this string thing. I'm not writing out the whole word string. I'm saying STR dot two uppercase, and I pass it in STR. Now here, things might look a little bit funky.

The first thing that looks funky is you say hey, Mehran, if you're telling the string to convert itself to uppercase, why do you need to assign it back to itself? Why can't you just say hey, string, convert yourself to uppercase? Why wouldn't that make sense in java's model? 'Cause strings are immutable. That's just beautiful. The basic idea is strings are immutable, so I can't tell a string to change itself to an uppercase version. I can say hey, string, create an uppercase version of yourself. You haven't changed yourself and give me that uppercase version. So it says oh, okay. I say string, create an uppercase version of yourself.

It says here you go. Here's the uppercase version, and it's all excited. It's like oh, here's an uppercase version of me. It's gonna be like me and my uppercase version, and what do you do? You just say yeah, it's not you anymore. I'm just slamming it over you with your uppercase version. So I've replaced the old string with its uppercase version, but for a brief, gleaming moment in time, I had this thing was a separate string until I signed it over here. I wasn't actually overwriting or changing STR. If I assigned it to some other string like string two, I would actually have the original, unchanged from the string two, which would be a different kind of thing.

A bunch of things you can also do on strings – [inaudible] you've actually seen before. I'll show you one more example of [inaudible], but you've been doing it this whole time. I have string S1, which will be CS106. I have string two. It's okay for a string to be a single character A. It is distinguished from the character A by having double quotes around it. So a character always one character. A string can be one or more characters. As a matter of fact, it can be zero characters. It can have double quote, double quote, which is the empty string.

So I can create a new string, string S3 by just [inaudible] using the plus operation. So I can say I got an plus string two plus N plus string one plus string two. What is that gonna give me? I got an A in CS106 A, and it just [inaudible] them all together. Be happy it's not like CS106 F. I don't know what's going on in CS106 X. We're just dealing with the A here. It's amazing how small the difference is between an A and an F. Just kidding. It's actually a huge, wide gulf. I'm sure all of you will get As. [Inaudible].

Another thing you might want to do with strings is say hey, the user's giving me some particular string like I do a read line over here. Can I actually check to see if that string is equal to something? So one thing I might be inclined to do is say hey, is that string STR equal equal to some other string, like maybe I had some other string up here, S2. And I might say is that equal to S2? Turns out, this looks like a perfectly reasonable thing to say. Bad times. This is not how we check for equality with strings. It's actually valid syntax. You will not get a complier error when you do this.

What this is actually doing is saying are STR and S2 the same object? Not do they contain the same characters but are they the same actual object, which most of the time when you're comparing two strings, they're not the same actual object. They are two different objects that may contain the same characters. So how do we actually test for equality? Well, there is a little method we use that is a method of the string class called equals, and the way we write it is kind of funky. We take one of the strings and we say S1 dot equals and then we give it as a parameter the other string, STR.

So what it says is it basically sends the equals message to string one and says hey, string one? You know what your own contents are. Are they happen to be equal to the contents of this other string I'm passing you as a parameter? And this returns true or false if these two strings contain the same characters. It is case sensitive. There's another version of it that's called equals ignore case that is case insensitive, but this version is case sensitive. The other one you can also see in the book. It's not a big deal. So these are some very useful methods. I'll show you some more useful methods of strings very briefly.

We'll talk about some of these in more detail next time, but I want you to see them briefly right now. The three made its way back up here. The string class has some methods like the length of a string. How many characters does that string contain? So for a particular string like STR, you take STR dot length and it would give you back as an integer the number of characters in the string. CHAR at you already saw. You give it some index starting at zero and it gives you back the character at that particular index. There's a couple things you can do – substring, where you can pull out a portion of the string.

Remember our friend hello? At some point, we want to say oh, just slide this over. Where we have some string that I'll call STR, which may be a set hello. What the substring method actually does is it says give me back a piece of yourself and the piece of yourself is determined by some indices P1 and P2. P1 is the beginning index of the substring you want to get. P2 is the end of the substring you want to get but not counting P2. So it's

exclusive of P2. What does that mean? So if I say string dot substring where I start at position one, that's gonna start at the E, 'cause that's position one.

Then I say go up to three. What it actually gives me back is L as a string, and so I can assign that somewhere else. Maybe I have some other string S. It does not change the string STR, 'cause it's immutable, but what it gives me back is starting at this position up to but not including this position. It's kind of funky. That's just the way it is. There's a version of it that only has one parameter, and if you specify just one parameter, it's [inaudible] start at a particular location. Give me everything to the end because sometimes you want to do that. You just want to say give me everything from this position on to the end of the string.

A couple other things – equals you just saw. This lets you know if two strings are equal to each other. You might say hey, Mehran, I don't want to just check to see if they're equal. Can I do greater than or less than? Well, you can't do greater than or less than using the greater than or less than signs. Those will not work properly. What you do is you use a function called compare to, and the way compare to works is it actually gives you back an integer. That integer is either negative, zero or positive. If it's negative, that means one string is less than the other string lexicographically.

If it's zero, it means they're equal, and if it's positive, it means one string is greater than the other string in terms of the ordering that you actually – what you send the message to and the parameter that you actually pass. It allows you to essentially check for not only equal to but greater than or less than as well in lexicographic order. A couple other things very quickly – index of allows you to search a string for a particular character or some other substring, so you tell a string, hey, string, I want the index of the character E in your string, and if I asked hello, the string STR for the index of E, the character E, it would give me back a one.

So if it finds it, it gives you back the index of the first instance that it found it. So if I ask for the L, it'll give me a back a two. It won't give me back the three. Or I can pass in some substring like EL and say hey, where's EL located and it'll say it's located starting at position one. If it doesn't find it, it gives me back a negative one to say it's not found. So just a few things that you should see, and we can kind of put these together in some interesting ways. Let's actually put them together in some interesting ways.

One of the things that's common to do is you want to iterate through a whole string to do something on every character in the string. So you might have some four loop, and it's I equals zero. You're gonna count up to the length of the string. So if our string is called STR, we'd say as long as I is less than STR's length, and then we would add our little friend, the I++ out here at the end.

This is going to go through essentially indexing the string from zero up to the number of characters that it has, and then inside here we might say CHAR CH equals STR dot CHAR at sub I, and what that means is one by one, you're gonna get each character in the string in CH, and then potentially you do something with those characters and then

you're done. So something you also sometimes do along the way is you do some work and you say hey, I want to build up some resulting string. Like, maybe I want to take a string and create a lowercase version of it.

So if I had some string STR and I wanted to build up some lowercase version of it, I would say, well, I need to keep track of what my result is. So I'll start off with some string result that I'll set to be the empty string. So it's two quotes in a row. That's the empty string. It means there's no characters in that string. It's still a valid string, but there's no characters. There's not even a space there. Then for every character that I'm gonna get in this loop, what I'm gonna do is I'm gonna tack on some version of it to result. So I might say result plus equals, which means [inaudible] onto the end of result, essentially what this is really doing.

Yeah, what it's really doing is creating a new string with an extra something added to the end and reassigning that back to result. So I might say result plus equals character dot two uppercase of CH. Let me erase this over here so we don't get anymore confusion. What this is actually gonna do is it goes through this string, character by character, pulls out the individual character, converts it to uppercase and pins it onto the result that I'm building up. So at the end, what the result will be is an uppercased version of the string that I'm originally processing.

That's an extremely common thing to do with strings – not necessarily converting them to uppercase, but pulling out characters one at a time, doing something on each character and building up some result as a result. Any questions about that? So I'm not checking the length of the result. I'm checking the length of string, but the length of result, if I were to actually compute it, is zero. Yeah, I need to put the double quotes here to say that results starts off empty. Otherwise I don't know what result starts as.

Last but not least, two uppercase and two lowercase I mentioned returns to you a new string that is the upper or lowercase version of that string. So basically, we've just written over there the equivalent of two uppercase if we returned whatever the result was. So let me show you one final example before you go, and that's reversing a string. You might say how might we reverse a string? Here's a little program that reverses a string.

It's gonna read in some line from the user and it's going to call a method reverse string that as you can notice over here, reverse string can't change the string in place because it's immutable, so it's gonna return to us a new string that we're gonna pass or we're gonna store in this thing called reverse. So we call – we ask the [inaudible] string. We're going to enter stressed, because I think by week four of the quarter, by the end of it, most people are feeling a little stressed, and so we're going to call reverse string, and reverse string comes over here doing this funkiness that we just talked about.

It starts off with some result, which is the empty string, and it's gonna count through every character in our original string, but it's gonna do it in a way so that by adding the characters one by one from the beginning and pinning them on, it's going to actually append it to the front instead of the back and that way, it's gonna reverse the string. So

that's the difference with this. Here, we're saying take whatever you have in result before and the thing you're gonna add is a character at the beginning instead of at the end, which ends up reversing our order.

So let's just go through this real quickly. I starts off at zero. It's less than length. I get the character at zero, which is gonna be an S, and I add that to result, so nothing exciting going on there. Result was empty before. Now it's just the string S. Add one to I. Still less than length, and what I now get in the character at one is the T. I'm gonna add the T to the beginning, right, so I'm gonna say T plus whatever's in result, so now I have TS in my result. The T doesn't go on at the end. I'm pre-pending it, basically, and then I add one.

I'm gonna get the R now added to the beginning and I keep doing this, adding the characters one by one at the beginning and at the end, I now have the value that's greater than the length. Even though the length is eight, I have the value that's eight, and so it's not less than the length, and so I return desserts, and back over here, stressed still remains stressed. People are still stressed out. Now they're just eating a little dessert along with it. You can actually multiply and divide characters, it's just sort of meaningless if you do. If you do, you sort of get – yeah. Sorry.

I'm sure you won't get an F, but you won't be multiplying characters, either. One final thing – we've still got a few minutes. Remember, we talked about how education is one of those things that if you get less of it, you're happy? If you bought a car and the car had three doors, you'd be really upset if it was supposed to be a four door car because you'd be like I paid \$10,000.00 for this car and it's only got three doors. It's missing one of the doors. But if I let you out of here five minutes early, you'd be like rock on! Less education! Same money.

I've never been able to understand – well, actually, I did understand that about 15 years ago when I was sitting there. But now I don't understand it anymore. I want to give you one more quick example, which is computing a palindrome. So what we're gonna do is write a function that computes whether or not something is a palindrome. Do you know what a palindrome is? How many people know what a palindrome is? It's a word or phrase that is the same forward or backwards. Most people know what it is.

Now, I will ask you the more difficult question – who created the world's longest palindrome? No, it was not me. It was, actually, however, my old boss, interestingly enough, a guy named Peter Norvig, who claims to have created the world's largest palindrome on November 23, 2003, and it's a palindrome that is 17,259 words long. Yeah. He did it with a computer. He's actually a wonderful guy, but created the world's longest palindrome, and you could probably create one that's longer using a computer. But how do you actually determine that something is a palindrome?

We're going to just do a little bit more math on strings than we might have otherwise done. Here's a function that computes is something a palindrome when we pass it a string. Now, to figure out if something's a palindrome, it has to be the same going

forward and backwards, which means to do that, one simple way of doing that is to say hey, something like the word racecar is a palindrome. One way I can figure out if it's a palindrome is the first letter equal to the last letter? If it's not, I stop immediately and if it is, then I check the next letter with the next to last letter, and I keep doing this.

Now, the question is how long do I keep doing this? How long do I need to keep doing this for? The length divided by two. I don't need to do it the whole length because eventually, I'll just cross over and I'll just redundantly check the other side. But if I check halfway this way, it means that if I was comparing the characters pair wise, I also checked halfway that way. The other thing that's fun about this is if I happen to have something that has a length three, like the word bib, the middle character is always equal to itself. I don't need to check it.

As a matter of fact, if I check the length three and divided it by two, I get the integer one, which means you really only need to do one check of the first letter and the last letter, and if those match up, don't even bother checking the middle. So it works nicely for words that have an odd number of letters by just optimizing out that middle character. If I actually have four letters like noon, you actually have four divided by two. You need to do two checks, and that will actually check all the characters you care about. So that's what we do here. We have a four loop up to the length divided by two.

It's doing integer division, and what we're gonna do is we say is the character at the beginning or at index 0 equal to the character at the end? Because we start counting from zero, the character at the end, if it has a length of nine, is actually at index eight, which is why we have this extra – when we do the subtraction, we add an extra one. It means subtract off an extra one 'cause if your length is nine, you really want the eighth character. So basically, as I increases, we check increasing characters from the front and decreasing characters 'cause we're subtracting off 1 from the back.

As soon as we find the pair that doesn't match, so as soon as these two things are not equal, we say hey, I don't need to check anymore. I return false in the middle of the function, which means the rest of the function doesn't need to worry about. All the local variables get cleaned up and I return false, because as soon as I find it, I don't even say hey, let me just check the rest just for laughs. Oh, you were just one character away. No, we just put you out of your misery immediately and we return false.

If you manage to make it through this four loop, it means you never hit a case where the two characters you were checking were not equal to each other, because if you did, you would have returned false. So if you never return false, you're good to go. You completed the whole loop, and so you'll return true at the end. Any questions about that? I will see you. Have a good weekend. I'll see you on Monday.

[End of Audio]

Duration: 51 minutes



## Programming Methodology-Lecture13

**Instructor (Mehran Sahami):** So welcome back to Week 5 of CS 106A. We're getting so close to the middle of the class, so a couple of quick announcements before we start. First of which, there are three handouts. Hopefully, you should pick them up. There's this week's section from some string examples, and kind of some more string examples that we're gonna go over today.

And now, since we're getting to the middle of the quarter, it's that time for the mid-term to be coming up. So, in fact, the mid-term is next week. It's a week from Tuesday. I know. The quarters go by so quickly. If you have a conflict with the mid-term, and by conflict, I mean an *unmovable academic conflict*.

Like, if you have another class at the same time, not like, oh, yeah, Tuesday night, you know, October 30, yeah, around dinnertime, I was gonna go out with some friends for dinner, but the mid-term is really cramping my style so I can't take it then. That's not an *unmovable academic conflict*. So if you have an *unmovable academic conflict* with the mid-term time, check the schedule.

Send me email by 5:00 p.m. this Wednesday, so you have a few days to do it, 5:00 p.m. by this Wednesday. Don't just tell me you have a conflict. Let me know all the times you are free next week to be able to take the alternate exam 'cause what I'm gonna do is a little constraint satisfaction problem. I'm gonna take everyone who's got a conflict with the mid-term, find out the time that everyone can take the exam an alternate time.

Hopefully, there will be one, and we'll schedule an alternate time. So only email me if you have an *unmovable academic conflict*. Make sure to list all the times you're free, and let me know by 5:00 p.m. Wednesday 'cause after 5:00 p.m. Wednesday, I figure out what the time is, and we ask for room scheduling. And that's it. I can't accommodate any more requests after that. So 5:00 p.m. Wednesday, know it, live it, learn it, love it, mid-term.

A couple of things about the mid-term you should also know, one of which is it is an *open-book exam*. It will be all written so leave your laptops at home. You won't be using a laptop. But if you ever get a job in computing where someone says, hey, guess what, you have to memorize, like, the Java Manual, don't take that job. So in computer science, we say, hey, you don't need to memorize all this stuff 'cause people, actually, look it up in the real world when they need it.

So it's an *open-book, open-note exam*. You can bring in printouts of your programs, and I would encourage you to do that. You can bring in your book, only the book for the class, by the way. Don't bring in a whole stack of books. And the reason why we make that restriction is because we – actually, in the past, there has been bad experiences with someone who brings in a whole stack of books. And they spend so much time trying to look things up in a whole stack of books that they just run out of time on the exam. It doesn't make any sense.

All the information you're gonna need for the exam will be in your book, and in the Karel Course Reader. So open book, that's course text, open Karel Course Reader, open any handouts or notes you've taken in the class, and printouts of your own programs. So you can bring all that stuff in. Another thing about the mid-term is you may be wondering, hey, what kind of stuff's on the mid-term, so on Wednesday I will give you an actual sample mid-term and the solutions for it.

I would encourage you to actually try to take the mid-term, the sample that I give you under sort of time constraint conditions. So you get a sense for 1.) What kind of stuff is on the mid-term? 2.) How much time you're actually spending thinking about the problems versus looking stuff up in books. And that will give you some notion of what you actually need to study to sort of remember quickly versus stuff you might want to look up.

Another good source of mid-term-type problems is the section problems. So like, the section handout this week, actually, has more problems than will probably be covered in section, just to give you some more practice. So section problems are real good. I'll give you a practice mid-term. That will give you even more sort of examples.

And if you want still more problems to work through, I'd encourage you to work through the problems of the exercise at the back of every chapter of the book. Those are also similar in many cases to the kinds of problems that'll be on exams. So you'll just have problems up the wazoo if you want to deal with them, okay. So any questions about the mid-term next week? We can all talk about it more as the week goes on, next week, Tuesday.

All right, so with that said, any questions about strings? We're gonna do a bunch more stuff today with strings and characters. But if there's any questions before we actually dive in to things, let me know now. And if you could use the microphone, that would be great. One more time, take those microphones out, hold them close to your heart. Air and gear, they're lots of fun, they're your friend. Keep the microphone with you.

**Student:** Actually, sorry, about the mid-term, is it going – what's the cutoff of the mid-term in terms of, like, caustes.

**Instructor (Mehran Sahami):** Right, so the mid-term for stuff you need to know, the cutoff will be Wednesday's class. So, basically, you'll have a whole week of material that you won't need to be responsible for that will be from this Wednesday up until the mid-term. The other thing, though, is to keep in mind that a few people have asked, well, do I need the book versus your lectures for the mid-term.

You need to know the lectures, and you need to know all the material from the book that is covered with respect to lectures, which is most of the material from the book. But there's a few cases where we go over something very quickly in class, or I say refer to this page of the book or whatever. That stuff you're responsible for knowing. Stuff that

I've, like, explicitly told you you don't need to know, like, polar coordinates, aren't gonna be on the exam, okay.

So the exam will be more heavily geared towards stuff from lecture, but you still should know all the stuff from the book that we've kind of referred to in lecture as we've gone along, allrighty. All right, so let's dive into our next great topic. Actually, it's a continuation of our last great topic, which is strings. And so, if we think about strings a little bit, one of the things we might want to do with strings, is we want to do some string processing that also involves some characters.

So how are we gonna do that? One thing we might want to do is let's just do a simple example to begin with, which is going through a string, and counting the number of uppercase characters in the string. And the reason why I'm gonna harp on strings a whole bunch – we talked about it last time – we're gonna talk about it this time – guess what your next assignments gonna be. It's gonna be all about string processing. So it's good stuff to know, okay.

So we might want to have some function. Count uppercase. And that's a function I've actually given to you in one of the handouts, so you don't need to worry about jotting down all my code real quickly, but you might want to pay close attention. And what this does, is it gets past some string, STR, and it's gonna count how many uppercase characters are in that string. So it's gonna return an int.

And let's just say this is part of some other program, so we'll call this private, although you could make it public if it was in some class that you wanted to make available for other people to use. So if we want to count the number of uppercase characters, what do we want to think about doing? What's the kind of standard idiom that we use for strings? Anyone remember? What? We want to have a foreloop, [inaudible] somewhere over here.

Yeah, it's just raining candy on you. We want to have a foreloop that goes through all the characters of the string, sort of counting through the character. So we can do that by just saying for N2i equals zero, "i" less than the length of the string, right. So STR.length is the method we use to get the length of the string, and then i++. And this is gonna loop through all the characters of the string. Okay, where actually it's gonna loop through some number of times, which is the number of characters in the string.

Now we want to pull out each one of the characters, individually, to check to see if it's an uppercase character. What method might we use to do that? Get a character out of a string in a particular position. Come on, I'm begging for it. Char at – it's like where'd it go? It's just gone, char at, and we'll just for the delayed reaction, we'll do it in slow mo. Anyone remember "The Six Million Dollar Man," that show? No, all right. Get another man.

I'm just getting so old, I gotta hang it up. And the thing is, I'm not that much older than you. But it's just amazing what big a difference a few years makes. So char CH is going

to be from this string. We're gonna pull out the char at apposition i. So now, we've actually each. We're gonna loop through each character of the string, pulling out that character, and we want to check to see if the character's uppercase.

We could actually have an if statement in here to check to see if that CH is in between uppercase A and uppercase Z, which is kind of how you saw last time we could do some math on characters. We're gonna use the new funky way, which is to actually use one of the methods from the character class, and just say if. And the way we use the methods from the character class, we specify the name of the class here as opposed to the name of an object because the methods from the character class are what we refer to as static method.

There is no object associated with them. There are just methods that you call and pass into character. Is uppercase because this returns a Boolean, and will pass at CH to see if CH is an uppercase character, okay. If it is an uppercase character, okay. If it is an uppercase character, we want to somehow keep track of the number of the uppercase characters we have. So how might we do that? Counter, right.

So have some int count equals zero, up here, that I want initialized. Who said that? It came from somewhere over here. Come on, raise your hand. Don't be shy. It's a candy extravaganza. So if character is uppercase, CH then got count, we're just gonna add 1 to. Otherwise we're not gonna increment the counts. It's not an uppercase character. And then, we end the foreloop. So this is gonna go through all the characters of the string. For every character check seeks the uppercase.

If it is, increment our count, and at the end, what we want to do is, basically, return that count, which tells us how many uppercase characters were actually in the string, okay. Is there any questions about this? This is kind of like an example of the sort of vanilla string processing you might do. You have some string. You go through all the characters of the string. You do some kind of thing for a character of the string. In this case, we're not creating a new resulting string. We're just counting up some number of characters that might be in the string.

So we can do something a little bit more funky. This is kind of fun, but it's sort of like, yeah, just basic kind of string and character stuff. Let's see something a little bit more funky, which is actually to do some string manipulation to break the string up into smaller pieces. And so what we want to do is replace some occurrence of a substring in a larger string with some other sting, sort of like when you work on [inaudible], when you do Find/Replace.

You say, hey, find me some little string, or some little work that's actually in my bigger document. I'm gonna replace it with some other word. We're actually gonna implement that as a little function, okay. So what this is gonna do, we'll call this replace occurrence just to keep the name short, but, in fact, all we're gonna do is replace the very first occurrence in a string. So we're gonna get past int some string, STR, and what we want to

do is, basically, have some original string, which is the thing that we want to replace, with some replacement string.

So we're gonna get past three parameters here. Will call this RPL for replace, okay. Which is the large string, a piece of text that I want to replace some word in, the original word that I want to replace, and the thing that I want to replace it with, okay. And so what I want to do because strings are immutable, right. I can't change the string in place. I have to actually return a new string, which has this original replaced by this string.

So, this puppy's gonna return the string, and we'll just make this private again, although we could have made it public if we wanted to have it in the library that other people would use, or a class that other people would use, okay. So how might we think about the algorithm for replacing this original string with the replacement. What's the first thing we might want to think about that we want to do with the original string.

Do, do, do, do, a little concentration music. We want to find it, right. We want to see if this original string appears somewhere on that string, right because if it doesn't we're done. Thanks for playing, right, but that's actually the good things for playing. It's sort of like you got no more work to do. And there's, actually, some methods from the string class that we can use to do that.

So there's a string in the string class called "index of." And what index of does is I can pass it some string, like the original string I want to look up, and it will return to me a number. That number is the index of the position of the first character of this string if it appears in the larger string. So the larger string is the one that I'm sending the message to, and I'm asking it do you have this original string somewhere inside you.

If you do, return me the index of its first occurrence. And if you don't, it returns a negative 1. So I'm gonna assign this thing to some variable I'll call index, and first of all, I want to check to see if I have any work to do. If index is not equal to negative 1, then I have some work to do. If it is equal to negative 1, that means hey, you know what, you want it to replace this original string, inside string STR. That original string doesn't exist, so I got no work to do.

You just called, like, find and replace in the word processor, and the thing you wanted to find wasn't there, okay. So in that case, all I would do is I would just return STR, right. Sort of unchanged, if I assume that I'm not doing what's inside the braces. If I do find that string, though, I'm gonna get some index, which is not negative 1, which is the position of this original string.

So let's do a little example just to make this a little bit more clear what's going on. So if we were to call this function – do, do, do, do, do – and pass in the string, STR. So here's STR that we're gonna pass in. We'll just put it in a big box, and we'll say, at this point in life, everyone's just friendly. So we say Stanford loves Cal, right. Sometimes you have to distort reality in order to make an example. All right, so we have Stanford loves Cal.

That's our original string, STR, and we might want to say, well, you know, this is, really, not always the way life is. Really, the way life is, is we want to replace the occurrence on STR of the word "loves" with kind of a more realistic example, like the word "beats," right. So what we want to do – and then we're gonna – this is gonna be some string that comes back, will find it back to STR.

And the question is, when we call this, what index are we actually gonna find in here of the original string. So strings we start counting from zero. Zero, 1, 2, 3, 4, 5, 6, 7, 8. The nine is where the L is at. And it keeps going. And 11, 12, 13, just put these all together – 15, 16, 17 is the L and that would be the end of the string. Sorry, the numbers are a little bit small. But the key is this L is at 9, okay.

So when I call string index up original, it says there's the word, or the string, "loves," up here somewhere in the larger string. Yeah, it does. It appears at Index 9 so that's what you get. So if I've just gotten Index 9, and what I want to do is construct some new string that, essentially, is going to have this portion removed from it, how do I want to do that.

What I want to think about is the way I construct that string, it's from three pieces. The first piece is everything up to the word I want to replace. That's Piece No. 1. The second piece is the thing that I actually want to replace, the string I'm replacing with, right. So this becomes Piece No. 2. And then, everything else after the piece I've replaced is Piece No. 3. So if I can concatenate those three pieces together, I'm going to essentially get the new string, which has this part replaced.

And the question is how do I find the appropriate indexes inside my larger string to be able to actually do the replacement, okay. So first thing that I'm gonna do here is say get me the first portion. So what is, essentially, the substring of the original string up to this L position. So the way I can do that is I can say STR, substring, and I'm gonna get the substring starting at zero 'cause I want to start at the beginning of the string, and I want to go all the way up, but not including the L.

That means the last position in substring. Remember, in substring you give it two indexes. You give it the starting point, and the position up to, but not including that last chapter. That's Position 9. Where am I getting Position 9 from this thing? From index, right. Index says where does love start. It starts at Position 9. I'm, like, hey, that's fantastic. So zero up to index, or zero up to 9 is Stanford and the states. It does not include the L. So I get that portion.

Then I say well, to that – I'm not done yet, so premature [inaudible] in there. Always gotta watch out for that, bad time. So what we're gonna add to that is we're gonna add the string that we want to replace in here, "beats," which happens to be the string called the replacement, or RPL. And then to that, we want to add one more string. And that's, essentially, everything from after "loves" over, to get that third piece, okay.

So what I want to know is what's the index of the position at which I need to get characters over to the end. That happens to be Position 14. What is 14 equal to, relative to

the kinds of things I have over here? It's index 'cause I have to first get over to the 9, then I need to jump over the length of this thing, which is the length of my original string. So if I add to index, what's my original dot link, what that gives me is the index from which I want to take a substring over to the end of the string.

So if I want to take a substring, this becomes an index to the substring function, or the substring method. And so from the string, what I do is I take the substring, starting at Position 14. Notice I haven't given a second index here. In this case I gave two indexes. I gave a start and end position. Here I just gave one index, and what happens if I only give one index? It goes to the end.

So that's part of the beauty is a lot of times you just say, hey, from this position go to the end. And so that's what I get when I put all these string things together. And what I need to do is these three things are just pieces. I'm concatenating them together. I assigned them back to STR. And then, when I return STR here, I've gotten those three pieces concatenated together. Is there any questions about that? Un huh.

If love appears more than once, index has just returned the index of the very first occurrence. There's actually a version of index sub that takes two parameters. One is the thing you're looking for, and the second is from which position you should start looking for it at. And so you could actually say look for love starting at Position, you know, 13, and then it wouldn't actually find love in the remainder of the string. So there's a different version of index of, but index of always returns the index of the very first occurrence of the string you're looking for in that string.

So let's actually do a little example of this in a running program. Do, do, do, do, do. And we'll do replace occurrence. And one thing that actually goes on at Stanford, which I thought was an interesting thing when I got here professionally, is we don't like to speak in full terms. So if we want to Stanfordize some strings, we do all these string replacements.

We sort of say, you know what, if you have Florence Moore in your string, that's really FloMo. And Memorial Church is memchu; AmerSc, [inaudible]; psychology is psyche; economics, econ; your most fun class, CS 106A. So it's just what Stanford's all about. And so if we go ahead and run this, right. Here's the function we just wrote. Here's our little friend, replace first occurrence. Over here we called it replace occurrence. I'm being explicit and saying it's only replacing the first occurrence.

You could think of a way to generalize this to replace all occurrences in a string if you wanted to. But I didn't give you that version 'cause I might give you that version on another problem set at some point. So what we're gonna do is we're gonna ask the user, enter a line to Stanfordize. Notice I want to put Stanfordize inside double quotes. So I put it in these characters, /quote, which just means a single, double-quote character. That's how I print double quotes.

So it says read line for Stanfordize in quotes. I want to keep reading lines and Stanfordizing them until the user gives me an empty line. How do I do that? I check to see if the line the user gives me is equal to a quote-quote. So if it's equal to a quote-quote, it's equal to the empty string. That means, hey, you entered in – if we ask the user for a string, they just hit enter. They didn't enter any characters. That's the empty string, so we would break out the loop. It's our little loop and a-half concept.

Otherwise, we say at Stanford we say, and we Stanfordize the line. And when someone's finally done, we say thank you for visiting Stanford, ha, ha, ha. That'll be \$45,000.00. [Laughter]. All right, so it's money well spent, trust me. Really. Okay, so replace occurrence string we want to run, and we come along, and it's running, it's running, it's running. Sometimes my computer's running a little bit slow.

I notice this weird thing last night. I'm gonna tell you a story while the computer's actually running. I couldn't type N's on my keyboard for some reason. And then I reset my computer, and I could. So at this point, I don't know if I can type N's. So let's just hope we can. So I live in – oh, I got the N – Florence – you should have been here last night. I was like, N, N, and I wasn't getting it – Florence Moore, major in economics – I can't even type today – and spend all my time on my most fun class.

And so, at Stanford we say I live in FloMo, major in Econ, and spend all my time on CS 106A, okay. And now, I hit return, Thank you for visiting Stanford. Go home. All right, so that's kind of a simple version of replace first occurrence. And notice you can actually replace multiple things in the same string, as long as the string that you're doing the replacement on you assign back to itself. And then we kind of do all bunch of these replacements in a row, okay. Is there any questions about that? Are you feeling okay about doing replacement. All right.

So now, it's time for something completely different. Although it's not completely different, it's just kind of different. And the idea is sometimes – and I always say that – sometimes you want to do this. Yeah, 'cause sometimes you want to do it, and other times you don't. Sometimes you feel like a nut. Sometimes you don't. Oh, man, I gotta start watching TV in this decade.

So, tokenizers. What is a tokenizer? A tokenizer is something, as they say it's a computer science term. All a tokenizer is, is we have some string of text. What we want to do is break it up into tokens. That's called tokenization. So you might say, Marilyn, what is a token? Like, last time I remember what a token was, is when I gave a dollar at the arcade and I got back, like, ten tokens instead of quarters. And you're like, yeah, Marilyn, I never did that. I had an XBox.

All right, so a tokenizer – anyone ever go to an arcade? All right, just checking. All right a token, basically, is a piece of string – a piece of string – is a string that has on the two sides of it, white space. So if I say, hello there, Mary, hello there and Mary are tokens. They are something that we refer to as delimited by what space, which means there is either spaces, or tabs, or returns, or whatever, in between the individual tokens.

We like to think of tokens as words, but computer scientists say token. Token is a more general term 'cause if I actually said hello there comma Mary, the "there comma" might actually be considered one token by itself 'cause it's just delimited by space. Here's a space here and has a space there, so the comma's in there. And you would think why comma's not part of the word. Yeah, that's why we call them tokens and not words.

So if we want to tokenize, there is a library that we can use in Java that actually has some fun stuff in it for tokenization. And that's Java util, so we would import Java.util.\*, and what we get for doing that, is we get something called the string tokenizer, which is a class that we can use to tokenize text. All right, so we get this thing called the string tokenizer.

How do I create one of these? Well, I paste string tokenizer as the type 'cause that's the class that I have, and I'll call it tokenizer equals I want to create a new tokenizer. So I say new string tokenizer, and the question that comes up here is well, what is the string you're gonna tokenize? That is the string that we passed to the string tokenizer's constructor when we create a new one.

So we might have some line here that we passed in. And now, line is just some string that maybe we got from the user for example by doing a read line. Maybe we were unfriendly and didn't give the user a prompt. We just like, if a blinking comes up, and there's like oh, I gotta turn and write something. It's just like when you're writing a paper, right. The blinking cursor comes up and there's nothing there. You just gotta fill it in.

So you write some line, and then we can say, hey, string tokenizer, I'm gonna create a new one of you, and the line I want you to tokenize is this line that I'm giving you to begin with. So once you get that line, there's a couple of things you can ask the string tokenizer. One of them is a method that returns a booleon, which is called has more tokens.

And the way this puppy works is you just ask this string tokenizer, like you would say tokenizer dot has more tokens, like; do you have more tokens? Have you processed the whole string yet? So if you've just created the new line, and this line is kind of sitting here like that, and it's saying do you have any more tokens. Yeah, I got tokens, man. I got tokens up the wazoo. You want tokens, I'll give you tokens. And so, has more tokens [inaudible] true.

If you process the whole string, when you will see when we get there, it'll say no, I don't have any more tokens. How do you get each token? Well, you ask for next token. And what next token does, when you call the tokenizer with next token, is it gives you the next token of the string that it's processing, as a separate string.

So if I started off the tokenizer with this line, I say hey, do you have more tokens. It says yeah. Well, give me the next token. So what it will return to you is hello. And it will be sort of sitting here waiting to give you the next token. You can ask if you have more tokens. It says yeah, give me the next token.

It will give you “there” and the comma ‘cause the default version of the tokenizer, the only think that delimits tokens – delimit is a funky word for splits between tokens – are spaces, or tabs, or return characters. But for a single line, you won’t have returns in that. And then you said you had more tokens. Yeah, give me the next token. It will give you “Mary” as a token that’s sitting here.

And then when you say do you have more tokens, that’s all, okay. And at that point, you shouldn’t call next token. You can if you want. You can experiment with this if you want to experiment with random error messages, but there’s no more tokens to give you. It’s all out of love. It’s so lost without you. It has no more tokens.

Yeah, Air Supply. Not that I would recommend that you have to listen to Air Supply, but sometimes you hear a song and you can’t get it out of your head as much as you wish you could. Sometimes selective brain surgery would not be a bad thing, but that’s important right now. What is important right now is how do we put all this together at the tokenizer line. So let me show you an example of the tokenizer. This one’s very simple.

All we’re gonna do here is we’re gonna ask the user – I’ll just scroll over a little bit. We’re gonna ask the user to enter some lines to tokenize and we’re gonna write out the tokens of the string R, and then we’re gonna call the message sprint token. What sprint token’s gonna do, it’s gonna take in the string you want to tokenize.

It creates one of these string tokenizers – I’m so lost without you. [Laughter]. Can we make Marilyn snap? No. I know it’s like great fun to listen to when you’re, like, 14, and you just broke up with a girlfriend for the first time. And then, after that, you want to kill the next time you hear it. Fine, So, tokenizer. I’m glad we’re having fun though.

So what I’m gonna do is I’m gonna count through all the tokens. So I’m gonna a foreloop interestingly enough. Here’s something funky. I’m gonna have a foreloop, but the thing I’m gonna do in my foreloop, my test, is not to check to see if I’ve reached some maximum number. But my test is actually gonna be to see if tokenizer has more tokens.

So I have a foreloop that’s just like a regular foreloop, but I start off with a count that’s equal to zero, and you’re like that looks okay. I do a count ++ over here, and you’re like that’s okay, what are you counting up to Marilyn. And I say I’m counting up to however many tokens you have. And you go, oh, interesting. So my condition’s to leave, or to continue on with the loop, is tokenizer has more tokens.

If it has more tokens, then I’m gonna do something here to get the next token. I’m gonna keep doing this loop. But what the counter’s gonna give me is a way to count through all my tokens. So I can write out token number count, and then a colon, and then write out the next token that the tokenizer gives me. Is there any questions about there? Let’s actually run this puppy [inaudible]. Do, de, do. You can feel free to keep singing now if you want, if you want.

All right, so we're gonna do our friend. What's our friend called? The tokenizer example. Do, do, do, do, we're running the tokenizer, interline's tokenized, so I might say "I, for one, love CS." We're very formal here. And it says the tokens of the string are on notice. It got the "I" and the comma together as one token because as we talked about, spaces are the delimiter. And so "for" and then "one" with the comma, and "love" and "CS," and that's all the tokens we got.

And so at this point you might be thinking, yeah, man, that's great, but you know what. I really don't like punctuation. And sometimes I don't like punctuation, but I can't stop the user from using punctuation because even though I don't like to be grammatically correct, they do. So how do I prevent them from being grammatically correct as well, which is kind of a fun thing to do.

What you can say is hey, what I want to do is change my tokenizer, so that it not only stops at spaces, but it's gonna stop or consider a delimiter, any of this list of characteristics that I give it. So you give it a list of characteristics as a string. So here I'm gonna give it a comma and a space, okay. And this version of the string tokenizer constructor, what it will do is it will actually tokenize the string.

But think of the thing that you're using as your delimiter, or what chops up your individual tokens as either a comma, or a space, or anything you want to put in that string there. Each of the individual characters in that string is treated as a potential delimiter. So if you say "I for one love CS," ah, no commas. Why? Because commas are considered delimiter.

So it just gives you everything up to a comma or a space, and you could imagine you could put in period, and exclamation point, and all that other stuff, if you just want to get out the non punctuation here. So tokenizing is something that's oftentimes useful if you get a bigger piece of text, and you want to break it up into any individual words, and then maybe do something on those individual words, okay. Any questions about tokenization? Hopefully, it's not too painful or scary. All right.

So the next thing I want to do, will just pay for the smorgasbord of string, is I want to teach you about something that's really gotten to be an important thing about computer science these last few years, which is, basically, this idea known as encryption. And encryption is something that's been around for thousands of years. All encryption is, is it's kind of like sending secret messages.

You have some particular message. You want to send it to someone else, but you want to send a secret version of that message. And people have been doing this for thousands of years, actually, interestingly enough. They just didn't have very good methods of doing it until about the last, oh, 50 years. But you know they did it for a long time, and people broke encryption.

As a matter of fact, there's this really interesting book by Simon Singh. I'll bring in a copy, perhaps next class, if you're really interested, about the whole history of encryption.

It goes back thousands of years, and how, like, wars, and queenships, and kingships, and stuff, were, basically, lost in one on the strength of how well someone could break a piece of code.

But the basic idea of encryption, and it probably dates back even further than this, but one of the most well-known ones is something that's known as the Caesar cipher, not to be confused with the salad. But the basic idea with the Caesar cipher – I picked up the wrong newspaper – the Caesar cipher is that what we want to do is, basically, take our alphabet and rotate it by some number of letters to get a replacement.

What does that mean? That's just a whole bunch of words. So let me show you a little slide that just makes that clear. So in Caesar's day – I will now play the role of Caesar. I actually considered wearing a toga to class today. I just thought that was fraught with way too much peril. So I just decided to bring my little Caesar crown. And that's what I'm trying to find my little crown of reason stuff, but I couldn't. So I just got a little hat. [Laughter].

And so the basic idea – say you are Caesar – well, I did crown myself, actually. I knew someone here could actually take the crown from [inaudible]. That was Napoleon, a whole different story. I really like to take history and mix it up. It's just to see if you're actually paying attention. All right, the basic way the Caesar cipher works is we take our original alphabet. Here's all of our letters from A through Z.

We take that whole alphabet, and we shift it over some number of letters. Like let's say we shift it over three letters. So I take this whole thing, I shift it over three letters, so now the D lines up over here where the A should have been so I've shifted over these bottom characters. And the characters that kind of went off the end here like the A, B, and C, were kind of like whoa, we're going off the end. Where do we go? We just kind of shuffle them back around over here.

So the basic idea is we're gonna rotate our alphabet by N letters, and N is 3 in the example here, and N is called the key. So the key of the Caesar cipher is how many letters you're actually shifting. And then we wrap around it again. And now, once we've done this little wraparound, we take our original message that we want to encrypt. That's something that's referred to as the plain text. The plain text is your actual original message.

And we want to encrypt that or change it to our cipher text, which is what the encrypted message is, by using this mapping. So every time an A appears in the original, we replace it by a D. And a D appears in the original, we replace it by G, and a C appears in the original, we replace it by an F, etc., for the whole alphabet. Is there any questions about the Caesar cipher?

This is actually an actual cipher that, evidently, historians tell us that Caesar used in the days of yore. And you know, evidently, he was killed, so it didn't work that well. But you know most people, that's one of the things that when you were a little kid, and you had

like the Super Secret Decoder Ring, you were probably getting a Caesar cipher. All right, any questions about the basics of the Caesar cipher.

So what we're gonna do is let's write a program that actually can be able to encrypt and decrypt text according to a Caesar cipher, and we'll do it doing pop-down design. So we'll actually just do it on the computer together 'cause it's more fun that way. And because I'm Caesar, I will drive. So we're gonna have my Caesar cipher, all right. And I just gave you a little bit of a run message here. It's kind of the very beginnings of the program.

But all this does – it's not a big deal. It says this program uses a Caesar cipher for encryption. It's going to ask for the encryption key. That means it's asking for the number by which it's gonna rotate the alphabet to create your Caesar key, or to create your Caesar cipher, and that's just our key that's an integer. So our plain text, that's the original message that we want to encrypt. We ask the user for the plain text, so we just get a line from the user.

And then what we're gonna do is we're gonna create our cipher text, or the encrypted text by calling a function called encrypt Caesar. We're sort of giving a directive. It's kind of like an inquisitive tape. Encrypt Caesar, and we give it the plain text, and we give it the number for the key that we want it to encrypt using. And then, hopefully, that will give us back the encrypted string, and we're just gonna write that out, okay. S

o how do we do this encryption? All right, so at this point, and it should be clear that the thing we want to write is probably encrypt Caesar. So what we're gonna do is we're gonna write a pleasant message, and what is this puppy gonna return to us? String, right 'cause that's what we're expecting, the encoded version of this particular message as a string. So we'll call this encrypt Caesar.

And what's it getting past? It's getting past the string, which we'll just call STR, and it's getting past an integer, which will we will refer to as the key. So if I want to think about doing the encryption, right, what I'm gonna do is, on a character-by-character basis, I want to do this replacement. I want to say for every character that I see in my original string, there is some shifted version of that character that I want to use in my encrypted string.

So in order to do that, I'm gonna use my standard kind of string building idiom, which says I start off with a string, which I'll call results, which starts of empty, right. It says, quote, quote, empty string. And I'm gonna do a foreloop through my string that I'm giving to encrypt. So up the string's length, I'm just gonna count through and get each character. So I'll so sort of a standard thing.

I'm gonna say CH and I'm gonna essentially get the character that I want to get from the string, so I'll say STR.char@chat@char@I. So I've now gotten my character. I want to figure out how to encrypt that character, okay. So I think to myself, wow, gee, while encrypting the character involves all this stuff, doing the shift and all that, that's kind of complicated. Maybe I should just create a function to do it. All right, that's the old notion

of pop-down design. Any time you get somewhere, well, you're, like, wow, that's kind of complicated.

Maybe I don't want to stick this all in here and figure it out. But it's the smaller piece, which is just dealing with a single character instead of dealing with the whole string. Let me write a function that will actually do it, or a method that'll actually do it. So what I'm gonna do, is I'm gonna append to my results what I get by calling encrypt, a single character.

So I'll just call it encrypt char and what I'm gonna pass to it is the character that I want encrypt, and I need to also pass to it the key so it knows how to do the appropriate shifting to encrypt that character. And after it does this encryption, I'm just gonna say hey, if you've successfully encrypted all of your strings, what I want to do is return, RTN, my results, right. That's your standard string idiom.

I start off with an empty string. I do some kind of loop through every character of the string. I'm gonna do the processing one character at a time, and return my results. Everything in that function that you see, or in that method that you see, except for that one line, should be something you can do in your sleep now. You've seen it, like, over and over. We just did it a couple of times today. We did it a couple times last time. It's the standard kind of thing for going through a string one character at a time.

And now, we reduced the whole problem of encrypting a whole string to the problem of just encrypting a single letter. So what I'm gonna have in here is private, and this is gonna return a single character called – and this puppy's called encrypt char. And it's gonna get passed in some character to encrypt as well as the key that it's gonna use to encrypt it. And now I want to figure out how do I encrypt that single character.

So what's something I could do to think about how this character actually gets encrypted. How do I want to do the appropriate shifting of the character. So let's say I've gotten an uppercase A. Let's assume for right now all my characters are uppercase. As a matter of fact, that's a perfectly fine assumption to make.

The solution you've gotten to, it assumes all the characters are uppercase, so assume all the plain text is uppercase, and I want to return to the encrypted cipher text also in uppercase. Let's say I've gotten an uppercase A, okay. And my T is 3. So I want to do, is take that A somehow, and convert it to a D. How do I do that?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Uh huh. I want to add 3 to the character. Now the only problem is I might go off the end of the character. If I just add 3, and I have a Z, I'm gonna – if I just have the A and go to D, that works perfectly fine, but if I have a Z, I'm gonna get something like an exclamation point, or something I don't know 'cause I go off the end of the character. So I need to do slightly a little bit more math. And what I'm gonna do is say take this character, and subtract from it uppercase A.

That's gonna tell me which character in the alphabet it is, which number character it is, right. Now, if I add the key, what I get is the number, or the index, of the shifted character. So if I had an uppercase A, and I subtract off uppercase A, I'm gonna get a zero. I now add the key, so I get 3. And you might say, well, if you just convert that to a character, you get a D. That's perfectly fine. Yeah, but if I had a Z and I subtract off an uppercase A, I get 25.

If I add 3 to 25, I get 28, which is now outside the bounds of the alphabet. How do I wrap around that 28 back to the beginning of the alphabet. Mod it by 26, or we do with the remainder operator by 26, right. So what that does is it says if you've gone off the end, basically, when you divide by 26 and take the remainder, if you've gone off the end, it kind of gets rid of the first 26, and wraps you back around the beginning.

So if I do that, this will actually work to get me the position of the character wrapped around, and once I've gotten the position of the character, here's the funky thing. I need to add the A back in because if I have, let's say, an uppercase A to being with, and I subtract out uppercase A, that gives me zero. I add the key. That gives me 3. I do the remainder by 26.

Three divided by 26 as the remainder is still 3. So now I have the number 3. I need to get that 3 converted to the letter D. How do I do that? I add the letter A to that 3, okay. Is there any questions about that?

Now, the final funky thing that I need to do, is if I want to assign this to a character, I can't do this directly. Notice if I try to do this directly, I get this little thingy here. And you might say Marilyn, what's going on? Like you told me characters were the same as numbers, and everything I've done so far has to do with numbers, so why can't I assign that to a character?

And this little error message comes up. And this has to do with the same thing when we talked about converting from real values to integers. Remember when we went from a real value to an integer. We said you'll lose some information if you try to truncate a real value, like a double to an integer. So you explicitly have to cast it from being a double-splint integer. Same thing with characters and integers. The set of possible integers is huge. It's like billions and billions. The set of characters is much smaller than that.

So if you want to go from an integer back to a character, you need to explicitly say convert that integer back to a character. So we need to explicitly do a cast here back to a character. And if we do that, then we're happy and friendly. Did I get all my friends right? One, two, three, one two three. All right, why is this still unhappy? Oh, duplicate variable CH, yeah. Let me call this C.

Actually, let me make my life easier. This is a thing I just want to return, so I'm just gonna return it. Do, do, do, do, do. I won't even assign it to any temporary variable. We'll just return it 'cause now I'm upset. No, I'm really not upset. We're just gonna return it.

So, hopefully, that will give us our little Caesar cipher. So let's go ahead and run this, and see if, in fact, it's working. Any questions about this while this is running?

I'll sort of scroll this down a little bit so you can see what's going on for that single character. So this was my Caesar cipher. So we say, et tu, brute. Illegal number format. Yeah 'cause that's not the thing I wanted to encrypt. My encryption here is 3, then I will give it the plain text I was to – everyone's like what did he do. Sometimes it's the obvious that's wrong, and you just need to read. All right, there we actually go.

Now, there's a little problem here. See, the little problem is the spaces actually got encrypted. We don't want to encrypt spaces. We only want to encrypt things that are actually valid characters. So we're not quite done yet. What we need to do is come back over here and say, hey, you know what, for my encrypt character, I wasn't quite as bright as I thought I was. I need to make sure this thing's actually in uppercase character before I try to encrypt it.

So we can sort of do that if I just call my little friend character. And the thing we want to say is, is uppercase, and I'll pass at CH. So if it's already – if it's an uppercase character, then I'll return this. Otherwise, what I'll do – I'll tab this in – is I will just return CH unchanged. So if I've gotten something that isn't actually a character, then I'll return – do, do – why is this unhappy again. Oh, semicolon, thank you. All right.

**Student:** [Inaudible]

Now I got an extra one. Notice it doesn't give me an error on the extra one 'cause, actually, semicolon without a statement is the emsin statement. It's perfectly fine, but thank you for catching the straight semicolon. So we'll go ahead and run this, and we'll try our friend, et tu, Brute, again. Sometimes it's all about texting, and so we have et tu, Brute, and now we're okay 'cause we're not encrypting anything that is not a letter.

So sometimes we think we're okay. We need to go back and just make sure we actually do the texting. Any questions about this? If this all made sense to you, nod your head. If this didn't make sense to you, shake your head. Feel no qualms about shaking your head. If you're someone in the middle, just stare and stare at me. No, if you're someone in the middle, shake your head. Okay, un huh.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Why don't I need an L statement, like, say here? 'Cause if I hit the return, I return from the function immediately and I never, actually, get it down to this return. So if I hit this return statement, I'm done with the method. As soon as I hit that return, it doesn't matter if there's any more lines in the method. I'm done. I actually return out.

So the one other thing we might like to do with this that doesn't quite actually work right now. Let's actually try running this, then I'll show you what happens, just to show you

that it's bad time. If I actually encrypt something like et tu, Brute, and I want to decrypt it, I might say hey, try to use minus 3 as your key, and if I try to put in the text – I don't even remember what the text was that I wanted to encrypt. I guess this funky thing with question marks, and it's just not working to move in the negative directly.

So I want to allow for my Caesar cipher to also be able to decrypt information, which means if I got a Caesar cipher by encrypting with a key of 3, if I give it the text that's been encoded, and I give it the key minus 3, it should shift it back three letters and actually work for me. So how do I do that? Well, it's something that has to deal with each individual character.

If I want to encrypt each individual character, I need to figure out what's the right way of using the key, okay. Think about a key of minus 3. What's a key of minus 3 equivalent to? A key of 23, right. A few people mumbled it, so we'll just throw out some candy. If I want to go 3 in the opposite direction, if I want to go 3 sort of this way, as opposed to this way. It's the same thing as going 23 characters in the opposite direction.

So if I want to think about doing that, I can say, if my key is a negative number, so if my key is less than zero, there's some shifting I need to do of the key to actually get this puppy to work. So if my key is less than zero – as a matter of fact, I'm gonna do this once down here. So rather than doing it, and encrypting each character, I'm gonna do it over here by saying you know what, once I shift my key over, I want to use that same key to encrypt all my characters.

So I want to do the shifting just once up here. It makes sense to do it once for the string, and then I'll use my updated version of key. So here's what I'm gonna do. I'm gonna say take the key, and the way I'm gonna update key is I'm gonna say it's 26. And this looks a little bit funky, but I'll explain it to you in just a second – modded by 26.

And you might say Marilyn, why do you need all this math to actually pull it off 'cause if you do something, you could say why can't you just take 26 and subtract from it your key. So if you want to say minus – or add toward your key. So if you want to have a key of minus 3, isn't that just the same as adding minus 3 to 26. You'll get 23, aren't you fine. Yeah, that's fine for sufficiently small values of key.

So if this thing actually is minus 3, minus, minus 3 gives me 3, and if I were to – oh, missing a minus in here. Sorry, my bad. I had two minuses. I want to have another minus right there. So if key is minus 3, and I take a negative of minus 3, that gives me 3. Twenty-six minus 3 by itself would give me 23, which is the value I care about and that's perfectly fine.

But what happens if this key that someone gives me is something, for example, that's larger than 26. That's kind of bad time because if I subtract a number that's larger than 26 from the 26, so if this happens to be minus, let's say 27, and I say minus minus 27 is positive 27. And I subtract 27 from 26, I get minus 1. That's bad time.

So the reason why I have this 26 in here, is it says first take the key. They gave me some negative value. Take the negative of that, which gives you some positive value. When you mod it by 26, you will guarantee that that value they've given you is less than 26 'cause if it was 26, 26 mod, 26 is zero. Something larger than 26 gives me a remainder.

So as long as I mod by 26, I will always get back the appropriately mapped value, less than 26, and then I will subtract that [inaudible]. So just to make sure this actually works, what I'm gonna do is in my main program, I'm gonna say encrypt Caesar using this key. And then, do, do, do, so I have some cipher text.

I'm going to now – well, actually, let me write out the cipher text so I'll still use this print link. And then, I'm going to have some other string, new plain. A new plain is just going to be doing encrypt Caesar on my cipher text, so that should be my encrypted text with the negative of the key. So I want to, essentially, switch back to what I've got. And so I'll have print link new plane quote dot whatever the new – man, I cannot type to save my life – L print link. Thank you.

So now I run this puppy in our final moment together, 3 et tu, Brute. Well, at least I got it back even though I misspelled it. I got my mixed-up characters, and then I got my new plain text, which is the same as my original text, which I got just by, essentially, shifting in the negative direction. So any questions about that. Allrighty, then we're done with strings for the time being, and I'll see you on Wednesday.

[End of Audio]

Duration: 50 minutes

## Programming Methodology-Lecture14

**Instructor (Mehran Sahami):** Are we on? All right. Looks like we're on. Let's go ahead and get started with a few announcements.

So hopefully you're still turning in, or you will be turning in, your assignment three soon. There are three hand-outs today. They're not all in the back there right now. There was a huge problem with the copier today, and Ben is actually gone to make — to finish up making all the copies of all the handouts by the end of the class. So at the end of class you can pick up all three handouts. Some of you may have already picked up one. There were some copies back there. But the three handouts are basically your next assignment, assignment number four, which is a hangman game. So you'll be doing another game. This one's a little less graphically involved, unless you actually want to add a lot of graphics, which is another fun extension you can add to it if you want. But it sort of involves much more algorithmics with strings and also gives you practice using multiple classes. So this whole time we've talked about, oh, you can have all these different classes. Well, this is your chance to actually use multiple classes starting now in hangman.

The other two handouts are the practice midterm and practice midterm solutions. So those will also be here at the end of class. That will give you a whole bunch of details about the midterm and what the midterm actually covers, but it will also give you examples of real exam problems that have been given in the past. So you can work on them. Do it in the time, you know, time sort of setting so you can see what you're slow on and what you're fast on. But it's kind of in flavor, very similar, to what the real exam's gonna be in terms of what it covers, the kind of complexity of the problems, et cetera. So you can get that after class.

Last time to announce it. If there is midterm conflicts, email me by 5:00 p.m. today. So if you have an unmovable academic conflict with the midterm next week on Tuesday, 7:00 to 8:30 p.m., send me an email by 5:00 p.m. today. After 5:00 p.m. today I'm gonna take all the emails that I've got and try to figure out a time that everyone can make based on those emails and schedule the alternate time. So after 5:00 p.m. today I can't accommodate any other requests, unless of course you happen to be free at the alternate time you schedule, in which case you sort of got lucky. But if you have a conflict with the midterm, 5:00 p.m. today, send me email. That's the deadline.

Just in case you're wondering where you would actually take the midterm, it's in Kresge Auditorium, one of the few buildings on campus that's actually large enough to accommodate this class with alternate seating. So Kresge Aud. Anyone know what the significance of the, "K," in "Kresge" is? It's the same as the, "K," in "Kmart." So anyone ever gone to Kmart? Ever seen a Kmart? That's like Kresge-mart. That's where the, "K," comes from. So you wonder where all that money came from to donate that huge building. So midterm's in Kresge.

Along somewhat different lines then, we're gonna make a slight change to the syllabus. This won't affect anything that's on the midterm because the midterm will only cover stuff through today. Anything after today is not fair game for the midterm, will be fair game for the final exam. But a small change to the syllabus is — Friday's class will remain the same. We'll talk about files. But next week on Monday we were gonna start talking about debugging strategies. And I'm gonna defer that lecture until after the midterm. So actually starting on Monday next week we're gonna start talking about arrays. So if you want to keep up with the reading in the syllabus, that's Chapter 11. We're gonna start that on Monday, as opposed to on Wednesday. So I've just moved basically things up by one day for arrays. It's not a big deal.

And we'll actually talk about debugging after the midterm. Part of the reason for that is for your assignment number three — or, sorry, your assignment number four for hangman, it's actually a three-part assignment. It's all one big program that you write, but we sort of break it up for you into three parts. That third part that you'll do at the very end, which is perhaps actually the simplest part, will make use of arrays which is why we're gonna start talking about arrays. So you will have seen all that by the time you actually need to do that third part.

Last but not least, as you know, assignment three is due today so quickly let's take the pain poll. Just wondering, how many people actually added extensions to break out? Wow, lovely. How many people were really gunning for the plus plus? Cool. Good times. I look forward to seeing those, so it'll be a good time. So the — in terms of the pain poll, I want you to just let me know how much time it took you to do the basic assignment. So if you did extensions, don't count the time for the extensions. What I really care about is the base assignment.

Anyone under two hours? Just wondering. It would have been frightening if you were. I wouldn't expect anyone to be. Two to four? A couple people in two to four. Four to six? That's a pretty reasonable contingent. Six to eight? Also pretty healthy size. You can tell that the bar drawing is a very scientific process. Eight to ten? Okay. Ten to twelve? Beginning to quickly fall off. Twelve to fourteen? Fourteen to sixteen? I put in 16 to 18 this time just for good measure. Sixteen to eighteen? A couple folks. Eighteen plus? Anyone taking a late day? A few folks for 18 plus, all right. And anyone taking a late day. All right. We'll see where you end you. I always have this tendency to think, like, oh, if you're taking a late day it's like I should just count you in here, but that's really not the case. Sometimes you're just like, hey, I decided to go play in the sun or whatever. And the weather has been lovely, hasn't it? All right. With that said, world is normal, life is good, average is still, on average, less than ten. Everything we sort of hope for. We sing — we hold hands, we sing Kumbaya, and the birds are out chirping. Good time.

Any qu — well, I shouldn't ask you if there was any questions about break out because some people are taking late days. Any questions about strings? Because after today we're leaving — well, actually after last time we're sort of leaving strings behind. After right now, we're leaving strings behind. You will use them just to no end on the hangman assignment because it's all about text and strings and characters and it's just a happy

camper that way. But if you have any questions about anything you've seen with strings, ask them now or I won't say forever hold your peace. Ask them now or ask them at office hours, right? You can ask later.

Alrighty. So today's the day when we lift the covers on the machine. It's actually when we sort of like rip open your computer, although I'm not gonna rip open a computer. I considered doing that. But we're gonna rip open your computer and look at what's inside. And it's really, really small so you have to bear with me, okay? But we're gonna talk about is memory. And after today you will be experts in memory. You'll just be like, "Hey, man, I might not be able to remember anything, but I know all about the memory in my computer," which is the important memory to actually think about.

So the first thing in memory that we want to think about, right. When we talk about memory some people have heard the term like RAM for Random Access Memory. And so, like, you buy some new computer and you go, "Oh, I got, like, two gigs of RAM in my computer," right? This is what we're gonna be talking about, is what that RAM in your computer actually means, okay? And how it relates to your job of programs, and why it's all important.

So the very basic notion, the very simplest concept of memory inside the computer is something that's called a bit. And all a bit is — it's actually shorthand for a binary digit. So if you take binary digit and you squeeze it together real hard, all of sudden the extraneous letters pop out, and you get bit, okay? And this is just a zero or a one. It is a single binary digit. Binary being base two numbers which means there is only zeros and ones as digits in that number system, okay? And that's the simplest kind of thing. So somewhere inside your computer there's actually, like, a little piece of silicon or whatever and a little transistor that can keep track of a zero or one. So when you get — we won't go below the transistor level. If you're really interested in that take, like, you know, an E class. But the bit is a zero or one.

And then we take a column — a whole bunch of bits, and we stick them together into something called the byte. You're like, oh, that whole term like gigabytes. Yeah, this is what it's all about. What a byte is is eight bits. Okay. So basically eight binary digits strung together is one byte. Okay. That's all it is. Now, one byte of memory would not be very interesting, right? You — it's like, oh, I could have one really small number in my computer, right? Because eight bits can represent, like, an integer between zero and 255. It's not very exciting if that's all the memory you had in your computer.

So when we think about bits and bytes, we also like to think about larger pieces. There's a piece sometimes people refer to as a word, which actually has nothing to do with English words. But a word is generally the size of what an integer is used to store in some particular language. And it turns out an integer in Java is four bytes of memory. So it's 32 bits. Thirty-two ones and zeros comprise the space that your computer actually sets aside for a single integer in memory, which is why an integer has some bounded sides, right? You can't just have a number very close to infinity in an integer because it just has some finite size. It turns out to be four bytes. And that's something that sometimes people refer

to as a word. We won't use the word, "word," very much but just so you know. What we will talk a lot about are bytes. And so bytes — there's a lot of terms related to bytes that you've probably heard. So how many people have heard of a K or a kilobyte? Right. Probably a lot of people. And you would like to think that the metric system, right, kilo means what in the metric system? Thousand, right. So you would like to think a kilobyte is a thousand bytes. But, no. Computer scientists come along and they're like, "No, no, no, no, no. That's much too round and we're not decimal people. We're binary people." And the closest number to a thousand in binary is two to the tenth which is equal to 1,024.

So a kilobyte — one K when you talk to a person who's actually a computer scientist or someone who's putting the RAM together for machine, you actually, sort of, get a little extra, right. It's kinda like there you're, "Oh, I got some bonus RAM." And, like, this whole time you didn't even know. They were just giving it to you for free, right? And so there's this notion of a meg, right? Like a megabyte. And a megabyte is basically just a thousand kilobytes. As a matter of fact, it's not just 1,000 kilobytes, it's 1,024 kilobytes. Because we do everything in powers of two, so it's actually two to the 20th.

And then there is probably you — something you've heard of called a gig or a gigabyte, right? These are the common ones. This is 1,024 meg. So when you start adding these all together, right, you're actually sort of getting a lot of bonus memory for your gig. Because it turns out you're getting, you know, 124 megs, and each one of your megs is getting 124 K, and each one of your K is getting 1,024. So you're getting a little bit of a bonus. So now it's time for the advanced course. You're like, "Yeah, man, I got so much RAM in my computer. I got gigs and gigs." And so then I would say, "Oh, yeah, really? You got 1,024 gigs?" Because what would that be if that's what you had?

**Student:** Terabyte.

**Instructor (Mehran Sahami):** Terabyte. All right. So the next one is a terabyte. What comes after terabyte?

**Student:** Petabyte.

**Instructor (Mehran Sahami):** Petabyte. Okay. And now we're getting into dangerous territory. So each one of these is a factor of 1,024 greater than the next one, right? What comes after petabyte? Exabyte. There was a few folks. We'll see how long we keep you. How — what comes after exabyte? Anyone?

**Student:** Zettabyte.

**Instructor (Mehran Sahami):** Zettabyte with two, Ts. Not to be confused with Catherine Zeta-Jones. That's an entirely different zettabyte. So this is a zettabyte, all right? Just horrible, wasn't it? And then after zetta, just to drive the point home, is a yattabyte. I always like to think of it as like a yoda byte, right? It's just, like, so much

information you can't even conceive of it. But it's a yattabyte. Oh, yeah, that's a yatta good time. We just not even close to this, okay?

A couple things if you just want to think of orders of magnitude just so you have some vague idea. If you took the printed collection in the U.S. Library of Congress, that's roughly equivalent to about ten terabytes. Is all the printed material if you thought of each character being represented by one byte. Okay. And you're like, "Oh, that's interesting." If you took all printed material ever, that's about 200 petabytes. So at this point we're getting pretty close. Like, you can go down to Fry's, right, or some other place. I shouldn't name a particular manufacturer or a particular vendor. And you can probably buy about a terabyte of storage relative — well, I shouldn't say relatively cheaply but for a couple hundred bucks, okay? And you kinda crank that up and you can imagine someone for a couple hundred thousand dollars could have a petabyte, right? And then you sort of have a couple million dollars and you could have enough storage to store all the printed manner ever, right? Which is not that far away.

And then the — I don't know how someone came up with this number, but I actually found this number and I was like, how do you measure that? Which is if you took the amount of storage that would be required to store all words ever spoken by human beings, okay? Like, where did they get that number, and it's increasing right now, right? They estimate that that would be about five exabytes. Right. So we're still sort of in this range, right? Like, yeah, Catherine, she's safe. And the little Zedi guy, yeah, we're not even close, all right? So we still got a lot more information to produce someday. And you're like, "Oh, I can do that now with my program. I can just have an infinite loop, and I can generate this." It'll still take a while. All right. But it's interesting just the amount of information that we're actually producing.

Now, when we think about actually storing all this stuff inside a computer — so now that you've got some notion of, kind of, orders or magnitude for these things and what these little bits and bytes really are — the other thing that we want to think about is what kind of representations do we have when we actually store this stuff? So it turns out because we like to think in binary, right, we have bits which are binary digits. Sometimes it's easier for us to think not just in terms of all the ones and zeros themselves but some other base system that's easier to keep track of. And so one you might think of is Octal, which is base eight numbers, right? So remember when we did the little asky chart and I showed that to you in Octal, and I was like, "Oh, these are base eight numbers." Like, the digits go between zero and seven. Yeah, that was Octal. That's something that some computer scientists do, but most computer scientists, at some point in their life, that — as a matter of fact, I would say all of them in some point in their life. And now is that point in your life.

We'll deal with something called hexadecimal. And hexadecimal, hex being six — decimal — yeah, it's hexicomical. Hexadecimal, six, and then, "deci," being ten is base 16. We put the six and the ten here. You're like, "Shouldn't that be 60?" No, it's 16. We add. So this is base 16 numbers. And you might say, "But, Maron, yeah, like, I remember these three, four, five, six, seven, eight and nine and those are all fun." And, like, when

you told me about binary I was like, “Yeah, it was just these two,” and when you told me about Octal you were like, “Yeah, cut here.” Now you’re telling me about base 16. I got no more digits. Where are my other digits coming from? And you’re like, “Well, I could have, like, a ten.” And I’m like, “No, ten is two digits.” And you’re like, “So where does it come from?”

Well, what else do I have beside digits? I got letters, right? So in fact, A is ten, B is 11, C is 12, D is 13, E is 14, and F is 15. And so when I want to write some number in hexadecimal — I’ll just write a number in hexadecimal because it’s fun to write numbers in hexadecimal for about two numbers. And then it stops being fun. But just so you actually see it. 2B — you’re like, “Yeah, over on the other side of the quad, this is Shakespeare.” On this side of the quad, this is hexadecimal. What is 2B? It’s base 16. What does that mean? That means this is the ones column. This, in decimal, would normally be the tens column. This becomes the 16s column. The next column, if we had one, would be the 256s column, et cetera, okay? So if we want to think of this number we have two in the 16 column so that’s 2 times 16 is 32. And we have a B in the ones column. Well, what’s a B? That’s 11. Okay. So we add 11 to that so we have 1 times B which is basically equal to 11 and what we get is 43. So 2B is actually the number 43 in decimal that we like to think of, okay?

So you might say, “All right, man. That’s great. Yeah, sure, whatever. Why are you telling me about this?” And the reason why I’m telling you about this is when we think about the computer’s memory, we think in terms of numbers that are hexadecimal. So a lot of times when you see memory drawn out — and I’ll draw memory because memory is so cheap these days I can just draw it on the board. So we have a bunch of cells in the computer’s memory. Each one of these cells is one byte, which you mean — which you know underneath the hood what that really means is that cell contains eight bits. Okay. And so each one of these cells has associated with it some location in memoria where it lives. You can kind of think of this whole thing kind of like the system of houses in the United States, right? How do you know where a house is, right? This little box is a house. How do I know how to get there? It has some address. It has some place that if I know its address, I know how to get to that box and see what’s inside the box.

And in the computer’s memory we refer to the address as using numbers, and they’re just numbered from zero up until however much memory the computer actually has. So if you have two gigs it’s somewhere on the order of two billion, whatever that number is. Except, we’d write those numbers in hexadecimal. So somewhere in the computer you have memory location A000. And that’s just some box. And after it comes A001 and A002 all the way down. And somewhere down in the computer’s memory there’s FFFF, and if your computer is large enough it’s got some more Fs in there too. And it’s in there. Trust me. You’re like, “Oh, I think that’s a little, you know, a little scary now to use the computer.” And up here there was — the very first address is 0000. But when we actually write these addresses, even though they would translate into some decimal value, we refer to them with hexadecimal. And the reason why we do that is we want to distinguish them as being addresses or locations of where things live in memory as opposed to the actual contents of memory, okay?

So somewhere — yeah, at this location there may be stored the value ten in some binary representation. But we want to distinguish that this ten is not the address ten. It's a number ten. It's an integer ten, whatever it may be. Whereas there is some address that we care about that's actually different. Okay. That's why we distinguish between these things. So when you're running your programs where is all this memory coming from, right? And it turns out there's two different sections. There's actually three different sections but two that are mostly relevant to you of where your computer actually says, "Hey, this person just declared an integer," or, "This person just called a method," or "This person just created an object." Where am I gonna get the memory to store all of the information that's associated with their integer or their object or their method called, okay? And so the places where this memory comes from — and I'll show you all three. There is some place where all of your static variables or your constants, you could kinda think of them if they're final variables. There are some special place that's set aside — so this is kinda special. All right. Or if we want to get multicultural it's especial, right? So it's special and basically we have static variables and constants that are just stored in this special location. Where is the special location? We don't really care. Okay. It's just our little lovely special occasion.

Now, beside their static variables and constants the thing — because these — at the beginning of our program it gets started and our — these — all these things come into being, all of our constants. And they never change, which means they don't need to be in some portion of memory that we access regularly. They just need to be somewhere in memory that we know how to refer to them and that's all, okay? So we can have dynamic variables, and what are dynamic variables? Dynamic variables are just any variable that you use new on, all right? So when you say, "new," some object — like you say new G oval, and you pass with some parameters, that's a dynamic variable. And the memory that comes from a dynamic variable when you do this thing called new is something called the heap.

And the way you can think about the heap is it's just like a big pile of clothes. When you need clothes you come along and you say, "Hey, I need some new clothes." And it gives you some clothes. And when you're done you sort of just say, "Hey, I'm not wearing those clothes anymore." And somewhere magically down the line when there's a piece of clothing that you stop wearing, the computer knows that you've stopped wearing it and it says, "Oh, I'm gonna go take that memory back." That's called garbage collection. It's c — wouldn't it be nice if there was, like, little gnomes or elves and, like, when you stopped wearing a piece of clothes and they knew you were done with it because you just had no way of ever getting back to that particular T-shirt again, they just went and threw it in the trash for you or they took it to recycling, which would be the more useful thing to actually do. Then, that would be a form of garbage collection. It's memory from the heap, just gets reclaimed when it's no longer being used.

Well, as long as you're using it — so if you have some G oval object over here, which we'll just call G. As long as G is still being used, this memory is still set aside for you in a place called the heap. And I'll show you how the heap is kinda set up in just a second. Okay. So last but not least, beside static variable and these dynamic variables, there is

another kind of variable that we care about which are local variables. Okay. So local variables come from a place called the stack, to be differentiated from the heap, okay? And the difference between the stack and the heap is that for the stack whenever you have local variables in a method or you also have parameters to a method — like, those parameters, remember, you get copies of the values of the parameters which means they have to live somewhere in memory. Some memory gets set aside for you automatically in this place called the stack. When you have your local variables and your parameters and your method is running those variables are alive. And when those variable go out of scope, like the method actually ends or you get some closing curly brace in which it can cause the variables to actually go away. The variables go out of scope in one sense or another. Then, the stack automatically says, “Hey, I’m gonna take that memory back from you.” Okay? So, again, this memory, at least in the Java world, is automatically managed for you. If you come from sort of the C or C++ world then you’re like, “Oh, how do I free memory?” You don’t worry about it in Java. It’s taken care of for you. It’s a good time, okay?

So what does this actually look like in the machine — in the computer’s memory? Let’s draw that over here. So where is the special memory? Where is the heap, and where is the stack? It’s kinda funky how it’s set up but you got to see it just so you can see what it looks like. Here is the area over here where all of our static or special variables are kept, okay. It’s just some section of memory; usually this is some low area in memory. What that means is the numbers, the addresses here, tend to be smaller. Like, let’s say around 1,000, okay? Which is actually 1,000 in hexadecimal, not in decimal. After the special loc — the special set of stuff we have the heap. And the heap grows. It grows downwards. What does that mean? That means if it starts at 2,000 and you say, “Hey, heap, I need some space for my new G oval.” It says, “Oh, okay. Well, I need to get you some more memory down here. Maybe this is location 2,004. I’ll get some more memory over there. And then you say, “Hey, I need a new G wreck.” It says, “Oh, okay. I’ll get to location 2,008 for your G wreck, and I’ll give you some memory over there.”

So the numbers that was associated with the heap increase, or what we refer to as the heap grows downward. Because you can think of memory — the low values are over here. The low addresses and the high addresses are over here. There is something else called the stack, right, that we just referred to, so let me stop having the heap kinda take over the whole board. The stack starts off in the very high addresses. So here is the stack. It starts off at a place like FFFF, right? Somewhere really — with a really large value for its address. And as it allocates more memory, it grows upward which means the addresses get smaller. And you might look at this diagram and say, “Hey, Maron, do they ever meet?” And they do sometimes. And if they do, it’s real bad news, all right. So if your heap and your stack ever overlap, they start writing over each other and usually what happens is your machine just crashes. But that’s why you got so much memory these days. You’re like, “Oh, I got a gig of memory.” Yeah, because there is a billion data addresses between the heap and the stack, usually not actually that much in a program. They put them a little bit closer together. But you got so much space in here it’s just frightening.

And you could probably use it in your program, right, by having some loop that just allocates a whole bunch of memory. But, really, the thing you want to keep in mind is that these things, at least for the programs that — for most programs that you will ever write, you never have to worry about the heap and the stack actually coming together, okay? So the one other thing that's associated with local variables, parameters and dynamic variables — mostly, actually, with dynamic variables, is how much space gets set aside for something, right? And that's one thing you want to think about is different kinds of variables actually have different amounts of space associated with them. So something like an integer, at least in Java, is four bytes. And this is not something you actually need to worry about because it turns out on different machines they may actually be implemented in slightly different ways.

A character — it uses a special kind of encoding where most characters are actually in two bytes. Sometimes if you get into some very funky languages that have lots and lots of characters or the way the special encoding for characters — the Unicode in coding is actually set up. Sometimes it can actually be as large as four bytes. So as far as you're concerned, you don't really care about the underlying numbers. It's an abstraction to you, and that's the key information, right? Even these little things like an integer or a car is just hiding information. It's hiding the fact that it's a bunch of ones and zeros underneath the hood, and you don't need to worry about it or how big it is. All you need to know is how to use it, okay?

So along with that, with dynamic variables there is also a little bit of overhead memory. And it's just an extra amount of memory besides how much space is taken up by, like, your integers or your cars or whatever variables are inside your object to keep track of other information about the object. And how much information is actually kept track of in that overhead is not important. But it's just important that you actually know that there's some overhead associated with it. So putting all this stuff together, it turns out that we can look at the mechanics of what actually happens underneath the hood when you run your program and you create new objects, right? So if we go to the slides for a second let's actually see, sort of, diagrammatically what's going on when we actually create new objects and declare variables, okay?

So what we're gonna have is some class called the points. And this is a super simple class. All it does is you can create a point with this constructor by saying, "Here's an X and Y location." And all it's gonna do is store an X and Y location for you and let you move them around, all right? It's real simple. It's something you could imagine might be, you know, convenient to have when you're doing graphics. So you pass an X and a Y, it sets its own internal instance variables PX and PY, which I just called PX and PY to remind us that they're our private X and our private Y. They're inside this particular class. And then there's another method called move that you pass some offset in the XY location, and it just changes its private X and Y by whatever offset you give it, okay? Fairly straightforward. Should be, you know, pretty easy to understand given all the stuff you've done with graphics so far.

And then we can say, "Okay. You have this class. You haven't done anything with the class yet, right?" We've just said, "Hey, I have this class." No memory has been allocated. Nothing's actually been done. What you're gonna have is some program somewhere, like my program, that's gonna come along and create new points and call methods on those points. And we want to understand what's actually going on in the computer's memory when this happens, okay? Is everyone happy with the point class? I'm gonna make it disappear in just a second. Can you remember what it does? There is two methods. There is the constructor and the move, and then there is two private variables, PX and PY, okay?

So let's see what happens when your program is actually running, okay? Now, just a little mnemonic. We have the heap and the stack. Heap is gonna grow down, stack is gonna grow up. Just so you can see that as we go along. And the first thing we're gonna do is our method run gets called, right? When your program first starts up the method run gets called. What happens when a method gets called? Well, when a method gets called, we create all the local variables and parameters. Run has no parameters, so we don't need to set aside any space for parameters. It does have two local variables, P1 and P2. So space — notice this is starting at the very bottom of memory like FFFF. There is some overhead for the fact that we create a method, and I just sort of blocked that off and called it overhead just to match, sort of, the same style that the book does it in. And then there's some space, some number of bytes. In this case, I just said four bytes that gets set aside for P1 and some number of bytes that gets set aside for P2.

Right now they don't contain anything because my program is just started. I haven't gotten to this new point stuff yet. All I've done is said, "Run." So it says, "Oh, okay. You have some local variables. I'm gonna set aside space for your local variables. Now what are you gonna do?" I'm gonna say, "Well, I want to create that first point. I'm gonna call new." If I'm calling new, that's gonna be a dynamic variable which means it's gonna come from the heap. So here is what happens. I call this line, and I say P1 equals new point two comma three. So it says, "Hey, you're calling new. You need some new memory. That's dynamic memory. I need to get it for you from the heap." Where does the heap start? The heap starts up at the top and goes down. What data is associated with a point? What values do you need to store to actually keep track of a point? Well, a point inside it has its private X and its private Y. So I have to set aside space for everything that a point encapsulates, which includes all of its private variables and its public variables as well. But all of its instance variables get space allocated for them.

And there is some amount of overhead associated with the fact that we need to keep track of some extra information about this point object, that you don't need to worry about, the computer needs to worry about it. So there is some overhead. Let's say there is four bytes for that. Four bytes for the integer PX and four bytes for the integer PY which is why you can see memories going from 1,000 to 1,004 to 1,008. So each one of the boxes up here is not one byte. I've just kinda stylized it to make it easier. Each box is now representing four bytes because integers are four bytes and the amount of overhead we're keeping we're just gonna say is four bytes, although you don't need to worry about that, okay?

And what did we do when we called the constructor? We set the private X to be equal to the value two that got passed in and the private Y could be equal to the value three that got passed in so that all the constructor did was it wrote into the little places for PX and PY, which means it wrote into the memory locations that the computer set aside for it the values two and three. Okay. Any questions about that? Um hm?

**Student:** Why is there [inaudible]?

**Instructor (Mehran Sahami):** The 1,000 — sorry, is on the — oh, excellent point. Why is there a 1,000 over here? Because the one last thing I didn't actually mention is after we set aside the memory for the new points, what are we gonna do with that point? We're gonna assign it to P1. So here is the critical concept. How does the computer know where P1 and all the information associated with P1 is? Every object lives in an address. The way the computer actually is referring to objects underneath the hood is by the address at which they live, okay?

So you can think of — there's some little house over here that's P1's house and there's, like, the little kids in the house, PX and PY. They're like, "Oh, we're the little children. We store integers." And then I say, "Well, how do I know how to access you, P1?" I need your address, which is 1,000. That's where you live. So when I do this new and I create the memory in off the heap, it says, "Hey, I'm gonna assign that thing over to P1." What's the thing that you're actually assigning? You're assigning the memory address for where P1 lives or its starting address. And from the starting address we can find everything that lives in P1. So it's getting stored here. This value, 1,000, is not an integer. You don't want to think of it like the value 1,000, it's an integer. You want to think of this as the memory address 1,000. Okay? Um hm?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Overhead. So it — that little overhead gives the additional information that you keep track of so — So how do we do the next line? Okay. Very similar kind of concept. We say, "Hey, I'm gonna create a new point. Oh, heap, I need some more memory." And so the heap says, "Okay, well everything up until 1,008 was allocated, so I'm gonna allocate — give you some more space and memory that's enough space to store another point which means I'm gonna give you space for your instance variables, PX and PY, and your overhead."

Notice this PX and PY are distinct from that PX and PY because this is the PX and PY that are the instance variables of P1. The second PX and PY over here are the instance variables of P2. They're distinct things, so they have separate memory to store them, okay? And same thing when we return a value that we're gonna assign to P2, what we're gonna assign is the address that's the beginning of the space that we set aside, the memory that we set aside, for P2. And that's at, you know, 1,000C so 1,000C gets stored there, okay? Any questions about that? It's basically the same thing except you can see the heap is growing down. All this memory has already been allocated, so any new memory we need requires the heap to keep growing downward.

Now, next funky thing, right? We're gonna do a method call. So we're gonna say, P1 dot move. We're gonna do a method call. When we do a method call what ends up happening is we're gonna have some parameters and some local variables which means we're gonna have to allocate some memory on the stack for that method call. So when we start the method call what ends up happening is we say, "Hey, P1, I want you to move yourself by 10 comma 11." So what actually happens in memory? What happens in memory starting at this point upward is we create a new stack frame. Remember when we talked about stack frames? We said when you call a method it's sort of like the previous method or previous place you were running just gets suspended, and we sort of create this new frame on top of it that contains all the information. We drew little boxes for the variables and all that when we did stuff like tracing palindromes and all that happy news. Remember that? If you remember that, nod your head, right? Yeah, what was going on underneath the hood? It wasn't just nice — I mean, at the time, it was nice pictures. This is what was going on underneath the hood.

So memory was getting created on the stack, and that's why we refer to this as a stack frame. It's keeping track of that frame of information for the method call and it does it on the stack, which is why it's a stack frame. What information goes on the stack frame? Well, you get some overhead for the fact that you're calling a method, so there's this — overhead is this first four bytes here. Then, you get all of the parameters that you're passing to the method. And you might say, "But, Maron, the parameters I'm passing to the method are DY and DX. What's this "this" thing all about?" Okay. There is a hidden parameter that you didn't know about until now and now you know about it. When I'm calling a method on a particular object, how do I know which object — once I'm actually getting into the details of things — I actually called the method on? And so when you make a method call on an object, the very first thing after the overhead that gets put onto the stack is, essentially, a pointer to the object, right? It's what we refer to as the this pointer. Remember when we talked about the this pointer, and we said this is basically a way for an object to refer to itself when it's been called? Well, how does it know how to refer to itself? It needs to know where itself lives.

You might say, "That's kind of weird, Maron. Why does an object need to know where it lives itself?" Because when I write some method, these methods are not yet associated with particular objects that I created. Once I create a particular object and call that method on them, this method over here needs to know, "Hey, when I'm doing these modifications on PX and PY, which PX and PY are you referring to?" That's where this comes in. This said, "Hey, I'm gonna tell you the place where the particular object lives whose method is being called." So whatever modifications you make over here, you know to make them to the object that lives at this address, which is why it's called this. Okay? So when you make a method call the first thing you get is the address of the particular object that you're making the method call on, right? We're making a method call on P1 and P1 lives at 1,000. All right. So P1 is just 1,000. So this is the value 1,000, and then I get the parameters in reverse order because the stack is growing upward, right? So I get DX first, then I get DY. And those just get the value 10 and 11 that are passed into them. I get copies of those values, okay? Any questions about the this pointer? Uh huh?

**Student:** I just had a question about graphics variables. And when you add like a G oval to the screen or something —

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

**Student:** — is it a dynamic variable or is it a local variable? It's — if you add it to the screen in a method and then you — it's still on the screen in a different method.

**Instructor (Mehran Sahami):** Yeah, a G oval — G oval is gonna be a dynamic variable because you called new on it, so anything you say new on is a dynamic variable.

**Student:** Then why is point A a local variable?

**Instructor (Mehran Sahami):** Because what's going on here is that P1 is just the address of where the actual stuff lives. All the data associated with that particular object, P1, is on the heap, okay? So P1 is a dynamic variable, right? The place P1 — where I store the address or the place I keep track of its address to know where to look it up is something that is local. So I keep track of P1 by itself as being local. The actual object itself that's created when I do the new is on the heap. Does that make sense? So the address is a local variable. So when I say, "Point P1," all I'm getting is an address. When I do the new, that's when I'm actually saying, "Hey, set aside space for a dynamically created object and let me know where it lives." So, yeah, there's a subtle distinction there but the important thing is that the variable itself, just the address can be a local variable, but the actual object itself is the dynamic variable. Uh huh?

**Student:** Well, so do all the different parameters that you — or all the different messages that you could possibly pass to like say a G oval get stored on the heap or, like, would the set fill be just false by default? Would that actually go on the heap or is there nothing that's put there until you actually send something to it?

**Instructor (Mehran Sahami):** Yeah, so you would need to know all the internals to G oval and G oval does actually keep track of an internal variable for whether or not it's filled, right? So when you create an object, whether or not you set its variables, those variables all need to have some initial value that the object is gonna set itself if you don't set them. So they're all on the heap. Uh huh?

**Student:** So if there is some sort of dec — like an integer declaration like you're saying N — X equals, like, ten or something, do you get, like, something on the stack for, like, where X is and then you get, like, ten in the heap?

**Instructor (Mehran Sahami):** No. So N — that's the funky thing, and we'll get to that in just a second. Ince, doubles, characters and bullions are different than objects, right? So when you use ince doubles and characters, you never say new. So if you don't say new, it's gonna show up only on the stack. Things that you say new on are the things that show up on the heap. So all — either your basic types of variables like cars and bullions and ince and doubles, those — when you declare those you never said, "Hey, I need a

new character." You just said, "Hey, car CH," and you got a character CH. That was all on the stack. The only time you actually created something on the heap was when you said new. Like you said, "Hey, I want some new G recs." Then you actually got one on the heap.

So let's keep following this through and see what happens when we make this move call, right? We got all this information on the stack. We haven't actually done any of the instructions and move yet. So move comes along and says, "Hey, I'm gonna do the first instruction. I'm gonna add whatever is in DX, which happens to be the value ten, over to PX." Which PX do I know it to refer to? It's the PX that I get to by following the this pointer or going to the address of this. So it goes up to 1,000 and looks at the overhead and so it can figure out where PX is starting at location 1,000. And it says, "Hey, it's over here. Its value was two. I add 10 to it, so now it gets the value 12." Okay? And it does the same thing on the next line when it's modifying PY. It says, "I want to modify PY. Which PY am I referring to?" The one that's at location 1,000, so it goes to 1,000 and says, "Hey, where can I find PY from starting at 1,000?" It's over here. And it adds 11 to the value 3 that was there before, and I get 14. Okay? Now this method is done, right? The move method, we executed everything in the move method. It's done.

Here is the funky thing. Remember I told you when a variable that's on the stack goes out of scope, when it's no longer being used, the computer automatically comes along and says, "Hey, that variable or that memory is not getting used anymore. I'm gonna take it back." So all the memory over here on the stack that's associated with the stack frame for the move method, right, which is the parameters to move. The this param. or the this variable, which is actually sort of a hidden parameter to move and the overhead associated with move, that was all the memory that got allocated when I did this move method. This move method is now done, which means all of the memory that's allocated just to execute the move movement automatically gets deallocated by the machine. It's what we refer to as popped off the stack. Because if you think about the stack growing and we get to a certain point and we're like, "Hey, we're done with this top part of the stack." We sort of say, "Boink," and we pop it off the top of the stack and it goes away. So when we pop it off the stack, moment of silence, it goes away, right? And then the program just keeps executing from wherever it left off, right? If there's another method called, guess what? The stack is gonna now grow from this point, okay?

Is there any questions about that? If we were to call the move method again, it would recreate all of the stuff it needed on the stack for another invocation of move because all that stuff is temporary, right? After it does move once it doesn't need it anymore. If you involve move again, it will create it again. That's just the way life is. All right? Now, any questions about that? Now, one thing to keep in mind that's important is that there's a duality here, right? I keep referring to this thing called the pointer. So I will just whip out the pointer, right? Think pointer like pointer. I always feel like a little evil when I actually hit the board with a pointer. But the way to think about the pointer is, right, just think of the value 1,000. Really, this is referring to address 1,000. This is just kinda pointing up to 1,000, right? We don't really care what the value was in here. All we know — it's some memory address up to 1,000.

So what are we going to do? Let's just draw it that way, okay? So rather than having this thing write out 1,000 in here and remember, oh, yeah, that 1,000 is supposed to be a memory address, et cetera, we can just draw it as a pointer and say, "Yeah, this thing is really some memory address, and the memory address is just pointing over here." Yeah, this 1,000 over here lets us know that it's 1,000, but graphically it's a fun way to keep track — and you're like, for some definition of fun — it's a fun way to keep track of it because then you know that you're not gonna muck with this variable in here as though it were an integer, right? This thing is just some — referring to some piece of memory where you're actually storing some object. And similarly, so is P2, okay? So that's what we like to think of as the pointer view points of the world, right? It's just the point that things that are memory addresses are really pointing somewhere. They're telling the machine basically, like, "Look, I know where you live."

Let's do another one real quickly, okay? So another memory allocation example. Here's, again, our points, right? Same point that you just saw, except I've left off the move method. And now I'm gonna create something called the line. And the line has two points associated with it. So when I create a line, I pass at two points to its constructor. Like, two things of this object. And it has a — a line has a beginning and an end which are private points that it keeps track of, okay? So I say, "Hey, beginning is gonna be the first point you give me. End is gonna be the second point you give me because you could have told me a long time ago that a line is defined by two points." So if I know those two points, then I know what the line is, okay? So what's going on if we actually think about this in the computer's memory? If we want to try to create two points and then create a new line. Okay? So everyone's sort of clear on points and lines? They're just two basic classes just to kinda illustrate the point. Ha, ha, yeah. Not funny. So when we actually create the two new points we get exactly the same sort of diagram we had before, right? We have memory allocated on the heap at location 1,000 for point one and 1,000C for point two.

The other thing we have is in this version of the run program, we have a local variable called line, which I've just abbreviated LN. And so we've gotten some memory allocated for it over here. Notice it doesn't have value yet. It doesn't have a value yet because we haven't asked the heap yet for the new memory that actually stores all the information for that line. All we've done is say, "Hey, you know what? I'm gonna have some local variable called line in here." When you get to the new line over here, then you're actually gonna ask the heap for memory, and things are gonna get set up. So let's see what happens when that line gets called, okay? We're now gonna go into new and excruciating detail. When we say new before we do any kind of assignment over here — so this still remains blank. When we say new what it does is it says, "Hey, I'm gonna go over to the heap, and I'm gonna allocate space for a new line." What are the variables that are part of a line? There's two variables, a beginning and an end, okay? And they're both points. I haven't initialized them yet. I'm gonna actually got into the details of the line constructor in just a second, but it sets aside space for you for its variables which are beginning and end.

Now, we call the constructor method — so, actually, I didn't show you this before when we did it for points, but what was actually happening when we called the constructor, is the constructor is just a method just like any other method. So what happens with this method? Except the constructor doesn't have a this pointer associated with it because it hasn't created the object yet. It's still in the process of creating the object. What it does have, though, is it has the parameters that get passed into it. Well, what are the parameters that get passed in there? P1 and P2. What are P1 and P2? They're pointers. They're just values that are — happen to be memory addresses. So what I do is I pass a copy of the address, 1,000, and a copy of the address 1,000 and C, and I don't create new versions of P1 and P2. All I do is I pass copies of their addresses, okay? Are we clear on that? That's the critical concept here.

So what happens after that? This guy comes along and says, "Hey, sa — beginning to be P1." Well, what's P1 that got passed into me? It's 1,000, so it just sets beginning to be 1,000. And then the next line comes across and says, "Hey, set end to be P2." Well, what's the P2 that got passed into me? It's 1,000 and C, so it just sends end — sets end to be P2. And then it says, "Hey, I'm done. I did my work. I'm the constructor. I did the two lines you told me. So it's done." And all the memory that's allocated with this particular method call, which are the two parameters in the overhead that's associated with calling this method, all get popped off the stack. So they get popped off the stack and they're gone. And now we say, "Hey, thanks, line constructor. You just created a, you know, first of all, thanks, new. You allocated the memory for us. Thank you, line constructor. You went ahead and initialized all the fields that I needed for a line. Thanks a lot. What do I do with that now?" And it says, "Yeah, what I need you to do is assign that object where that object lives to the variable line."

And so the variable line, which is over here on the stack, is a local variable, finally gets assigned the address of the line object on the heap, which is at 1,018. So when the whole line executes, basically what it's done is it's called the con — set aside memory with new, called the constructor, had the constructor fill in the initial values. And then what actually gets assigned to the line over here is the beginning memory address for this object that was allocated on the heap. Okay? Any questions about that? Now here is the funky thing. Let's take a look at this in the pointer viewpoint of the world, right? So the first thing we could do is say, "Hey, let's look at the pointers over here, right? These are all pointers. This points over to 1,000. This points over to 1,000 and C. This points over to 1,018. So we could just draw that looking like this.

But we're not done with all the pointers, right? There's still more pointers left. Guess what? Beginning and end, they're just pointers back into the heap, right? Beginning, it just says, "Hey, the point that you gave to me is just a point that starts at memory location 1,000 and end is just location memory of memory 1,000 and C." So really these guys are also just pointers back to the respective locations to those particular objects, okay? Any questions about that? I know that looks kinda funky and the reason why I did it on slides as opposed to on the board is because I want to post the slides so you can refer to them back after class if there's any questions. But that's the funky thing. You can actually have things on the heap that refer to other things on the heap. Okay? Uh huh?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Well, and d — yeah. It's — I mean, you need to convert it to decimal, right, if it's a hexadecimal number. You don't care what the actual value is, all you care about is that it's an address. It's not, you know, a number that you would manipulate like an integer.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** We can talk about that offline. I mean, it's easier, as far as the book's concerned, to be able to do differentiation, to differentiate between addresses and non-addresses, but there is also some history to it in computer science.

So I just want to show you one thing very quickly, and then I'll give you a quick video. So if we have point, P1 equals new point one comma one and then we have some point, P2, that equals P1. What's really going on the in computer's memory, right, is P1 is just some address somewhere. And this is on the stack. And P2 is also just some location that's also gonna be an address because it's just an object. Now, this is the stack. Here is the heap over here. When I create P1, I sort of create some space over here that's gonna store a one and a one and there's some overhead down here that I don't care about. And this guy's gonna point over to here. Let's say this is a memory location ABCDF. Okay? It's just some memory location. When P2 comes along and I set P2 equal to P1, it just gets the same value as P1. It gets ABCDF. Which means this guy is pointing to the same location, okay?

So if that's the case, then if I do something like say P2 dot move ten comma ten, what happens there? It says, "Hey, I'm gonna go and move based on P2." Well, where is P2 pointing? It's pointing over here. It's pointing the same place P1 is and says, "Oh, I'm gonna move it by 10 10 so I get 11 and 11. And you might look at that and you say, "But, Maron, you just changed P1 without ever referring to P1." Yeah. If you have two objects that are pointing to the same place because you set them equal to each other, what you've done is set their pointers not created copies of objects, okay? That's why — it's sort of like you're old enough to know what's going on now, right? Before we just said, "Hey, every time you just create a new one and that way they'll all be distinct." If you don't create a new one and you set some local variable to be equal directly without doing a new — notice I didn't do another new point here. All you're doing is setting up pointers to be equal to each other. Which is why in the days of yore when we talked about our friend the string — remember the string, and we said if you had some string S and this was equal to hello and you might want to say — and this was S1 and I had some other string over here, S2, and this was also equal to hello. And I might want to say if S1 is equal to S2 and I told you, "Yeah, this doesn't work because you need to call S1 dot equals S2." The reason why it didn't work is because when you check for equality here, it's not checking to see if these strings have the same characters. These strings are objects. They have memory associated with them. When you check to see if the actual objects are equal to each other, it looking to see, "Are they referring to the same actual thing underneath

the hood?" These are not. These have two versions of hello in different places in memory. One is over here and one's over here.

So S1's a pointer or S2's a pointer here. S1's a point here. These guys are not equal to each other. They would only be equal to each other if they were, in fact, the same object and I set S2 equals S1, okay? Is there any questions about that? Alrighty. So if you want to go you can go now. And if you don't, I'll show you a little video. It's just a little fun claymation.

[Video Plays]

Male Speaker: Hey, Binky. Wake up. It's time for pointer fun.

Binky: What's that? Learn about pointers? Oh, goody.

Male Speaker: Well, to get started, I guess we're gonna need a couple pointers.

Binky: Okay. This code allocates two pointers, which can point to integers.

Male Speaker: Okay. Well, I see the two pointers, but they don't seem to be pointing to anything.

Binky: That's right. Initially pointers don't point to anything. The things they point to are called pointees and setting them up is a separate step.

Male Speaker: Oh, right. I knew that. The pointees are separate. So how do you allocate a pointee?

Binky: Okay. Well, this code allocates a new integer pointee, and this part sets X to point to it.

Male Speaker: Hey, that looks better. So make it do something.

Binky: Okay. I'll deference the pointer X to store the number 42 into its pointee. For this trick I'll need my magic wand of dereferencing.

Male Speaker: Your magic wand of dereferencing? That's great.

Binky: This is what the code looks like. I'll just set up the number and —

Male Speaker: Hey, look, there it goes. So doing a dereference on X follows the arrow to access its pointee, in this case to store 42 in there. Hey, try using it to store the number 13 through the other pointer, Y.

Binky: Okay. I'll just go over here to Y and get the number 13 set up and then take the wand of dereferencing and just —

Male Speaker: Oh, hey, that didn't work. Say, Binky, I don't think dereferencing Y is a good idea because, you know, setting up the pointee is a separate step, and I don't think we ever did it.

Binky: Good point.

Male Speaker: Yeah, we allocated the pointer Y, but we never set it to point to a pointee.

Binky: Very observant.

Male Speaker: Hey, you're looking good there, Binky. Can you fix it so that Y points to the same pointee as X?

Binky: Sure. I'll use my magic wand of pointer assignment.

Male Speaker: Is that gonna be a problem like before?

Binky: No. This doesn't touch the pointees. It just changes one pointer to point to the same thing as another.

Male Speaker: Oh, I see. Now Y points to the same place as X. So wait, now Y is fixed. It has a pointee. So you can try the wand of dereferencing again to send the 13 over.

Binky: Okay. Here it goes.

Male Speaker: Hey, look at that. Now dereferencing works on Y, and because the pointers are sharing that one pointee, they both see the 13.

Binky: Yeah, sharing, whatever. So are we gonna switch places now?

Male Speaker: Oh, look, we're out of time.

Binky: But —

Male Speaker: Just remember the three pointer rules. 1.) The basic structure is that you have a pointer, and it points over to a pointee. But the pointer and pointee are separate. And the common error is to set up a pointer but to forget to give it a pointee. 2.) Pointer dereferencing starts at the pointer and follows its arrow over to access its pointee. As we all know, this only works if there is a pointee which kinda gets back to rule number one. 3.) Pointer assignment takes one pointer and changes it to point to the same pointee as another pointer. So after the assignment the two pointers will point to the same pointee. Sometimes that's called sharing. And that's all there is to it really. Bye-bye now.

[End of Audio]

Duration: 55 minutes

## 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. Excellent.

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 Mona 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 our 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 think 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 do 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 one 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 an 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

## Programming Methodology-Lecture16

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

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

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

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

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

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

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

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

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

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

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

So how do we actually create one of these things?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Ooh, a lot of chalk dust.

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

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

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

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

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

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

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

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

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

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

Uh huh?

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

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

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

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

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

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

Uh huh?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Let me write that a little more clearly.

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

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

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

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

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

**Student:**Zero.

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

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

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

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

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

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

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

**Student:**Yeah.

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

**Student:**[Inaudible].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Okay, any questions about that? Uh huh?

**Student:**[Inaudible].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Uh huh?

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

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

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

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

Uh huh?

**Student:**Can arrays be ivars?

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

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

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

Uh huh?

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Okay, and questions about that? Uh huh?

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

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

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

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

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

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

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

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

**Student:**[Inaudible].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[End of Audio]

Duration: 52 minutes

## 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 looks like 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  $I \text{ sub } I, \text{ sub } J = 1$ . 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 get 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.add(5)` and give it X directly where here I might have `int X = 5`, and you would say, "But Miron, you told me this ArrayList can only store integer objects and you're passing an int. You told me that doesn't work." Well, this is what Java 5.0 is doing for you automatically. It says, "Hey, this thing is 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.get(0)`, 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 wrapper 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 mouse 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 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

## Programming Methodology-Lecture18

**Instructor (Mehran Sahami):** So welcome back to yet another fun filled exciting day of CS106A. I'm feeling a little bit more relaxed. Are you feeling a little bit more relaxed? A few people are nodding. A few people are like, "Yeah, Ron, we took the midterm in your class, but we have like six other classes to deal with. You can feel relaxed, but we don't." But don't worry, things in this class are just – it's all downhill after this. It's just a good time.

So a few quick announcements before we jump into things. One of which is that there's four handouts, and if you didn't get the handouts – if they weren't there when you came in, they're all in the back now. You can pick them up after class. So you don't need to pick them up right now. You can pick them up after class.

The midterms will also be back after class. I'll show the performance of the whole class. I was very pleased. It was just – it was a heartwarming kind of thing to just see so many people doing so well. But you'll get them back after class. So you'll get them today.

You – one of the handouts also solutions for the midterm, and so you can – if you got anything wrong you can compare your answers with the solutions. One thing to also keep in mind is that the actual solutions we were looking for are shorter than the solutions that I give you. The solutions that I give you have comments. For one of the problems I actually gave you two different ways of doing it just so you see different approaches. But you weren't expected to actually write that much code. All we wanted was sort of the code without the comments, which is actually pretty slim if you consider how much code there is there without comments. But you'll get those back after class.

And then as you hopefully know, assignment number four is due today, and assignment number five, you're next handout – one of your handouts is going out. For assignment number five you will do yet another game because it's just so much fun to do games. And this time we're gonna be testing your abilities with – we're not testing your abilities. That's an awful way of saying it. We're going to be watching your abilities in the use of arrays flower in terms of designing another game. Okay.

So with that said, it's time to see how much pain we actually caused on you assignment number four. Hopefully not too much pain, although, I did hear there was quite a few people on [inaudible] on the wee hours of the night last night.

So anyone in the zero to two hour category? Was that actually someone back there? No. Just joking – yeah, kind of funny. Two to four hours? All right. A few folks. Good to see. Four to six hours? Oh, nice. Six to eight? Pretty large contingent there in the old six to eight category. Eight to ten? Wow, also pretty large contingent in the eight to ten category. Ten to 12? All right. We begin to see the drop off at ten to 12. Twelve to 14? Few folks at 12 to 14. Fourteen to 16? Sixteen to 18? Eighteen plus? Taking a late day? That's – yeah. Alrighty.

Once again – it happens every time. I don't make this up, right? You saw people raising their hand. The world is just – it's amazingly normal. It really is. There are two things the world is – it's linear, and it's normal, not necessarily at the same time. So again, ten hours, and less life is good. Even if you're in the ten to 12, you're still in pretty good shape. I didn't see actually too many people up in this range, which is good to see.

If you're in the 12 to 14, it's still not very bad, right? It's just some – there might have a bug or some particular thing that happened that caused a little extra time. If there was some conceptual thing that did not get cleared up as a result of actually doing the assignment, that's the whole point, right? Some times you might run into a roadblock, but the point is once you get through that roadblock, that concept is cleared up for you. That's the real point of doing assignments as opposed to just reading the book.

Reading the book's easy, and a matter of fact, it's surprisingly easy to read code, and think that you could write it, right? That's what most people do. They look at code, and they're like, "Oh, I could've written that." Right? It's like Jackson Pollock painting. You look at that, and you're like, "I could've painted that." Right? But there's a difference, right? One, he painted it first, and that's why it's worth like \$20 million, and two, trying to actually do the same thing later on requires a different creativity on your part than just looking at what someone else did. So it's important to not just read code, and understand it, but to be able to write code, and understand it, which is the whole point of the assignments.

Alrighty.

So with that said, time for some midterms statistics. So here is the stats on the midterm, and that's just a beautiful thing to see. Right? This little hump over here – I should've just tacked on to this, right? Then we would've actually had like a little inverse Poisson distribution that you would've said, "But, Marron, it's not normal." And in fact, you can approximate a Poisson to the normal, so it is.

But the important thing is that in fact the class did very well. There were six perfect scores on the exam, which is why I broke out the 90's as a separate category, which it just warms the cockles of my heart. What the cockles are, I'm not sure, but it warms them. And a lot of people who did very well, right? A lot of people in the 70 or higher category. Nice to see a very large contingent in the class in the 80 plus category.

And so the stats – this is out of 90, right. So this in not percentages, but this is out of 90 points. The mean was a 73.2. Median was a 77, which percentage wise translates into almost 85 percent. And then the standard deviation was 13.6 points. So if you're kind of over in this range over here, good times, happy days, life is good.

If you're kind of in this range over here, you want to make sure conceptually things get clarified. You want to look at your exam, and figure out whatever problems there were, look at the solutions, get them clarified, get the issues clarified in your mind. If they're not clear, talk to your section leader this week, talk to me, talk to the TA to get any concepts cleared up.

If you're kind of in this range or lower, it's – to be honest, it's time to worry a little bit, and you need to really – think of the midterm not as just, "Oh, my God, my grade is ruined," because the point of having midterms, at least as far as I'm concerned, is not just grades. It's great if everyone does well, and we just give more A's, and life is good. But really it's a diagnostic for you. It's a chance for you to get a realistic calibration of how you're doing relative to the rest of the class, relative to what we expect you to know.

And so if you find yourself from that calibration falling into a range that worries you a little bit, that you shouldn't think of as, "Oh, it's just time to pull your hair out, and panic." All that really is is an indication that you need to come clarify some concepts. And maybe they're clear for you, and something bad just happened during the exam, like an earthquake. Or maybe it's some conceptual issue that needs to be cleared up, and this is a chance while we're still well before the final exam, and well before several other assignments in the class to get that clarified for you. So think of it as a diagnostic. Okay.

So any questions about anything related to the midterm before we sort of push off into our next great topic in computer science? But I was just very pleased. It just made me happy. It made me think like you had the easy button with you, and you just saw the exam, and you were like, [Easy Button] That was easy. Yeah.

It's just like – I got this thing – well, I got – I'll tell you the story. I bought this thing like a year ago, and I was like, "Oh, this will be great for class." And then it sat on a counter for a year. So now I'm happy that I can actually use it. And it is not particularly branded because I cut that off just in case someone wants to sue. All right.

So with that said it's time to actually get into our next great topic in computer science. So I'll just leave this up here if you want to write down these numbers for whatever reason. But to wrap up a little bit about one of our last great topics, which was arrays, and specifically multi-dimensional arrays. And I'll just call them multi-dim arrays. And you've seen a couple examples of these right now, but I want to – already, right, in the last couple classes you saw some examples of multidimensional arrays. But I want to show you yet another example to show in a very nuts, and bolts kind of way how funky things get when you have arrays of arrays, and you pass them around as parameters in a program – just so you see yet another example to drive the point home.

So let's say I was gonna write a program to, oh, keep track of midterm, and final scores, for example. And let's say the class was smaller than this. So I had at most 100 students because I was teaching some class like basket weaving or something, and it's just small. It's cuddly. We all hold hands. So I have some two-dimensional array that I want to have because I want to have midterm scores, and final scores, but I want to keep them separate. So I'm gonna have one array of midterm scores, and another array of final exam scores.

But in some sense what I really have is a two-dimensional array that's two by the number of scores that I have as a maximum. So let's say 100 is the maximum number of scores I'm gonna have. So I'm gonna a two by 100 array, and I'll call this scores, and I'm just

gonna create this by basically saying new int. And you've seen the syntax before. Two – and this is 100. And these would probably be constants that I would have in a program, but just to save space, and save time, I'm not – you actually declare public static final int., and saw max size 100. So I'm just gonna put 100 in there.

And as we talked about a little bit before, some particular entry of this grid, right – because this is just a two-dimensional grid – like score zero comma zero is just – this is just a single int. basically. But then we talked about this funky thing where we said, "Hey, you know what? A multi-dimensional array is actually an array of arrays." So a two-dimensional array's an array of arrays. A three-dimensional array is an array of arrays of arrays, right?

So we could actually think about, "Well, what happens if I only specify one of the dimensions of the array?" So what do I get if I say, for example, scores sub zero, right? That's not one entry, so it's not an integer. That's in fact a whole array, right? The way to think about this is I have two arrays that have size 100. All right. So we have a little break in there. And so score sub zero is this array, and score sub one is this array, and scores is kind of this array of arrays, right? It's really an array that has two elements. So this is element No. 1 over here in the heavy chalk line of the array or element No. 0 – it's the first element in here is element number one. Okay.

So this thing – the type of this thing is really an integer array. And that's the funky thing to kind of think about. Okay. So score sub zero behaves in your program just like if you had declared a one-dimensional array, which means you can pass score sub zero to a function that expects a one-dimensional array, and that's perfectly fine. What it's doing is passing this element of the scores array of arrays, which happens to be a whole array of integers in its self. Okay. So any questions conceptually about that? All right.

So why don't we look at a little code that actually drives that point home. All right. So we'll put this away. Bye-bye midterm scores. Don't save. And we'll come to our friend the Eclipse, and we'll say, "Hey, Eclipse –" oh, we'll get to Roulette in just a moment. Don't worry. It's all about gambling in this class. All right. So it's not really gambling. All right.

So I'll tell you we have a program that wants to keep track of test scores. So here what I'm gonna do is I'm gonna have a program test scores, again, we're just gonna set the font to be large. It's not a big deal. And I'm gonna ask the user for the number of scores. So I'm gonna ask the user for number of scores, and whatever the user gives me is num. scores. I'm going to create my scores array to be a new array of integers that's two by however many scores you have.

And you should see this right now, and you should go, "But, Marron, you're missing the declaration of scores, right? This creates a new array of arrays, but where's the declaration of scores?" Ah, instance variable. So I want to show you an example of where we have an instance variable that's an array of arrays. So here I declare the instance variable. This just tells me that score's is going to be a two-dimensional array.

It is not yet created the two-dimensional array. It is just set – I'm gonna have this thing called scores that gonna be a two-dimensional array, I still need to create it. So when my program runs up here, I ask the user for the number of scores, and I actually create the two-dimensional array. I don't need to declare scores because scores was declared as an instance variable or an ivar, so it's already declared. I need to create all the space for the all the elements of that array. Any questions about that? Hopefully that's clear. All right.

So first thing I'm gonna do is I'm gonna initialize my scores, and here – and its scores. All this is gonna do is have two for loops that are gonna go through, and set all the scores in the grid to be zero to begin with. Now, I told you before, when you declare an array – when you create an array, the values of the array get initialized to be the default value for that type. And the default value for integers is zero. So you might say, "But, Marron, they're already zeros anyway, so why are you doing this?" I'm just doing this to kind of exemplify the point. I could put in a one here, and just say, "Hey, everyone starts with a one on the exam." Right? But we'll just say it's zero.

How do I actually think about this two-dimensional array? Well, first of all, I want to think about the rows of the array, and then for every row it's got a certain number of entries or essentially the number of columns in the grid. So how many rows are there in the array? If I look at the two-dimensional grid scores, and I say scores dot length, and scores is an array of arrays, it tells me what is the size of the first dimension, right? So if scores is a two by 100 array, what's the size of the first dimension? It's two.

So scores dot length – the way you want to think about it is this is scores. How big is scores? Scores actually is of size two. Each of the elements of scores of size 100, but if each of the individual entries of scores – score sub zero, and score sub one – is an array. So there's only two arrays inside scores. So scores dot length, back over here in my program, will in this case have the value too. Okay. So that's gonna be my outer loop. My outer loop's gonna say, "I want you to loop over however many rows you have in your grid, " is the way to think about it.

For every row, how many entries do you have in that row? What I'm gonna do is I'm gonna say, "Hey, all the rows have the same size, so I'm just gonna arbitrarily pick the first row because I know there's always gonna be one row. If I knew there was always going to be ten rows, I could say, "Score sub nine." But for programmer, they're just like, "Why are you doing score sub nine?" So the general idiom – the way we using do it, is just score sub zero because you always know that there is a first – always a first entry in an array – or in an array of arrays.

So score sub zero, what is that? Score sub zero I just told you is a whole array of integers, right? It's this thing right here. So its length is 100 because there's 100 integers in it. So in this for loop we loop through 100 elements, and so all we're gonna do inside here is we're gonna set scores I sub J. I's gonna vary from zero to one because it's actually gonna count from one to two, but not include two. And J's gonna vary from zero to 99. And so it's – we're gonna set all the elements of the grid to be zero. Any questions about that?

The syntax just kind of looks funky because here we have scores without a sub script on it. Here we have scores with a subscript on it. In both cases we're taking the length. And the important thing to think of conceptually is it's an array of arrays. So when we just take the first length, we're saying, "Hey, how many sub arrays do you contain?" It says, "Hey, I contain two arrays. I'm an array of arrays." And when we take the length of the first sub array, we get the actual number of columns for a grid. Okay. And you can generalize this to three, and four dimensional arrays, but for this class really all you're gonna need to worry about is two-dimensional arrays at most. Okay.

So that's gonna initialize all the grid to be zero. So that's what an int. scores is gonna do. And then we're gonna write out score sub zero before we do any incrementing – action increment all the scores inside because it's just good to give everyone points.

But before the increment, we're gonna print out the scores array at score sub zero. So if we look at print list – right – all print list does is – this is a really simple thing. It just gets passed an array of integers. It goes through all the elements of that array, right? So we just are calling the array list in this case, so it's just counting up from zero up to the length of the list, and just writing out all the individual elements, one on each line. So it's a trivial thing to write. You've seen this a whole bunch of times with arrays so far.

But what are we passing to this guy, right? This guy's expecting a one-dimensional array. What we're passing is score sub zero here, which means what we're passing to it is just this single entry from the scores grid. We're passing a sub array of the scores grid. We're passing the very first array that's at score sub zero, which means score sub zero here is a whole array of integers, and that's the type this thing's expecting, so everything's perfectly fine. It may seem weird that we're sort of taking this grid, and dissecting it into its individual rows, but the important concept – and I'll say it once again – each one of those individual rows is just an array in itself. Okay.

So last thing I want to do besides just printing out the scores, I want to go through, and increment the score list. Now, here's something that's funky. An increment score list – I'm gonna pass in one of the sub arrays. I'm gonna pass in score sub zero.

What is increment, and score list do? It's expecting an array of integers. It just has a loop through that list, and it's gonna go through, and just add one to every entry in that list, right? Very standard thing you would think of doing maybe with some array of numbers. You just want to go, and add one to it. And that's all this method is doing. The only thing that's funky about it is when we call that method, we're passing it a portion of the scores grid, which is just the first array in that scores grid. Okay.

So any questions about any of this? If this is all making sense nod your head. Excellent. If it's not making sense shake your head. Feel no qualms to shake your head. Good, good times.

So let's run this. We'll be happy. We'll just make sure it's working. I'll give it a small number of scores because it'll be like a tiny wee class. How many – I should just ask.

How many people are in the smallest class that you happen to be in not counting the instructor? Anyone lower than five? One. How many people are in that class?

**Student:** Four.

**Instructor (Mehran Sahami):** Four. That's less than five. And as a matter of fact, that's all you need for candy because one is – one here – and that's amazing, right? It's just a class of four people. That's a good time. Sometimes – all right.

So we'll say the number of scores is four just to emulate your class, and everyone started with a zero, and then they all got a one, which in a class of four people probably means 100 percent. All right. So that's all it's doing, right? And all of this is in terms of score sub zero. Score sub one has the value zero. It still maintains the value zero. We never touched it when we were doing the incrementing. So any questions about that? No. No. No. All right.

So besides just doing life in this array way, you can also do things in the array list way. Okay. So we talked about arrays, and array lists. So here I'm just gonna have a single array of – array list of scores. And what I'm gonna do here, just so you can see the syntax, is again, I'm gonna have a private instance variable that's my array list scores that's an array list of integers. I only have one, and I'm not asking the user how big it needs to be, right, because array lists are dynamically sized. It doesn't matter. As the user keeps giving me scores, it's just gonna get bigger, and bigger. So I don't need to have the size a priori.

So in my instance variable, I actually not only declare scores, but I actually create the array list. So I create a new array list. It's empty to begin with. I can just add stuff to it. Not a whole lot of excitement going on there.

What do I do? I'm still gonna ask the user for the number of scores, but here when I'm initializing my scores, I don't need to worry about creating an array of a particular size. Here I just say, "Hey, you gave me some number of scores. You want me to initialize this array with how ever many scores you gave me." So I'm just gonna go, and do a for loop, and add that many elements to the array list because every time I add, the array list just automatically grows in size to be able to store all the scores. And I'm gonna store zeros.

Well, as you know a priori, right, zero is just an int. It is not an integer. So in Java 5.0, and later because life is happy for us now, we get this automatic boxing, which means the computer sees the zero, and goes, "Oh, that zero's an int." Your array list stores integers, right? That's the parameterized type of your array list. So I'm automatically gonna box up your little zero like you're at the integer restaurant, right, and it's like, "Oh, I got this O." And it's like, "Yeah. Yeah. I can't eat the O, let me box it up for you as this thing called an integer, and then you can have it." And then it says, "Oh, I can add an integer to scores." And it just takes care of that for you. Okay.

So when it scores it just goes through, and adds that many zeros to the list, and then we can write out the scores before we increment, and we'll print list our scores. And all print

list does is – you've actually saw this exact same code last time. It has an array list. We don't need to specify the parameterized type here. This will just work on any kind of array list. It goes through up to the size of the array, and it gets each of the individual elements, and prints them out. Okay.

So all we're gonna do now – the interesting part is to increment the score list. Here again, we're gonna go through scores up to the size of scores, but the syntax looks different. The syntax is actually a little bulkier than our array syntax, so we could just say, "Scores sub, you know, I plus>equals one." Right? To add one or plus plus if we just wanted to add one.

Here the funky thing if we want to set a score is inside the scores list we have to use the set method. There's two parameters we sent to the set method. The first is the index. So we're gonna say, "Set at location I." So whatever the old value was, we don't care. We're just gonna slam something new over it, but set something at val – in place I.

What are we gonna place at location I? We're going to get whatever was previously at location I, which is an integer, and we're gonna add one to it, which means if I have this thing that's an integer, and I want to add one to it, I take that integer, I un-box it to become an int. This happens for you automatically. So I get the int., which is zero, which is what I put in there before. I add one to it. So now I have the int. one, and now I want to store it back into something that's stores integers, so it automatically gets boxed up for you as this integer, and gets stored back.

So the syntax looks a little bulky, and that's really the point of showing this to you. It – sometimes using array lists can actually be a bulky thing to do. But if we go ahead, and run this – do te do te do – we're running. We're running. Life is good. It chugs along. I got to get a faster Mac. Alrighty.

Test scores with array lists – and it's gonna ask me for the number of scores. We'll say three this time because someone in your class dropped. And we had three zeros that got entered to begin with, and then we went through, and for each one of those zeros we sort of took the zero, added one to it, and stuffed it back in. So we got three ones. Not a whole lot of excitement going on there, but you see the difference.

So when you see these two things one natural question that kind of comes up in your head is, "Okay, Marron, I sort of see two ways to do this. Which one should I use? Should I use an array list or should I use an array? And as a matter of fact you just told me on this next assignment I'm gonna be working my array muscles. So what's a good idea to think about?" Okay.

And here are the pros, and cons. So if we think of array lists – and I'll just give you the pros, and cons relative to array lists versus arrays. The pros of an array list, right, are that at – the biggest pro is that it has this dynamic resizing. So if I need to add more elements – if I don't know the size a priori, this is a great way to go because it automatically resizes. Okay. And the other pro that it has is there is a bunch of high-level operations

that automatically supports things like contains, right? So you can just see if an array list contains a particular element or not. So it has other operations – I'll just say ops. that are provided for you. Right? Which is a nice thing to have – so you don't need to worry about going, and finding some element in array.

It does have some cons though, and the cons are that it's less efficient in terms of how the computer actually deals with an array list versus an array than an array. So it's less efficient than an array. The other con that it has is its syntax can get kind of bulky. Syntax is bulky because if a lot – one thing that I really want to do often is if I want to have some array, I want to make some modifications to values in that array. Well, in an array I just say, "Hey, you know, I have score sub three, and I stick in some value, and life is good." Or I say, "Score sub three plus>equals five, and I add five to it, and life is good."

Here the syntax is bulky because I need to say, "Get the old value, do the modification to it, and then set it as the new value." And so I'm really doing two method calls every time I want to get, and set a meth – a value, and that's just bulky. And bulk means it becomes more error prone.

The other thing is in the pre-Java 5.0 world. So we'll just say pre-5.0, which would you sad, and weep, and any time any – or if you run into any friends that are using a version of Java earlier than 5.0, you should just pat them on the shoulder, and take a moment of silence with them, and then like – I wouldn't say slap them around, but I would say, "Encourage them strongly to get a newer version of Java." Pre-5.0 – this was horrendously bulky to use because all of that boxing/un-boxing stuff didn't happen automatically, and you had to worry about, "Oh, I need to box, and I need to un-box." It was a bad time.

The main difference – and you might say, "Yeah, well, you know, I don't care about efficiency, right, Marron? I got a whole bunch of memory, and my computer's fast. And yeah, bulky syntax, I can deal with that because I cut, and paste anyway. And I'm using Java 5.0 or later, so I don't really care about any of this, man. I'm just – I'm going the easy route."

And the real big difference – the thing to keep in mind for one versus the other is if you know the size of your array beforehand – if you have a fixed size, almost always go with a regular array, not an array list. Keep that in your mind when you're thinking about the program that you're going to be writing. Fixed size – there's a lot of things for which you know they will have a fixed size or some maximum size. Hum. Let's take a moment to think about that. Fixed size – your next program – array. Good? All right. So just in case it wasn't clear. All right.

So now, with that said, it's time to actually move onto the main topic for today. So any questions about array versus array list? Hopefully that should be clarified. All right. And you should just be thinking right now, "Oh, Marron, [Easy Button] That was easy." Can you even hear that? [Easy Button] "That was easy." Yeah. That's what I'm talking about.

That was easy broadcast in stereo. All right – which means it's easy from two different sides.

So now, the thing to think about is our friend debugging. All right. Which is something you've done this whole time, and you're like, "But, Marron, I've been doing debugging this whole time. How can we have a whole lecture on debugging?" Well, because it's time to think a little bit about that zen, and the art of debugging, and actually a bunch of practical tools I'm gonna show you to do debugging. Okay.

The first thing to think about debugging is when you are a computer programmer or software engineer or in the larger sense computer scientist, there are different roles you play when you're building software. There's a portion of your time which is spent on designing software, and here you're really like an architect. You're kind of looking at big pieces, and how they fit together, and drawing schematics, and figuring out what your sub pieces are, and doing decomposition, and the whole deal.

Then you go into coding. And when you're a coding – when you're coding, you're an engineer. Woo-woo. You got you're little hat on. You're in your train. No. You're writing code, and you're thinking about good software engineering, but you're building something, and you're driving a process forward.

Then there's this thing called testing, which you do to your code. And a lot of times that means it's 4:00 a.m., you think your code works, the project is due tomorrow, and you decide, "Hey, I'm gonna do some testing." And this is the role that does not get taken seriously because it's 4:00 a.m., and your project is due. Often times testing means, "Hey, I ran it once – maybe twice, and I felt good, and I stopped. And I tried the simplest possible case I could try."

That's not the role you want to play when you're testing. When you're testing, you're the vandal. And that – like you got the spray can in your hand, and you're like, "Yeah, we're going out." You are thinking about, "What do I need to do to break this program?" You're not thinking about, "What's the least I can do to please make the program run once, and get through, and hopefully there's no bugs, and if it makes it through once, I'll just believe it's correct, and that's a good time, and I'll turn it in?"

You're just banging on the thing until you break it. As a matter of fact, a lot of software companies – most good software companies have people who this is their whole job, right? They don't do this. They get code from other people, and just try to break it. And this is a skill. You actually – you see some amazing things that people do to break code, but they do it, right? And you got to appreciate that because when you're writing your paper on your word processor at 4:00 in the morning, and it crashes, this person wasn't doing their job because you found the crash, and they didn't. Okay. So that's what you want to think about. You really want these people to be vandalizing the software. Okay.

And then after the vandal comes along, and smashes everything up, then you have this job that's left to you, which is debugging. And when you're debugging, you could think,

"Well, I'm the police officer, and I go, and track down the vandal, and make them stop." In fact, no, you want the vandal to keep vandalizing. You're the vandal. You got to keep trying to break your software, but at the same time you're a detective to try to find out where the breaks are happening, and how to fix them. Okay.

Now, here's something that people find a little odd, and I like to call them the four – well, some people don't find them odd. The people who do software engineering don't find them odd. I like to call them the four D's of software. Okay. And the first D is design. You've already seen that. And then there's development, which is actually writing your code. And then there's debugging. And then – anyone know what comes after debugging? You're like, "D party." No. Deployment. Okay.

And in this class so far we've focused on design, and development. And now, we're gonna spend some time thinking about debugging. And deployment is kind of like when you give it to your section leader. We don't want to worry about like you released software version 1.0 of your hangman program, and then we're like, "Oh, so when is 2.0 coming out?" And you're like, "Hum? Let me get back to you on that." And then you issue a press release, and the dates slip, and shareholders get angry. All right.

Yeah, like – see we should have CS106a shareholders for like, "When is hangman coming out? Oh, my God, you slipped the deadline." All right. Then you'll know how serious those late days are.

Then there's deployment. Okay. And the one thing that I would pause at, and actually a lot of people agree with, is that any problem that cascades from one step to another causes a factor of ten times in the cost to fix it. Okay. Which means you come up with a bad design for something, you're gonna spend ten times as much time writing the code for that bad design. And the bugs that come up as a result of bad design are gonna be a 100 times as costly to fix. And once that software gets shipped out into the field, and the only way to fix it is to issue a new version, that's a 1,000 times more expensive.

And people look at this, and they're like, "But, Marron, you can't be serious, right? How could it be a 1,000 times more expensive?" Well, let me give you a little example. How about our friend the space shuttle? All right. We have a space shuttle mission going on. Space shuttle had an initial set of software that was written for it. Okay. Right.

And we're thinking like, "Oh, you know, like in tex." How much do you think one line of code costs when they were all said, and done? If you think of the total amount of money spent on the software development effort for the space shuttle versus the number of lines of code that were written, how much did it cost per line of code? Anyone want to venture a guess?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Like dollars. \$10.00. That's kind of fun. That's probably a job I'd take. Anyone want to venture another guess?

**Student:** [Inaudible] thousand.

**Student:** \$1.00.

**Instructor (Mehran Sahami):** \$1.00. Higher – higher. Does that scare you?

**Student:** No.

**Instructor (Mehran Sahami):** Does that scare you? I'd take that job. I'd be like, "\$1,000.00 a line? Yeah, today in tex. Tomorrow, [inaudible]." Right? Like I'll take that job. That's the right number. Okay.

There is about 500,000 lines of code – and now there is more – but in the initial version of the space shuttle, and it took a total cost of \$500 million to develop that software. Okay.

So this is serious, right? This isn't something where we're like, "Yeah, it's fine, and we're vandals, and we're debugging." Right? When you're on the space shuttle, you better hope someone was doing their job doing this, right? Because if something's wrong, you can't fix it up in space, right? In space no one can hear you scream – well, if you're in the movie "Alien." But if you're on the space shuttle, it's real difficult to actually fix software. Okay.

Mars Rover was actually lost because of a software issue. Not actually – well, it was a software design problem that came up. All right. Anyone know what the design problem was?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah, it was basically two different – it wasn't – it was all basically the same difference as feet versus meters, right? One person was coding to a spec that said, "Oh, the input you're gonna get is in feet." Another one thought it was in meters. Right? Those are off by a factor of three. That's enough to kill the project. And guess how much money was spent on that thing. A lot. All right. So it's important to think of this – well, so how do you actually think about doing good debugging?

So the way we want to think about debugging is in some sense it's a river. And you're like, "What? What is Marron talking about?" It's – this is a river. This is the river of your program flowing. All right. And it's just nice. It's wonderful. And you're like sitting there, and you're having a picnic down here, and the sun is shining, and the birds are chirping, and you're like, "If my program gets here, and life is good, everything's just fine."

But what happens when the river is flowing? You run your program, and you get all this sewage down here. And you're like, "Having a picnic next to sewage is no fun." Well, what happened? So you go all the way upstream, and you look at the beginning of your program. You say, "Well, at the beginning I knew I had no bugs because my program

didn't contain any lines of code." And somewhere along the way as my program's running there was – oh, say this factory that was belching sewage into your water stream which showed up at the end when you're running your program, and this thing is a bug.

And often times the problem is there is not just one. If there was one you'd be like, "Oh, I can it. That's all good." Yeah, there's like factories all down, and up the river. There's some on the other side of the river. There's some in the middle of the river. All right. It's a bad time.

So how do you find those factories that are belching sewage? So here are some things to think about. First of which is the mindset of what causes – what tends to cause bugs. Okay. There's three things. I wouldn't say they cause all bugs, but they cause the vast majority of bugs. One is bad variable values. Okay. Some where in your program you have a value that is not what you expected, and as a result your program is doing something that it shouldn't do. What that tells you if you think about this mind frame is knowing the values of your variables at different points in your program is critical to understanding when the bad values entered your water supply.

Second of which is faulty logic, right? You meant for the computer to do X. You programmed the computer to do Y. Right? X, and Y were not the same. There's no such thing as a computer glitch. The computer does what you tell it to do. And so you might say, "Hey, the computer's not doing what I told it to do." Yeah, it's doing exactly what you told it to do, you just didn't tell it to do the right thing.

And last, but not least, and this is probably the biggest one is unwarranted assumptions. Right? "Hey, I thought the value were gonna be meters. You didn't tell me they were feet until after the mission launched, and we lost the Rover." Think about what you're assumptions are. And that's why when we talked a long time ago about pre and post conditions – right – when we talked about it in the little world of Carol – "Oh, we just didn't want them to walk into a wall." That was what assumptions were all about. What can your method assume when it's called, and what can you assume after your method's actually done, and do those match up?

So the things to keep in mind – and these are s – part of the zen of programming – debugging. And then I'll show the actual doing of debugging. Okay. People have a tendency to look for complex problems. Most of the problems are simple. So don't fool yourself into thinking a problem always has to be complex to be a bug. Some times it's just a very simple piece of faulty logic or some value that you mistakenly added one to or added three to some where that you didn't mean to under some conditions. Okay.

Be systematic. There's a tendency, and this tendency tends to increase as the amount of time left until the deadline for the assignment decreases, which is to just haphazardly jump around code, and be like, "It's got to be here some where. Is it over here?" No, that's the game I play with my one and a half year old son. "Is he over here? No. Is he over –" and he's just standing there with his hands over his eyes because he has the simple idea in

mind, and I'm looking for something complex, but I'm not being systematic. I'm just like, "Is it over here? Is it over here?"

And then finally at the end, right, had I just started at the beginning of my program, and began to trace it through, I would've found my problem. But I have a tendency not to do that. And part of the reason I have a tendency not to do that is that I make assumptions about where the problem is.

Now, hunches are great. It's great to think about, "Oh, I think the problem should be here," and have some intuition, and hunch about that. But when you have an intuition versus a fact, and the two collide – this is in your eleven rules of debugging on your next – on one of your handouts that you get. The fact always wins. Right? Your hunch never beats out a fact. So if you make assumptions about the problem, and you don't see the problem in the place of the code where you thought the problem should be, maybe the problem's not there, and you need to reassess your assumptions about those pieces of code that you thought were simple, and so they were error free, and in fact the error could be there. Okay.

Other thing is be critical of your code. Right? People have a tendency to look at some function or some method, and they think, "Oh, it's super simple. There can't be an error there." Or they just glance over it. I've actually seen at major software development companies, very large pieces of software written by brilliant, top of the line software engineers who the problem in their code was this instead of this. Right. It was harder for them to find that bug than to find issues with, like, memory, and garbage collection, and leakage, and all this other stuff because they were just used to that. Right?

They weren't critical of things like this. And when they finally saw this error you could just see like the eyes just nearly pop out of their head, and their head was gonna explode. But that's all it was. It was something simple they just – the code – they glanced through that code a thousand times and couldn't find it because they just didn't look at it critically. Okay.

And last but not least – and this is the important thing at 5:00 in the morning where you're programs not working. Remember this. Trust me. Don't panic. Panic is the worst thing you can do with bugs. And why? Because people sit there, and they think, "Oh, my God, there's this bug in my program, and it's over here. No, it's over here. No, it's over there."

No. Your code's not changing unless you change it. If you have a bug in your code, there's a bug. It's in one place. It remains in one place. Every time you run your program it's in that same place just begging you. It's like, "Come on. Come find me. Did he find now? No. Come on. Come find me." It's just there. It's that same little line of code, right? How many total lines of code are there? Like a couple hundred at most, right? It's in there. It's not moving. It's up to you to find.

Don't panic about it. Be systematic, question your assumptions, be critical of your code, and eventually you'll get there. But if you panic you won't because you won't be seeing it. Okay.

So what are some approaches we can take to debugging now that we have some of the basic sort of philosophy down, and the fact that the real critical thing is don't panic? The simplest approach to debugging is something that's called printf debugging, and printf is actually something that comes from the language C.

The way you can think about it is println debugging. And the basic idea is right – if bad values are the problem with your program, which in most cases they are, put some extra printlns in your code that tell you what the value of your variables are at different places in your code. Use printlns to let you know that you called a particular method. So at the top of a method just have a println that says, "Hey, method X just got called." And right before that method's supposed to be done, you add another println that says, "Hey, method X is just about to finish." And if you notice some problem that happens between those lines probably something's going on in method X.

Another thing you can do is just write out the values of your variables. And that's probably the thing I would recommend the most. At different places in your program just stick in printlns, and see if the values that you expect there matching what you thought was gonna actually be. Right? And if they're not you can sort of see at what point did the variables change the values that you didn't expect, and so something bad must have happened between the last time you saw the values that were okay, and the time in which they became bad. Okay.

The other thing – and this is a little bit more involved, but not a big deal. It's something called unit testing. Right? People have a tendency to test their entire program at once. They write the program, and they run it, and something doesn't work. Well, why not actually test out individual units or individual methods in your program, right? Write a call to a particular method where you pass in the values for all the parameters with known values, and see if the thing that it gives you back is what you expected it to do.

That's called unit testing because you're just testing one unit at a time. And when you're done doing the testing you can take those calls out, and run your whole program. But when you write some method, especially if it's a complicated method, just write a couple lines at the top of run that the first thing they do is call that method to see if it's actually doing something reasonable with some artificial values that you give it. Okay.

So, one way that you can combine a lot of this stuff is if you happen to be working in a development environment that gives you facilities to do debugging more easily. And so I'll show you, if we come to the computer, Eclipse actually has a very nice debugging environment. Okay. And so rather than these printlns, you can do sort of in any program, right, because even if you're doing a program that you don't have a nice debugging environment, you can still do println, and life will be good.

So let's actually debug a program using the debugger in Eclipse. So here's a little Roulette program, and the way Roulette works is we're gonna have some constants to begin with, how much money you start with, and how much you're basically wagering or betting on every spin of the Roulette wheel. So the way Roulette works is we have a big wheel that we spin. And here's all the rules of Roulette down here. I'll just show you the rules. Ah, too much text to read.

The basic idea is the wheel has a bunch of different numbers on it from zero to 36 all sort of consecutive numbers. And the way you can bet on numbers is you can either bet for an odd number to come up, an even number to come up, a high number to come up, which means a number in the top half from 17 to 36, or a low number to come up, which means from one to 16. And you might – or if – one to 18.

You might say, "But, Marron, what about zero?" Zero is always a loser. So, zero doesn't count as an even number. It doesn't count as a low. It's just always the losing value. And the reason why that exists is because that's the house's advantage. It gives them a slight advantage of about three percent over the player, and that's how in the long term they make money because zero is always a winner for the casino, never a winner for the low – the player, other than that, everything's even money. If I bet \$10.00 on low, and a five comes up, which is in the low half of the range, then I win \$5.00. Okay.

So that's what the instructions basically say. Just a bunch of `println`s they write out in instructions. Okay. We're gonna have a random number generator, `rgen`. That's an instance variable that's just generates random numbers, and it gets an instance of the random number generator. Hopefully after Breakout, and Hangman, this should be fairly comfortable to you.

And now we kind of come to the rest of the code. Okay. First thing I'm gonna do is set the font to be big. I'm gonna write out the instructions, and then I'm gonna play Roulette. And so I say, "Hey, in playing Roulette I have some starting amount of money that the user gets. As long as their money is greater than zero, I tell the user you have some amount of money. I ask them which betting category they want to bet." That means either even or odd or high or low. So they type in a string. There's four betting categories.

I spin the Roulette wheel to get a value between zero and 36, and then if the winning – if there's a winning category based on the outcome that they have, I give them – I basically say, "You won this amount," and I increase their amount. If they didn't get a win in that winning category, then I say, "The number is not –" basically, "That number is not whatever you bet. So you're gonna lose." And I subtract the amount of wager from the total money. And if they happen to get out of this loop then their money is down to zero, and I say, "You ran out of money," and we stop. Okay.

So I might just run this program. Do te do te do – and see what's gonna happen. Notice I didn't show you all the other code yet? Because I want to show you the facilities of the debugger rather than just having you stare at code for a bunch of time, and try to debug it

because that's not the way you want to debug by just staring at code. You want to look at what the symptoms are.

So here we're gonna run Roulette. And it writes out all the instructions, and it says, "Enter betting category." And so we'll say, "Hey, I'm gonna bet even." "Ball lands on 33. That number's not even so you lose." I'm like, "Okay. I'm gonna go for even again." Good old even. Even always wins. "Number 35. You lose." "I'm going for even again." "27, you lose." "Even come on." "Nine – 31 – 15 –" and then you're like, "I had so much faith. I lost all this money, and then ball landed in two, and I still lost money."

And you just kind of sit there, and you're like, "Yeah, this casino's not really what I had in mind." And you try to leave before the bouncers get involved. But you realize you actually have a bug in your program, right? You bet on even. The ball landed in two. That should be an even number. And you still lost money. So then you think, "Okay.

Something's wrong. What's wrong? Well, let me figure out right – this winning category thing, I should've said, 'If the outcome moded by two or remaindered by two was equal to one that should be odd. And if the remainder after I divided by two is zero that should be even.' So I would've thought this would've been the case. What's going on here?"

And so what I can do in Eclipse is I can set up what's called a breakpoint. The way you set a breakpoint is that a particular line over here on the gray bar you click twice. And if you click twice, notice I got that little circle there. Can you see the circle on the very edge of the screen? That's called a breakpoint. That means if I run the program now with the debugger my program will stop at that point, and allow me to look at, like, values of variables, and other kinds of things. Okay.

So I set that breakpoint by double clicking, and now I need to run again. So the way I run again is see this little bug up here. That means run with the debugger. See up here? It looks like a little tiny bug. So I click that to run with the debugger, and now it starts rerunning my program again. And it says, "Enter a bet." And so I say, "Even." And it says, "This kind of launch is configurative. Open the Stanford debugger perspective when it suspends. Should I open this perspective now?" All right. It looks all big, and nasty, and you just say, "Yes." Okay.

And it says, "Hey, your program got here. Look at what values you actually have." And so there's a couple things to look at. Up in this debug window up here, it shows me – remember when we talked about stack frames. This is what's called the call stack. The run method called the play Roulette method which called the is winning category method which is where I'm currently suspended. So it tells you all the functions that are currently – or the methods that are currently active to get you at your current point.

Over here in the variables window, it actually tells you the value of the variables. So my bet is even, outcome 13. I say, "Huh? It says the ball lands in ten. I thought that was supposed to be even, and I was gonna even. What's going on? Over here it tells me that my outcome, which is the thing that I'm gonna mod or divide by two to see if it's odd or even is 13. That's not what I got, which means at this point sewage has entered the

system. The value that I thought I should have is not the value that my program actually has." Okay.

So what am I gonna do? There's two things I'm gonna do. First of all I want to say, "Oh, I'm in this debugging perspective. How do I get back to the little editor that's my friend?" You come to the Stanford menu, and you pick switch to editor. Okay. That takes you back to what you're used to. Notice we're still stopped here, but you're back just in this editor view.

There's a couple things we're gonna do. First thing we're gonna do is I'm just gonna say, "Whoa. Stop the program. Something bad is going on." Next thing I'm gonna do is say, "Hey, my program involves random numbers. So I'm gonna set the seed for my random number generator, so I can count on always getting the values the same every time." So if you're debugging with something that involves random numbers, set the seed before you go into debugging, and that'll guarantee you always get the same numbers. So I set the seed.

The other thing I'm gonna do is say, "Hey, the number that got printed on the screen was different than the number that I actually got. So let me look at the code that prints a number on the screen." So here's spin Roulette wheel. The ball lands then it generates a random number, and then it returns a random number. What's the problem here? Anyone want to venture a guess?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** They're not the same random number, right? It generates one random number to say, "Hey, the ball lands in some number." That was kind of a social. And then it says, "Hey, you know what? Yeah, our casino – we just remove that ball, and we just pick another number randomly to say that's what it really was." Okay. That's – maybe a lot of fun for the casino, not so much fun for you.

So what we really need to do is inside here we'll have something like spin, and spin is gonna be whatever we get from the random number generator. Okay. So we'll just say, "Hey, spin is this random number." And then what we're gonna write is spin over here. And what we're gonna return is also spin. So we only generate one random number.

Let's save that, and we're like, "Oh, we only generated one random number. That must be the bug. I must be done. Oh, yeah. Yeah, good times. Good times." So I run my program again. Do te do – and I get this whole thing, and it says, "Enter betting category." "Even." "27, you lose." I'm like, "Oh, it must be working." "Even." "15." I'm like, "Oh, yeah, my program is so good. I'm such good –" yeah. And you think you've found your bug, and then you realize when you're the vandal, and you're trying to see if that error condition still exists, "Hey, we found one bug. That was great. That didn't solve this problem."

So you say, "Okay. Back to the drawing board. Let me quit out of here. I'm gonna go, and set another breakpoint over here. Okay. So let me clear this breakpoint. I'm gonna set

another breakpoint. I'm gonna run it with the debugger. Okay – which is what I didn't do before." So now I run with the debugger, and it says, "Enter a bet." And I say, "Even." And it says, "You've stopped here." And so, "Yes. I want to do this debugger perspective."

And I say, "Hey, the outcome is 27, and so my – if I look at – my bet is not even, right? It should be odd." So I say, "Oh, that's great. Well, it shouldn't be odd, so that shouldn't match." There are some tools up here I can use to say, "Step through my program." There's one in the middle up here if you can look at this line that has sort of this curve to it. That means step over. And all of these – all these different icons are actually explained in your debugging handout, but step over is the critical one. It says, "Just execute this line of code, and go down to the next line of code."

You have two other options. You have step into, which actually makes a function call or method call if you happen to be in a method, and step return, which says, "Step out of the current method I'm in." They get a little funky.

So step over is what we're gonna do. So we step over, and odd doesn't execute, and you're like, "Yeah, odd shouldn't have executed. Even – yeah, my bet is even, right? I know over here my bet has the value even, so I execute another line. Whoa. What's going on there? I should've gotten bet was equal to even, and return the fact that my outcome was actually true that I actually won." What's the problem here?

**Student:** [Inaudible].

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Pardon?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Ah, strings. These are strings. I can't use equal equal on strings, right? It just doesn't work. It doesn't work when you're not looking either. So I say, "Hey, come back to the editor, and when you want to make some changes – yeah, instead of checking for equal equals, I actually need to say dot equals here."

As a matter of fact I'm gonna use equals ignore case because that'll make the life of the user even better. They can either put in odd or even in upper case or lower case. So I'm just gonna do a little copy, and paste, which is always a dangerous thing to do. Do te do – so close. Do do – and then for high – and at this point I could go ahead, and save. Oh, it's still in this perspective. I want to quit out of my program.

And now there's one final thing I'm gonna do here. Let me switch to the editor, and I'm gonna run one – let me save, and run one last time. And this becomes very subtle. So if I say, "Hey, I'm gonna bet on odd. Yeah, look I'm winning now because I know that odd

numbers come up. I should've remembered this from last time because I set the seed, right?" And I keep betting on odd.

And then one interesting thing that happens is zero came up, right. This is something that may take a while to come up. So I start betting on even because I lost with zero before. It's gonna take me a while before zero might ever come up again, but I need to be – I need to persevere. And it's probably not gonna come up in this run. So let me quit this, and run it again. Do te do – even. And I should get the same sequence of numbers. Even, even, even – oh, I lost on zero. Right?

There's one subtle bug that's still remains in my code which is for the case for even I not only need to check for even, I also need to make sure that the value is not equal to zero. So the other final thing I need to put in here is and and outcome outcome not equal to zero. Okay. And then my code is actually fixed.

So sometimes it may take me a while to find the actual error in my code through running, but now you can use the debugger to find it. All right. And if there's any questions I'll take them after class, and you can pick up your finals –

**Student:** Thank you.

**Instructor (Mehran Sahami):** – over from – or the –

**Student:** Thank you.

**Instructor (Mehran Sahami):** – midterm exams over from Ben.

[End of Audio]

Duration: 54 minutes

## Programming Methodology-Lecture19

**Instructor (Mehran Sahami):** Howdy. So welcome back to yet another fun filled, exciting day in CS106A. I don't know if there's any days actually we started where I didn't say that. I don't know. Someday, I should go back and watch the video. But they're all fun filled and exciting, aren't they? So it's not like false advertising. There's no handouts today. A little breather after the four handouts you got last time. And another quick announcement, if you didn't pick up your midterm already, you can pick it up. They're along the back wall over there in alphabetical order, so hopefully you can pick it up if you haven't already gotten yours. If you're an SITN student and you're worrying about where you can get your midterm, it will be sent back to you through the SITN courier, unless you come into class and you picked it up, in which case it won't be sent back to you because then you already have it. All righty. So any questions about anything we've done before we delve into our next great topic? All right. So time for our next topic. And our next topic is really a little bit of a revisiting of an old topic, kind of an old friend of ours, and then we're gonna push a little bit further. So remember our old interface. I always love it when math books say like "recall" and they have some concept, and I look at that, and I'm like, "Oh, recall the interface. Oh, what good times the interface and I had, like we were holding hands and running through a garden, the interface and I, and I recall our happy times together." So now's the time to recall the interface. What was an interface? Right? Last time when we talked about interface, we talked about something really generic, which was basically – it was a set of methods. And this was a set of methods that we sort of designate that some sort of classes actually shares. So it's a common set of functionality – common functionality among a certain set of classes.

And we talked a little bit about how yeah, there's – if you had some notions of [inaudible] classic standing in other class, you get sort of that same idea, but the real generality of interface is with – that you could have certain things that weren't related to each other in an object hierarchy or a class hierarchy that you still wanted to have some common methodology. And you sort of saw this all before, so in the days of yore when we talked about G objects. Remember? Those little fun graphical objects like the GLabel and the GRect, and all that happy stuff. And we said there were a bunch of interfaces there. Like for example, there was an interface called GFillable, and the GFillable interface had certain methods associated with it, like you could set something to be filled, or you could check to see if it was filled. And we said some of the objects actually implemented this interface, so for example, GRect, and GOval, and a couple others actually implement this GFillable interface. And there were other things like GLabel where it didn't make sense to have a filled or unfilled label, so GLabel didn't implement this interface. Okay? And it was just kind of a set of functionality. We're gonna kinda return to this idea of interface to talk about some of the things that we've actually done so far, like array lists, and some new concepts you're gonna learn, and how it relates to our notion of interfaces.

But the more general thing to take away from interfaces is in terms of thinking about how they're actually implemented for their syntax. Sometimes what you'll see in the actual

code when you see that a particular class implements the methods that are considered part of some interface, the way we write is that is we say public class, and then we have the class name just like we usually do. So I'll write class name here, and I'll underline it to indicate that this is just a placeholder for the actual name of the class as opposed to writing class name. And then what we would write is implement – this would all generally be on the same line, much in the same way when you've seen before like your class, my program extends console program, or extends graphics program. Here we have a notion of implements, which is an actual word in Java, and then the name of the interface that it implements. So this over here would be the interface name. So you can imagine somewhere in the definition of the GRect class, there is a GRect class that implements GFillable. And GFillable is defined to be some interface somewhere which just specifies a set of methods, and any class that implements that interface needs to provide its own version of the methods. That's all it means, so for a class to implement some interface just means that that interface specifies some set of methods, and this class provides all of those methods. By providing all of those methods, it what's called implements the interface. Okay? And it's perfectly fine for a class to not only implement an interface, but also to extend some other class. That's perfectly fine, so the syntax is gonna get longer up here, but I just want you to see the implements syntax.

And then inside here, just like you would be used to – actually, I'll just draw the opening brace here and inside here – this would be all of your code for the implementation of the methods would go in there. So it's the same kind of syntax for writing a regular class, but now you're just specifically telling the machine, "Hey, this class is gonna support all the stuff from some particular or other interface," like GFillable or whatever the case may be. Okay? So that's kinda the concept. Now why do we sort of revisit this idea of interface is that now it's time to talk about some new things and the interfaces that they actually implement. So any questions about the basic notion of an interface or implementing an interface? Good call on the microphone.

**Student:** How's this different from – does this work?

**Instructor (Mehran Sahami):** Sure. Just press the button. We'll pretend it's working.

**Student:** How is this different from extending a class or just calling it an instance of another class?

**Instructor (Mehran Sahami):** Right. So that's a good question. The difference between this and extending a class is for example the notion of a hierarchy that we talked about. So if we have some class that extends some other class, we basically say, right – like when we talked about – remember in the very first day – primates and all humans are primates? We would basically say any human is a primate. The difference is there might actually be some things that are not primates, and humans may actually implement a class like the GIelligence class, which means you have a brain that's larger than a pea or something like that. It turns out there's this other thing over here called the dolphin which also implements the Intelligence class. Right? We generally like to think of dolphins as – a point of debate. I don't know how they actually measure this. There's like the dolphin SAT or

something, the DSAT, and they're like [inaudible] – I don't know how they do it, but evidently someone has figured out how to measure intelligence in dolphins.

So you could say this dolphin class actually implements the methods of being intelligent, which might be some method like it responds to some stimulus in the world, but a dolphin's not a primate, so – at least as far as I know. Dolphin's not a primate, so there's no extends relationship here, and that's the difference. Right? So an interface in some sense provides more generality. When you extend the class, you're saying there's a direct hierarchical relationship among those objects. Interfaces, you're just saying there's this thing like the intelligence interface, and sometimes these get drawn with dash lines when you see them in diagrams. Both humans and dolphins provide all the methods of the intelligence interface, but only a human is a primate. That's the difference. Okay? So any other questions? Uh huh?

**Student:** Is the code in this instance the code for that particular class or the interface?

**Instructor (Mehran Sahami):** Pardon?

**Student:** You wrote code there as in – is that the code for that class or for the interface?

**Instructor (Mehran Sahami):** Yeah. So the code here is the code for class name, and somewhere else there would actually be the code for the interface. And that's covered in the book. We're not actually gonna be writing an interface together, which is why I'm not showing that to you here in detail, but the basic idea is just so that you would actually see that when a class implements an interface what that syntax would look like. Uh huh?

**Student:** Can you implement more than one interface?

**Instructor (Mehran Sahami):** Yes. You can implement multiple interface. For example, GRect implements both the GFillable interface and the GResizable interface, and that's perfectly fine. It's a good time. All right, so let me move on a little bit and talk about something that we're actually gonna deal with which is an interface, which is why we kinda revisit the subject. And the particular thing we're gonna talk about is something that's known as a map. Okay? So a map – and you might look at this, and you're like, "Map? Is that like oh yeah, the travel guide, like I got one of these, and I kinda look in here, and I'm like yeah, where was I going? Oh, here's a map of all the interstates on the United States, and I'm sure this is copyright, mapquest.com. There's this map, right? Is this what you're talking about?" No. This is not at all to do with what I'm referring to here as maps.

So when you think of the word map, you need to let go of what your previous notion of what a map might have been is, and think of the computer science notion of a map. So basically, all a map is – it's an interface in Java, which means it's some collection of methods that will implement some functionality. What are the methods? What is a actual map? What does it do? The way to think about a map is there's two key terms you wanna think about for map. One is called the key, and one is called the value. And basically, all

a map is it's a way of associating a particular key with a particular value. You might say, "Whoa, man. That's kinda weird." Yeah, that's a very abstract idea. So let me give you an example of a map that you've probably used throughout the majority of your life, but no one told you up until now that it was map, something called the dictionary.

Anyone ever use the dictionary? The number's getting smaller and smaller. No, I just let my word processor spell correct for me. A dictionary is a map. Its keys are the words that exist in the dictionary, and the values are the definitions of those words. So what a dictionary gives you is the ability to associate a particular key with a value. It associates words with their definitions. And the important idea to think about a map is in a map what you do is you add key value pairs – so like in a dictionary like the Oxford English Dictionary, right every year there's a little convention, and they actually figure out some new words that are considered part of English, and they add them to the map that is the OED.

Anyone have a copy of the OED? Yeah, a couple folks. It's a good time. Just get it. You get the little magnifying glass version where it's got like four pages on one page, hurts your eyes, but it's a good time. That's what it is. We're just adding key value pairs. Now what you do in a dictionary is when you look things up in a map, you don't look them up by the value. Right? It would make no sense if I told you look up the word for me that has this as its definition. You'd kind of be like, "Yeah, Mehran. That's just cruel and unusual." The way a map works is you look things up by their key. You have a particular word in mind in a dictionary, you go to the dictionary, you look up that word, and then you get its corresponding definition. Okay? So what we like to think of a map in this abstract way it's an association. It's an association of a key to a value where when we enter things we enter both the key and the value together, and when we look things up, we look things up by saying get me the value associated with this key. Okay?

So dictionary's a simple example of that. There's other examples. Your phone book is same thing. It's a map. The keys in the case of your phone book happen to be names that you wanna look up, and the values in the case of your phone book happen to be phone numbers, but it's also a map. There's maps kind of all around you. Everywhere in life, there's a whole bunch of maps. No one told you they were maps before. And if I told you all before this class started, "Yeah, a dictionary and a phone book are really the same thing," you might kinda look at me sorta weird and be like, "No, Mehran. One is like storing names and numbers, and the other one's storing words and definitions." But when you tell this to a computer scientist, they're like, "Yeah. They're both maps," because now you know what a map is. All right. So any questions about the general idea of a map? Uh huh?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** As far as we're dealing with right now, you can think of a key having just one value, but that value is really some abstract notion, right? So you could actually think we have some word – like let's say we have a dictionary that has multiple definitions. Its key could be a word. Its value could be an array of definitions. So really, in the philosophical sense the value is one thing. That one thing happens to be an

array that stores multiple values. So philosophically it's one thing. In terms of the practicality of what you're storing, you might actually be storing multiple things. And oh, maybe for Assignment No. 6, you might actually do something like that, but that's not important right now because you're still working on Assignment No. 5. All right? So [inaudible] something, you're like, "Quick. Write that down." Yeah, we'll talk about it again when we get to Assignment No. 6. All right? So map is a particular interface, so that means we need to define some class that actually implements this interface, that provides some notions of these key value abstractions for us, and so in Java we have something called a hashmap. Okay? A hashmap is an actual class. It is a class that implements the map interface, which means everything that is a hashmap implements map. Okay?

Map is not actually a class in itself. Hashmap is a particular class. Map is just a set of methods that I think are actually useful to have. Now where did it get its name? Why do we call it a hashmap? What's this hashing thing all about? And in the book in excruciating detail, it talks about hashing, what hashing is, and how to implement hashing, and hash codes, and all this other stuff. You don't need to know any of that for the class. All you need to know is how to use a hashmap. You don't need to build a hashmap. If you wanna build a hashmap, in CS106B a couple weeks from now, you will build a hashmap. You will have the joy of seeing the underlying implementation of hashmaps in C++ of all things. But for right now, you just don't need to worry about it. The reason why it's called a hashmap is it uses a particular strategy to store these key value pairs called hashing. That's all you need to know, and because it's called hashing, we call it a hashmap. That's life in the city. Okay? But so this particular hashmap is a template. Right? Remember we talked about templates, and we talked about how the array list was a template, and you could have an array list of strings, or you could have an array list of integers, or whatever the case may be. A hashmap is also a template, but the key here is there are two types involved. It's like two scoops of raisins. You have two types involved in the template for a hashmap. You're gonna have a type for your keys, and you're gonna have a type for your values. So what does that actually look like in terms of syntax? Let me create a hashmap for you.

Now that you know all about interfaces, we can erase this part. So let's create a hashmap. The class is called hashmap. Because it's a template, it has this funky angled bracket notation, but it's gonna have two parameters for two types for this templated class. The first type is what are you gonna have for your keys. So let's say we wanna implement a dictionary. A dictionary for its key type is gonna have strings because it's gonna be words. What type – the next type – so you have a comma in here, and then you specify the next type, the type for your value. For a dictionary again, if we assume a simple definition rather than multiple definitions which could be an array, we'll also just gonna have a string for the value type. So hashmap string comma string, or sometimes we refer to this as a map from strings to strings because it maps from keys to values is how we would actually specify this. We need to give it some name, so we might call this "dict" for dictionary, and then this is going to – we are gonna create one by saying a new hashmap – and this is where the syntax gets a little bit bulky, but you just have to stick

with it – string, string, and we’re calling a constructor, so again we have the open paren, close paren. And that will create for you a hashmap that maps from strings to strings.

Now we could also create a hashmap for a phonebook, so let’s create a hashmap for a phonebook. In the case of a phonebook, you might say, “Hey, names are still gonna be strings, but phone numbers” – well, phone numbers if they’re generally seven digit phone numbers – let’s just say it’s seven digit phone numbers. We won’t worry about the ten digit phone numbers. Seven digit phone numbers, I can store that in an integer. And so you might say, “Hey, I’m gonna have a hashmap of string to int.” And at this point you should stop because your compiler will give you an error. What’s the problem with this?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Int is a primitive. Right? All these templative types require objects for their types. They require classes, which means we can’t give it a primitive. We have to give it the wrapper class integer. So it’s gonna be a map from strings to integer, and we’ll call this thing phone book. And this is just going to be – we’ll give you the syntax in full excruciating detail – hashmap from strings to integer [inaudible] the constructor, and that’s gonna create a hashmap from strings to integers for you. The first thing doesn’t always have to be a string. It can be whatever you want, right? You can have a hashmap from integers to some other integer, where you wanna map from integers to integers if you wanna do that. Some people do. I don’t know why. Sometimes you do. Now the other interesting thing is because a hashmap actually implements this map interface, sometimes the way you actually see this written is you’ll see it written without the hash in front. Okay? Just for the declaration. Here you still have to have the hash, and here you still have to have the hash, but just in terms of the type, sometimes it will be written map string to strings is a hashmap of strings to strings.

And you should look at that, and at first you get a little bit tweaked out because you’re like, “But Mehran, you told me a map wasn’t a class. You told me it was an interface.” Yeah. And a hashmap implements that interface. So what I’m saying is I’m saying I wanna create a map, that map that I’m gonna call a dictionary, and what’s the actual thing that implements the dictionary? It’s a hashmap. Okay? Because the hashmap implements the map interface, it will have all the methods the map expects, so any time I use this guy, and I say, “Hey, it’s a map,” and I call any of the methods for a map, it’s perfectly fine because a hashmap is guaranteed to have implemented those because it implements the interface. So just so you see this in code, sometimes you’ll actually see the hash out here, sometimes you won’t. Sometimes you feel like a nut, sometimes you don’t. You should see it both ways. It makes sense both ways, but you can’t drop the hash here because a map is an abstract idea. You can’t say give me a new abstract idea. Okay? That’s not gonna work. You can say give me a new hashmap. That’s the concrete thing. And what am I gonna assign that concrete thing to? Well, this concrete thing implements the abstract idea, so it’s perfectly fine to say, “Yeah, here’s the concrete thing. Use it in the place of the abstract idea.”

Any questions about that? Kinda funky, but that's the way it is. So when we have these key value pairs, how do we actually add stuff to our hashmap and pull stuff out of our hashmap? So there's two methods. There's a method called put and a method called get. These are the two most commonly used methods of the hashmap. And the way these things work is put – let me write them more toward the middle of the board. Put strangely enough – I know this is gonna be difficult to see, but put actually puts something in your hashmap. So the way put works is you give it a key and a value. The types of those things have to be the types corresponding to whatever object you've created, so if I wanna have – for example, I wanna put something in my dictionary, I'd send the put message to the dictionary hashmap, and I'd say, "Hey, add this key value pair," and key and value type strings to match this thing. Or if I have some phone book, I could say put key value where value better be an integer and key better be a string.

Besides putting things – it's kinda fun if you can put a lot of things in a hashmap, but it's not so much fun if you can't get them out. It's kinda like you had your phone book was kinda like your cell phone that you put all these numbers in, and then you tried to get a number out, and it just said, "No, I'm not gonna give you any numbers." You would probably quickly stop putting numbers in. Some people wouldn't. I don't know why, but I've seen that happen.

You're like, "You've seen people not be able to get numbers out of their cell phone?" Yeah, like their cell phone's busted and they just can't deal with that. But that's not important right now. What's important right now is you actually wanna be able to get values out. So out of our dictionary, you might say, "Hey, Mehran. I wanna get some particular word." I wanna get the definition for a word, really. This word is my key, so one way I could think of it is it's a word I'm looking up, but in the abstract notion, it's really a key. And so what this gives me back when I call get on the key is it goes over here and says, "Hey, dictionary. Do you have some value associated with the particular key?" So if this key exists in the hashmap, what it gives you back – so it will return to you the corresponding value. If that key doesn't exist in the hashmap, it gives you back null if key not found. Okay? So that's the way you can determine is this thing actually exist in my hashmap or not. If I try to do a get on it and it's not there, I'm gonna get back a null. Okay? So any questions about that? Uh huh?

**Student:** If the value's an integer, will it still give you back null?

**Instructor (Mehran Sahami):** Yeah, because integer is a class, so you're gonna get back object of type integer, and if you happen to look up something that's not there, null is a – it's the empty object. That's why you can't do int because there is no null int. Good question.

So let's create a little hashmap, add a couple values to it. So if let's say I had my phone book, I could say phone book dot put – always a dangerous thing to give out your phone number, but it's on the first handout. It's 7236059. Call. It's a good time. So that's just gonna put – this thing is an int, right? So in order for it to become integer, what's gonna happen is this is automatically gonna get boxed up for you as an integer, and then that's

what's gonna get stored. So this autoboxing is what's happening to you, which makes it a little bit more seamless to use it with an int, but if you try to do this in an older version than Java 5.0, it'll actually give you an error because it didn't do the boxing for you. Notice there's no dash in there by the way because then my phone number would be some negative number because it'd be 723 minus 6059, and that's a bad time. So then we could add someone else to the phone book. Put Jenny – anyone know Jenny's phone number?

**Student:** 8675309.

**Instructor (Mehran Sahami):** 8675309. It's amazing how much longevity that song – like I remember listening to that song when I was in high school. It's such a catchy tune. And if you don't – you have no idea what I'm talking about, look it up. Just query 8675309. You'll find it on Wikipedia. It's not important. It's really totally irrelevant, but it's just fun. All right. So once I have these entries in the phone book, I could say something integer mnum, because that's what I wanna be Mehran's number, equals phone book dot get and give it Mehran. Okay? Important thing to remember is these are – the keys are all case sensitive, so if I put in a lower case M on Mehran here, it would return null because it wouldn't be able to it. So keys are always case sensitive in maps. Just an important thing to think about, okay? So with that said, we can actually get a little bit more concrete with the hashmap. I need a volunteer. Come on down. It's easy. You're up close. You're gonna be a hashmap. Here's your hashmap. I've just created you. You exist. Rock on. All right. So I hope you don't plan on taking notes any time soon because it might be difficult as the hashmap. So what we're gonna do is we're actually gonna enter – we're gonna call these commands on the hashmap so we can actually store these things in there, and then see what other commands – other things we can do. So normally the thing that gives bulk underneath the hood to a hashmap is our friend the Ding Dong.

So what we're gonna have is certain things that we're gonna enter in our hashmap, which is basically a Ding Dong sandwich, and on one side we're gonna have the key, and on the other side, we're gonna have the value. So what I'm gonna do is I'm gonna write the keys in black and the values in red so we can keep track of them. That way we won't get them confused in our minds. So the first key I'm actually adding is I'm gonna add myself. Mehran – I guess I could've written this before, but I didn't, so I have my key. And that key has with it a value, and the value sticks with it, so 7236059 – I shouldn't have put the dash in there, but I accidentally put the dash in there, and I say put. And this goes into the hashmap. Okay? At this point, I have no notion of what's in my hashmap, or ordering, or whatever. I just created something. I tossed it in the hashmap. So then I say, "Well, Jenny, she was always a good time." You've gotta listen to the song, all right? If you have no context for the song, you're like, "That's not funny." 8675309 – we put Jenny in the hashmap. Now that we have these things in the hashmap, there's an interesting thing that comes up, which is when I had an array list, I knew how many things were in my array list. I could refer to the first element. I could refer to the second element. What's the first element that's in my hashmap? You're like, "Well, you put in Mehran first," but it's not guaranteed to necessarily be the first element. A map has no intrinsic ordering. It's all just kinda mashed in there. So there are some things we might additionally wanna do on

the hash map to give us some notion of what's in the hashmap and how big the hashmap is.

So here's a couple more methods, and you could just bear with me. I swear it will be over soon, the pain and the agony. No, that's a good time. All right. So a couple other methods – I might have to ask you to just move over slightly – is we can remove a key. So if I ask to remove a key, what I pass it is the key, so I might say something like phone book dot remove and I give it some key. Now if that key exists in the phone book, it removes it from the hash map, and it is now gone. So if I call remove on Mehran – oh, hash map, I need Mehran. So you look somewhere inside the hashmap and you say, "Oh, here's Mehran. Remove. Phone number goes with it." That's just the way it is. The key and the value always stay together. Now besides removing something – and if Mehran didn't exist – so I could actually say remove Bob. You look in there. There's no Bob. You don't need to do anything with the hashmap. It's sort of this simple operation. There's no exception thrown, nothing like that. I asked to remove something. It's not in there. We just keep sticking with it. But what I can actually ask is, "Oh phone book, do you contain a key?" And so there's a method, contains key, that I give it a key and it returns to me a Boolean. So I can ask you, "Oh phone book, do you contains key Jenny?"

**Student:** True.

**Instructor (Mehran Sahami):** True. Good times. Phone book, do you contains key Mehran?

**Student:** False.

**Instructor (Mehran Sahami):** False because the key's been thrown out, right? We already removed that key. And last but not least, we can ask the phone book for its size. So this will return an int. Phone book, what's your size?

**Student:** One.

**Instructor (Mehran Sahami):** Excellent. Thank you very much. And you're done, phone book. Thank you very much. Nice work. And for that, we'll just – you and everyone around you. It's an array list of Ding Dongs.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** All right. So that's the way you wanna think about it. It's this bag. You put stuff in the bag. You take stuff out of the bag. Whenever you put stuff in the bag, you put them in in pairs. When you take stuff out of the bag, it's always pairs. And all the work that you're doing is always in terms of your key, but the value is associated with it. So when you wanna remove stuff or check to see if it contains, you never look based on the value. You always look based on the key. That's just the way things are. Now this whole notion of maps is part of a bigger framework, so if we can get the computer for a second, I'll show you the bigger framework just real briefly. There's

this notion called the collections hierarchy, which is – you’re like, “Oh my god. It’s big. It’s huge.” No.

Most of the stuff you don’t need to worry about. There’s some of this stuff you’ve already seen. An array list you’ve seen. An array list is something called an abstract list, which is just the abstract concept of a list, and that’s part – that actually implements an interface called list, which implements an interface called a collection. So array list, this thing you’ve been using this whole time, we didn’t tell you until now is actually something that we could refer to as a collection in the same way that we could refer to a hash map as a map. Now there’s a few other things in here that we’re just not gonna deal with in this class, like a tree set. Or you might notice there’s something over here called a hash set. Hash set and hashmap, not the same thing. So you should just know that. You don’t need to know what a hash set is. You just need to know that it’s not a hashmap because the words aren’t the same. If they were the same, then it would be called a hashmap. But there’s this notion of an abstract set that they are subclasses of that implement some abstract notion of a set, so there’s this big kinda complicated hierarchy, but the important thing to remember is all of this stuff at the end of the day is a collection. So there might be some operations we might learn how to do on collections that applies to all of it because all of these things at the end of the day implement the collections interface.

There’s a same kind of notion for hashmap. The hierarchy’s just a little bit smaller so it’s easier to see. A hashmap is this notion of an abstract map, which we talked about. It’s kind of an abstract concept that has key value pairs, and what really defines that is an interface that’s called a map. Now it turns out that besides hashmap there’s another kind of map called a tree map, and a tree map also allows you to put things into it and get things out of it. The underlying implementation is just a different way underneath the hood of actually implementing a map. And that’s the whole critical concept here. The critical concept is the abstraction that the idea of a map interface gives you. When you think about something that implements the map interface, all you need to worry about is what are the methods that are involved in a map. Underneath the hood, hashmap may have some different methods than tree map for some specialized kind of things, but we don’t care. As long as we just treat everything as a map, someone someday comes along and there’s the hashmap, the tree map, and then they create the super cool funky map. And you’re like, “Wow, the super cool funky map is just way more efficient than all the other maps that existed,” because someone had this great idea about how to implement maps like in 2027.

Well, if you go back through your code and everything’s written in terms of a map interface, and super funky cool map – was that super funky efficient map? Gotta get the name right – actually implements the map interface, none of your code changes, except in one line when you create your map. Rather than saying map string string hashmap, you say map string string super funky cool map. And everywhere else in your code when you refer to dictionary, you’re just using the methods of the map interface and no other code changes. So that’s the power of thinking of the notion of object oriented programming and object oriented design in these abstraction hierarchies is the fact that as new

implementations come along – and they really do, right? It's not like computer science is a static thing. Ten years from now, there will be other implementations of these things which will be more efficient and better than what exists now. But if we relied on always thinking about hashmap, then our code would be much more rigid than if we just think about this abstract concept of map. Any questions about that? All right. So think abstractly. That's kinda the bottom line from that whole little diatribe. So the one thing that we wanna think about is when we have these things like maps and array lists, and I told you an array list is part of the collection, what do I actually get with these things? What does a collection buy me other than the fact that I get to draw this complicated picture and be like, "Oh, look. This is a collection framework?"

What it gives you that's interesting is a notion of what we call an iterator. What is an iterator? An iterator is just an idea basically. It's a way to list through a set of values. What does that mean? What that means is if I have some array list, and that array list contains like ten different items in it, one way I could think about that array list is having some for loop that counts from zero up to nine, and I get each one of the individual values. Another way I can think about that array list because an array list is really a collection is if all collections allow me to have an iterator, this iterator will allow me to very easily – and I'll show you the syntax – have a sequential process of going through all the values in my array list. And for an array list, it's not as exciting as some other things because you're like, "But I already have a way of going sequentially through my array list." So let's just compare and contrast them. The idea when you think about having a list of values and having an iterator is – first of all, let's create an array list, and I'll show you how to get an iterator from it. So let's say we have an array list of strings. And we'll call this names because it's gonna just store a bunch of names for us, and so we say new array list. It's an array list of string. We're calling a constructor so we have the prints. Then what I say is, "Hey, array list. I want some way to be able to go through all your elements in a very easy way in an abstract way because I don't wanna have this grungy syntax of asking you for your size and having a for loop." That's something that's very specific to an array list. What I really wanna do is kind of have this generalization to say, "Hey, you contain some set of values, and I just wanna go through each one of your values one by one." So the way I do that is I create an iterator. An iterator has a templated type because if I have an array list of strings and I wanna go through all the values one at a time, the values are all strings, which means I need to have an iterator that iterates over strings. So the types are generally the same. If I have an array list that I'm gonna iterate over, its iterator is always gonna be the same type as the type of the array list.

And I'll just call this it. And it, which is – oftentimes you'll see iterators often called I or it just for shorthand – is named dot, and now it's time for you to see a new method of the array list, iterator. So what this method does is it says, "Hey, array list. Give me an iterator over your values." And what you get is something whose type is iterator of strings. Now how do you actually use that? Here's how you use it. So what you're gonna do is you're gonna have a while loop. And the way an iterator works is you can basically ask the iterator two questions. You can say, "Hey, iterator. Are there any values left in you? And if there are, give me the next values." So the way you ask to see if there's any

values left in the iterator is you say, “Hey, it. Do you have a next value?” so has next. That returns to you a Boolean if it has a next value or not. And then the way you get the next value interestingly enough is you just say, “Hey, iterator. Give me the next value,” which returns to you whatever your type your iterator’s templated type is. So if you have an array list of strings, and you have an iterator over strings, when you call it dot next, you will get a single string. So if our array list over here – let’s just say names – contain the names Bob, Alice, and a C name, Cal. Not very popular, maybe it’ll come into vogue someday. Name your child – I was thinking, yeah, I should name my child Cal Sahami. That would be so wrong on so many levels, but that’s not important right now, besides the fact that my wife would probably beat me to death.

But what this really gives you – names is this array list is when I create the iterator – you can think of the iterator sort of sitting at the first value. And I say, “Hey, do you have a next value?” and it says, “Yeah, I do.” It doesn’t give it to you. I just says, “Yeah, I have a next value.” “Oh, give me your next value.” So it gives you back Bob as a string. Bob still is part of your array list. It doesn’t go away, but now your iterator sort of automatically moves itself to the next element. So when you say, “Do you have a next element?” “Yeah.” “Give it to me.” You get Alice. And it moves itself down, and then yes for the third value, and you get Cal. And after you get Cal, you say, “Hey, iterator. Do you have a next value?” So after it’s given you Cal, it’s sort of gone off the end, and it says, “I don’t have a next value.” And that’s why you always wanna check if it has a next value before you ask for it because if you ask for a next value when it does a next value, that’s bad times. And if you really wanna know what that does, just try it in your code, and you can find out. It’s always good to try to try some of the error conditions yourself, but this is the simple idea.

Notice there’s no notion of asking the array list for its size, or anything like that, or needing to know that the array list is actually ordered. And the important thing there is because there are sometimes – there are some lists like array lists which are ordered. We know that Bob is element zero, Alice is element one, Cal is element two. But we just talked about our friend the hashmap, and now you’re probably thinking, “Yeah, Mehran. Why were you talking about all this array list stuff when you’re talking about the hashmap?” Because the hashmap didn’t have any ordering over its keys. And so one of the things we can do is we can ask the hashmap, “Hey, what I want is an iterator over your keys.” So let’s see an example of that. So in terms of thinking about the hashmap – and we could’ve done the same thing with a for loop in the case of an array list. It wouldn’t have been very exciting, but I would completely trust that you could probably do that, or maybe you already did for your hangman assignment. Okay?

When we think about a hashmap, a hashmap by itself doesn’t – is not a collection, so it doesn’t provide you with an iterator directly. And the reason why it doesn’t provide you with an iterator directly is because if you were to say, “Hey, hashmap. Give me an iterator,” it would say, “But I store key value pairs. I can’t give you an iterator over key value pairs because an iterator is only defined by one type. But what I can give you is I can give you a set of my keys, and that set of keys is a collection, so you can ask it for an iterator.” So the way that works is if I have some hashmap – let’s say I have my phone

book hashmap, I can say, “Oh phone book dot” – and the method is called keyset. Lower case K – it’s sometimes hard to draw a lower case K, upper case S, so what that gives you back is a set of just the keys, so just the strings of the names in the phone book. And then you can ask that keyset dot – so this would all be on one line. “Hey, keyset. You’re a collection. Give me an iterator.” So you would say iterator. And what you’re gonna get back is you’re gonna fum phone book. “Phone book,” you’re gonna say, “What’s your keyset?” and you get a set of keys. What’s the type for the keys of phone book? Not a rhetorical question. Strings. So if I say, “You’re a string set. String set, give me an iterator,” what you get back is an iterator over string, so I can assign that over here by having iterator of strings, and I’ll just call this I equals phone book dot keyset. I need to have these parens in here because I’m actually making a method call. This looks a little bit funky, but the way to think about it is phone book dot keyset is returning to you a object which is a set of keys, and then to that object, you’re sending it the iterator message which is giving you back this iterator of strings. Okay? So once I get back this keyset from the hashmap and get its iterator, this exact same code over here works on that same iterator. So I could use this code on an iterator that was generated from an array list, or I could share exactly that same code from an iterator that comes from the keys of my phone book. So if I had my phone book that had Mehran and Jenny in it, what I would get is an iterator that’s going to iterate over Mehran and Jenny.

It has no notion of the actual values that are associated with those keys because the iterator is just over the set of keys, not the corresponding values. Any questions about that? The other thing that’s funky, just to point one thing out, is in an array list, an array list is ordered, so I’m guaranteed to always get my set of keys from the iterator in the same order as they’re stored in the array list. So I’ll always get as the first key whatever’s at index zero. The second key will be whatever’s at index one, and so forth. Hashmap has no ordering. Even though I entered Mehran first and then I entered Jenny second, there is no actually no reason why Jenny couldn’t show up as the first value. The only thing I’m guaranteed is that every value will show up once, but I get no guarantee over the ordering. So no order, and this one you always get order, just something to keep in mind. Was there a question? Or was that the question? Yeah, just for thinking of it, one step ahead of me. So one other piece of syntax that’s a little bit interesting to see that you should also see – so any questions about the notion of a keyset or an iterator? If you’re feeling okay with iterators, nod your head. If you’re not feeling okay with iterators, shake your head. All right, question back there?

**Student:** Does keyset return an array or an array list? What does keyset return?

**Instructor (Mehran Sahami):** It actually returns a set, and as far as you need to worry about it for this class, you don’t need to worry about a set, so the only time you’ll actually use the keyset syntax is with an iterator to get its iterator. But it actually is returning an object that is a set, or it actually implements the set interface. Okay? All righty. So one last thing to see, and then I’ll show you all of this kind of stuck together in some code. Java actually gives you – because iterators have become such a common thing to do, like people were writing this code and left right – they were writing this code until the cows come home. And they were like, “Ah, but it’s such a pain. I gotta create the iterator, and

then I go through all the values, and that's such a common thing to do that I wanna go through the values, wouldn't it be nice if there were some simpler way of doing this?"

And enough people sort of yelled and screamed this over the years that the people who created Java 5.0 said, "Okay. We're actually going to give you a version of the for loop which is totally different than what you've seen so far." So now you're old enough to see the new improved for loop. And the way this new, improved for loop looks like is you say for, you specify the type of the thing that you're going to get as every element of the loop – so in this case you would say something like string name. Then you give a colon, and then you specify over here the collection over which you would like an iterator. What does that mean? That means the collection that you would like an iterator over is whatever thing you are calling the iterator method on. You do not actually call the iterator method. You just specify the thing that you would've called the iterator method on. So over here what we would actually have is phone book dot keyset. Phone book dot – this would all be on the same line – keyset. Okay? And then inside this loop, what you can do is this word, this name of this variable is a declaration of a variable that sequentially will go through all of the elements of this collection over here. So this collection could be the keyset from a phone book. It could actually be an array list because over here we were asking the array list directly for an iterator. So we could say for every string and name colon names, and that means give me the name one at a time out of names, and inside here we could do something like print lin name. So name is just a variable. The way you can think of name is name is getting the value one at a time of an iterator over whatever you specified here. But the funky thing about this is you never created the iterator. You never said, "Hey, I need an iterator." You just said, "Yeah. This thing? You can get an iterator from that. And I wanna get things one at a time from this iterator, and the name I'm gonna specify for that thing getting one at a time is this variable over here, and it's gonna have this type."

So this syntax, Java automatically what it will do is construct for you basically underneath the hood – you never need to worry about the syntax stuff for that – it will create an iterator for you. It will automatically check to see if the iterator has a next value. If it does, it will assign the next value to name, and then execute the body of your loop. And after it executes the body of your loop, it comes back up here, checks to see if iterator has a next value. If it doesn't, then it immediately leave the loop. If it does, it gets the next value, sticks it into this variable – so it clobbers the old value, but it sticks it into this variable, and then you can do whatever you want. So this will actually write out the whole list of keys from the phone book. And this only exists in Java 5.0 or later. And sometimes people refer to this as the for each construct, but the name's not as important as just understanding the syntax. And if you try to do it with something from Java in an earlier version than 5.0, it just doesn't work. So any questions about that? So let me show you some code that kinda puts this all together, a little code. So here's a little hashmap example. What I'm gonna do, and I put in a little bit of print lins here, just so you could see – like remember we talked about our little lecture on debugging? Where we said, "Hey, you know what? Before you call a function, just add a print lin. You know you got to that function. That's all you need to do." Then you know before you're gonna call this function, I'm about to call that function. It's the simplest form of debugging if you didn't

have the full power of Eclipse available to you. So what we're gonna do is we're gonna read in phone numbers by reading in a list of phone numbers. What are we gonna read them into because we're not passing a parameter here? We must have some instance variable. So down here I have an instance variable that is – oh, and I forgot private in front of it. My bad. So it's a private instance variable.

It's a map from strings to integers because this is our friend the phone book that we just talked about, and I'm gonna create for this map – notice I'm just saying map here. The actual implementation I'm gonna use is a hashmap from strings to integers. So it's the syntax you saw before, before I erased it. And then what we're gonna do is to read in the phone numbers is we're just gonna ask the user, "Give me a name." So we're gonna have a while true loop. Give me a name. We read in that name. If the name is equal to empty string, then I know the user's done entering names. And if they gave me a valid name, then I say, "Hey, give me a phone number as an integer." So I get the phone number's integer, and then I will add that name number pair to the phone book. Notice because this is an int, there's boxing going on here. This number's automatically getting boxed up for you to go from being an int to an integer so I can stick in a string and an integer into the phone book. Again, that only works in Java 5.0 or later, but that's what you should be using. So after I read in all these numbers from the user, I'm gonna allow you to look up numbers. And the way I'm gonna look up numbers is I'm gonna say – I'm gonna keep looking up numbers until you tell me you're done. So I'm gonna ask for a name to look up. If you're a done, you'll give me a name that equals the empty string to indicate that you're done looking for names. Otherwise, what I'll do is I'm gonna ask the phone book to get that name. What it's gonna give me is it's gonna go look up the name. It's gonna go look up Jenny in the phone book and say, "Hey, Jenny. Are you there?" And if Jenny's there, what I get back is the value associated with Jenny, 8675309, as this integer. I don't get back an int. I get back an integer. And this is critical.

I shouldn't declare this as an int here, and the reason I shouldn't declare this as an int is because there's a chance that number could come back null. So if I say, "Hey, phone book. Give me the number for Bob," and it's like there's no Bob in the phone book, so here's null. I can't treat that null like an integer. I can't box and unbox the null, so I need to specifically check for that, and then I'll say that name is not in the phone book. Otherwise, what I'll do is I'll print it out. When I print it out, then it does the unboxing for me and prints out the integer. So that's look up number. Last but not least, I wanna display all the phone numbers at the end that are in the book, so what I'm gonna do, I can actually show you two versions of that. One version uses our happy friend the iterator. It says, "Oh, phone book. Give me your keyset, and from the keyset give me an iterator, and I will assign that to it which is an iterator of type string." And then I'm gonna do the loop that you just saw before. While the iterator has a next value, I'm gonna get its next value which is a name. All the iterator is just iterating over all the keys which are names in the phone book. To get the associated number, I need to say, "Phone book, get the number associated with that name, and then I print it out." Note here I don't check for an error condition because I know all of the names that I have are valid names in the phone book because I got them from the phone book, so I know when I go look them up, they'll always be there. And the alternative way I could've done this is using this for each

syntax, which is just so lovely, right? I don't create an iterator. I just say for every name that's in my phone book keyset, get me the number by looking up that name in the phone book. And then I just print it out.

So just so you believe that this actually works, and then we'll go in our final minute together, so we put in Mehran, 7236059, I won't give you my home number ever. Just kidding. I did one quarter. That was a mistake. Jenny, 8675309 – I won't ask you to give me a number because it will be [inaudible] like random people will call you and be like, "Is this your real number?" Because it turned out when that song came up about Jenny, a lot of people called that number, and there were people who got really angry because some people really have that number. So we're done entering numbers, so we're done entering names, and we wanna look up someone. I wanna look up Mehran. Mehran's not in the phone book? Case sensitive. I need capital Mehran. So Mehran is in the phone book. I'm done looking folks up. And then it displays the phone numbers. Notice it displays Jenny first even though I entered Jenny second because a hashmap has no guarantee of ordering over the iterator, important thing to remember. So any questions about anything you've seen? All righty. Then I will see you on Friday.

[End of Audio]

Duration: 50 minutes

## Programming Methodology-Lecture20

**Instructor (Mehran Sahami):** All right. Let's go ahead and get started. A couple quick announcements. [Audio cuts in and out]

Sorry. A little technical difficulty. So when last we left off – hopefully you can hear me – the idea is to create the coolest thing you possibly can while using graphics, okay? So there's two categories that we're actually going to judge these on. One is esthetics, which is just the nicest looking thing. It can still involve interaction, and probably should involve interaction or animation, but just the coolest looking thing. Then the coolest thing allorhythmically, depending on what kind of stuff you put in there. Is it a game? It doesn't have to be a game. Whatever it might be.

So to give you an appropriate incentive for this, what we're going to do is when you turn them in, Ben and I are going to look through them. We're going to take a first pass and take a sub-set of them that we think are the best. Then we'll have the section leaders in the class collectively vote as to what the winners are in both of the categories. You get, at most, one entry. Your entry, you don't have to designate for a category. We'll look at every entry as being entered in both categories. You just get one entry.

If you want to focus on a particular category, you can, but we'll look at both of them. Then we'll have the section leaders vote and see who the winners are in the two categories. Just to make your life a little bit more interesting, your prize, if you're a winner, is 100 percent on whatever thing in the class ends up being your lowest score. It could be the midterm, could be one of the programs, or it could be the final exam. Whatever ends up dragging your score down the most.

So at this point, if you're a computer scientist, which means you're thinking ahead, you could think to yourself and say, hey, if I know I win the contest, because you're going to announce the contest winners on the last day of class, and I know my lowest score is going to get replaced by 100 percent, were I to not take the final exam, would that not be the lowest score in the class and would get replaced by 100 percent? Yes, in fact, that would be the case, which means if you win the contest – if you choose. If you still want to take the final, that's cool, and if it turns out that something else is lower, you can drop that lower thing just as an added bonus if you want, but you don't have to take the final exam. You just get 100 percent on it if you want.

Now, for those of you who are thinking, hey, doesn't that blow out the curve and cause all these other bad things to happen, no, as a matter of fact, it's goodness for you, too. The reason why it's good for you is whoever wins the contest, we're not going to count whatever that score was that we replaced with 100 percent as part of the curve. So it's a very tiny thing. In a class of 300 people, it really doesn't make a difference, but even for you, the person who won the contest, probably was going to do pretty well on the final anyway or the midterm or whatever. So taking them out of the curve is actually a little bonus to you as well.

The other thing, just to sort of say, if you enter and you're not one of the winners, we don't want to say, all your work was for nothing, immediately. What we're going to do is say, of all the people who entered the contest who have a serious entry – a serious entry means you really put some reasonable amount of effort into it. You didn't just say, here's my hangman program. I'm just going to turn that in as a contest entry. We want to see some real effort. But for all entries that show some real effort into the contest, what you'll get is we'll put you into a random drawing, and on that last day, when we announce the winners in the two categories, we'll also do a random drawing of everyone who actually entered the contest for one additional winner. That randomly drawn person will also get 100 percent on whatever their lowest score was in the class.

So just a little incentive for you. If you don't have time, you're like, oh, I'm so swamped with all this other stuff. That's fine. You can take the little slip of paper about the contest, put it in the recycle bin and just put it out of your head. It won't affect your grade at all. But if you want to go for it, you're welcome to go for it. So any questions about that?

All right. So a couple other announcements. One thing that was kind of interesting was I read all the evaluations, so the mid-quarter evaluations. Hopefully you already did for [inaudible]. If you haven't you should go do them online. But I actually do read them all, and I'm very serious about trying to improve things. So I was looking through, and there were some that were kind of interesting that I wanted to share with you.

One of them was, when I review the lectures online, I noticed you say, Do, do, do, a lot. I had no idea I did that, but do, do, do, now I do. Better throwing accuracy was a pretty common theme. There were some things that I'm not sure how to deal with, so I'll give you these two comments in a row.

He could go a little slower, which I can fully appreciate, and then, perhaps he could go a little faster. I look at how many comments I get on each side to try to weigh that, but there's a lot of comment of that form. There was one that I saw that made me go, hmm. It was, I have never been to lecture. I was like, thanks for sharing, and you probably won't actually ever see this because you don't go to lecture. One person asked – actually, a couple people, I was amazed they asked for more baby or toddler pictures. That one's easier fixed. When I get the overhead camera. There's life in the Sohami household. Notice the book he's reading.

After a little while, you kind of go through the book, and when he gets later in the book, he realizes this whole time, he hasn't actually been using JAVA 5.0. He gets a little upset, but that's life in the city. There you go. There's more toddler pictures. Then the last one, which I thought was interesting, out of all 300 that came in, but I saw Professor Sohami – you can call me Maron. It's fine –at [inaudible] buying a hard drive yesterday. I wanted to say hi, but I forgot how to pronounce his name. I was so embarrassed. It's Maron, and thanks for sharing. All right.

So with that said – and then there were the other comments that I actually pay attention to, not that I don't pay attention to these. If you saw me buying a hard drive, hey, yo also

works. I'm happy to chat with you about the latest hard drive configurations. Anyway, the topic for today's class is our friend, the GUI. And you look at that, and you're like, GUI, what's that all about?

This is what we refer to as a Graphical User Interface. But the way it's usually pronounced, graphical – graphica. It's sort of more avant-garde that way. At times, you'll hear people say this as a GUI. Like, oh, it's GUI. It's like taffy. They're just pronouncing the letters, GUI.

The basic idea in a GUI, right – and actually, if you think about it so far, you've been doing some stuff that involved the graphical interface before where you might've had a program, and you did something that involved mouse clicks. So when the user clicked on the mouse or they moved the mouse, like in breakdown, you were essentially creating an interface for them that was graphical. But the notion of graphical interfaces that most people are familiar with when they talk about GUIs are these things that you interact with on the screen. Things like buttons that you may press or what we refer to as sliders, which are little things that look kind of like this, and you move this button back and forth along some scale from high to low. Check boxes, a lot of times, you see on forums on the web. It's just a little box. Sometimes you can put a check mark in it, or you can click it again, and it takes the check mark.

Something referred to as radio buttons, and this should hopefully be familiar because this was on your midterm exam, except we had you do this very stylized version of it with the graphics library. Radio buttons are just basically where you have a list of choices with buttons, and when you pick one, it becomes selected. When you pick another one, this one becomes unselected, and the other one becomes selected. So if you want to give someone a short-list of options.

There are some other things that come up. For example, something that's called a combo box. This also goes by other names. Sometimes people call it a drop-down box or a chooser. It's one of those things that kind of looks like this. Sometimes you see a little triangle next to it, and it might have some value in it like blue. You click on the triangle, and you kind of get this thing that drops down, and you get other choices like black or red or green or whatever the case may be. That's called a combo box or a drop-down box because when you click on it, this thing kind of drops down.

Or just our friend the text box. A text box is just where there's some form you fill out. You can enter some text in it. Okay? These are, collectively, what we refer to as interactors because what they are, are things that allow the user to interact with your application and provide some information to your application interactively, by clicking on buttons, moving around sliders, setting check boxes or selecting from radio buttons or picking some option in the combo box. That's all they are, so we refer to them collectively as interactors.

Now, the interesting thing is, how do we actually use these in the context of a JAVA program to allow someone to interact with our program more than they've done before?

So far, people have been able to enter text in a console window, or they've been able to move a mouse or click. We want to be able to do something more. So in order to do something more, we need to have the use of some libraries. The libraries that you need to use for interactions, besides the standard ACM libraries that you use, like `ACM.program.star`, if you want to input two JAVA libraries. One is called `JAVA.awt.event.star`. These are all in the book, but you can write them down if you want.

You also want to import something that looks a little funkier. It's JAVA X, so just keep that in mind. It's not JAVA. It's `JAVAX.swing.star`. And collectively, what these do is these help you keep track of events when the user's interacting with something. Before, when you had mouse events when the user clicked, and JAVA X swing is actually a package that's part of the standard JAVA libraries that has a bunch of stuff that allows you to create these graphical interactors very easily on the screen.

So those are the libraries that you're going to have. So to give you an idea of what's actually going to be going on in your program, if we can go to the slides, let me show you how the program or the interactor hierarchy looks. So in the JAVA world – this is in `JAVAX.swing`. All interactors, in some sense, at the end of the day, are what we refer to as J-components. J-component is kind of the basic, generic thing that all interactors are. It's kind of like when you think about the graphics library. All the elements that are in the graphics library like a G-oval or a G-rect or a G-label, are all G-objects at the end of the day.

Same kind of way to think about it with interactors. At the end of the day, all the interactors are J-components. Notice this is a J instead of a G because we refer to these as – they're really JAVA objects, in some sense, so they all start with a J. That's just a naming scheme in JAVA. Then there's various kinds of things we can have like a J-button is a J-component. It's just going to be a particular button that we're going to display on the screen. There's different kinds of buttons. There's your standard button.

There's other buttons like a toggle button, which we're not going to talk about because no one ever uses toggle buttons in their raw form. But, for example, a check box and a radio button, if you think about them, in some sense, are ways of doing some kind of selection like a button. A button, you just click one thing. A check box, you set some box to either be checked or unchecked, and a radio button, you pick one of many options. So in some sense, it's just how much flexibility you have with them, but at the end of the day, you're doing some interaction with something that involves clicking somewhere to turn something on or off, basically.

There's a slider, which we talked about, a little slider thing that basically is some spectrum that you can move some controller on. There's something called a J-label, not to be confused with a G-label, but a J-label is very similar. It's just a piece of text that you can put next to some of these other components to label what it is. It doesn't actually do anything other than just sitting there, being a pretty label.

Combo box that we talked about, and a text field, which is basically like a text box. Now, there's a few other things that are in here that kind of come up like entfield and doublefield. We won't be talking about those. Most of the things in here, we're actually going to cover today, and it's just important to see how they're related. They're all components at the end of the day, much the same way that when we did graphics, all the individual graphical kinds of objects were G-objects.

Now, with these interactors, how do we actually put them on the screen? We don't just put them anywhere on the screen. They actually have a special way that they get laid out on the screen. It turns out, now, again, you're old enough to find out something that we sort of didn't tell you about this whole time, even though it existed this whole time. Now it's time for you to know about it.

Your program window, whether it's a console program or a graphics program, actually has five regions on it, okay? So far, you never used any of the four regions around the side. You always just used the center region, but there was actually five regions labeled, sort of by the points of the compass, north, south, east, west and then the center.

So the way this actually works is the center is where all the action was taking place in your programs before. So when you had a console program, what you really got was a text console that took up the center region, which was basically the whole screen at that time. Anything you wrote out, it got written into the console. On the flip side, if you had a graphics program, what a graphic's program did – remember, we talked about a G-canvas. What it did was it put a G-canvas in the center and sort of made it big enough to take up the whole screen. So that's what was going on this whole time.

You might say, but Maron, what happened to these other regions around the side? I didn't see any space getting taken up by the regions. It turned out that the other regions are only visible and only take up any space at all if you add interactors to them, which means if they had buttons on them or sliders or combo boxes or whatever, when you put them on, when you're going to put those interactors on, you're going to say which one of these regions, north, south, east, west or center. Most of the time, you won't put them in center because the action will still be going on in the center.

When you put them in one of the regions around the sides, it says, hey, I have some interactor in the southern region. I need to now show the southern region. It will actually take up space on your screen. Any questions about that?

I'll show you an example of this in just a second. So that's the basic idea of window regions and what's actually going on with them. The one thing that is important to keep in mind, just in terms of name, is when we place interactors in one of these regions, we refer to that region as a control bar. So if we put some buttons, let's say, in the southern region, what you'll see when your program runs is you'll actually get sort of this gray bar down at the bottom, and your buttons will show up on it. We refer to that gray bar with the buttons on it as just a control bar. If we want to be specific, the southern control bar. That's just the name for it.

With that said, let's actually create our first interactor. It's time to actually make one of these things and put it to use and see what it all looks like. Then we'll build something super cool and complicated with them. But let's just start with the most basic one right now. A little side point. Computer science career panel next week. Go there, Wednesday, November 14th in Packard room 101. It will show you a wide spectrum of things that you can do in computer science at 5:30.

There will be people there who are working at start-up companies, people there who went into academia, people there who are, for example, doing product management or marketing who all are graduates of Stanford's computer science program. It just shows you the wide breadth of stuff that's going on. So if you have any inkling at all that maybe computer science is for you, check that out. That's just a little side point in the middle of lecture.

So let's create our first interactor. The button. Okay. The way we create this is we're gonna need to have a constructor. We're going to need to create something called a J-button. So the type is J-button. We give it a name. I'll just call it but for our button because we have one. So what I'm going to do is I'm going to create a new one of these things in the standard style I use for creating something new. It's a new J-button, and what I give the button when I create it is I give it a single parameter, which is the name or the string which I want displayed on that button.

So let's say I want this button to have the word, hi, on it. Okay? What that does, it creates a button object that the button is labeled with the word hi. This button does not yet show up on the screen. It doesn't yet do anything. All I've done is create this button object. Now, the next thing I want to think about when I want to do anything involving these interactors, right? After I create one of these, I'm going to put it somewhere. Before I put it somewhere, one thing I'm going to keep in mind is I need to listen for these interactors.

So before, when we talked about the mouse, we added mouse listeners. We talked about keyboard events, we talked about adding keyboards listeners. If I want to do something with interactors, I need to add what's called an action listener. So somewhere in my program, say in my innate function or my innate method or my run method, what I'm going to do is say, add action listeners. Much in the same vein I would say, add mouse listeners. As a matter of fact, I could have both. Most of the times, in programs that are interactive, I probably will have both.

But add action listeners, when I add that, it says, oh, yoo-hoo. I'm going to be listening for things happening, like when someone clicks this button. So I'll show you in just a second, there's a corresponding function that gets called – or method that gets called when some action offense happens, and that's the idea. When you had a mouse, and your mouse moved or you had a mouse click, you got mouse events, and so you created methods that were called automatically for you when those mouse events happened. Like, when the mouse got clicked, one of your methods might've gotten called.

Here it's going to be a special method that's gonna get called every time some particular action happens, and I'll show you that in just a second. So we need to add action listeners at the very beginning of our program, and then somewhere, we're going to create a button. Then we need to add the button to one of our five regions. So the way we do this is we use add, and we're going to use a version of add with two parameters. What we're going to add is the object, so we specify the name of the object. This should look real similar to you from graphics when you added, say, a G-rect to the window. But because these are interactors, we don't just add them to the window. We add them to a particular region. So we might say South, so capital north, south, east and west are all predefined constants for you that let you know what region you're referring to.

So add a button to south says, put this button in the southern region. Now that there's an interactor on the southern region, it will automatically show up. So any questions about the basics?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Ah, good question. How does it know where in the region they show up? Well, what it does is it will automatically manage the layout for you. So what it does is any interactors that are in a region will be centered in that region, and they show up in the order in which they were added to the region. So if you add three things to the region, one, two and three, they will show up in the order, one, two and three, and they will be centered, with respect to the entire bar. I'll show you what that looks like in just a second. Let me do a complete program. Let's write a complete program together on the board, and then we'll see what that code actually does when we run it.

So we're going to have some innate method. A lot of times, in interactive programs, rather than having a run to begin execution, you just have an innate [inaudible]. I'm going to set things up, and then I'm going to wait for interactive events to happen. When they do, other methods of mine will get called. So I'm not actually running something. I'm just kind of initializing the world. So oftentimes, you'll see these programs written with an innate method rather than a run method. That's just the way they are.

So what I'm going to do, is I'm just going to do a shorthand. I'm going to do the construction and the add all in one line so you see that. Add new J-button with the label hi to the south region. Okay? So all I've done is I don't have a variable to keep track of this J-button anymore. I just kind of create it and add it, just like you may have done with some graphical objects when you created them and added them immediately to the canvas. We're doing the same thing here because we don't need some method or variable to keep referring to this button. That button, when it's pressed, is going to invoke some other method for us automatically, as long as we add our action listeners.

So then here, we would say Add action listeners. So what we generally do in our innate method is we create all the interactors that we want to create, and before we're done, we

say, add action listeners, to say, okay. I created everything. Now go ahead and start listening for events on them.

Now, when some action actually gets performed, and you've added action listener, what happens is a method gets called for you called action perform. So this method is public void, and you should always declare it as being public void, and it's called action perform. Lower-case a, upper-case P. Action perform. What this gets is a particular parameter, and the parameter it gets is something called an action event, and I'll just call that E. That's the parameter that it gets automatically. Just like when you had mouse events, when you had your mouse clicked function, for example. Exactly the same idea going on here, except this gets called when, for example, someone clicks one of your buttons.

Now, how do you figure out what actually happened when this interaction was actually performed? Action performed, and important thing to keep in mind, only gets called when buttons get clicked. So later on, when we talk about some other things, I'll show you, when someone clicks a check box or whatever, how we deal with that. But the important thing to think about is action performed only called buttons get clicked. So right now, we only have one button, so you might just say, hey, if I click that one button, this gets called, I know it's that one button. Life is good, right?

Well, sort of. You still want to keep track of which button was actually called because you might have an application that has multiple buttons in it. So how do you do that? What you do is you're going to have a string that we'll call CMD for command, which is going to be, essentially, the command that caused this action to be performed to be called. What does that mean? What it means is this action event, E, we're going to pass it a method called get action command. Get action command returns to you a string that we're just going to assign to this variable command.

So we say, hey, an action was performed. Get for me some string that refers to what action was performed. It says, okay. The way I keep track of this command, what is this referring to, this will have the same name as the name of whatever button was pressed. So if a button has the label Hi on the screen, it shows up with the label Hi, ad if it gets pressed, get action will return Hi. It will essentially return whatever label was on the button that go pressed.

So I can check to see if command was equals Hi because I know that's my button, Hi, because I coded Hi up there. I could've actually made this some constant, a string constant, if I wanted to keep them in mind. If it was equal, then maybe I want to do something, like I want to print some lines of the screen.

Printlin, and so this printlin is still going to print to the center region because that's still where the console is in a console program, and I might write out hello when someone actually clicks the Hi button.

So there is no run method here, right? I just set up the state of the world with a button. I start listening for what's going on with that button, and any time that button gets click and action perform gets called, I ask this event that gets passed in, hey, what command was actually issued? I get that as a string, and then I can check to see what button it actually was. So let me show you a couple examples of this in terms of what that actually looks like when it's run.

Okay. So we can quit out of here. Oh, don't save. It's never fun to save. All right. So here's a program, which is basically the program we just wrote. I called the first button. It extends the console program. It includes the libraries that I care about in addition to the ACM library. Other than the fact I set the font to be larger, I just add this new button, Hi, to the southern region, add action listeners, and I have this event here, or I have this method, action perform, which is basically exactly the code I just wrote up on the board.

So when I run this, what I actually get is – here's first button. I get the southern region, which now shows up as a control bar. It has my single button, Hi, on it, and nothing's going on in my program, right? It just creates the state of the world. It puts the button. It says, now I'm listening. Now, every time I click on the Hi button, it executes the command.

After it goes off the screen, it keeps scrolling. You're like, how small can I make this roll button – the scroll bar over here on the side. It's the things you do at 4:00 a.m. when you get your programs working that just make it that much fun.

So let's look at a slightly more complicated program. Same idea, but something more complicated. This is called Button Press, and all Button Press is doing is it's creating three buttons instead of one button, ad it's adding them in different places. So it's creating a button that says hello in the northern region, creating a button that says CS106A in the southern region, and creating another button that's called Basket Weaving 101, also in the southern region.

After it creates all those buttons, it adds them, right. You always remember to add the buttons. It adds action listeners to listen for those buttons. So what happens if a button gets clicked? Action perform is going to get called. Here, I get the action command, and depending on what the action command was, I know which button was clicked. So if it equals hello, I know the hello button was clicked, and I write out hello there.

If it equals CS106A, I write out CS106A rocks, and if it's basket weaving 101, I write out not so much. So here I run my Button Press program, and notice now, the northern and southern regions now show up. So the regions automatically show up any time there's an interactor there. You don't have to tell the region to show up. It knows to show up if you put an interactor there. If I click hello, I get hello there. 106A rocks, and basket weaving 101, not so much.

You're just like, oh, click a little here, click a little there. A little under the arm. Yeah. There you go. It's ten lines of code, and I can put buttons all over and just click them

wherever I want. So buttons are kind of fun, but we can do some cooler things. Let's actually create an interactive program and bring back – remember Click for Face? Do you remember that from a long time ago where we click the button, and we got a face on the screen?

Does anyone remember? If you remember that, raise your hand. Yeah, click for face. This is supposed up click for face. The basic idea is we're going to allow someone to click for faces on the screen, but they can pick the size of their face using radio buttons, either small, medium or large. They can pick the color of the face, black, blue, green or red. We'll start with black, and they can choose to either show the front of the face or the back of the face by saying, hey show the front or don't show the front.

So let me show you an example of this. So I'm going to draw a medium-sized face, the front of the face, in black. So I click here. Oh, front of the face, medium-sized. Front of the face, medium size. Large face. Large face. Small face. Small face. Red face, red face. You're like, okay, that's fun, but what does the back of the face look like?

It's the back of the head. The guy's bald. There's nothing. It's an oval, but that's the thing we want to keep track of, right? Do we draw a face or draw an oval, depending on if front is clicked or not. So how do we get all these buttons and these check boxes, these radio buttons, this drop-down box or combo box. And then we have clear, which is just a button that erases everything. Then we're like, yay, large face. And we just start all over again.

So how can we create this whole program in the span of one lecture? So here we go. We're going to go do it. All right? So first thing we're going to do, it's speed. It's sort of like name that tune, but it's sort of name that interactor. It's like, how many interactors can you go through in half an hour? It turns out they're actually not that complicated, so we can go through a whole bunch of them.

So you already saw a button. What are some other things that we want to do? Check box, right? So check box, you saw that check box, that's allowed us to either show the front of the face if it was checked or the back of the face if it was not checked. How do we create a check box?

Well, the type here is called J-check box. For a J-check box, what I need is a variable of that type. I'll call it check. The way I create this is I say new J-check box. The parameter I give the check box is the name that I want displayed next to the check box like front, to indicate is that the front of the face. So it will right out front and put a little box after it.

Now other things I can do with the check box, which are kind of cool, I can set its initial state, to either be checked or unchecked. So what I can do is I can say, check dot set selected and give it either a true or false, a [inaudible] to say do you start off checked or not checked.

So if set select it to be true, it starts off checked. If I set it to be false, it starts off unchecked. Then when I create this thing and set its initial state, I need to add it to some region to show up on a control bar. So I can say, hey, add check to the south region. Now I've gotten a check on the south region. So the thing you should be asking is, hey, now you've gotten a check box on the south region. How do you actually know when someone clicks the check box? Do you get this action performed method called for you when someone clicks the box?

Actually, you don't. It's a little bit different, how you actually check for the check box. It turns out that for the check box, you can check its state, whether or not its checked or unchecked at any point in your program. The way you do that is as long as you have a way of referring to the variable check, you can just say check dot is selected. What the will give you back is a boolean. So we might have some boolean T equals check is selected.

The tells you any time you call that, whenever you call it in your program, is this thing checked or not? So if you think about this, in order to be able to refer to check, what that means is when you actually declare this puppy, this thing is probably not a local variable. This thing is probably actually an instance variable.

So what that means is somewhere in your program, when you're doing innate, say this is your innate method over here. And then public void innate. You would say check equals new check box, but really, somewhere down here where you declare your instance variables, you would say something like private J check box check, which means I'm going to have this instance variables. There's something I need to be able to keep track of in between method calls.

So I'm going to declare it as an instance variable. I will create a new one in innate, but this one is actually just setting this instance variable. So somewhere else in my program, I can refer to that instance variable.

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** Yeah, you can actually do the creation there and say J-check box check equals new J-check box and then whatever the name of it is. Oftentimes, you just don't see that, so I'm following that convention here, but sometimes you'll actually see someone do that in a program, yeah.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** The reason, really, for doing it this way is that in innate, you can sort of see how everything's being created. You don't need to look at two different places in the program. There's some things where it actually makes sense to create them when you declare them and other things that you can't create them when you declare them. It doesn't make sense to do that, and it's better to just create them all in one place, basically, is the general thinking.

So when check is selected, this is something that you might check somewhere in your program. For example, when the user clicks the mouse somewhere, right? So when we click somewhere and get a face, at that time, we want to see, are we showing the front of the face or the back of the face? So in our mouse-clicked method, that is getting called automatically when we click the mouse, that's when we're actually checking to see if check is selected.

I'll show you the code in just a little while, but that's an example of how it might actually be done. So instance variable's the important thing to keep in mind there. So besides the check box, we had this radio button for the size being small, medium or large. Radio buttons get a little bit more involved, but not a whole bunch.

So the first thing we're going to do is we're going to create all the individual radio buttons. So I might have small equals new – actually, let me abbreviate these just so it's easier to keep track of them. We'll have SM, which is small, is a new J-radio button, and it's going to have some text associated with it, which is just the word small. So this actually creates a single button, right, the single circle with the word small next to it.

This is not – yeah might say, okay, if it does that, how do I know that it's related to medium and large and all that? We'll get to that. Small, notice I haven't declared a variable out here. Again, a radio button is something that we want to be able to keep track of over time. So small, we would actually declare over here. Let me just increase the amount of space we have for variable declarations by getting rid of innate for right now.

We would declare a small over here by saying private small J-radio – backwards. Private J-radio button small. Okay. So that's going to be another private variable for us. We're going to create it in our innate method somewhere, or perhaps another method that gets called for innate.

We create the small radio button. We can create the medium radio button by doing the same thing, new J-radio button, we'll just put the label, medium, on it, and we can do the same thing for large. New J-radio button, large. So we create the three radio buttons.

Once we create the three radio buttons, we need to say, hey, all these radio buttons are in a group. They're all related to each other because if you click small, medium and large better turn off. If you click medium and small is on, small better turn off. So how do I know that they're all related to each other?

In a program, you may actually have multiple radio button groups, and you don't want them to conflict with each other. So what you do is you create something called a button group. A button group is just another type, and I'll call this size because size is small, medium or large. So our group is going to keep track of size. Equals new button group.

Now, the interesting thing about button groups is I never actually care about referring to the button group after I create it and assign all my buttons to it, that I'll show you in just a

second. So this button group, oftentimes, is actually a local variable. It's not an instance variable because I don't need to keep track of it after I create the original button group.

I need to keep track of the buttons, so medium and large would also be instance variables, just like small, but I don't need to keep track of the group. So what do I do? How do I put all these guys in a group? I say the name of the group size, dot add, and I add all the buttons to it. I would add small, and I would say size dot add and add medium, and I would say size dot add, and I would add large.

That says, hey, now you're group has these three buttons in it. Once I do that, still not done. Oh, let me just slide this over here. The power, the magic, of boards. So once I create my initial group, I want to say one of my radio buttons is selected to begin with. So I pick the one I want selected, like medium, and say set selected to be true.

I only set selected after I've added all my buttons to the group. So the general idea is I create the buttons. I create the group. I add all the buttons to the group, and then I pick which one is selected because it doesn't make sense to select one until I know everything that's actually in the group.

After I do that, the funky thing is I need to add my radio buttons to the control, one of the control bars, but I don't add the whole groups at once, as strange as that may be. I add all the individual buttons. So I need to say add small, south. Add medium, south. You could actually put them in different regions if you want, but then that's going to mess up the user real bad. And large in south.

Usually it's a good idea if they come right after each other, so they're all sequential. Otherwise, if you have some interactor in the middle of them, the user's going to get very confused. So this basically sets up my buttons. I create the buttons. I create the group. I put them in a group. I pick which one out of the group is initially selected, and then I add all the buttons from the group onto the control.

Now, how do I figure out which one of these buttons is actually selected somewhere in my program. It's because I made these things private. Small, medium and large should all be private instance variables. I can refer to them later in the program by saying something like small dot is selected. This gives me back a boulion, which tells me, out of that group of radio buttons, is small the selected one?

So if small is the selected one, the other two are probably going to be false. But I can check for all three. I can say medium is selected and large is selected, same kind of thing. Anywhere in my program, it will just give me back the appropriate state of whichever radio button is selected at that given time. Again, it's not calling the method, that action performed method only gets called when buttons are pressed.

Things like J-radio button is not a standard button, so it doesn't call that method or check box doesn't call that method either. Any questions so far? All right.

So one last thing we need to have before I can show you a program that puts it all together, and the last thing – anybody remember what the last thing that was that we showed in our little sample program for the face? Combo box. Always a good idea to have a combo box.

I'm not actually going to show you the slider. The slider is pretty straight-forward. It's all in the book. It's not worth spending lecture time on it, and the truth is, when was the last time you saw a slider on a web page. You're like, today. Yeah. Yeah, that's for today. That's for tomorrow, and that's for the rest of your life.

Yeah, slider you can read about – oh, the snag. Denied. That was one of the other comments, too. The social candies come too far in front, and I just can't go that far back. There's only – I've been hearing, oh, you should really practice your – a lot of the comments were, you should really practice your throws. I've been doing this for 15 years. It doesn't get any better, okay?

All right. So we want to have the combo box, okay? The combox. I always want to write combox, but it's combo box. They should just rename it the combox because really, you can just reuse these two letters and reduce global warming. So the combo box. How does the combo box work?

We have some variable called the J-combo box. We'll call it pick for our pick for the color. We're going to create a new J-combo box. Once again, combo box is something that we want to keep track of between method calls. So we would not actually declare the variable here. We would refer to the variable, but we would declare it over here in our private instance variables and have private J-combo box pick. Okay?

So we start off by initializing the pick to be an empty combo box. This just creates a new combo box. Then what we need to say is we're going to add all the elements that we want in that combo box in the order in which they're going to appear in the drop-down menu of the combo box. We do that by saying pick dot add item.

Then the item you give it is a string. We would have black for the first item, and then the next item pick add item is blue. I just put these all in alphabetical order, but you can put them in whatever order you want. Green, and then pick add item red. Those are all of the choices that are in the combo box. The reason why it's called a combo box, interestingly enough. You might say, but isn't it just a choice box or drop-down box? Why is it called a combo box?

The reason why it's called a combo box is in the days of yore when these things were first created, they not only let you pick from a choice of items, but you could actually type in a value if it allowed you to, and just make up your own value that wasn't in the list. But oftentimes, we don't want people making up their own value. If we said, hey, from this combo box, pick your year.

You could pick freshman, junior – we don't have sophomores. Freshmen, sophomores, juniors, seniors, grad students or other. If we let you just type something in, you're like, well, I'm supersenior. First of all, I'd be like, sorry, but second of all, I'd be like supersenior doesn't exist. You're still just a senior. You could be super senior, but you're not supersenior, all one word.

So what we want to do is tell this J-combo box, hey, we're not going to allow the user to type anything in. We're just going to allow them to pick one of the choices. The way we do that is there's a property called set editable, that we're going to set to false. So we say set editable false. If we set editable to true, you will get a combo box that allows someone to pick any one of the choices or just click on it and type something in, if you really want to do that.

But if you don't know how to handle what they type in, don't let them do it, and set editable to false. The other thing that we need to do is we need to pick an initial value for that combo box. All right. So you create a combo box. Which one of the choices is initially picked? We do that just like you've seen for all the other interactors. Pick dot set selected. Except here we say selected item, for some reason. With combo box, we just use the word item when we add and set selected item because they're just items. That's the way life is.

Item, and then we give it the string of whichever one is selected, like black will make our initial choice. After we do all this, we create the combo box. We add all the choices to it. We choose whether or not it's editable, and we pick the initial selection. Always remember to say, hey, now I need to add it to one of my control areas on the screen.

So I say add pick to, for example, the south controller. Okay. Question?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** I think the default, that's actually – offhand, I don't know if the default is false or true. I think it might actually be true. But just for finding something that I don't know off the top of my head, there's a candy. And for asking a question. You can just try it out yourself. So don't set it and see what happens. Offhand, I don't remember. We could actually just play with it in the program.

So here's how all those things actually get put together in our program for drawing this interactive face. So first of all, what I'm going to do is in my initialization method, I'm going to create the buttons that I care about.

I only had one button, which was the clear button. So I have one button, clear, that's in the southern region. What do I do when I get the clear button? Remember, buttons are the only thing that the action performed method is called for you automatically when they're clicked. So down here, I have my action performed method. It's pretty similar to what you saw before with the hi button except this one gets an action event. It gets the action command. So here, I didn't take this and assign it to a separate string.

I'm just saying, hey, E, get your action command. I know that's going to be a string, so just directly call the equals method on it. Oftentimes, you'll see the shorthand when there's only one thing that we want to check for instead of a bunch. We don't get the string separately.

So E action command equals clear, which means they click the clear button. If that's the case, then I remove all. Remove all is one of the methods of a graphics program that removes everything that's on the canvas, so it clears the canvas. So that's how my button gets set up. Then I have a check box for whether or not I'm displaying the front or the back of the face. So what do I do? I create a new check box that's name is front. Notice I need to keep track of the state of the check box, right, because I need to know later on, was it checked or not.

So check box is actually down here as an instance variable. We use our instance variables to keep track of the interactors whose state we need to refer to as the program runs. We need to say, hey, check box, are you checked or not, somewhere later in the program. So we need to be able to refer to check box.

My radio buttons are all also instance variables, and my combo box pick, just like you saw up there, right? Radio buttons, combo box, and the check box were all instance variables. Same thing going on here.

So check box, I create it. I set it selected to be true. So when I first start off, I'm going to be drawing faces, and I add the check box to the southern region. Now there were two other things that I had. I had a radio button for small, medium and large, and I had a color chooser, which was a combo box. [Inaudible] to take all of that code and abstract it a little bit and modularize it, I've just created two methods that I'm going to call to do that initialization for me.

So innate radio button. What that does is it does the code you just saw. It comes here, and it says, hey, create a button group. Create three buttons for small, medium and large, and add all the buttons to the button group. Same code you saw. Actually, you could've created the button group. You could've taken this line of code and done it after you created the buttons. It doesn't make a difference, but here, I just created the group, created three buttons, then added all my buttons to my button group here.

Once all my buttons are in there, I say, hey, medium is the one that starts off selected, and then add all of my radio buttons to the southern controller. Any questions about that? It's exactly the same code we just wrote on the board, except just with nicer variable names. Uh huh?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Right. They all get drawn from left to right, but the entire collection of stuff is centered in the screen. So you'll see that when we run the program. It's not going to happen.

So last but not least, I initialize my color chooser, and my color chooser is just a combo box. So I create the combo box. Again, pick color is an instance variable. I add my items to it, black, blue, green and red. I set its editability to be false, so I don't allow the user to type in. I set the selected item, initially, to be black.

Now here's something funky that I'm going to do. I'm saying, hey, here's my combo box. My combo box doesn't have a name associated with it. When this thing actually gets drawn on the screen, what I get is a little drop-down box that looks like – actually, I just get a box that starts with black in it. If I click on this little thingy over here, I get blue and red and green and the other choices.

What I would like to have, is I would like to have the word color, if this chalk actually writes, appear before the combo box so the person knows, yeah, that's what this black is referring to. It's referring to the color of the face. So the way I do that, is I'm just going to create a J-label. I don't worry about assigning it to anything because the J-label doesn't do any interaction. It just puts a label in the control bar, so I create a new J-label.

I add some spaces here to separate it a little bit more from whatever the last control was. So whatever interactor I put in before, I say, hey, before you draw the label, put in some space, then draw the label. So I add that label to the southern region. After the label, I add the combo box.

So what will show up in whatever interactors I had up to this point, then a little bit of space in the label color, and then my combo box after it, okay?

Last but not least, I say, hey, add mouse listeners, and add action listeners. So I set everything up in my program, and I'm done. Now the only thing my program will do is do something when the user interacts with it. So when the user interacts with it, there's two things they can do. They can either action perform, clicking the clear button, which you already saw, or they can click the mouse somewhere on the canvas.

If they click the mouse somewhere on the canvas, we call the mouse clicked method, just like you saw before, clicking the mouse before. Here's where all the fun goes on with checking the state of the check boxes and all that. We say, hey, I'm going to have some generic objects, some G-object.

The diameter of it is going to be whatever size should be appropriately set. So I'm going to call some method set get diameter size. Diameter size comes along and says, hey, I'm going to start off by setting size to be zero, and then I'm going to check which radio button is selected. Hey, small, are you selected? If you're selected, then the size of the small diameter. If you're not selected, then I'll check medium. If medium's selected, then the diameter's the medium size. If it's not, I'll check for large, and I'll set the large size.

Small, medium or large are just constants that are set in my program to be 20, 40 and 60. So all this does is check to see which of the three radio buttons is actually selected and

returns an appropriate value for the corresponding sides. That's what we set for the diameter of the object.

Then we come here, and we say, hey, check box, are you selected? If you are selected, the I want to draw a face. So my object is going to be a new G-face. You're just using that G-face class I wrote two weeks ago. If the check box is not selected, your object's going to be a G-oval because the back of the face is just the circle.

Then lastly, I need to set the color. I'm going to set this object's color, just using the set color method that hopefully, by now, you're very comfortable with, to be whatever color the current color is picked in the combo box. So I write a method called get current color that returns to me a color object.

It goes through, and it says – here's how it works. Hey, color-picker, what's your selected item? So it returns a selected item. I need to cast that to a string. That's just a little JAVA-esque thing. This is all in the book. You need to cast whatever the selected item is to a string, and then based on the string, I say, hey, is the string blue? If it is, I'll return the color blue. If the string is green, I'll return the color green. If it's red, I'll return the color red, and if it's some random thing I don't know or it's black, presumably, I'm just going to return black. So that returns the color object black, which is what I set the object's color to.

Now I've set the size of the object, whether or not it's the front or the back of the face, the color of the object, and now I'm just ready to add that object wherever the mouse was clicked at a [inaudible] location. So that's the whole program. When I run, here's how I get the layout. I added the button first. Then I added the check box. Then I added the radio buttons in this border. Then I said, hey, a little bit of space, and the label color, and then here's my combo box for this lecture.

Select green, set this to be large, I have front. I get the green face. Any questions about interactors? All righty. If that's making sense, nod your head. If it's not making sense, shake your head. All right. Have a good weekend. I will see you on Monday.

[End of Audio]

Duration: 51 minutes

## Programming Methodology-Lecture21

**Instructor (Mehran Sahami):** So welcome back to the beginning of week eight. We're getting down to the end. Well, we've got a few more weeks to go. It feels like we're getting down to the end, though. It's one of those inverse relation things for the amount of work. The amount of work you do as the quarter sort of reaches the end goes up. The amount of work I do goes down, which I shouldn't laugh about, but that's just the way it is.

There's two handouts today. One handout's your section handout for this week. Sections are still happening this week. The other handout is some code on interactors that you saw last time as well as some additional code that we'll go over today on interactors. Other announcement, computer science career panel. So just a quick show of hands, how many people are thinking about computers science as a major?

A few folks. How many people thought about computer science as a major prior to taking this class? And now they're not thinking about doing computer science as a major? All right. Sorry. How many people were not thinking about computer science prior to this class that are now potentially thinking about computer science? A few folks in there, too. That's good to see.

So this computer science career panel, one thing that people make the mistake of doing is to equate computer science with programming, which is not at all the case. So if you're trying to make a decision about whether or not to major in computer science just based on this class, if you like programming, you should do computer science. You'll be a great computer science. No doubt.

If you don't like programming, but you were thinking, hey, computer science might be for me, in fact, many computer science majors don't end up programming after they graduate. But having a computer science degree opens up doors for them to go into the industry, in roles which, otherwise, they may not have been able to go to. So if you want to find out about careers in computer science, and so the folks on this panel include some people who are engineers, people who are at startups, people who are big companies, people who are doing design, people who are doing product management. There's also someone on the panel who went into academia, so not me, but another new professor on the department. He'll also be talking about careers in academia on this panel.

It's Wednesday, November 14, 5:30 p.m. in Packard 101. There will be food, so don't worry about missing dinner because there will be food there. There'll be fun, and there will be computer science. So you just got to make everything start with an F, and there aren't that many C-words.

So with that said, it's time to delve into a continuation of our last great topic. So it's time to continue a bit with our friend, the interactor. If we think about the interactor and action listeners – so last time we talked about having buttons and buttons generated action

events. Remember that? So we're going to do a brief review of that and push it a little bit further.

So one of the things we talked about is having your program say in your innate method, innate, somewhere you might have public, void, innate and inside here, you would set up the parts of your program that you want to actually do something like the various interactors, that when someone clicks on them, something happens. Then you would say, add action listeners. What this would do is basically say, hey, I got some buttons in my program. I want you to be listening for buttons.

So when someone clicks on a button, I want you to call a particular method for me, called action performed, and then based on when you call action performed, I'll figure out what button was clicked and then actually do something. Okay? So over here, we had our friend, public void action performed. And action performed would get, as its parameter, something called an action event. An action event, we'll just refer to it as E, was basically what it would check to see what action was actually taken or basically which button was actually clicked.

So hopefully you remember that. That's just a little bit of review from last time. Now when we've got this action event, and we said there are a couple things you could do with it. There was actually one main thing we talked about that you could do with it, and you could figure out which command was actually the thing that caused this action event to be generated by saying, hey, you know what I want to do? I want to pull out, as a string, and I'll just call it CMD for command, the command or the name of the interactor that caused this action performed method to be called.

So here I would say E dot command equals E dot get action command. And what get action command does, it's just a method of this action event that says, I'll return to you the name of the interactor as a string, and button's names are basically just whatever display's on the button.

So then I can have some ifs in here based on this command. If command dot equals – and I can check for some name that I might want to take some action based on that button. It turns out there's something else you can ask this action event, E, for, other than the action command. You saw this very briefly last time, and the program that we did, and you're going to see it a little bit more now.

So I want to spend a little bit more time on it. It's something where you can say, E, what I want to get from you is not the action command. I want to get the source of the action. Now, the interesting thing about what get source returns to you – actually, let me not put the semicolon here right now – is get source actually returns to you an object.

It returns to you the object that caused this even to be generated, which means if a button was clicked, E dot get action command will get the name of the button. E dot get source will actually give you a reference to the button object. So what you're getting back from this is an object. You're getting a reference to that object.

So what does that mean for you in your everyday life? What that means is over here, when you want to set up your initialization, you could say, I want to create a button. So I'll have some button I want to create. So I'll say new J button, and maybe that button, I want it to say hi on it. Okay? So one thing I can do is I can say hi equals new J button. Hi, what I'm going to do is make that an instance variable. So somewhere down here in my program where I have my I vars, my instance variables, I would have private J button hi.

So I just do the declaration of a variable called hi, which is of type J button, and then in my initialization method, I actually create that button with the label hi on it. Then I go ahead and add it somewhere to one of the control bars in my program. So I would say add hi maybe to the south control bar because we really like adding things to the south control bar. It's just fun when buttons show up on the bottom of our screen.

So we say add it there, and then wait for something to happen. So add my action listener in case this button gets clicked. Now when the button gets clicked over here, what I can do – I could actually ask command to get its name. Or I could ask action event to get the action command name, and then I could say something like if command dot equals and the name of the particular button over there happens to be hi, then there's something I want to do. Maybe I want to print something on the screen or whatever the case may be.

That's one way I could write this, and that's the classic way that you've seen it written before. That's the way you saw it last time. The other way I can write it, with my friend get source, is rather than getting the name of the command and checking to see if the command is equal to hi, I can actually say, Maron told me about this thing called E dot get source. As a matter of fact, I don't even need this line for command anymore. Let me just comment it out so I don't erase him.

I can say if E dot get source – this returns an object to me. I want to check to see if that object that it returns is my hi button. So here, I check directly, is it equal to hi, and then I do whatever I was going to do. So this has exactly the same effect as before. It's checking to see if I've gotten a button that is the hi button that was clicked. So the difference between these two things, if you kind of think about them, one of them is I'm just using a name as a string, and the other ones, I'm using the actual object.

Now, if you think about it more deeply what that means, if I think about the name over here, if I think just in terms of the name, I never need to be able to refer to the actual object. Which means if I don't need to refer to the actual object again over here, I don't necessarily need it as an instance variable. I only need it as an instance variable if I'm going to refer to it again in some place that's in a different method than some other method I may have already used it in.

So let me show you an example of what I mean by that in code to make that more concrete. So if we come over here to code, here's essentially the code I just wrote for basically reacting a button. So it's just the code I wrote on the board, except I made the font bigger. I create a button with the name hi. I put it in the southern region, and I add my action listeners to listen for that button getting clicked.

When the button gets clicked, I say, is the thing that got clicked this button that I created? Here I actually called it hi button instead of just hi over there. I shortened it to hi so it'd take up less board space.

If it's actually the source of that action, is my hi button, then I will print out hello there. So I can go ahead and run this program. If I run this program, this is – I click hi, I get the same thing I saw before. Every time I click hi, I get hello there. Now, alternatively, I could've written this slightly differently, which is the way you saw it last time.

What I can do here is I can say, when I'm going to do the add, just go ahead and create that button and add it, all in one line because I don't need to have some variable that stores the button. Down here, I don't need to check for the source of what that action event was. I'm going to say, action event, give me your command. The command is going to be the name of the button, so I no longer need a variable to actually store a reference to the actual button object because this is going to give me the name whenever I need it.

So as a result, notice here I don't have an instance variable. So this is one of those things that's a tradeoff. It should also give you a little bit more insight into when you have instance variables versus when you don't have instance variables. You need to have the instance variable in the case, where you need to – wrong line. I want the new one. You want the instance variable in the case where you want to be able to refer to this variable in some method that's different than perhaps the method, which got created.

So I created the button over here, and I stored it somewhere, but I need to be able to refer to it in some other method, so it's got to be an instance variable. If I don't need to refer to it in any other method, which is what I saw in the second case, I don't need to refer to it again. As a matter of fact, there's no other place I need to refer to it after I create it. Then I don't need to store it anywhere. Any questions about that?

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** So that's a bug in your logic. The computer shouldn't try to figure out which one because if you give it two buttons with the same name, it says, I have no idea. It's going to cause you problems, so don't do it. If you want to see what happens, go ahead and try. It's a bug in logic, not a bug in what the computer's executing. Any other questions? Uh huh?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It's still going to get the actual button, so you're saying in this other case over here, what is this thing going to return if I didn't create a variable over here? This thing's still going to return some reference to your object. The only issue for you now, though, is you have no way of checking for equality with some object. If you don't have this as an instance variable, you can't check to see if that thing's equal to hi button.

So if you created hi button over here and just immediately added it and never kept track of it over here, this guy would return to you a pointer to hi button, and you'd say great, I got a pointer to hi button. How do you know it's hi button? You don't. You have no way of comparing it to the actual hi button you created. That's why we need to store it.

All right. So why do I show you these two different ways of doing it? The reason why I show you these is because now you're going to actually make use of this with respect to some other interactors that you're actually going to see where we care about viewing get source as opposed to the action command.

So what we're going to do next is we're going to say, you know, a lot of times in programs, what you really want to have is some way of letting the user have some way to specify some text in a program that's running interactively that's not in the console. They'd like to be able to type something in, so let me just show you an example of this.

So here's a little program that's go what we refer to as a text field down here, and I call that name. So if I say, hey, my name's Maron, it says, hello, Maron. I say, no, my name is really Sally. Most of you don't know this. It says, oh, hello, Sally. It's just some way of being able to have some text field over here that the user fills in. This is an interactor, right? This is just one field. It's not on the console. Then do some action, and the action we happen to do here is just to write something to the console that makes use of the text that the user actually typed in.

So how do we get something like that to work? So what we need to do is have an interactor that's called the text field. Basically, text field is just that thing you saw. It's a little place where someone can type some text as an interactor. So it shows up in one of the control bars, and then potentially, when they hit enter, you get some action event that tells you, you need to actually – or if you want, you can do something with this text. So that's the basic idea.

What you really get is a box, and that's all you get with it. If want to add a label to that box, like we added name over here, we need to specify that. I'll show you how to do that in just a second, but what you get is the box. The user types in something and then hits the enter key. Then potentially some event is generated for you. So how is that actually set up?

So the thing we want to create is a J text field. It's just another one of these interactor like you saw before with check boxes and combo boxes and all that stuff. It's just called a J text field. I'll name this one TF to stand for text field. What you do when you create a new one of these is you say, new J text field. What you give it as a parameter, and here's the funky thing.

You don't give it its label. The label doesn't come with the text field. You need to create the label separately. What you give it is the size of that text field, how big it should be in terms of the maximum number of characters that would show up in there. So if you say ten, for example, what you're saying is I want to have some text field that will hold, at

most, ten characters. If you use some font that's variable with it, it automatically gives you the size of 10 Ms because M is the widest character, in case you didn't know. That's just life in the city.

Now the funky think about this, relative to this action performed, is when the user hits enter, if I didn't do anything else, you would not actually get this call to action performed because action performed is only called for you for buttons. So what you need to do is after you actually create this text field, you need to say, you know what, I need to let you know about this text field as something that can generate actions.

So the way you'd do this, because it looks a little bit funky, but you tell the text field dot add action listener this. Okay? You don't need to worry about all the way does that actually mean at a very low level. All you need to know is you're telling this text field, you're going to be able to generate actions now, and the thing that you're going to let people know when you generate some actions is yourself, which is why we passed this.

But anytime you create a text field, and you don't just do this once for all text fields. If you have multiple text fields, you need to send this add action listener this message to each one independently. We only have one here, so we only need to do it once here.

But what this basically does is says, text field, you can now generate these action events as well. So after you create it and you set up this line – and you would want to add it somewhere into your program. So in your program, you would probably say add TF, and we might add TF, for example, in the south because we add everything in the south.

When someone types something in to TF and hits enter, then it will generate some call to action event for you or action performed and pass you an action event. Now once that gets set up, how do you actually figure out what was the text field that generated this event. You could have multiple text fields that someone could've typed into and hit the enter key. What you're going to do is you're going to add E dot get source. So inside here, what you're going to say is if E dot get source is equal to TF. At this point, all kinds of warning bells should go off for you. So maybe inside here, you want to do a println where you want to say hi and then add to it the text that's in that text box.

The way you do that is you just say the name of whatever the text field is and the message you send it is get text. What it will give you back is it will just return to you this thing by itself. It just returns a string of whatever is in that box when the user hits enter. So this will write out hi and then whatever text you typed in, just like you saw in the program. Except that was writing hello, and maybe that's what we want to do.

But the warning bells that should be going off now, what's the problem if I've just written a code like this?

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** It's not an instance variable, right? So I have no way, if this is my innate method over here, of being able to refer to TF again out here. So I need to create the instance variable, right? If this is my innate method, and I have public void innate in front of it. What I need to do is this TF, somewhere else in my class, let's say over here, which is where I declare my I bars. It's just lower down in the class somewhere. I need to actually have private J text field TF.

Then over here, rather than declaring it, I just create the new TF. So I need to set it up as an instance variable, just like you saw in the example with the buttons. The same kind of thing's going on here, except it's a text field.

So let me show you an example of this code in action. So here's a little text field example. What I'm going to do is I'm going to extend the console program. So I'm still going to have a console. In my innate, I'm going to have something called name field. What's name field? It's just a private J text field. It's an instance variable, so I can save off name field. Name field, I initialized over here to be some new J text field of size ten. This is exactly the code you just saw over there.

Now what I also want to do, here's the one extra funkiness in the program. I want to give that box a label. So before I add this box to my control bar, I'm going to add a new J label that just says name. So all J label does, it just says I'm going to create something that's just this name or just this particular piece of text. The text happens to be name, and I'm going to add that to my southern control bar.

So first it's just going to write name out there, and then after I write name, I'm going to add my name field, which is going to create the box after name. Then I'm going to do exactly what I told you where you have to tell name field, you're going to add action listeners of yourself so that if you do anything, you're going to let someone else know that you actually have done some action when the user types into you and hits enter. That means action performed is going to get called for you because now you're going to be able to generate events to an action listener.

In action performed, we check E dot get source. We can compare against name field because we have that saved off down here as an instance variable. We'll just write out hello and then the text associated with name field. Let me increase the text size, here, just so it's a little bit bigger and we can see what's going on.

We'll do Courier 24. Make it bigger.

So here, once again, Maron, hello, Maron. See? It knows it's getting the event when I hit enter. Enter, enter, enter. See, that text just stays there. It's another one of those things. It's only so much fun. Sally, we're scrolling. Not a whole lot of excitement going on here. This is good for about two minutes.

So we can go ahead and do it this way. Now you can get information from a text box. Any questions about the text box? Yeah, in the back?

**Student:** [Inaudible]?

**Instructor (Mehran Sahami):** Yeah, so basically the way the layout works is every time you add things, they just get added sequentially from left to right in whatever region you're adding them to. In this case, the southern region, and the whole set of stuff gets centered. So if you want to space stuff out, what you actually need to do or add, for example, more J labels, they might just have more spaces in them. That will create more spaces between stuff. There's no way I can hit you. I'll just leave it here, and you can pick it up after class.

So there's one other things that we can do that's kind of funky. We can actually name the text field. So you might say, but Maron, this whole get source thing, keeping it on the instance variable, I'm not so keen on that. What I am kind of more keen on is giving things names so I can just refer to them by their name. That's cool. You can have a name.

So here's that exactly same example, slightly differently. What I'm going to do is I'm going to add just one more line here. So this is exactly the same code I had before, except after I create the name field, I say, hey, I'm going to give you an action command, and your action command is going to be name. So whenever you generate these events, yeah, I can check to see if the source of that event is you.

Or, if I've given you a name, I can do the same thing I just did with buttons. Down here, I can get action command. That gives me the string, which is the name of the object that created this event. I can see if it's equal to name, which is the name that I gave it. So this just shows you a little back and forth. With the buttons, I kind of show you, with buttons, you just name them because you always name buttons and you check against names. You could actually check against the source of the button if you wanted to.

J text fields, it's kind of backwards. J text field, you always, in some sense, have the text field that you can get with get source, but if you want to refer to it by name, you have to explicitly give it a name because name doesn't show up as part of it. If we want the label, we still need to add this separate label name over here. This is just naming the particular event that comes from that box. That's all it does. Any questions about that?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** That's the maximum amount it shows.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah, name field is still an I var here. It's really actually no longer necessary because I don't need to refer to it over here. So I wanted to, I could just do this. A little cut and paste. Thanks, I vars, thanks for playing. That's real nice of you. Yeah.

Oh, no, I can't. That's why I had it in here. I still need to refer to it over here to get this text. To be honest, actually, what I could do is I could just call E get source here and get its source and then get its text. So I really don't need to, but it's just better stock because it makes it cleaner that I'm getting the text here. So there is a way around it, but the cleaner way is to actually do it this way.

Let me get rid of that. So any questions about that J text field?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Oh, yeah. You can. I'll show you that in about 20 minutes. And three seats away. Before we get there, it's time for something completely different is to say – it kind of gets to the question in the back of the room. These things are all sort of showing up centered on the bottom on the screen. Could I actually have these interactors laid out a different way than this way that they're getting laid out for me.

In fact, you can, and strangely enough, the thing you use to do that is called a layout. So a layout controls the layout of the particular interactors. It turns out, when you used your friend, the console program, or your friend the graphics program, what you got was a layout that was called the border layout. As a matter of fact, you already saw the border layout. You saw the border layout last time. It looked like this.

You had some center region. You had a north, south, east and west borders, which is why this thing's called the border layout. What happened with this border layout is that the center was where all the action takes place. The console program would add a console to the center automatically. That's just what happens in a console program. And a graphics program would add a G canvas to the center automatically, which is where you're going to draw all your stuff.

The other regions are only visible if you add stuff do them. So in the very early days, when you had a graphics program that was all just graphics, you would say, hey, nothing showed up in the south region. Yeah, because we didn't put any interactors there. So these interactor regions only show up if we actually put interactors on them.

We said these are referred to as control bars. So you saw these last time. How do I consider different kinds of layouts? So there's a couple other layouts to also think about. There's something called a grid layout. The way a grid layout works is you actually create and object called the grid layout, and you specify in that grid layout how many rows and columns are in the grid layout. So we might say two rows and three columns, which means we're going to have a layout that looks something like this.

It's just a grid with two rows and three columns. I'll show you the code for this in just a second so we can get into the nitty gritty details. Conceptually, here's what it is. Now, when I add items, what I do is I say, you know what, I want to set my layout to be this grid layout. What now happens when I add items is it will add items, the items being the interactors, one by one, starting at the topmost row and the leftmost square.

Every time I add a new element, it moves over by one until I get to the end of the row, and then it automatically comes down. It goes sequentially across, row by row. It allows me to sort of lay things out in a grid if I want to actually be able to do things in a grid. So let me show you an example of what a grid layout might look like.

So grid layout, here's a simple program that does it. What we do is we start off in our innate method by saying I want to create a layout. So I want to set the existing layout that the program is going to use to be a new grid layout that's two, three. Two rows by three columns.

Now one thing that's interesting about this program, if you look at grid layout example, it does not extend console program. It does not extend graphics program. These are not its beautiful house and its beautiful wife and children. What have I done? What I've done is said I'm just going to extend the program. I don't want you to create a console for me, and I don't want you to create a G canvas for me.

I want to take up the whole screen with my buttons, baby, so that's what I'm going to do. I'm going to add six new buttons, and these buttons are just going to get sequentially added in the order that you saw. Then I'm going to say add my action listeners. I'm not going to do anything. I'm just going to ignore the buttons.

The reason why I'm doing this is I just want to see some big, fat buttons. Yeah. Look at that. Six buttons that take up the whole screen. It's a grid. My interactors fill up the grid. The layout takes up as much space as possible in the screen. More importantly, each of the interactors that I put into a grid cell takes up as much cells, as much space as possible. So this button comes along.

Oh, yeah, I got so much space. You're like, why does it do this? This is the most brain-damaged thing ever. I don't need a two inch by three-inch button. The reason why is – how did you find that so quickly. We won't talk about it right now. Maybe afterwards. It's like having a sound effects guy.

Check this out as I resize the window. All buttons small and cute, big buttons. That's why we have layout managers because the layout managers just give conceptual – it says, this is how your layout's going to be. It says, I'm going to handle all the dynamics of resizing and all that stuff for you because people resize the window. But I need to know how things are laid out. If you give me more space than I need, I'm just going to take it up.

Grid layout, not so useful, just something to see. If you see it in the book, you know what it's talking about. There's another kind of layout which is called a table layout. There's actually another kind of layout called the flow layout. We're not going to talk about it, but there's something called a table layout.

A table layout is basically just like a grid layout except for the niceties. So you also give it a number of rows and columns, except what it says is rather than having each one of

the interactors fill up itself, the maximum possible size, I'm just going to give that interactor as much space as it needs in that cell and no more.

So what does that mean? That means if I come in here, rather than a grid layout, I say I want to create a new table layout, and I run this – I need to add more imports. Let me just grab imports from over here. Come on table layout. Let me just show you the nicer example of table layout.

Sometimes in life, you just got to get ugly with it. We got ugly with it. Oh, table layout? There's table layout. Six buttons still. We can still resize the window, but the buttons are just given as much size as they would actually need. They don't fill up the whole region that they actually want. So table layout's actually something slightly more useful for us than grid layout.

This question came up before which was, can I actually link buttons and text fields together to create something a little bit more funky? In fact, I can do that, and I'm going to show you that in a context of something a little bit more interesting. It's a program that allows for conversion in temperature. So this one's actually in the book. I didn't give you the code because the code is all in the book. So I didn't give it to you on a separate handout.

Basically, the idea is we write out a label called degrees Fahrenheit, a label called degrees Celsius, and inside here, we can type in some value. If we click Fahrenheit to Celsius, it will automatically fill in the Celsius field with the corresponding value. So 32 is zero Celsius.

The other thing that's kind of funky is I don't necessarily have to click the button. I can type in, say, some value and hit enter, and that's just like clicking the button. Interesting. So how do I create this program? Well, if you think about this program, first thing I'm going to need is these things are not supersized, but they're all laid out in a grid. So I'm going to need a table layout that has two rows and three columns.

The first element that I have here is just a label. Then I'm going to have a field that's a text field. As a matter of fact, I'm going to have a specialized kind of text field. There's two specialized kinds of text fields. Something called an intfield and a double field. They work just like text fields except you're guaranteed to get an integer value or a double value from them.

You might ask what happens if someone types in A and wants to convert A to a temperature? Oh, I clicked the wrong button. I want to convert A to a temperature. It says enter an integer, and it brings up this pop-up box and gets in their face. Then you say, sorry, my bad. So it guarantees you get an integer.

Then I'm going to have a button, and somehow, I want to link the button and the text fields to do the same action. So let me show you the code for that. It's actually a lot shorter than it looks.

First thing I'm going to do is set the layout to be a table layout. Notice once again here, I'm extending a program because I don't want a console or a canvas created for me. I want to be able to specify the whole layout, so I'm just extending a program. I set the layout to be a table layout, 2, 3. Again, we're going to go sequentially through all the elements. So what I want to have in the first element, the first thing I'm going to add to my layout, I don't specify until I'm down here.

The first thing I'm going to add to my layout is degrees Fahrenheit as a label. Then I'm going to add some Fahrenheit field. How did I create that Fahrenheit field? I actually created it up here. What I did, first, was declaring it as an instance variable. So Fahrenheit field is an intfield, not a J text field, but an intfield, which is just a specialization of a J text field to just give you that integer.

Other than that, it works just like a text field except here, I just wanted to show you intfield, so it's an intfield. So I create a new intfield. I specify its initial value, not its initial size. Its initial value is 32. Then what I say is Fahrenheit field, I'm going to set your action command so that when you generate actions, the name associated with the actions that you generate is going to be F dash greater than, which you can just think of arrow, C.

That's going to be your name. So I set its name, and then I say you're going to generate action events, so I'm going to add an action listener to you of yourself. Just like you saw before with the text [inaudible], except now we're dealing with an intfield. We do exactly that same thing with something called the Celsius field. Celsius field is also declared to be an intfield.

It starts off with an initial value of zero. We set its action command to be C goes to F as opposed to F goes to C. So we give it a slightly different name, and we also set it to listen to action events or to generate action events.

Then we're going to lay out our grid. So first element to the grid is the label, as we talked about before. Next element to our grid is our little text box that's going to actually have the numeric value in it. Last element of our grid on the first row of the grid is a button that's name is F goes to C. And you look at this, and you say, hey, if I have a button that's name is F goes to C, and I named this guy F goes to C, aren't I getting back to the previous point over here of – wasn't this the logical problem where I actually have two elements that have the same name?

Yeah, I have two elements that have the same name, but I want to do exactly the same thing in both cases, so it doesn't make a difference. So what I want to do is say if someone clicks the button, I'm going to do the conversion. So I'm going to have some code that's going to do the conversion. If someone types something into the text field and hits enter, I'm going to do the same thing.

So this is something you'd see a lot of times on the web where, for example, if there's a search engine you use, you can type in the search engine and click search, or you can just

hit enter. How many people actually click the search button? One. How many just hit enter. Yeah. Isn't it nice that you can just hit enter? Yeah.

That's the same thing we're doing in this program. That's why we went through this extra rigamarole of setting this action command here because sometimes it's just nice to hit enter. We do exactly the same thing for degrees Celsius. So we add the label, degrees Celsius, we add the Celsius field, and then we create a new button whose name is the same as the action command for the Celsius field. Then we add action listeners.

So that sets up our entire user interface or our entire graphical user interface, or the GUI. Then when the user clicks on a button, we say, let me get the action command. If the action command is F goes to C, which means either they typed something into the Fahrenheit field and hit enter or they clicked the button, then I'll get the value in the Fahrenheit field because Fahrenheit field is an integer field. It just always gives me back an integer. And I do a little bit of math. If you don't know the math conversion from Fahrenheit to Celsius, don't worry about it. This is just how you convert from Fahrenheit to Celsius.

You take nine fifths times the Fahrenheit value minus 32, and that gives you the Celsius value. Now you know. And what I do, more interestingly, is I set the value in the Celsius field to be whatever value I computed. So someone just typed something into the Fahrenheit field and hit enter or clicked the F to C button, but what I do to update the screen is I change whatever value is in the Celsius field.

I do the compliment of that – the mirror image. How many words can you come up with for the same thing? If someone does C to F, which is I get the value that's in the Celsius field. I do the math that's necessary to convert from Celsius to Fahrenheit, and I set the Fahrenheit field. And that's the whole program. Here's my instance variables.

So if I run my little temperature program, I have my label. I have my initial value, and I have my Fahrenheit to Celsius. If I put in some value here like 100 degrees Fahrenheit is 38 degrees Celsius. 212 degrees Fahrenheit, we'll not touch the mouse, just hit the enter key. Enter does the same thing as if I click the mouse.

Same thing on this side. I can say 0 Celsius. Yeah, 32. Good times. Now I've created a whole program with the graphical user interface, and I can resize. It just does – oh, it always centers for me. Isn't that nice. If I make it too small, well, these things don't get to small. I can't see the screen.

Any questions about that?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Oh, can you use the mics, please? I got to keep reminding everyone to use the microphones.

**Student:** Can you adjust the width of the grid, each cell within the grid?

**Instructor (Mehran Sahami):** There are ways with table layout, you can actually give it what are referred to as hints to actually specify different sizes for things. I just didn't do that here. It's in the book, if you want to do it, but we're not going to push it that far in this class. But there are ways that you can.

So one final thing that we want to do, you might say, this is kind of fun, but what I like is some text and some graphics together, and I want some interactors. So I kind of want it all. I want text, I want graphics, I want interactors. You think back to the days, you think Hangman.

Hangman, you had text and you had graphics, but you didn't have interactors. Here you have interactors, and I showed you an example with interactors and text where you click the button, and it said Hi and gave your name or whatever.

Now it's time to roll the enchilada and put them all together in our friend, text and graphics. So what text and graphics is going to do, is we want to think about having some console in the program and the graphics canvas in the program and interactors in the program so we can just go to town and do whatever we want to do. How do we make this happen?

First thing we're going to do, let me write a little bit of text on the board. So text and graphics. Two great tastes that taste great together. You can decide which one's chocolate and which one's peanut butter, but it's text and graphics. So it's like Hangman, but with interactors.

So what we're going to do, is we're going to extend a console program. The reason why we're going to extend the console program is we still need our friend, the console. That's where we're going to get the text portion of doing this interaction is from having a console program. So when we have a console program, what this gives us is the borders that we've come to know and love.

It gives us the north, south, east and west borders. These are places where we can still place our interactors. The interesting thing is what's going on in the center region. What I told you before, the console program fills up the center region with a place where you can put text, and that's all you can do with it. So what we're going to do is say, console program, what I want to do is in the center region, I want to give you a different layout and put stuff in that layout so I can potentially have some text and some graphics.

So what am I going to do? The first thing I'm going to do is I'm going to think about having some layout. My layout's going to apply to this middle region. The important thing to keep in mind is the console program, what it used to do, was create a console that filled up the entire region. Now what I'm going to get is a console as my first element. This means however I do the layout and whatever I do in it, the very first thing – let's say I have a grid that has three elements to it in one row. The first element of that will be my

console. That you don't have any control over just because of the way the console program works.

The first elements of whatever layout you use when you extend the console program and create a layout for it will always be whatever your text is. You said, you were also going to tell me about graphics, but if I'm doing this in a console program, how do I get graphics?

We do the little trick we did in hangman. There was this thing called the G canvas. What we're going to do is create a G canvas. It importantly, is actually something that we can add to a layout. So what I can do is say, create my console program. I'm going to create some layout. Let's say I'm going to have a grid that's – I'm going to create some sort of layout.

Maybe I'll have a grid layout that's one, three, which would give me this. My first thing is already taken up by my console. What I'm going to do is create a G canvas and add that G canvas as my second element. Just to be super cool, to give you something that normally, you'd have to pay \$12.95 for, but I'm going to give it to you for free, we're going to create another G canvas and add it over here.

So that's G canvas dos. So what we get is console and two different G canvases. Plus, we can still add interactors all around our screen. At this point, you should be looking at this in shock, horror and delight and going, okay. Let's all put it together in five minutes because it's just that easy. So here's how it looks.

Text and graphics. I extend console program. That's where I'm going to get my console. In my innate, I say set the layout. I want a new grid layout. Remember, grid layout, the elements expand to take however much space you give them. That's what I want in this case because what I want to say is I want to have a grid. I want to give the console one third of the whole grid, and two canvases another third of those grids and grow them to as large as they can be.

Then what I'm going to do is I'm going to create two canvases. So I need to have some instance variables to refer to these canvases. I'm going to have two canvases, which are just – type is G canvas. They're private variables. I will call them the right canvas and the left canvas. I'm also going to have a text field in this program just for laughs, just because I can. That's going to be one of my interactors. I want to have interactors, text and graphics.

What am I going to do? First thing I'm going to do is I'm going say, left canvas, create a new canvas. Add that canvas, and when I do this add, it's adding it to my layout. I'm adding a whole canvas. So what does that do? It says, you told me you got a grid layout here. I've already filled in the first thing with the console because that's what I do. I'm a console program.

You just told me to add a canvas. Element No. 2 will be the canvas. I do the same thing again for right canvas. Element No. 3 is now the right canvas. So I have two big canvases on there as the second and third elements of my grid.

Now I got a console, canvas, canvas. Now I'm going to add some interactors because it's just that cool. I'm going to create a text field. We'll say new J text field. Text field, I declare it as a private instance variable. I just showed you that. Maximum size is ten. I will add a label to it, and the label's just going to be called some text. So the right some text in the southern region. Then I will add my text field in the southern region.

As of this point, you should come to know and love, you always got to remember to add your action listener. Very common thing that happens, people create a text field, and they're typing it and stuff in their program, and nothing's happening. They're tearing their hair out, and they're wondering why. They just forgot to add the action listener. Know it, learn it, live it, love it. It's a good time.

Then I'm going to add two buttons, just for good times. So I have my text field. I'm going to add two more buttons. A button that says draw on the left and a button that says draw on the right. So let me show you what all these things are going to do before I show you the rest of the program. So I want to show you text and graphics.

I have my console over here. I have two – you can't see them, but they're side by side. Two different canvas windows over here. Here's some text. I can type in hi. You typed hi. Wow. It's that exciting. Draw left, in my left canvas. I'm just drawing rectangles. I'll show you how I do that in just a second. Draw right. Drawing in my right canvas.

How did I make that happen? Let me show you the rest of the program. I've set everything up. Console, two canvases, text field and two buttons at the bottom. Here's where all the action's going on. When action performed is called, that means someone's interacting with one of the interactors.

There's nothing else I can do in the program except interacting with one of the interactors. First I check for the text field. If the interaction with the text field – so if the source of the interaction was the text field, I write out, you type in the text field. This will go into the console because anytime you do a println, the text always goes into the console. So it just shows up in the console. Not a whole lot of excitement going on there.

Alternatively, if the thing they did was not to type into the text field, they clicked one of the buttons. I could've either done all with get source or all with get action command. I'm using both just to show you that you can mix and match if you want. So I say, what was the command? Get action command. If it was draw left, then what I'm going to do is I'm going to create a new filled rectangle. Let me show you.

Create new filled rectangle. It's very simple. It just creates a rectangle that's 50 by 20, and yes, they should've been constants, but I didn't make them constant so I wouldn't have to

scroll down and show you the constants. I set it to be filled, and I return the rectangles. All it does is create a filled rectangle and say, here you go.

All I do is I take that filled rectangle, and I add it to my left canvas. So because it's not a graphics program, I can't just say add with the rectangle and add it. If I want to add the rectangle somewhere, I need to specify which canvas I'm adding it to. I'm adding it to the left canvas. So I say, left canvas, add to yourself this rectangle. Where are you going to add it? At X location 20 and at Y location left Y.

Left Y starts out with the value ten, and every time I add something, I space down my Y. So I'm just making Y go down by some spacer amount, which is 30. So all it's doing is drawing a rectangle and essentially moving down. So I'll draw my next rectangle below it, moving down to draw the next rectangle below it.

I do exactly the same thing for the right hand side, except after I create the filled rectangle, I have a separate right Y, which keeps track of how low I've gotten on that side, in terms of the Y coordinate. I add to the right canvas. That's the only difference.

So when I run this program, some text. Again, if I type in great, great. If I typed in great and hit enter, it generates the event, which does this println on the screen. It generates this event over here, this action performed. The source was text field, and I write out the text on the screen.

If I click on one of the buttons, draw left, it draws the filled rectangle, and it's incremented the Y on the left hand side. So the next time I click draw left, it draws it lower and lower and lower. Draw right does the same thing.

Notice that the X location for both this canvas and this canvas, when I add the rectangles, are both at 20. The reason why it shows up two different places in the screen is because there are two different canvases, and there's kind of an invisible border here. So you can create text, graphics and interactors all together and just go to town.

Any questions? All right. I will see you on Wednesday.

[End of Audio]

Duration: 50 minutes

## Programming Methodology-Lecture22

**Instructor (Mehran Sahami):** Howdy! So welcome back to yet another fun-filled, exciting day of CS106a. We're getting close to that Thanksgiving recess, which is always a good time. In the days of yore, it used not be a whole week. It used to be you got like one or two days off. You got like Thursday and Friday, which means you would have gotten only one day off from this class and now you get a whole week to mellow in style or catch up on all your other work as the case may be.

So there's one handout today. It's Assignment No. 6, which is the next assignment, name, circle, watch. They talk a lot about that assignment in this class today. As a matter of fact, we're very fortunate for coming today because I will give you so much gratuitous information that presents that assignment. It's just almost laughable.

And then Assignment No. 5, as you know, is due today, so we can do painful to see how much time it actually took you. Just wondering, I heard from my little sources that exist was that last night, a lot of people were at the Lair and the computers at the Lair. Just wondering how many people were at the Lair last night. A fair number. And some of the computers at the Lair just mysteriously didn't have Eclipse anymore. Anyone run into that problem? One person, okay.

Yeah, there was some reimaging that went on, which I had nothing to do with and didn't know about until some of the section leaders said, hey, Eclipse is missing on these machines and, you know, so I sent a very anxious email, I should say, to academic computing, and there had been some mix up and Eclipse is now reinstalled on all the machines, so I'm glad it didn't affect too many people, but if it affected you, I apologize, although we really didn't have anything to do with it, but I'm sorry that you had to experience that pain.

So in terms of the painful, how much time it actually should give you the assignment on the Yahtzee assignment? Just wondering, how many of you actually liked the Yahtzee assignment? Just a quick show of hands. And how many thought it was just more pain than it was worth? Anyone? You folks, all right. And that's good to know. I always try to – actually try to gauge these things.

So anyone in the zero to two-hour category? You just did it and it worked and life was good. Two to four? A couple people; that's always good to see. Four to six? A reasonable contingent. Six to eight? Good times. Eight to ten? Wow, pretty healthy distribution there. Ten to twelve? Okay, we're falling off a bit. Twelve to fourteen? You folks. Fourteen to sixteen? Maybe a couple. Sixteen to eighteen? Eighteen plus? Taking a late day? Yeah.

As you may have read – I should refresh your memory because it pertains to Assignment No. 6, but as you may have read on Assignment – on Handout No. 1 at the very beginning of class, because Assignment No. 7 is due the last day of the quarter, you can't use late days on that assignment. So if you have late days and you're like, hey, I was just

saving them up for No. 7, well, seven's due the last day so there's no classes beyond seven and you don't say, hey, I'm taking a late day. That gives me an infinite amount of time to turn it in because there are no more classes. No, it doesn't work like that.

So Assignment No. 7 you can't use late days on. So again, I would discourage you from using your late days, but should you have them and you're like, hey, I really wanna use my late days, Assignment No. 6 is kind of your last chance to actually use your free late days. Again, not that I would encourage you to use them, but just so you know up here that you don't get Assignment 7 and say, hey, I thought I could use my late days for this. You're getting plenty of advanced warning that you actually can't.

One other quick announcement; CS Career Panel is today, so after today, I'll stop mentioning it because I believe most of you can't actually time travel, so today, 5:30 p.m. in Packard 101. Be there. We have a stellar panel of folks from academia, from industry, a lot of people who actually – every one is a computer science major and four of the five people graduated from Stanford's department. The fifth is actually a faculty member here that didn't graduate from my department, but is a faculty member here.

And you'll actually see there's a very broad range of stuff to do. Out of those five people, though, actually, I think only one, maybe one and a half who actually do programming as their sort of fulltime job. But all of them are computer science undergraduates, so you get sort of a breadth of what computer sciences can do.

So I'm so happy to see the fifth region. You'll see the variance gets larger, right. As the quarter goes on, that kind of happens, but it's good to see that the mean is still where we generally want it to be.

So now it's time to talk a little bit about your next assignment. You just turned in your last assignment. You're like, no, no, it's not time to talk about the next assignment. I'm still working on the last assignment. Well, just pay attention.

Here's what you're going to do for your next assignment. It's a little program called NameSurfer, okay? And the idea of this program is it turns out that the government actually keeps track of statistics of the popularity of names over time, okay. So every decade when they do the census, they kind of keep track of the number of names and they rank the top 1,000 names in terms of popularity for boys' names and for girls' names, and they actually make this data available. We thought wouldn't it be cool if you actually were to display names to the user in terms of how that popularity changed over time.

So that's what NameSurfer does. Here's the program. What it does is it shows you the decades. You can notice that this is a graphics, or involves graphics, right, has the decades from 1900 up to 2000 and so relatively recent date up to the last census. And what we can do is we can type in a name and then click graph, and what it will do is it will show the rank of that name over time, on a decade-by-decade basis. So let's try Fred, for example. If you graph Fred, you can see Fred real popular at the turn of the prior century and then Fred just had this quick decent into oblivion, where now he's the 974th

most popular boys name, at least in the last Census 2000. We can do multiple graphs on here, so if I type in someone else – so one thing that's also interesting is you'll notice popular culture comes in, and so if I type in Britney, the alternative non-traditional British spelling, it didn't exist until 1980, in which case it totally shot up to like the top 100 and maybe with the most recent things that are going on, it's dropped a little bit.

But through this whole region, what we have in our data, and I'll show you the data in just a second, is that Britney did not rank in the top 1,000 until the 1980's and then it jumped up. Because it started ranking in the top 1,000 in the 1980's, it still shows up in our data file because at some point it started ranking higher than that. And it turns out that there are some people who just have names that, well, are not so popular. So we do Mehran and nothing comes up because contrary to popular opinion, Mehran is not in the top 1,000 most popular names. In a few years, you can change that if you have children. Your choice. Not that I would advocate it, man. It's a rough childhood. Mehran, doesn't that sound like moron? Yeah, I haven't heard that one in the last two hours. Yeah, I know. I'm just – rough childhood. Not that it affected me at all. So in terms of actually doing this assignment, you might say, oh, that's kind of interesting. Now there are a couple of things going on. You might look at that and say, yeah, I could do that, Mehran. I could do that in the zero to two-hour category, right, except for one little thing and the one little thing is check out this bad boy. Whoa. As I resize the window – I will continue to resize the window until I get you in a hypnotic trance – everything in the graphics display automatically resizes to display for that window size, okay? And what you're going to find out today is how do you actually do this kind of thing to keep track of as the windows resizing, how do you get events that come to your program that tell you, hey, some resizing happened. I need you to redisplay what's going on.

The other thing that's interesting about this is I cannot only when I resize, it keeps track of all the information that's on there, Fred and Britney. If I ask someone else like, oh, my son, William, who's always been a popular choice – ah ha ha – just like in the top ten dips briefly over here because maybe it was getting over used and then it comes back up. Keeps them all. They're in different colors. The color sort of cycles, so if we add someone else, like Ethel – Ethel, Ethel – yeah, Ethel, sorry, not so popular anymore. When Lucille Ball – when "I Love Lucy" was on or earlier, it was a popular time. The cycles of colors eventually will go back to black, so if we put in something like Bob, it goes back to black. Then you'll see red again, then blue, then magenta. That's all given to you in the handout.

If you hit clear, everything goes away, except for the gridlines, and you can sort of go again from there, like here's George and you can see George has a slight decline. Here's four back here and now he's like somewhere in the hundreds. It's okay if it goes off the window occasionally, okay? But the resizing is kind of the key to this and that's what we're going to – one of the things we're gonna look at today. Okay.

So that's what you want to keep in mind for the assignment, resizing, because the demo that's on the website because it runs in a web browser, doesn't resize and it sort just has it as the little warning down at the bottom. It just shows you how the program actually

works so you can try out typing in different names and see the behavior. But it doesn't resize because its size is stuck by the browser and changing the browser window doesn't actually change the size of the applet. That's just life in the city, okay? But that's the way NameSurfer works.

So what we wanna think about is actually building programs that have this kind of property and to understand how programs that have this kind of property work, what we need to think about is something known as a component and related to that is a container. So first of all, I'll show you all the classes in Java that pertain to the components and containers. Not all of them, but a good number of them. So here's the classes in the component hierarchy and some of these will look familiar to you. So programs of which Console program and Graphics program that you've been using the whole time are j-applet, which is an applet, which is a panel, which is a container, which is a component.

So somewhere all the way up the food chain, right, the food chain before we didn't go that high. We just went up to applet when we showed you sort of this picture before. Turns out all your programs are components and containers. Every component is a container, or every container is a component. And so you may ask, well, what are containers and components? So the way you wanna think about this is at some generic level, a component is anything that can appear in a window. It's anything that can in some sense be displayed. So your programs are appearing in a window that's getting displayed. That's why they're components, okay? Now, that also means that a graphics program and a console program are also components because they're subclasses of program. Now at the same time, there's a notion of a container. What's a container? A container is simply something that contains other components, so a container just contains a set of components and you might say, hey, Mehran, that's really weird. A container is a component, but you're telling me it contains a set of components? What's that all about? And the simplest way to think about it is it's this, okay? This is a component, this bag. It turns out this bag is also a container because it contains things and it's perfectly fine for something to be a container and also to be a component.

So here's what I mean, right. Your programs, as it turns out, are all components, which means they display in the screen, but they're also containers because your programs contain other things. For example, in the display of your program, it may contain a button. Right? It may also contain, say, a text field. And the other thing that it may contain, if you have a graphics program besides these kind of components, which buttons and text fields are also components – I'll show you an example of that in just a second – it may contain another container. And it's perfectly fine for containers to contain containers. As a matter of fact, you've been using them for the whole time. Your graphics program, especially if you have a graphics program that has some interactors on it. It could have buttons. It could have text fields. It could have a G-canvas, and gee, guess what? A G-canvas is a container and a component. Because it's a component, it can be contained inside some other container, and because it itself is a container, it may have certain things in it.

Oh, like here's our friend, the frog image from the mid-term. Yeah, that was just something that was in another container, okay? So there's no reason why something that is a container can't be a component, which means it can be contained in some other container, that's fine, and all containers themselves are components, so they can be stuck in somewhere else. That's the basic idea. You should think of it as bags and you can put bags inside of other bags, and bags contain things. So even though you see the hierarchy this way, the concept itself is not actually that difficult. Now, the thing you wanna think about this is I just mentioned to you your programs are actually components and containers. G-canvas is a container and a component, but components that we talked about before, j-components, like our friends, say, oh, j-button – you actually saw this slide a couple classes ago, j-button or j-label or j-combo box, all these things you saw before – are actually j-components. They all are subclasses of j-component, which means by being subclasses of j-component, they were also containers, strangely enough, and they are also components. We just think of them generally as components and we try to put them in other containers, like we put them inside the display for a program and that's its container. But that's the general idea that's kinda going on here. So when you wanna think about containers and components, what you also wanna think about is that these components themselves, before we thought about like action listeners, right, like something happens like I click on some component and some particular event happens, like I clicked on it, something like that. We can think of the whole component as also listening for other events because a component is something that display stuff on the screen.

Stuff can happen to be components, like they can be resized and when a component's resized, some event can get generated and sent to that component that says, hey, guess what? You got resized. You need to do something. Like the canvas that shows all the graphics, it can get some event that says, hey, canvas, yeah, you were just resized. Do whatever you wanna do. Up until now, you didn't do anything. Someone came and resized your canvas. You said, yeah, you're resizing my canvas. All of the pieces I have in my canvas stay where they are. Nothing's changing. Now what you're going to get is the ability to have the canvas be told, hey, you got resized or something else happened to you. You can respond to it however you want, which means you can redraw everything that's in your canvas, so that, for example, you can have all the layout come out just fine when people resize your window. So what does that actually mean in terms of these components and containers? What it means is that we can have listeners in the same way that you kinda thought about listeners from mouse events and listeners for j-components. You can have listeners for components, okay, like we talked about resizing the window.

Now, here's a little simple example of something that can go on with one of these components. So I'll come back to Eclipse, I'll run a little program here – I gotta compile – and I'll show you the very sort of simple program that makes use of this concept of a component having a listener and being able to get certain events. So it's just called My Programming. Here's what my program does. It draws a square. That's all it does is it draws in a filled-in square. You're like why are you showing me this, Mehran? I could've done this in the first week of class and I would say, no, you were doing terrible in the first week of class. And then you say, no, I could've done this in the second week of class and

be like, okay, but could you do this? Oh. Yeah, when you did this before, that square just stayed where it was and the size of the window changed. Nothing happened. Now the square always remains in the center of the window no matter how I resize. It's super small. It's still in the center. Super big, yeah. Another one of those things good for about 20 seconds. But that's the idea. How does it know when the window's getting resized that it needs to do something to update its display? So the way this is going to work, in terms of having this square that recenters itself, is we wanna think about having a program that is a container. And so, up until now, you had console programs and graphics programs, and they were containers and what they did for you automatically for console programming, it created a console for you and put it in your container so that anything you printed showed up in that console.

And for a graphics program, it created a G-canvas for you and put it in the container and anything you drew showed up on that canvas. And now, what you're going to do is you're going to create your own program that's not a graphics program or a console program, but to which you're going to create your own canvas and add it. And so the way that would look is you would say something like "Public Class" and give it the name, whatever name you want, so I call this "My Program" and this is gonna extend just the generic program. It's not gonna extend console or graphics program. It's just gonna extend program. And so inside here, it's still gonna have some init method, like you've come to know and love in other programs. And inside init, what we're gonna do is say I'm gonna create a new canvas. But the canvas I'm going to create is a canvas that is a new object I'm going to create myself to define what that canvas is, rather than the basic G-canvas. So I'm gonna call this My Canvas. I'm going to do top down design. My canvas is a class that doesn't exist yet, all right? It's not a built-in class. It's my canvas. It's something that I'm gonna write the code for in just a second and I'm gonna have some object called "canvas," which is off top My Canvas and I will create a new My Canvas. At this point, you should be getting a little bit worried unless you've internalized top down design. If you've internalized top down design, you say, yeah, I'm just gonna go create My Canvas when it's time for me to write that code. For right now, I'm just going to assume it exists.

And then what I'm gonna do is I'm going to add this canvas to my container. When I do an add of something that is a component, in this case, canvas will be a component – you'll see why it's a component in just a second – it says, hey, program, add this particular element to yourself, and program says, hey, I'm a container, right? I don't already have a canvas. I don't already have a console, but I'm a container. You're giving me a canvas, whoosh, the canvas goes up into my container, and if it's the only thing in my container, its size will expand to fill up the whole screen. Kind of like you saw last time. And that's it. So what I've don't in this program is said, hey, I'm gonna have a program. I'm gonna create a canvas where all the action isn't gonna happen in that canvas, and I'm going to add that canvas to the display and then I'm gonna let the canvas do all the work. Is there any questions about this, just creating the canvas? So now we can actually sort of define what is My Canvas and I'll show you that on the computer because it's a little bit more involved. And you don't need to worry about writing this all down. Actually, on Friday, I'll give you a handout that has all this code in it. I just didn't

want to give it to you now because then you wouldn't pay attention to what's going on on the screen, okay? So what I'm gonna do is I'm gonna have a class My Canvas. And My Canvas is going to be a G-canvas, so it's going to be an extension of a G-canvas. It's gonna extend G-canvas. So all My Canvases are G-canvases. Because it's a G-canvas, that means it's a component. Because it's a component, I can add it to a container. So that's why add works there.

Now, what am I gonna do with this thing? The other thing I wanna do is I wanna say, hey, this is a component that's gonna listen for stuff, and so the way I do this is rather than doing add mouse listeners like I did before, or add action listeners, which look at actions of things like buttons, the way a component actually listens is it implements the component listener interface. So when you wanna have some component that is actually gonna get events that it's listening for, component events, the way you do that is when you create the class, you say it implements component listener. That's just the way you say it, so it's going to implement the interface. There's some set of methods it needs to provide that implement that interface and I'll show you the list of methods real quickly. They're just down here. These four methods here, component resize, component hidden, component moved, and component shown are the methods that comprise that interface, the component listener interface. So anything that implements the component listener interface needs to provide these four methods. In a lot of cases, you won't necessarily care that something happened and so, for example, if the component is hidden, which means some other window comes over it, the component is moved, or the component is shown, you're like, yeah, whatever, like I don't care. Those are good things. I need to have methods that correspond to those things because in order to implement the interface, I need to provide methods that correspond to all the methods that the interface is expecting, so I need to have these. But in three of these four cases, I'm not going to do anything, right? It's just open brace, close brace.

Someone comes along and says, hey, guess what? You know you were occluded. Someone hid you, so your component's hidden, and you're like, yeah, that's cool. You're like, aren't you going to do anything? No, I like hiding. And it's like, oh, look, your component was moved. Someone came along and moved your window. Aren't you gonna do something? And you're like, no. No, I'm cool with that. That's fine. You moved me, all right? I won't get into the music, but if you can think of that, what that reference actually is, it's not that difficult. Louie, Louie. Component shown, again, definitely, so the only time we actually care about doing something is when we're resized because when we're resized, we wanna say, hey, our component was resized. Let me move that square to wherever the new center is of that window. So whenever my component's resized, I'm gonna get the component resized method called and I'm just gonna call the method called Update. And what's Update gonna do? Update's right up here. All Update says is remove all, so who's the remove all message being sent to? It's being sent to myself because I'm not specifying any particular object to send this message to. So when I say remove all, I'm sending it to My Canvas. Well, My Canvas is a G-canvas, which means remove all, removes everything that's on the canvas. It clears the canvas, which means wherever the square was before, it just got cleared.

And what I'm gonna do is I'm gonna add some rectangle, and I'll show you the rectangle. It's pretty simple. So what I have is I have a private G-rect that I call rect. It's just gonna be some private variable that I keep track of. In the constructor for my class, what I'm going to do is I'm gonna – before I tell you what add component listener this does, I'm going to create a new rectangle, who's width and height are whatever I specify as constants is the box width and box height. They're just constants, not a big deal. And I set it to be filled. So what I get is a filled square. I created this rectangle as a filled square. I haven't done anything yet with it other than create it, but I've created it and I have some variable rect that refers to it. So all Update does is it says, hey, you know what? Remove the square if it was anywhere on the screen before and now add that rectangle back at the center location of the square, right? What's the center location of this point? You should be good, real good, with figuring out how to center stuff on the screen. You take the width of the screen, subtract off the width of the object you're displaying, and divide by two. That's the x location. You take the height of the screen, subtract off the height and divide by two. That's the y location. And this get width and get height will get the current size of the component, which means when the components been resized, this update method gets called, it closed the screen, and it gets rid of the old box and draws a new box at wherever the new center of the screen is because this gets called, which means Update gets called after the component's been resized. So that way it's gonna redraw the square always in the center of the screen.

The only other thing about this program that you need to know is in the constructor for My Canvas, My Canvas implements the component listener, so you not only need to say, hey, I implemented the component listener, but you need to let the – you need to actually say, hey, you know what? I am a component listener, so add me to the component listener list, and the way it does that is you just call add component listener and you pass this, which means myself. So somewhere, I say, hey, you know what? Add me to the list of component listeners. I'm giving you a parameter to myself so you know who I am, so when something happens to me, you can call my component resize method to get resized. And that part up there, that Add Component Listener, is just a broiler plate, but it's gonna, for example, show up in your NameSurfer assignment because guess what? In your NameSurfer assignment, when your component gets resized, you're going to do something very similar to this. You're gonna find out what the new dimensions of the screen are and redraw everything that should actually be on the screen, which means you need to have some way of being able to keep track of the stuff that's on the screen. So any questions about this? All righty, so that's the basic notion of this idea of a component listener, okay, and what we wanna think about next is how do I create a larger program with these kind of things. So this is just the simple hint. Now I'm gonna give you something that's even a bigger hint to what's going on in NameSurfer.

And so what I'm gonna do is one thing that's real popular these days is kind of like music, right, like online music and albums and the whole deal. So what I wanna do is implement a program that keeps track of data related to music and at best, I'm gonna have a little music store, the program for the music score. And the basic idea here is that a lot of what – well, I shouldn't say a lot, but so far you've been – I don't know. You did a lot of games in this class and games are great and I totally encourage games, and I'm a

big fan of games. I got into computer science because of games. It's a good time. But a lot of what computers are actually used for as well, like when you go to some .dot com site or whatever to like buy music, and I won't name a particular one because I might get sued, you know, it's doing data management, right. For them, when you go to their site, all they really care about are what are the songs they're providing, do they – you know, like you're buying a physical CD, do they have it stock? Sometimes you just get stuff downloaded that's all digital. And the other thing they worry about is stuff like prices and inventory management and stuff like that. But that's really all that whole side is doing is doing management of data.

So we're gonna write a little program that does management of data, and the basic idea here is someone comes along and gives us a data file called Music Data and the idea of this data file is it's a very simple format. And the basic format – I'll just draw it out here – is every line of the file contains information about one album that's in the inventory and the basic idea is there's a bracket, here's the name of the album, right. We can't just use spaces in this case because there might be spaces in the name of the album, but we kind of assume there's no brackets in the names of albums. So there's the name. And then there's a space and a bracket, and then the band's name that actually produced that album, and then there's an integer over here, which is the number in stock of that particular album. So we have ten copies of Snakes and Arrows. Any Rush fans in here? No? There's like two? Times have changed, I've gotta say. All right. There's not gonna be any fans of this. Dokken? Anyone? Yeah, I would not advise it. And Smashing Pumpkins. There might be a few – yeah, it's a good time. It's a good time. Old stuff a little better than the new stuff, but that's not important right now. What is important is that we're gonna have some data and our program is basically gonna manage this data. It's gonna read in this data file and allow someone to ask questions about for different kinds of albums, do we have that album in stock, and how many of that album do we have in stock. And do some nice little graphic display on it to make use of some of the graphics concepts that we've actually shown.

So what I wanna do in this program – let me actually show you the program in operation so you can see what's kinda going on with the whole display and resizing the display and why it's the coolest thing ever. So when we run the program, I'm just gonna call this program Music Shop. When I run Music Shop, what I get is basically a blank screen that asks me for an album name, so I can come along and say, oh, do you have any copies of Plans by Death Cab for Cutie – any Death Cab fans? It's a strange name for a band. What it does, it might be difficult to see, but it writes that album Plans by Death Cab for Cutie, ten in stock, and it writes out little squares to indicate that there's ten. It writes out ten squares and you might say, okay, Mehran, that's not the greatest display in the world, and it's not. This display I wrote in a half an hour last night, but what's kind of cool is, ah, look, it centers this way. And here's the other cool thing. Look at the size of the boxes. They get smaller and they expand, so they always have the same spacing, but as I expand out or get smaller, they resize. The fonts don't resize. They just recenter. I could, for example, put in perhaps one of the greatest albums of all time, "So" by Peter Gabriel. There's 20 copies in stock as everyone should own a copy, and if you don't, that's okay. It's perfectly fine. And sometimes I might type in a band that doesn't exist or an album

that doesn't exist, like I might type in – I don't know – cow, which is – there's probably an album out there called Cow. I don't know why I always pick Cow. I just do. In which case, it definitely disappears. Like we don't have that in stock. Just back off, man.

So that's what's going on in this program. So we need to be able to resize the bars and we need to vertically center the stuff we display in the screen. How do we actually do that? So one thing we wanna keep track of is you wanna go back to the data, right? You actually wanna say if I need to do this, in order to display anything, the first thing I need to do is figure out some information about the data that I wanna store. So what data do I wanna store? Well, I probably wanna store some information about albums because that's what I have in my data file. So if I think about my data file and wanting to encapsulate all the information about one album, okay, I wanna do something on a per album basis because every album is gonna have, for example, an album name. It's gonna have some band name that produced that album. And some number that's in stock, like Num Stocked. Oh, and this is a string and this is a string and this is an int and it's good to put semicolons at the end of them and it's probably a real good idea to take this whole thing and turn it into a class that's gonna encompass all this because what I want to have is modularity in terms of my data. If there's a whole bunch of data I wanna manage, the first question I wanna ask myself is what's the basic element of data that I want to have? In this case, the base element is one album. I'm gonna have a bunch of albums, but the basic idea is one album and an album's gonna contain an album name, a band name, and a Num Stock, which are, for example, gonna be private fields of that album.

So one of the things I might wanna do and I'll just show this to you real quickly, is I would create album.java, which is just some class. It doesn't extend anything. It's just gonna be a class to keep track of data and what it's gonna do is it's gonna have some private instance variables, album name, band name, and Num Stock, that's gonna keep track of the information from one album. And then it's gonna have some other methods. One of the methods it's gonna have is the Constructor. What does the Constructor do? It says, hey, you want an album object? Give me a name for the album, a name for the band, and a number of stock. I'll set all my internal variables to those things. I'll do a little annoying pop-up on the screen and what you'll get from your constructor is you'll get an album object that encapsulates that information. Now, I made all this stuff private, right, album name, band name, and Num Stock, because what I wanna do is when you create the album, you say, hey, this album's in stock. I don't want you to go back and say, hey, no, no, no, the name of that album changed. That's not what happens, right? When an album's released, the name stays the same, generally. We won't talk about special cases. So in order to get the information out of an album object, I need to have some getters. I can potentially have setters as well, but here I have some getters. Get album name, get band name, and get Num Stock. Pretty straightforward getters. You've seen getters in the past; not a whole lot of excitement going on there.

The only other thing I might add, which I told you in the days of yore that you didn't have much need for until now, is to have a 2-string method, just returns a string representation of that particular object if you ask for string representation. So it just writes out "album name" by band name colon the number stock in stock. So it can return

to you a string that basically has all the information about the object in some nice little display format. That's the whole class, okay? This is just a simple class that keeps track of information about one album because we're gonna have a bunch of albums that we wanna keep track of. So now that we can keep track of information about one album, it kind of brings up a deeper question. The deeper question is we don't wanna keep track of just information about one album. We, in some sense, wanna keep track of a whole set of albums, a database of albums, and what we wanna do if we think about a running program – so I'll go back to the running program – is we wanna allow someone, given a particular name for an album, to be able to go look up all the information on that album. So the way you wanna think about it is given a name, I wanna go and look up the record for that particular thing. And this happens all the time. It happens with your student I.D. records, except we don't do it by name. We use student I.D. reenact records. We say given your Stanford student I.D., let's go grab a record of all of your information. It's exactly the same problem. This is just the simple version with albums. So if we wanna think about having some mapping, where, for example, from some name, we can go get the whole information about that record. What kind of data structure might we think about? What kind of thing that you've seen before?

Ah, yeah, the whispered HashMap. Could it be the HashMap? And in fact, yes, it could be a HashMap. And the thing you wanna think about with the HashMap, that's very concentrated. It's a low variance event where it's kind of tougher the whole time and you're like, ah, everyone. Yeah, it's early. It's before Thanksgiving. Have some sugar. We can think about a HashMap. Now when we think about having a HashMap – suddenly, everyone wakes up. Food? Food? Bark, bark. I know, sometimes. Anyway, HashMap, what are we gonna map to? What are gonna be the two types that our HashMap is parameterized by? What's the first type? What are we gonna look things up by? What's the key? String, right? We're gonna look things up by the name, which is a string. When we look something up, what do we wanna get back? An album. So what we're gonna have is a HashMap that maps from strings to albums, and I think someone said album right over here, although I think I missed you. Sorry. I missed you again. So the basic idea is given some name, I'll go look up the whole album and at this point, you might say but Mehran, isn't the name inside the album? Yeah, that's fine. In your student record, it also contains your student I.D. and your name. We just happen to look it up by your student I.D. and it's the same thing here. The user's gonna give us the name to look it up. When we go look it up, it's fine if the name's also contained in the album, but there's a whole bunch of other information we care about there as well. So if we have some HashMap, we'll call this particular HashMap "Inventory" and we might say new HashMap that's gonna map from strings to album, and there's the constructor. And so that's how we might actually create this particular HashMap.

So once we have this HashMap, if we're gonna have some object inventory that's a HashMap of all these things, we need to load it up. We need to say, hey, all my data is actually sitting in a file somewhere. What I need to do is read the data in from the file and as I read the file line by line, every line I'm gonna create one of these album objects because every line is information about one album, and after I create this album object, I'm gonna add it to my HashMap and so my inventory's gonna be all of my albums

mapped to by their name. Any questions about that basic idea? So let me show you the code for them. What does that actually look like? So here's album. You saw album. What I'm gonna have is I'm gonna have My Music Shop. What's My Music Shop gonna do? Before I show you everything else the Music Shop's gonna do, it's gonna have this HashMap. It's gonna have a HashMap for inventory that's a map from strings to albums, and it's initialized just the same way you saw it over here, and the way I set this up is I'm gonna have some method I'm gonna call – called Load Inventory. What's Load Inventory gonna do? It's going to have a buffered reader because I need to open a file and I'm just gonna hard code my files called musicdata.txt. I go and I read the file line by line, so I grab a line from the reader. If it's the last line of the file or there's no more lines left in the file, I'm done. Same thing you did with file crossing before. If there is a line there, then I'm going to write some method that's top down design called Parse Line, which says, hey, let me give you this string that I just read in. It's a whole line of the file. You break up the string into all of its fields and create an album object that contains all that information and return that to me and I will assign that to something I'll call album, which is a type capital A Album. And then what I'm gonna do is I need to put that in my inventory, so in my inventory, I'm going to put it. How am I going to put it in there? I'm gonna put it in by the albums name is the key, so I just say, hey, I have an album object that contains all my information now, so album.getalbumname will give me the name. That'll be the key, and the thing that I wanna store relative to that key is the whole album object. So this just line by line reads the line, parses it – I'll show you how to parse it in just a second – to create an object, and then adds that object to this HashMap I'm creating that's gonna store everything. Know this, live this, learn this, love this. You'll do this for NameSurfer. So after I do all this stuff, I close off my file and I'm done because presumably, I should've read the whole file and put everything into my database and I'm done. I do my little exceptions with file reading, just in case I had exceptions. So the only code you haven't seen so far is parse line. What's parse line doing? Parse line is a string manipulation extravaganza. So what's parse line doing? And I'll just go through this very simply. It's a lot easier than it looks. Basically, it's just a lot of indexing for strings. What I do is I say, hey, I got a whole string that has – oh, I erased it up here, but basically, starts off with a bracket, has the name of the album, the name of al, has the name of the band, and has some number that's an integer. So how do I break this up into sub pieces that I can actually store in my structure?

Well, the first thing I'm gonna do is I wanna pull out the name of the album. How do I find the name of the album? The name album starts after the first opening bracket, so I find the index of the first opening bracket and add one to it. That's the first character of the album name. How do I find where the end of the album is? I look for the index of the closing bracket, and so basically, if I take a sub string from the start of the album name to the end of the album name, I pull out that piece of text that's just the album name and I'm gonna store that in a string called Album Name, which is just a local variable. So now I've pulled out the album name. How do I get the band name? Band name, I look for the index of the first bracket after the album name. That's the critical thing. If I don't look starting after the album name, I'm gonna read the album name again. So I look for the first bracket after the album name, which is ed album name end plus one. That will get

me the index of the bracket where the band name starts and I add one to that, which gets me to the first character of the band's name.

And then I do the same thing over here. Where's the end of the band's name? Get the index of the closing bracket starting at the end of the album name so I don't get the closing bracket for the album name. I get the closing bracket for the band name. So I start at the album name plus one. So now I have the boundaries for the band's name and so the band's name is just the substring I get in the line from the starting index for the band's name to the ending index for the band's name. Now, the final thing I need to do and this is funky thing. It's probably the funkiest thing of this whole function, of this whole method, and it's pretty simple, which is say that last thing that's on the line is actually integer, so I don't wanna pull it out as just a string. I need to actually convert that string to a real integer. How do I do that? Well, let me first find out where that number actually lives. Where do I find the number? I look for the number by finding the first space after the end of the band name. The first space after the band name is gonna be this index right here. The number starts on the next location, so if I take that particular location, after the end of the band name, and add one to it, I now have an index that's the very beginning of the number of that numeric sequence. So what I wanna do is I wanna say pull out that numeric sequence as a string and convert that string to an actual integer. How do I do that?

Well, the way I pulled it out is I say take the substring of the line that starts at the starting position of the number of the number stock. That's where I just computed in the last line was where that number starts. Because I don't specify an ending index for this substring, it goes till the end of the line, which means it takes all the characters. If there happens to be like 100 in stock, it starts at the one and takes the 100, it takes that whole substring as this substring right here, and what do I do to that? There's a method that's a static method of our friend the integer class called parse ins, and so if you say integer.parseins, you give it a string, it converts that string into its integer equivalent. So what it does is it says, hey, I'm giving you the number that's the number stock as a string. It says okay, I'm gonna parse that and turn it into an integer and what I will give you back is something you can store as an int. So that's how we convert that last portion of line from a string to an integer. We first extract it and we get the integer. And then what are we gonna return to the person who called our function? We're gonna return a new album object, we're gonna create a new album object, where we initialize the album object to have the album name, the band name, and the number stock that we just parsed out of that line. Are there any questions about that? Standard thing you do in files, you pull out a line as a string, you break up that string using some string manipulation operation, however you wanna break it up, and then you potentially create some object out of it so that you can store all of the nice things that you extracted out of the string into nice little name fields.

So this gives you an album object. Then back up here, that album object is getting returned by parse line. We put those album objects into our inventory HashMap indexed by the album name. Are there any questions about that? If that's clear, nod your head. Good time. All right. And that's basically the role loading of the database. Now the only other things going on here is we need to figure out our little interactors, right? So in our

program, we have a label that asks for the album name where the user can type the album name, so we put in a label. Then we put in a text field that has a maximum size of 20, right? At this point, this should all be sort of old hat. We add the label to the southern region. We add the album name, which is just a text field that's gonna take in the album name for the southern region so we get those two interactors. We get the label and the text box, okay? And the things we wanna do after we set up the interactors is we say, hey, I need to display the stuff somewhere. Mehran, I remember he told me, oh, about 20 minutes ago, that if I wanted to actually have some canvas that could resize itself automatically when the user changed the window, what I need to do is extend the G-canvas. I need to create in my own version of canvas and make it a component listener. That's the same thing I'm gonna do here. I'm gonna create something called a Music Shop display and Music Shop display, which I'll show you the class for in just a second, is something that I'm going to store as my canvas as a private variable inside my class. And I'll just go ahead and say, after you put your little interactors on this screen, create your new canvas and add the canvas, just like I did before in the previous program, now load all your inventory, so go read the file, do all that parsing, the funky stuff, set up the inventory. Add action listeners because I wanna be able to listen for events that actually happen on the buttons and also add the text field album name as an action listener. So it basically wires everything up. It says put in the interactors, create the canvas that I'm gonna draw stuff on, load the inventory into data, and get ready to listen for stuff. And then it's ready to go. It's not gonna do anything until I get some events, but now it's ready to go. So load inventory you saw, parse line you saw. The only thing you haven't seen is what do I do when an action's performed.

I don't have some button the user compressed. All I have is the text field, so when they hit enter, I check to see if the source is the text field album name. If it is, what I'm going to do is ask the canvas to display the inventory of a particular album. How do I get the particular album? What I do is I'm gonna say what I want you to display, I'm gonna give you an album object that's gonna contain all the information you need to display, so I'm gonna call display inventory pass in album object. How do I get the album object? I say, oh, text field, give me the text that's in you. It says, oh, here you go, and here's what these are typed in. That should be the name of an album. I can use that name for an album in a HashMap to look up the album object, so in my inventory HashMap, I say get on the album name and if there is an album object in my inventory HashMap, that's what I'll get back. If there isn't, I'll get back null and I'll go ahead and call display inventory with null, so it needs to know how to handle that. So all the action to do the display is gonna happen in display inventory. So let me show you musicshop.display, which kind of pulls this final thing together. These are shock display extends G-canvas, just like you saw in the previous example with the little square, implements component listener, just like you saw in the previous example, and it just has a little bit more complexity. The only additional complexity it has is in its constructor. As before, it needs to add itself as a component listener. That's broiler plate. But it says, hey, you know what? I'm gonna keep track of the last album the user actually typed in because when you resize the window, what I need is the information about that album to redraw everything in the window relative to that last album. So when I start off, there was no last album, so I set it to null, but as soon as you give me a real album, that's what I'm gonna store in last

album, is the last album from the user. I'll show you a few things down here. So last album is basically just album that I keep track of in album object, in last album [inaudible]. I have a few constants that indicate for me how big things are gonna be on my display and I can get all the methods of a component listener. Again, the only one I care about is the resizing event. The other ones I ignore. On the resizing event, I'll call Update. If I call Update on the net display, display from my inventory the last album that I displayed. So if I had something in my display and my display got resized, basically all I wanna do is redisplay that last album in the same graphics window.

So Update calls display inventory or I can call display inventory directly from my main program to display something. And this is way more complicated than it looks, so I'll just briefly tell you what it does. It clears everything that's in the display from G-canvas currently by calling Remove All. That is you just gave me an album to display. The last album that I wanna keep track of, the thing that I wanna keep track of in display next time I get resized or whatever is the album you just gave me because you just saved me a new album to display, so that's gonna be the album I keep track of to display on resize events. If that album was not null – if it was null, I'd just clear the screen. I wouldn't do anything else. If it wasn't null, it means that you had a real object in your inventory that was an album, so there's some work for me to do. What am I gonna do? I'm gonna ask that album for the number that are in stock and I'm gonna create a label that has album and the name of the album by and the name of the band, and I'm gonna place it on the screen at a location that's censored based on the height of the current display. Then I'm gonna have a fore loop that displays squares that indicate – shouldn't be dictated; it should be indicated – how many are in by inventory. I won't go through all the math because the math is not interesting. The only thing that's interesting about this is I have a fore loop that goes through the number in stock and draws one filled in rectangle for every number in stock. What's the size of that rectangle? The length of that rectangle is called the broad length that depends on the total size of the window. So I look at the total size of the window and I divide by however many maximum squares I can display in the window, which happens to be 20, to get my size. So as the window gets smaller, the display object's gonna get smaller and as the window gets larger, broad length will get larger. So it depends on the size of the width of the window.

And I do this in a loop so basically, it just draws all the squares. And you can go through the math on your own if you're interested in it, but the basic idea is the squares just size depending on the width of the window. Last but not least, I write out another label that says how many are in stock and where I display this on the screen depends on the height of the window because I'm in the center. So as the height of the window changes, this will always recenter as well. And that's where all the action is, so when I run this program, it starts off not doing anything. And when I type in an album – let's use Snakes and Arrows because I think that's in there – I get a bunch of squares and it's just sitting here. If I type in an album that doesn't exist, like Cow, it clears the display and there's nothing to display. But if I happen to type in – I'm gonna do Snakes and Arrows again – something actually does display. When I do the resize, it's resizing based on the size of the window because it knows what the last thing displayed was, right. It knows the last thing displayed was the album Snakes and Arrows and it's storing that in its own local

variable, so that when I resize, it knows what information to redisplay on the screen to resize. Any questions about that? We'll do exactly the same thing in NameSurfer and I'll give you all the code you just saw.

[End of Audio]

Duration: 50 minutes

## Programming Methodology-Lecture23

**Instructor (Mehran Sahami):** So welcome to the 106A lecture just before Thanksgiving. I thought attendance might not be spectacular today. People are still filtering in, but it's good to see so many of you here, though. I'm sure some of you have already headed home.

You may be wondering who I am. I would have thought before we got back mid-quarter evaluations that you stood a chance of recognizing me as the TA of your class, but the comment of more than half of the people who responded to the question of how is Ben doing was, "I don't know Ben. I've never interacted with Ben. So I assume he's doing a great job."

Most of you jumped to that conclusion. But – so I sort of chased at this, but not much. I realize, though, that I would enjoy the opportunity to develop a little more exposure. And so Maron and I decided to switch roles for today. In fact, he's making copies as we speak, and there's going to be one handout that's sort of pertinent to what you'll be doing for [inaudible] the next assignment. And he should be back any time, but I certainly can't blame him for being – running a little late since I was supposed to make the copies before class myself. So anyway, in my capacity as Maron, I don't hope to be as spontaneous or as funny as he is, but I do hope to talk about something that is a little bit off of the sort of charted path of CS106A's curriculum.

So one way of describing the things that you're learning in 106A is that you're learning how to take the ways that you already know of doing things that may take you some time and maybe error-prone in the doing of them and tell a computer precisely how to do them so that you reap the benefits of a machine being able to execute those instructions. And they'll be faster. There'll be fewer errors as long as you get the instructions right in the first place. And that's got to be an exhilarating feeling. It's got to be empowering, or at least it was when I was sitting in your place as a freshman, to know that, all of a sudden, if you have a good idea – if you can come up with a procedure for solving a problem that you've always had and tell a computer how to do it, then the computer will be able to do that over and over and over again. All right? So that's 106A in a nutshell. So it's sort of – it's not a perfect metaphor. What does it mean to run breakout by hands? I don't know.

So what I – the distinction I'm trying to make between what you've been learning so far and what you will learn, hopefully, in this lecture – and certainly if you go on to CS106B – is that there are things that computers can do that you couldn't possibly have simulated by hand. Computers can handle so much more data than you could ever manage to sort through on your own manually that it's worth you're learning a bit about how to make them do that – how to think about instructing a computer to do something that you couldn't possibly have done. So today, we're going to talk about two of the most important kinds of those problems in computer science, and I'm going to stop just talking at you and show you some examples in a second. But those two topics are searching and sorting. How do you find something in a set of things that you have? And how, if you could impose some structure on that set, would you both find it more quickly, and – I

guess I got ahead of myself – how would you go about imposing the sort of structure that would help you find it more quickly?

All right. So if I'm seeming to ramble, that stops now.

So searching and sorting – I say they are important, and I hope you believe me, and you think that for that reason, it's worth talking about for an entire lecture. But they really are just a way of getting to talk about something a little deeper, which is this concept of algorithmic efficiency that we haven't said much about in the class so far. So that's the third part of this two-part lecture. What's the deal with searching? Well, searching doesn't quite fit the mold as something you couldn't do by hand. You find things all the time. Chapter 12 of the book looks at two operations of the searching and sorting. This is pretty generic. All right. So searching is simpler. You can define a search problem – say you have an array or some other collection of things, and you have something you want to find. The typical way this is done is that you want to find the index into the array – where that element was – or, in a more generic case, you want to find out if that element is even in the array. So you want to have some way of determining that it wasn't actually found.

And that may be all you care about. It may be the case that, if the search routine returns anything other than negative one, you don't actually care what its value was. But we adopt – I will adopt the convention for this lecture that if you don't find what you're searching for, then the method that you wrote to do the search should return a negative value since that's not a valid index into the array. All right. So if you have a set of things that you're trying to search through, the easiest way of doing that is just to look at all of them and see if each one, in turn, is what you're looking for. That's called a linear search because, to sort of pre-figure what we're going to talk about at the end of the lecture, the number of operations you have to do to find the element that you're looking for is linearly proportional to the number of elements that you had to begin with. Now it may not be obvious right now that there's something better that you can do, and with an arbitrary set of data, there is not anything better that you can do. But this procedure that I've written up here is an implementation of what you already could have written – a procedure for finding an element in an array of integers.

So there's nothing very tricky about this. If it had been a string, you might have even avoided having to write the function and called it something like, "Index of" with a character. And inside of "Index of", though you don't have to write the code, it would have had a four loop that iterated over each of the characters of the string, tested each one for equality with the key that you were looking for and then, presuming it found one that equaled that character – or in this case, that integer – it would return the index of it. And if the four loop completes, then you must not have examined any elements that matched, and so you return negative one. Okay? So here's a simulation of how that would work, although the – leaving nothing to the imagination here, this is pretty easy to sort of envision.

So we have this array of primes, and it has 17 in it. So we are going to expect to find that this linear search shouldn't return negative one. But the second one, we're looking for 27 – which sort of looks prime, but of course, it's not because it's nine times three. All right? Okay. So we called linear search, and here's our new stat frame, which has the local variables I and the parameters P and array in it. This is sort of Eric Robert's – who wrote the textbooks – way of displaying the execution model of JAVA. All right? So we're just looping over it, and we're testing as I equals zero and then one and then two and then three and four, five, six – where should we find it? Well, we were looking for seven – no. Right now, we're looking for 27. So we're just gonna go to the – I think there was a problem with the way that – well, anyway. The console here prints out that, indeed, we found 17 at position six. There may have been a problem converting PowerPoint slides to Keynote slides, so I apologize for the sort of shakiness of these animations. But the content of the slides should be adequate. Cool.

All right. So now we're in our second called linear search, and the only thing that's changed at this point from the beginning of the first called was that the key is different. So I is still gonna start at zero, and it's still gonna iterate over all of the positions of the array. But we're gonna go through the entire array, which means going all the way up to ten where I is no longer less than ten. And we didn't find 27 anywhere. So I hope this is all fairly clear. But these are the results that you would have expected. So we found the position of 17, and then we looked for 27 but didn't find it and returned negative one. Cool.

How many people think they could have written that in their sleep? That that was too much time to spend on such a simple idea? All right. Well, it gets more interesting from here on out, but – so talk to me. What is the problem with linear search?

**Student:** It takes a lot of time.

**Instructor (Mehran Sahami):** It takes a lot of time, right? It takes as much time as you have elements. So if I asked a question, which was – if I knew the SUID number of one of you in the room, and I asked, "Is this your SUID number?"

And you say, "No."

And I ask you, "Is 05179164 your SUID number?"

Is it yours? Be kind. Tell me it's yours.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It's not. So it actually turns out that it's mine, so I would have had to search the entire room before I finally got back to myself if I didn't make that shortcut. Okay.

So another illustration of what is really a very simple idea – that if you’re forced to look at all of the elements that you are searching among, then that’s bad in and of itself if you could do better. So we’ll see if we can do better, but here is a rather larger data set – 286 area codes – the three-digit numbers that come before the rest of the seven digits in your phone numbers.

But we’re trying to find 650, which is this county’s area code. So here’s a larger array. What would happen if we were searching in this? Well, it depends. If we have a fast computer, it may not matter. These are – there’s only about 300 numbers here, and doing something 300 times is something that computers are reasonably good at.

But I thought I’d put together this sort of compelling example of why this can become a bad idea.

All right. So I wrote a program in JAVA, which has this implementation of linear search that we were just looking at. And the difference that you may notice is that I have an instance variable, which is sort of behaving as though it were an array, and you’ll see why that is in a second. But all that I need it for is to get values from it at certain positions, and I wanna know what its length is so that I can iterate over it.

And I’m also – so why is it called DIS? It’s short for display, and it’s this – an instance of this number display class that I’ve written. And if there’s time and interest, I’ll talk about how that was implemented. But it’s intended to sort of mimic the behavior of an array and, at the same time, show you a linear search in action.

All right. So this is a dialogue program that’s just reading line to ask me for some input. And here are all of those numbers.

So I slowed it down – even if this looks like it’s going relatively fast – so that you could see it sequentially considering each of these numbers. All right?

You may notice that you can – you could probably beat this. Right? It’s going slowly enough that, since these numbers are already in sorted order, you can jump ahead.

So what is it – I want you to think about – while you’re waiting on this to finish, I want you to think about what it is that you’re doing when you search for the number in this list of numbers that happens to be sorted, which allows you to go so much more quickly than the computer. All right.

So why didn’t we find 650 here?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It wasn’t even there? Yeah. Tricky. Let’s fix that.

Where do we wanna stick it? Probably between 630 – they skipped whole swaths of – I just pulled this off of the Internet, which is notoriously unreliable. All right. They probably aren't gonna let this run all the way through the next time. But 650, as you will notice, is now right here. So it would eventually be found. I'll let that tell me when it's finished.

So now I'm going to switch over to a different kind of search, which I hope has something to do with the way a human looks at a sorted array and finds the element that they're looking for quickly. But – no. Help me out before I just show you that implementation. What is the insight that lets you find elements more quickly if there's some structure – if there's a sorted order to the elements that you're looking for?

Go ahead.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Okay. So if I start with the first number in the array, and I ask myself, "Is the number I'm looking for higher or lower than the first number in the array?"

What's the answer probably going to be?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It's probably higher if it's a sorted array, right?

So I know that's not quite what you were getting at, but I'm pushing on the definition of this procedure so that you – would you like to add a clarification to what you were intending?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah. The number in the middle. Okay?

So if you look at the number in the middle, and you see that that number is less than the number you're looking for, there's a whole set of numbers that you now don't have to worry about. Right? It's all of those numbers up to and including the number in the middle of the array. Right?

So now you're looking at half of the array, and in some sense, your problem is simpler. In fact, it's a lot simpler because if you apply the same procedure again, then you can cut that remaining half in half, and you can cut the remaining half in half. Right?

So this kind of searching is called binary searching because you make a binary choice at each decision point, and you can then cut the space that you're searching in half. So what would that look like if we wrote it out in code?

Well, if you trust me, then it would look something like this. And the insight behind these admittedly too-short variable names is that we're keeping track of the LH or left-hand side of the portion of the array that we're considering and also the right-hand side. So if you look at these two initializations, it should make sense that you start with the left-hand side as far left as it can be at zero and the right-hand side at the index of the very last element in the array, which is one less than the length of the array. Right?

And so you may already be able to anticipate that we're going to sort of move these in closer and closer to each other until they identify an array that has only one element or, potentially, zero elements, in which case we would decide that we were not able to find the element that we were looking for. All right?

So we're going to do this halving procedure until it is the case that LH and RH have collided with each other in some sense. And the idea of getting the middle element is captured by this next line here where we take the average of left hand and right hand.

Now searching is such an important operation, and it's so often needed for very large data problems that talking to people in industry who have to do a lot of searching – you may hear someone say that there's a bug on this line. And it's the sort of bug that has nothing to do with computer science in the abstract, but it's just one of those details that is unfortunate – the bug being that you don't really have to add LH to RH before dividing them. You can divide each one by two and then add their halves to get the average.

This is just sort of a side comment, but why might it be problematic for you to add them together?

All right. If this rings no bells – why would you say?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Integer division? That's – I think if you work with some small representative cases, you'll see that that doesn't prove to be a problem, although that's a good insight. But there is a maximum value – yeah?

**Student:** What if they're too large?

**Instructor (Mehran Sahami):** What if they're too large? So an integer can be as large as – an [inaudible] integer can be as large as, like, four billion. But what if each one of LH and RH is, at some point, three billion or greater? Right? And there's going to be a problem adding them together.

But we've already spent too long talking about that. I would just throw it out as sort of an interesting detail. But after many, many years of using a binary search that looks exactly like this, only recently have people begun to realize that little problems like that can sort of spell disaster with really, really big problems – really, really big sets of data.

Okay. So anyway. That is somewhat an immaterial point.

We now have the index of the middle of the array – or at least the middle of the portion of the array that we’re considering right now – and we ask our sort of array-like object what number is at that index. And if it’s equal to our key, then we’re done. We can just return that we found the index that we – of something that matched the key. So in the linear search case, if there were multiple copies of the key that we were looking for, which one would be returned?

The first one, right?

Do you have any insight yet about which one might be returned in this case?

It’s not at all clear to me. And in fact, binary searches generally don’t guarantee anything about which element they return if there are multiple copies.

Anyway. Assuming that we don’t get lucky, though, and that we don’t find the element that we were looking for on our first try, then we have to modify the size – the part of the array that we’re looking at. So how do we do that?

Well, we need to determine, as was suggested, whether the element that we’re looking for is less than the middle or greater than or equal to the middle. And in the case that it’s less than the element of the middle of the array, then we want to adjust which side.

So we know that it must be in the first portion of the array, and we’re going to ignore the second portion of the array. So the right-hand marker that we’re – is what we need to update this time. So that’s what this does. It updates the right-hand marker to mid minus one. And why is it minus one?

These sorts of issues can be really frustrating to be off by one errors in otherwise already complicated algorithms.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Okay. Arrays start at zero. Well, could it be the case after we examine this test that the element is actually at the mid position? Okay, why is that?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah. We already – oops. We already checked, and the way that we checked is because it’s just a less than sign. It’s strictly less than, so that implies that the two are not equal. So there may be different ways of interpreting why there’s a minus one here, but my interpretation is that we never want to consider that index again. So we want to move one past the middle index that we now know is not equal to the element that we were looking for. Okay?

So in the other case, we want to adjust the other end of this portion of the array that we're considering, and so we update the left-hand side to mid plus one. All right? Why plus one?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It's a similar reason. Right. We don't want to consider the midpoint ever again. Right?

So at the very least, we've gotten rid of the midpoint. But more importantly, we've gotten rid of everything on one or the other sides of it. Okay?

So you can now see that we are – we've got an instance of the original problem, and when we go back to the top of the Y loop, we calculate a new midpoint, and we do the comparison again. But now we're in a much smaller array, and eventually, the only directions in which RH and LH can move are directions toward each other. And no matter what happens, unless we go ahead and return mid, we're changing one of them each time. So you should be able to see that the array that we're considering is definitely getting smaller. Okay?

So that's important. This is a Y loop with an interesting test. So it's always worth considering whether the Y loop is really going to exit. But in this case, I hope that I've persuaded you that, in fact, it should exit.

Okay? So what will the search look like if we use binary search now. I'll just change this name. All right. That was quick. We could have fiddled with the delay that I added. But holding the delays the same, it should be clear that the difference in running times between these two algorithms is pretty substantial. All right?

So that's searching. Any questions about what I've explained? Yeah.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** That's an excellent question. So it depends on – I could put the question back to you. I haven't explained a good way of thinking about exactly how fast these two operations are. And I haven't even shown you how one would go about sorting the array, much less how fast that would be. But if you were going to do a bunch of look-ups, then the savings that you would reap by being able to search in a sorted array would potentially, for some number of look-ups, dwarf whatever cost it took to sort the array the very first time. Right?

But if you're in a situation where you've got all sorts of arrays, and for this array you want to find this element, and next to it's going to be some completely different array – some completely different element that you're trying to find, then your intuition – or the intuition that I imagine is behind that question that it's more costly to sort the array and then search it is perfectly valid. It would be better in those situations just to do a linear

search. And sometimes, that's all you can do. There are situations where there's no easy way to put the elements into sorted order. It's not clear what that would mean, so in situations like that, sometimes linear search, which is sort of our null hypothesis – our default algorithm – is the only thing that's available. All right. So we've already gone through the idea of binary search. Here is a prose version of the algorithm that we just walked through. And here is – I inherited these slides from someone else's, as you may be able to tell. But here is – you can see that part of the array is no longer being considered. Successively, that portion gets smaller and smaller and smaller until finally, there's only one element. Okay? Okay. So we've already talked about how linear search depends on its running time on the number of elements. Do you have a question?

**Student:** Yeah. I was wondering what type of search mechanisms does [inaudible] use [inaudible]?

**Instructor (Mehran Sahami):** Yeah. Yeah, so – we'll talk about exactly how efficient binary search is, but the basic insight is that it – the number of steps that you take is the number of times you would have to divide the original number of elements by two in order to get it down to just one element. Right?

So that's a logarithmic proportion. Right? That's just sort of jargon, and actually, I didn't think much of logarithms until I started being a computer scientist, and now they make a lot of sense. They pop up everywhere. But I'll talk about that in a second.

But yes. Okay, so back to your question about the hashmap. The thing about a hashmap is that, for all you can tell, it takes the same amount of time to find the element or to insert the element that you're trying to find or insert, no matter how many elements you put into the hashmap so far. So how is that even possible?

So the book talks about this in some detail, but the basic idea is that the hashmap has a bunch of buckets behind the scenes, and it can figure out in constant time – that is in an amount of time that doesn't depend on the number of buckets or the number of elements – which bucket the element that you're inserting or looking for should go or should be found. And then hoping that the number of things that end up in that bucket – presuming you have enough buckets – is relatively small, you can just do a linear search on the list of things for each bucket.

So the idea is that very quickly, you pair it down to a single small list to search in, and then you just search in that list. So for very short lists – or I guess I should say for very small numbers of elements – I have said actually has a lot of overhead that may outweigh the cost of just doing a linear search on the elements. But for – again, the way to think about these algorithms in relationship to each other is to consider really big examples and think about how the running time would sort of grow as a function of the input.

All right. So let's say a little bit about binary search and its efficiency. I've already suggested that we divide the number of elements that we're considering by two each time. So if you're mathematically inclined, you may notice that we want to solve this

equation at the bottom where we find  $N$  such that  $N$  divided by two to the  $K$  times – no,  $N$  we already know.  $N$  is the size of our data set. We want to find  $K$ , which is the number of times we divide  $N$  by two. Okay?

So we can rearrange that by multiplying each side, and then the only way of solving that is to find  $K$  by taking the base two logarithm of  $N$ . That's – you could find it by inspection, but you could also type this into your calculator. There's an equivalence between these two things that, if you've forgotten, may be worth remembering now, but that is certainly not something we'll ever test you about.

I would also say that the content of this lecture is not something that you are going to be responsible for in your assignments or the exams. So if you find it interesting, then let that be its own reward. And if not, I'm terribly, terribly sorry. Okay.

So what – how do we compare these two numbers? So you look at an expression like log base two of  $N$ , and that may not mean anything to you. It certainly didn't mean much to me. But intuitively, if you remember that what we're talking about is dividing this array in half repeatedly, then it should seem like that's a lot faster than searching every element. And indeed, it is considerably, considerably faster. If it had been log base ten, the number would be approximately the number of digits in –  $NN$ , if you wrote it out as a decimal number. In log base two, it's the number of binary digits if you were to write it out as a binary number.

So you know that it's easy to write very large numbers with a string of digits. And in this case, if we scale in all the way up to a billion here, then a billion has approximately 30 binary digits – or another way of saying this is that the logarithm of  $N$  base two when  $N$  is a billion is 30. So notice how log base two of  $N$  is growing relative to  $N$ . The difference is just enormous. So by imposing a little bit of structure on the data that we wanted to search through, we made it much, much easier to find elements. Okay. All right. So that's searching, and I hope the points there are well taken because sorting is now sort of like the dual concept of the searching that we've just been doing. If it's easier to search when we have imposed some order on the elements that we're trying to search among, then how do we go about imposing that order? So 106B spends quite a bit of time talking about different sorting algorithms. There's a great assignment where they give you a program that implements these sorting algorithms, but not the source code. And you just have to run it on a bunch of different kinds of input. And given what you know about how those sorting algorithms behave, figure out which one of them is which. So it's sort of like a practical lab from a natural science. It's the only time I've done anything like that in the computer science world, but if that sounds exciting, think about taking 106B. All right. So here's the sorting algorithm that corresponds, perhaps the best, to the way that the humans sort things. So if you had a deck of cards, and you wanted to put those cards in – back in order after sort of – after shuffling them, you would look at the first card, and you'd think to yourself, "Gosh. This is great. I've already got another deck of cards that is sorted. It happens to have only one card in it." So it's sort of sorted vacuously. But starting with this sorted deck, can I add more cards to it and keep them in the same sorted order? Right? So you would take the next card off of the larger deck, and

you would either put it in front of or behind the card that you had. And now you'll have a deck of two cards that are sorted with respect to each other. And you would continue on in this way until you had exhausted the original deck, and you had a fully sorted secondary deck. Would anybody go about this differently? Is there a better way of sorting? Okay. Well, that's certainly what I would have done, and that's what is implemented by this as long as you trust the defined smallest function. So we just step through the array – and this is a bit different. We're not – we don't have two separate arrays that we're building up. But for each element, we look for the smallest element that's after that. And on the assumption that there shouldn't be any smaller elements after that, if we find such a smaller element, then we swap it with the element that we're currently looking at. Right? So by the time we get to the end of the array, and we swapped out each position with anything that was smaller than the element at that position, then our array should be sorted. Right? As a bit of review, why do we call swap elements? You'll have to imagine how swap elements is implemented. But based on what you can assume, why do we have to call swap elements with three arguments? Why couldn't we just pass an array sub LH and array sub RH?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Yeah. It only passes N. So those are just the values from the array, and our goal is to modify the array in place. Right? So the swap elements function needs to know about that array, too.

Okay? So I don't know if I can trust the simulation, but I actually coded up a version of select and sort myself and an eclipse, so let us see a somewhat more interesting simulation of how that would work.

Okay. So I'm gonna use a much smaller file because as I have – as I've already suggested, sorting takes more time than even linear search. All right.

So just to give you some bearings as to what's going on in this program, I have this number display, which is just a G canvas that has these operations that let us highlight numbers and swap them around. And I'm creating one of those down here as an instance variable and adding it and run and then initializing it with the contents of the file that's typed in. Right?

So if I were to just run the program as it is right now – I use a smaller text file. It should slow this down, but it's going to take enough time anyway that you can see sort of what's happening.

Each number is turning red when we examine it, so in some sense, you can think of the running time of the program as depending on the number of times, given the input that you have to examine elements of the array. So the way I structured this was just that examining any element of this pseudo-array would have the side effect of making it a little bigger and turning it red – and also pausing for about a quarter of a second.

Okay. So actually, I think it would be worthwhile to slow that down and look at what's happening. The value of constants – that was easy. Okay.

So we're considering four right now. One of the things that this doesn't keep track of very well is what number we're considering. Right? But after four, the smallest number that we found was two, and so we swapped them out. Two happened to occur immediately after four. And so we considered four. Again, this is just sort of a special case where four happened to be the – happened to be swapped into the position that we next considered.

So now is anything less than eight? Well, seven is. Oh, four is even less than seven. Now we swap the eight with the four. It's going to turn out that nothing is less than – or everything is less than nine – five in particular. Okay.

So how long did that take? Was anybody timing that? What would it mean? I mean, if we could just change the amount of delay arbitrarily, right, we could have made this take any amount of time. And in some sense, when computers get faster and faster because the hardware people are doing their jobs, that's what's happening. The delay that someone artificially – or in that case, not artificially – put into the system is just being reduced. Right? So everything seems to run faster. The rising tide raises all boats. So what would – I mean, would it even be useful to have timed that – to have a number of seconds representing how long it took to sort these ten elements? It's a lot harder to count when you're standing in front of a bunch of people. But it – yeah, I think that's ten. Okay. Right?

So clearly, it's sort of a result of a number of factors exactly how much time these operations take. So I – with what remains of the lecture, I'd like to give you sort of a better way of talking about how fast a program is. So if you buy my contention that you can't just time your program without making sure that all of the other factors stay the same, then what ideally we would want to be able to say is that, given some kind of input, about how long is it going to take? And in particular, how long would it take if, for instance, I doubled the size of the input – if there were twice as many numbers here.

For linear search, if we were searching for the five in this array – or let's just take the pathological case. If we were searching for the nine, we would have to examine all of the elements before we found it. If we double the length of the array and made sure that nine didn't appear anywhere except the end, again we would have doubled the time that it took to find the nine. But would that be true in the binary search case?

Well, no. We had only – we doubled N. And remember that the running time is, in some sense, proportional to the logarithm of N base two. So we wouldn't have doubled the running time. We would have only increased it by a factor of logarithm of N times two minus – or divided by the logarithm of N. I didn't say the base there because it actually doesn't matter. If you divide two logarithms by each other, you get the same answer no matter what base you took them at originally. It's sort of a nice property of logarithms.

Okay. So I think my simulation was better. Feel free to disagree. But I'm trying to skip – oh. This actually has a little finger pointing to the elements. Maybe there's something to be said for this. Anyway.

I'll post that program online incase anyone is curious. This is taking a bit longer to blow through than I was hoping. All right.

So how efficient is selection sort? Well, it's not okay to just give a number seconds. We want to give some kind of proportion, as we did for the searches. Right?

And if you noticed, for every element of the array, we had to consider every remaining element of the array. If we wanted to be a little less efficient in some sense, we could have just – well, I was going to say we could have considered all of the elements and just ignored the ones that did not actually come afterwards. But we can calculate sort of in a rough sense what that proportion should be in this case. But before I do that, here's a slide where I attempted to do the timing test. So I held all the factors constant that I could, except that I changed the size of the input. So I went from having just ten elements, which took me a very, very short amount of time, to having 1,000 elements, which took me a matter of minutes – 10,000 elements. Excuse me. Right? So this sort of illustrates the point, and it shows in a heuristic sense how the running time grows with the size of the input. But what's going on here? The – in the far column, we have the standard deviation, which is a measure of how different the timings were. And if you look on a particular row for any of these trials, all of those numbers are really supposed to be the same because I'm sorting the same number of elements. Right? But why are they different? Why such variation?

So I'm sorting 100 elements, and sometimes it takes .146 seconds, and sometimes it takes as much as .176. Just chalk it up to the randomness of the universe? I guess, but it's even easier to explain that problem away because your computer is doing so many more things than just running your eclipse program. It's written in JAVA. It may have to pay attention to activity on the network. You may be running iTunes at the same time, and it has to load a file in from the disc, which takes away time.

So these time measurements are just in terms of absolute time, and for that reason, they don't give a very accurate indication of the job that your program was doing all by itself. All right.

So [inaudible] is sort of crazy values. For 1,000 for the most part, it took just 13 seconds almost as much as 20 and almost as little as ten. The average was 21, though, because one of those time trials took 81 seconds. Right? Just because something else on the computer was happening that was taking time away. Right?

So clearly, we can begin to supply a better intuition for the question that was asked earlier about whether it would be okay to sort an array and then search it if all we wanted was to do a single search. All right.

So on the very first time through, we're looking at all of the elements in the array after the first one. How many are we looking at on the second time through?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** One less than that? Okay. And on the third time through, one less than that. So the total number that we're looking for is – or the total number of times we have to examine some element of the array is  $N + N - 1 + N - 2$  all the way down to one. Right?

Does anyone know how – what this sums up to? There's actually a closed form for it. Yeah?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Okay. Yeah. So if you work it out with small examples, you'll find that this is true, and you can probably then convince yourself of it. This sums to  $N \times N + 1 / 2$ . Okay?

So how do we make sense of that? How do you look at that and tell intuitively what kind of algorithm this is – how efficient this selection sort is?

Well, in computer science – this is one of the things I love about computer science – when we're talking about algorithmic efficiency because we really don't care about any of the sort of details. We only care about the aggregate behavior of these programs. We can drop pieces of information like this, too. Right?

Who cares if the algorithm is twice as fast or half as fast as another algorithm? If it's – you just need a computer that's twice as fast or half as fast. But if, say, an algorithm takes the square of the amount of time that another algorithm took, then really, getting a faster computer is not the answer for sufficiently large inputs.

So – whereas we can drop constant factors like two and just pretend that this is – well, you can see that this simplifies to  $N^2 + N$ . Right? Not only can we drop constant factors, but we can drop anything that doesn't really contribute in a big way to the running time. So it doesn't even matter that we have this plus in here. So we can look at this and sort of think of it intuitively as a quadratic that is squared function of the size of the input.

So what does this mean? It means that if we double the size of our input, then the selection sort that we just walked through is going to take the square of the amount of time that it took originally. All right?

So this is a computation of that. You can also think about it geometrically if you treat each of the accesses of array elements as these circles. All right.

So this isn't quite as bad as the difference between – no. Actually, I should say that we – to demonstrate the difference between linear search and binary search, we had to make  $N$  really large, while  $\log$  base two of  $N$  was still relatively small. Here, it's sort of the other way around. I guess explaining slides that I didn't write is a bit of a waste of time for you guys. So do you have questions about how we did that calculation? Why I modified that to make it simpler?

If you do afterwards, then feel free to ask.

So I sort of wanted to leave you with some insight about how we could possibly do a little bit better than selection sort. And it turns out there are algorithms that do substantially better than selection sort. And one of the easiest of them to describe is an algorithm called radix – or radix sort. And it comes from back in the days when we had these punch cards with holes in them to represent numbers. And – so say you had a bunch of three digit numbers that you wanted to sort. Well, the basic idea with radix sort is that you would set the – you would have this machine that could sort of read the holes and sort the elements into buckets such that every – the elements in this case being the cards. So each one of the cards is sorted into a bucket according to its – the first digit of the number that it represents. Right?

So you've got these buckets that have cards that aren't in any order within the bucket, but then you at least know that each bucket contains all the cards that are less than all the cards in the next bucket. So you can take them all out and stick them back together as a single thing, and then set the machine – the machine has this thing called the hopper, which is just a place to put the cards – all right. So run the machine again, but the second time, you run it – have it sort them in the order of the second digit in the numbers, and assuming that we have three digit numbers, then – now the cards are all going to be sorted within their bins with respect to both the last digit and the second to last digit. And so if we do this just on more time and sort the resulting set of cards on the one remaining digit, then we end up with bins that actually represent a complete ordering. If we took out the dividers between the bins, we could stick a stack of cards together, and that would be ordered.

So this again is an animation that didn't come through so well. I think we're about out of time. So I wish that I could have given a better intuition about how much better radix sort is than selection sort, but thank you very much for paying attention to me. And have a great Thanksgiving.

[End of Audio]

Duration: 51 minutes

## Programming Methodology-Lecture24

**Instructor (Mehran Sahami):** So welcome back! So yet another fun-filled, exciting day of 26A. After the break, it almost feels like – I came into the office this morning, not that I wasn't in the office during the entire break, but I came into the office this morning and it felt like a new quarter had started. And I was like, oh, it's been a whole week. And I'm sure for you, it feels like you just wish a new quarter was starting because we still have two weeks left.

So a couple quick announcements before we get into things. One is there is one handout, which is your section handout for this week. And kind of one of the themes of this week is bigness. In some sense writing bigger programs, bigger data structures, that's the whole deal. And we'll kind of talk about that as we go along.

Another quick announcement, just wondering how many people tried the Name Surfer demo online and had a problem with it? You folks, yeah, we updated it. So evidently there was some issue that only shows up on Windows XP with Java 1.6. And like, if you had a Mac you didn't see it, if you had Vista, presumably, you didn't see it, if you had Java 1.5 you didn't see it. But in that one case, it would come up, so the name surfer web applet demo was updated a few days ago, I think on Friday, maybe. So now it should work for everyone, hopefully. If you still have an issue, let me know. The only thing that you'll see now, though, if you'll try running this applet, is that the interactors, instead of it being on the south border of the screen are on the north border of the screen. That was just a little hackler we had to put in there to get things to work. The functionality is exactly the same.

As a matter of fact, if you want, you can put your interactor and border on the north instead of the south. It'll make no difference to the rest of your program other than where you say south for adding your interactors, you say north, that's the only place it makes a difference. But you actually see that in the web applet version, the interactors are just in the north border instead of the south. Otherwise, it doesn't make any difference. But in case you saw that and was like freaked out, there's nothing to worry about. Okay. So also, I hope you had a good break. Just wondering, how many of you actually enjoyed their week break? Good time. And how many were working most of that break? Yeah. Good times. Hopefully, it didn't cause you too much pain, but if it did, hopefully, you're like, all caught up or ahead of the game in all your classes, now, so life is good.

So I want to spend a little bit more time talking about today, well, actually a lot of time talking about today, is thinking about data structures, building large-scale data structures. And we begin to talk about it just a little bit before the break and it's been a while so we're going to review it a little bit and kind of build up even more. But one of the things we talked about, in the past, right, what a lot of our computers do is they manage data. They manage lots of data. And in fact, I would venture to guess that there's a whole bunch of applications out there that manage a whole bunch of data about you, but you may not have thought about all the data they actually manage. So some of the things that

actually come up, for example, online stores, right. Anyone actually bought anything online, just wondering. Yeah.

There's a huge amount of data's that involved with that. Not only the particular transactions you make when you buy something, but keeping track of accurate transactions, figuring out things like people who buy product X also, tend to produce Y. All of that is data management. And what makes those companies successful is they just do a very good job of managing their data. Okay. There are other things like, I'm almost frightened to ask, but social networks, like, Facebook, or MySpace, or ORCHID, or Friendster, or LinkedIn, or you could just keep going on. Anyone on a social network? Just wondering. Yeah. That's good because your next assignment is going to be to implement one so you can see what it's actually like. But that will be coming in a couple of days. And they're not that hard, really. But what it is is a data management problem. Right? And it keeps track of things like who you are and information in your profile in the social network, and who your friends are, and all that happy news.

Or you know, even things like a friend web search, right? There's a huge amount of data you need to be able to keep track of to be able to web search, right? So all these things are all about managing data well and so part of this class, right, is you've got a whole bunch experience in terms of building up code, and different kinds of classes, and doing nice things with user interfaces, and the whole deal. And one of the things that we need to spend a little bit more time on is talking about how do you manage lots of data and then do something interesting with that. Okay. So here's some principles to think about, if we think about good software engineering, some of the principles of thinking about data kind of in the large. Okay.

When you think about keeping track of lots of data, one of the things you want to think about is the information you want to keep track of, what are the nouns you want to keep track. And you're like, I don't want any nouns, what do you mean by the nouns? Let's say I was writing an application that was an online store, to keep track of, oh, let's say, music. And so one of the things I would want to think about is, where are the nouns that are associated with music? You're like, okay, now you're really getting weird. No, it's pretty simple. Things like a song, right, is a noun, that's associated with music, or an album, or an artist, right. And so what you want to think about is the things that are the nouns in the domain that you're dealing with oftentimes end up translating into what your classes are. So you may end up having a class that keeps track of information about a particular song or class that keeps track of information about a particular album.

So the good linguists out there tell me we not only nouns but we also have verbs, not unless you happen to be talking to my son, who seems to only have nouns, but that's a different story. And he loves jarens by the way. But like, why are you telling me this? Just cause it's fun, because I just spent a whole week dealing with it. In terms of verbs, these are oftentimes the methods that are associated with your classes, right. So when you want to do something, some noun takes some action, which is a verb, which is some class, has some method that operates on that class. So at an abstract level that's what you want to think about in terms of high-level principles of design. Now, there are some other

sort of more concrete things that you might want to think about, things that have to do with what are the characteristics of the data you actually want to store so one thing that comes up, oftentimes, is thinking of the notion of having a unique identifier, identifier. What do I mean by unique identifier? All of you have unique identifiers, whether or not you like it or not, as a result of being at Stanford. Your Stanford University I.D. number is a unique identifier for you at Stanford. Every student has an I.D. number. Okay. So it identifies you and it's unique. No two students share the same I.D. number.

So you get issued this number when you show up here and you have it for life. When you leave it's still with you. I know, I left, I came back, I have the same student I.D. number. It just exists and this uniquely identifies you. And in different cases you might want to think about what are unique identifiers. Right. So in some cases, for example, if you had a social network you might consider the names of people and not the social network to be identifiers, or say the names of their profiles, for example. In other cases, you might have something different. If you're managing a store you might have some I.D. number for books, an ISBN number, or if you're keeping track of music, you might say that the combination of the songs' name and the band that plays it is a unique identifier for that song. In some cases the unique identifier can be a combination of things. But if you think about your data having a unique identifier that also gives you some insights about what kind of data structures you might want to use to keep track of certain things. On other unique identifier that some of you've already grapple with is saying Name Surfer. Right? If you think about the data in Name Surfer what's the unique identifier there?

**Student:** Name.

**Instructor (Mehran Sahami):** Name, right? Name is a unique identifier and for every name you have some list of values associated with it, which was the rank of that name over the last century in terms of how popular it was for names. But every name, well, I shouldn't say every name has some value associated with it, but every unique identifier in the system has some value associated with it and only one set of values. And so the important thing to keep track of there is when you actually are doing your Name Surfer assignment to fact of this thing is a unique identifier can potentially help you keep track of the data that you're using. And we'll sort of go into that as we go along in the class. Okay?

So some other principles we can kind of think about in terms of designing data structure, in terms of actually doing the design, there's some questions you want to ask yourself. And the questions you want to ask yourself is, are you keeping track of some collection of objects? Right? So there comes some collection of objects through data that you want to have. And if you have a collection of objects, say in an online music store, you might have a collection of songs that you want to keep track of, this word should be a tip off too, that perhaps there's an interesting collection that exist in Java that would be a way of keeping track of that information. It may not be in Java if you're programming in some other language. But the fact that Java has something called a collection and the reason why they gave the name of collections to a certain group of stuff is because they're used

to keep track of a collection of objects. And the question that you ask yourself then is what collections do you actually want to use? Okay.

So with that said, what we can do is spend a moment, and it will be a brief moment, revisiting the collection hierarchy. Right? You've seen this picture before but I'm just showing it to you again, because the last time you saw it was like two weeks ago, which is a lifetime in a quarter. Right? I think it too, it's about a fifth of the quarter. You're like, oh, what was I doing two weeks ago? Was the break out, was I learning Printland? No, no it wasn't that long ago. But what you were learning about, a little bit, was collections. And o there are some collections, for example, like an ArrayList that going all the way up the chain of the hierarchy is itself a collection. Or there are other things, for example, like a HashMap. And a HashMap, if you said hey, I have some HashMap, the set of keys in that HashMap ends up actually being a set, which happens to be a collection. Okay. And so what you want to think about, do I have different things that I can keep track of? Like an ArrayList is one way to keep track of things. A HashMap may be another way of keeping track of things. When is the appropriate time to use one thing versus another? And so when you want to think about the appropriate times of one versus the other, you want to think about what are the methods that a collection provides to you. And it turns out all collections that implement the collection interface, like the ArrayList or the key set of the HashMap, have all of these properties.

And some of you have seen them before, but just to review. You can add a value, right? So this is a parameterized values type. Like you can have an ArrayList of strings and you can add some value to it, and it adds it to the collection, and little did you know, or maybe you did know, but at the time we didn't really care about it, was it returned a bullion. Most of the times we just returned the bullion, we didn't care about it. But actually returned true the collection changed. So in an ArrayList, it always returned true because when you were adding a value, it didn't care about duplicates, it would always just add them to the end and always return true. Some collections, like sets, actually don't allow you to have duplicates. So if you try to add something to a set that already has the value you're trying to add, it will not change the set and return false, because it says, hey, I already have that value and nothing changed.

A couple other things that you should know about, most of these you've seen. Remove, removes the first instance of an element as it appears and returns true if a match is found or returns false if it didn't find anything to actually remove. And clear, basically, just sort of nukes the whole collection. It just says get rid of everything in the collection. I'm done with that and the collection is dead. Actually, the collection is not dead to you, it still exists, it's just an empty shell of what it was before. There are violins playing in the background. And then size, you can get the size of the collection. You've seen this, you've probably used a lot of these before in your programs. Contains, that's an important one, right? You actually want to see if a collection contains some particular value, if a collection is empty. And here's one that's sort of interesting that we talked about a little bit but we didn't actually talk about the fact that a collection or all collections can give you one these. All collections can give you an iterator.

So we talked about, for example, having an iterator over the key set of HashMap. That's one thing we did before. We said we had some HashMap that lets a map from strings from some other strings. And we want, say, hey what I want to see is get a set of all the keys and I want to iterate all over those keys. That's great! You can do that and that's perfectly fine. When we used ArrayList we always had like a four loop and said, oh, from zero up to the size of the ArrayList do something. But if we actually wanted to, we could have an iterator over the ArrayList and then this would give us the elements of that ArrayList one at a time. So because an ArrayList is a collection it can also give us an iterator. And that's just something to keep in mind is that there's common patterns that get used in programming. One of the common patterns that get used is known as an iteration pattern, which again, is an iterator over some collection and you just go through and do something like printout the values for every element of that collection. And if you want to write it in the most general case, you don't care if that collection happens to be the key set of the HashMap, or an ArrayList, or whatever, you just say, hey, you're a collection, give me an iterator and I can go through all your elements one at a time and, for example, print them out. Okay. So there's just simple pattern's that we get into.

Now, you're like, okay, Marilyn, that's fine, you told me some design principles over here, you told me about some collections over here. Show me something concrete, like put it all together. So let's actually put it all together. Okay. And we'll view a little example, which is going to an online music store. And because many names for online music stores are already taken, our music store is going to be called Flytunes cause they're tunes that will fly. All right. Yeah, man, when you're like in your mid 30's you just can't be that cool. But trust me, it is. Okay? So we're going to make a little store that just keeps track of music and albums, and that music and actually lets us keep track of information and prices. And so what we want to think about is what are the things that we actually are going to do in that store, okay? So one of the nouns of that store is going to be a song, okay? So a song is some basic thing that we're going to sell. This is what we want to be able to do with the song. Now, you could say, well, what does that mean, do I have some method called sell? If we're doing inventory management we might not actually have a method called selling a song, but we might, for example, want to add for inventory to do things like add songs.

And similarly, songs, oftentimes, are put together into albums. Okay, so we may also want to keep track of albums and do things like add albums to our inventory. Now, the interesting thing with an online music store that differentiates it from say a physical music store, is you can do interesting things, right? You can actually have songs that are not on any albums. And that works, right? It's kind of like thinking of a single, right. When you go and buy a single somewhere. In the days of yore, you could actually buy a little record single that had two sides on it so you got two songs, so it wasn't really a notion of a real single, single. I guess now, there are like CD singles. But who wants a CD single when it comes down to it? You can get songs that are on albums. At the same time, you can have the same song be on multiple albums, right? That always happens. There's a band, I won't mention their name, but I remember from the early '80s, they had two albums. They had their first album and they had their best of albums, which were half the songs from their first album. Just anything you can do to milk the consumer. But

basically, what that meant was songs can show up on multiple albums. Okay, so we want to begin to think about how that might actually affect our design.

Now, if we think about putting the information together, right, nouns become our classes. So if we're going to have song as a noun, we're probably going to have some class song that's going to keep track of all the information associated with a song. And so just for the sake of brevity, I'll tell you what information's going to be associated with songs that we care about in our store. There's a notion of the name of the song, the band or artists that perform that song, and then a price, because we're going to allow for songs to be sold individually, so individual songs, as opposed to whole albums, have prices. Okay. And you can think about these things and think about, oh, what data types do you want to have for them. Right, so what type data type makes sense for a name, for example, string type, or if you want to have a band name, this would probably be a string. Price is always an interesting one. You could sort of say, well, now, and there's multiple things I could have it be. I could have it be an [inaudible], for example, if I was going to have it be the number of cents. In the simplest case, I'm just going to have it be a double. Even though we know there's no fractional money unless you're a banker, in which case there is fractional money. But we won't talk about that right now.

It was just like Superman III. Anyone see that movie? No. It's not worth watching, trust me. But fractional money does exist outside of movies, bad movies in Hollywood. So that's the information we want to keep track of for a song and then we want to think about what are some of the things that we want to be able to do in relations to those songs. The other thing we also want to think about is our friend the unique identifier. Is there some unique identifier for a song? And this is one of those things you really need to think about the application that you're using, what assumptions you can make. We might like to say that the name is a unique identifier for a song, but unfortunately, there are many songs that have the same name. Okay.

But I would venture to guess that the combination of the name and the band would perhaps be a unique identifier for a song. The only thing is we don't have one string that we keep track of that keeps name and band in it. So that's another thing that we need to think about, and we'll get into code when we get into code, that we need to think about the design of that particular object. The other thing we need to think about is what changes in an object during its lifetime and what doesn't change. Like, so if I have a song its name and the band that made it for a particular song, like, some band can go uncover the song they learn, but that's a different song, the name and the band name don't change for a song. But hey, it can go on sale and you know, I can jack up the price at the holidays and all that kind of stuff. So the price is something that's malleable.

So another thing you think about in terms of the principles of design is, of the data that I have associated with a particular object, what's going to remain static when that object's created and what's going to be potentially changed? And that's what gives you some insight about what's some of the data, for example, that you only get from an object, what's some data that you can potentially set in the object, and if you think about what potentially uniquely identifies that object, what data do you actually need at the time that

you construct the object, right. To say this object is actually some particular unique thing that I care about. Okay. So let's turn that into a little bit of code just to make it a little more concrete. So we'll get rid of our friend, Power Point, and we fire up our friend, Blitz. Ah, and look, a song, how convenient. So here's the information to keep track of a song. It's just a class called song. And what we want to do is keep track of song, the song's name, the band name, and the price.

So when we create the song, one of the things we might do is say, hey, give me all that information to start with. Because if you're going to put some song in your store and you're going to sell it, it better have some song name and band name that I can use to refer to it by, because that's going to be its unique identifier and give me some initial starting price. Now, we might necessarily not require an initial starting price, because it's something that's going to change during the duration of the program, and isn't in support of our unique identifier. But in this case, we're just going to ask for an initial price. The thing we do care about, in terms of the malleability of what's actually in this data structure, is thinking about song name, band name, and price. So song name, we only have a getter for there. There's no setter. Once the object's created, you can't change the band name for that song. You can't say, oh yeah, you know, that was "In Your Eyes," by Peter Gabriel and now it's going to be like, "In Your Eyes," by Kanye West. Like, that's a different song. And I don't know if that's happened. It's probably not a good idea. But the song remains the same, if you're a Led Zeppelin fan, right? And the band name, actually, the band name is also going to remain the same for that particular song. But the price has both a getter and a setter. Right? Because it's something that's malleable. After we create that song, yeah, we might change its price. And because we know that we provide both of those things in the definition of the class.

Now, as we talked about, in days of yore, whenever you create any class it should also have a method called Two String. And Two String just returns a string representation of the data in that class. So this just prints out inside double quotes, which is why we have this backslash, quote, that's a single double quote character, the title of the song in double quotes by the band name, and then it says cost, and it has the price associater with the cost. So it just returns a string to baseline caps lets the data. And here's the private instance variables of that particular class. Right. There's a title, a band, and a price for the title of the song, the band that made the song, and the price of the song. And that's all the information that's in there. But it captures and encapsulates the notion of having a song and what parts of the song are static or can't change, and what parts of the song are mutable or can change. Okay. So besides songs, we also have this thing called albums. Any question about the song portion? If you're sort of feeling good with song, nod your head. All right, good times. If you're not feeling good song, shake your head. If you're awake nod your head. There's a few that's not nodding, but that's okay. That's cool, too. So let's do the class for an album. So the class for an album is another thing we care about. And albums become a little more interesting because an album not only has a name, right, so this is going to be a name, and yeah, the name will probably be some string. And there's also a band, potentially, that produces the album. Now, the interesting thing is the band – you might say, but Marilyn, isn't that redundant? Like, don't I have

some album and it's going to have a bunch of songs on it, and so I already have names for the band for those songs? So why do I need the name of the band for the album?

Anyone know? Want to venture a guess? Anyone have an album that's like this, '80s compilation is the critical word? Right. You can have an album that's band isn't actually a real band name. Its band name could just be something like compilation. And it's going to have a bunch of songs on it, each of one which has a distinct band. Okay. So that's perfectly fine. There's no reason why an album, especially in the online world when you can sort of create mixes all the time, needs to have a single band. And so there wouldn't be a need for having bands associated with songs. We still need to have bands associated with the songs. And potentially, at a higher level, we might want to be able to say, is this whole album by one band, or one artist, or is it actually a compilation. Okay? Now, the interesting part though, is that an album not only has a band and name, but it has a list of songs. So how might we keep track of that list of songs? What would be a reasonable data structure we could use?

**Student:**

[Inaudible].

**Instructor (Mehran Sahami):** An Array, our friend an Array. Well, the only problem with an Array is, right, it needs to have some fixed size. There's some albums out there that are very short, like "In A Gadda Da Vida," Iron Butterfly, there's one song that's one side of the album, if you were back on the LP days, and what a fine album it is. And there's other albums that are just like, oh, look there's like 300 songs on here. Okay.

So an Array with just a fixed size might potentially waste a lot of space. What's the more malleable version we could use?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Oh, yeah. I love it when it's just all around. All right.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** [Inaudible] one, I think. Like that post Thanksgiving. It's like the tryptophan, still like working its way. Yeah. You know.

– albums, to begin with. How do we actually add some list of songs on it. We need to have a way to be able to add songs to this album, and once we actually add songs to a list of songs on the album, we need to have some way of being able to list the [inaudible], or perhaps, iterating over them. The only thing with an ArrayList is enter implements collection interface so that it actually provides you enter it. Okay. So let's look at the code for that, just real quickly and then things will become more interesting, afterwards. Okay.

So here's an album. Inside an album we have an album name and a band, those are the things that are going to start off by constructing an album. So we say here's the initial album name and band, and what I want to do is build up the contents of that album. So it lets you get the album name and get the band name but you can't set them. Those things are fixed. Okay.

The other thing that I'm actually going to assume here, which is something I didn't assume for songs, is that the name of the album is a unique identifier for the album. Because if I can potentially have compilation albums that's a compilation of multiple bands, so the band name is just something like compilation or maybe the band name is empty string, the album name by itself should be a unique identifier.

Now, you might say, but Marilyn, that's not true in the real world. I have multiple albums that have the same title on them. We're just going to assume that for the purposes of what we're doing here, and it'll be okay.

How do we build up the album? We have a notion of adding a song to an album and getting an iterator over the songs on the album. And so the way we do that is we're going to have something called songs. Let me show you songs down here. Songs is just an ArrayList of songs. Okay. And so if I want to add a song to the album, I pass it in an actual song object and it adds it to its ArrayList. And if I want to list out all these songs that are on the album, I ask for an iterator over all the songs on the album. So what I actually get is an iterator over song objects. Okay.

Two strings just returns the title and the band, it doesn't actually list out all the songs. It just says, hey, it's just this name of this title and this band, and that's all that's in an album. Okay. Again, we think about what's mutable and what's not mutable.

Now, to put the whole store together, this is where things get a little more interesting. To put the whole store together, you need to think about what's the store going to do. So let me show you a simple store running and this is the basic text interface for a store. It's kind of like online store circuit of 1995. Okay. So I can list out all the songs, I can list out all the albums, I can add a song, I can add an album. When the store starts, I have no songs or albums in the store. I need to add them all. I can list all the songs on a particular album and I can also update the price for a song. Okay.

So if I list out all the songs. It says all songs carried by the store and says nothing, because there's no songs that the store currently has. And list out all the albums carried by the store and list out nothing here, because there're no albums. But I can go ahead and do something like add a song. And let's say the song I want to add is "In Your Eyes," Peter Gabriel. Any Peter Gabriel fans out there? No? A little bit? Come on. Oh, man. I give up. It's all over. I just don't believe it. All right. We'll say the song is, I say, okay, it'll be 99 cents. Go get it. All right.

So we add a song and if we list all the songs, now we have here's the string representation of a song, "In Your Eyes," by Peter Gabriel, cost 99 cents. We still have

no albums, rights, we just have a particular song that we can potentially sell by itself and we don't have any albums. So we'll come back to this. But this is the basic idea. We want to be able to list all the songs and albums, add songs, add albums, and then list the information for a particular album. Okay.

So if we think about that, what we need is a bigger data structure to keep track of all this information about multiple songs and multiple albums. Okay.

Now, if we want to manage an inventor, the two things we have to keep in mind are also what I mentioned before. A song can exist in our data that is not on any particular album. So as a result it's not sufficient to just say what albums are carried by the store, because some songs may not be on any album, but we still sell them individually. So we need to have some notion of keeping track of a list of songs.

Now there's different things we could think of for a data structure to keep track of songs. One thing is an ArrayList, right. That's what we're using in albums to keep track of a whole list of songs. Another thing we could consider is a HashMap of songs. And so if we think about a map versus an ArrayList, what question that you want to think about gets back to this identifier question, right. Because if you want to have a map, say for example, some string to song, and you want this string to uniquely identify a song, this string needs to be something that is a unique identifier. But a song doesn't have one string that's a unique identifier, it's unique identifier's a combination of a name and a band.

And so all kinds of funky things that are things that people consider. Oh, how can I connect those two strings together? People actually do that in real applications. We're not going to do that here. We're just gonna say, there's too much complexity in dealing with this, we're going to go for a much simpler approach and just say we're going to have an ArrayList of all of our songs and not worry about the unique identifier issue. So here we have an ArrayList of type song, and we'll just call this songs, that's all the songs in our database. And so here we create a new ArrayList of song and we call it constructor. Okay. Now, life in the album world's a little bit different. Besides just keeping track of a list of songs, we also need to keep track of albums. But in the album world the name is actually a unique identifier. And if we want to be able to look up albums quickly, it might make sense to use a HashMap. So part of doing this whole example is to actually show you both ArrayList and HashMap in one application.

So what we could do is have a HashMap that maps from strings to albums where the map, this string, is in some sense the name of the album and this is the actual album object. And we'll call this albums and we can do all the new, you know, la de da HashMap we actually created. Okay. So now we have these two big data structures that actually keep track of stuff for us.

Now, here's where things get a little bit funky. And when things get funky, what you're going to need when you deal with big data structures, you need a guide. And you'll see this in just a second because you're going to see some of the code that we write gets very

long when we deal with big data structures. So I'll be your guide. All right. So in the days of yore, I almost bought the whole outfit. But it's a little hot in here, under the lights. So in order to actually think about how you get the information and store the information when you have a large data structure, paper and pencil is your friend. Right If you spend all your time just staring at a computer screen it doesn't really allow you to internalize what is your data structure really look like and what's going on. So break out some pencil and paper, not right now but when you're working on data structures, and draw out, potentially, what things look like.

So here's songs and songs, and songs is an ArrayList. And it's going to have multiple, let's say at this point, three songs in it. And over here we have albums and albums is a HashMap, albums, that maps from names of an album to a particular album object. Now, the important thing to keep in mind in objects, and this is kind of the whole key to big data structure, is all objects, when you refer to them in Java, are references to objects. Remember when we talked about that. When you pass an object to a particular method in some application, you're passing a reference to the object. You're passing where that object lives. Okay.

Which means that when you have an ArrayList of songs, which what you really have here are a bunch of references, which we can think of as pointers that refer to the actual objects that contain the songs. Okay. So over here there's a "In Your Eyes," by Peter Gabriel and it was 99 cents. And over here we might have say, "Ramble On," tell me there's some Zeppelin fans out there. All right, good, good. We will not have to end lecture early. And "Ramble On" is such a great song, it's like \$12.99 by itself, single son. That's probably why most people don't listen to it. And over here we have the master, "Stairway to Heaven," Stairway to H, we'll just abbreviate it. Because it's that good, we'll just have a moment of silence, also, by Led Zeppelin, and we'll just say that one should be like, 49 cents so everyone can listen to it. It's just kind of like the bonus tune. All right. And so that's what we have in a list of songs. Now, here's the interesting part, right. If I'm going to have some albums, so I add some albums. So let's say add some album on here like "Soul," by Peter Gabriel, and Soul actually has the song, "In Your Eyes" on it. Okay.

Now there's two things that come up we will need to think about when we actually do this. We need to say, hey, this has got some ArrayList associate in there, and so I can create a new object that is a song for "In Your Eyes" and set my ArrayList to be a reference to that object. And that's a reasonable thing to do in some cases. The only problem is what happens if I go into my store and say, hey, I want to change my song "In Your Eyes" from being 99 cents, because no one's heard of it before, to 9 cents. Okay. So if I go thought my list of songs I say, oh, here it is, I'll change it's cost to be 9 cents. Now, unless I go through all of my albums and find for every album go though every song that's listed on the album and see if I can find that same song duplicated, I'm going to create an inconsistency in my data. What I really want to have is say, hey, there's only one object that is that song. And if that song happens to be a song that's sold individually, or it's a song that's both in my list of songs and on some albums, there's only one object ever that I refer to for that song, which means, I never create the second object out here

for that same song. What I do, is when I'm creating the album "Soul," and someone tells me, oh, it's got the song, "In Your Eyes," on it, I say, hey does that already exist in my store. If it does exist in my store, I'm going to add that object to my ArrayList. I'm not going to create a new object, which means each song only ever gets created once, but it can potentially get added to multiple ArrayLists. And it's the same single underlying object that has multiple references to it.

Why is that cool? That's cool because now, when I come along and a whole bunch of people start listening to "In Your Eyes," and I'm like, Peter Gabriel, he just deserves a lot more money, we're going to make this \$9.99. It's \$9.99 everywhere by changing it once. And that's the real key to large-scale software engineering. You think about not only reusing – you remember for a long time we talked about having methods that you reuse and how you generalize your methods, this is about reusing your data. Thinking about your data, sort of, if it's only one thing, exists in one place ,and everything refers to it. Okay. So any questions about that idea? This is what we refer to as a shallow copy, because what you're getting, after you've created that song once, when you want to add that song somewhere else, you're just setting a reference to it, you're creating a shallow copy, there's only one copy. The thing we did before, where we actually created a whole separate structure, is referred to as a deep copy. And sometimes, deep copies make sense in some particular cases. Most of the time they actually, well, I won't say mot of the time, they don't, it depends on the application, but most of the time what you'll actually be using is your friend, the shallow copy. Okay.

So what does that actually look like if we try to turn that into some code? Well, what does that mean in the application? Let me show you what that means in the application. So we're going to add some songs. We're going to go through another example. All right, let me add the song and I'll just abbreviate, "In Your Eyes," Peter Gabriel, \$1.99. Then I'm going to add "Ramble On," oops, "Ramble On," Led Zeppelin, and we'll make that, oh, I don't know, \$2.99. Okay. Now, at this point I have two songs. Now, I'm going to add an album. So I add a particular album and the album I'm going to add is "Soul," by Peter Gabriel and it says enter a song name. It's going to have "In Your Eyes" on it. And it asks me because the unique identifier is both the song and the band name, it still needs to ask me for the band name, and the band name I give it is Peter Gabriel. And it says, hey, that song is already in the store. It's just letting you know, hey, I found that song in my store, so when I add it to the album, I'm adding that same object that's also in my store to the album. And then you could say, well, there's other stuff on there like there happens to be a tune called, "Red Rain," which is also by Peter Gabriel, and you know it's a fine tune, but let's just say it's 1 cent, okay. And it says new song to add to the store. What did it do here? What it did in this case, it says, hey, you want to have a new song called "Red Rain," by Peter Gabriel. That song costs 1 cent, you want to add it to your album.

Well, if you want to add it to your album, it's also a song that I'm going to see in the store. So it actually adds it to the store and adds it to the album. And there's still only one copy of that object ever. It just needs to make sure that when it creates a new song to add to an album that's not already in the store, it adds it to the store, as well as to the album.

If the song already exists in the store then it just adds a reference to the album. Okay. That's the critical idea here. All right. So now, if we sort of list – I'll hit enter quit – and if we list all the songs, right, the song "Red Rain" has now been added to the store and costs 1 cent. And if I list all the albums that are sold by Peter Gabriel, and if I list all the songs on that album, it has the songs "In Your Eyes" and "Red Rain" so it matches the picture that I think.

That's why having a piece of paper, where you draw pictures, is useful. Because you look at what your application is doing and you say, does it match what I actually think should be happening in my picture. And if doesn't, then you know one of two things is wrong. Either your picture's wrong or your code that's supposed to be dealing with that picture is wrong. But in either case, you've already figured out a bug, even though the program hasn't crashed or anything, you just know there's an inconsistency. Okay. And so now, if I update the price for a song, like I update the song, "In Your Eyes," by Peter Gabriel, and I change its price to, I just go crazy, no one's going to buy the song anymore, the price is updated. Now, if I list all the songs, that song is \$999.99 in the store, and if I also list the songs on any album – five, so lower case, the price is also updated on each of the individual albums, because there's only one object. Okay. That's where the consistency comes in. That's why the consistency's key. Okay.

So what does this actually look like in code? How do we do this? Let me show you what the actual application looks like for our little friend, the Flytune Store. Okay. So there's a bunch of stuff at the beginning that just asks for the user selection, basically print some stuff out to allow you to make a selection, and then gets your selection for you. And then there's a big case statement that calls an appropriate method, depending what selection you made. So I'll go through some of the simple ones pretty quickly. You can list out all the songs carried in the store. In order to be able to do that, we need to keep track of how this information's actually stored, it's exactly in these data structures I just showed you. Song is kept track of in an ArrayList of songs and albums is kept track of in a HashMap that maps from the name of the album to the actual album data structure, itself. Okay. Any questions about that, hopefully, that's all clear. I will take off the hat.

So how do we print these things out? To list all the songs, we just go through our ArrayList up to its size, and this is why you want to think of data structure as your needed guide, because you're going on a journey. At any given point, when you're dealing with a data structure, you want to think, what is the type I'll be dealing with right now? What does that mean? It means, when I want to print something out, what I need is a string that prints out. How do I get a string? If I started at songs, songs is an ArrayList. I don't have a string I can print out. But from an ArrayList I can get an individual element. When I get an individual element of that ArrayList, what do I have? I still don't have a string I have a song. What can I ask the song for? I can ask to get the string version of the song and I have a string to print out. Okay.

So you always want to think of it as you're going on a journey. Where do you start your journey? Your journey starts at the data structures you have available to you. In this case, we have a data structure called songs, another data structure called albums, that's

what's available to us. And what we want to do is go from that starting point through a series of steps to get to the thing that we actually care about at the end, that little piece of data that we want to display or interact with somehow. So here's another example. If I want to list all the albums, how do I list all the albums? Well, to list all the albums, albums is a HashSet. So in order to do something with a HashSet I need to say, hey, I want an iterator over all the keys of that HashSet. So albums is the HashSet, I get the keys of the HashSet, which is a collection, and I get an iterator for that collection, which is an iterator over all the keys of the HashSet. And now, as long as my album iterator, which is just my iterator over the keys, has an element, what do I do? I start at albums. I need say I need to get a particular album. Okay. Get. Which album am I going to get? I'm going to get the album whose name is associated with the next elements of the iterator. Right, because it's an iterator over all the names of albums. So get, gives me a particular album. Then, when I have the particular album, I can call two strings on it to get the string form of the album. Okay, any questions about that? Because they're going to get even longer, so if there are any questions about sort of the chain of things we call.

If it's making sense, the chain of things we call, nod your head. All right, and if it's not making sense, shake your head. And if it's kind of making sense, just keep looking and ask a question if a question comes to mind. All right. So how do I find a particular song? This is something where I'm going to use the helper method, so it's private to find a particular song. Songs, our unique identifier, is a combination of both the band name or the name of the song and the band name. So how do I check for that? I'm going to go through all my songs, it's an ArrayList so I can count through all the songs. Here's where things get long. How do I check to see if a song, that's actually in my data set, matches on its name with the name that's passed in? I start at songs, get the I song, and I have one particular song. For that particular object I get the song and name. Now, I have a string. I want to check to see if that string is equals to the name that's passed in. Okay.

And I do the same thing with band names. Song, get the I song, get the band name of that song, and then check to see if that's equal to the band. And if both of these are equal, then, hey, I found the song, and so I'm going to return an index, which is the index location of that song in my ArrayList, and I can just break out of the four-loop, here. Because once I find it, I say, hey, I found that, I don't need to keep looking, so actually this is one of the rare cases where you'll see a break in a four-loop, is you don't need to finish the loop. You got to what you were looking for and get out of the loop. If you manage to get through this whole loop without ever finding something that matches on both, the name and the band, well, your index remains negative one. So you return negative one to indicate, hey, I didn't find it, because you know negative one's not a valid index for an ArrayList. So if you return it that means you didn't find a valid element. Okay. How do we use find song? Here's how add song works. Okay. When you want to think about add song, you want to think about this property that we're only ever going to create an object once, and everything else is going to be references to that object. So the way add song is going to work is it's going to return a song object. Okay. And what it's going to do, is it's going to ask us for the name of a song, if the user enters blank line that means they want to stop adding songs so it just returns null to say, hey, you want to stop adding songs, I didn't create a new song, here's a null to indicate you

are done. But if they don't impress enter quick, I also ask for a band name, and then I ask to find the song. Okay.

I call that find song method I just wrote and I say, does that song exist. If the song exists, the song index is not going to be minus one. And that means, that song already exists in the store. So you told me to add a song that already existed in the store. So I'm not going to create a new song because it's already an object in the store that encapsulates all the information for that song, I will return to you a reference to that object, which means I just returned from the songs ArrayList whatever song happens to be at the index that that song actually lives at. Okay. So this just returns an actual object. It actually returns a reference. If you can, think of it as returning a pointer to the object. If I didn't find it in there, then, hey, I need to create the new song, right. It's sort of like "Red Rain" at the end. You wanted to add a song. It didn't exist in the store, let me get the price for that song. I'll create a new song object and now. Here's the funky thing, I will add that song to my ArrayList of songs for the whole store, write out to that the new song was added to the store, and I'll return that new song to you so you can do whatever you want with it.

And so now, you might ask, okay, Marilyn, if I just added a song to the store I don't really care about doing anything with that song, why are you returning the song to me? And that's true. If I just add a song to the store, if that's all I care about, I ignore the return value. That's actually what I do up here, which is very funky. Right. If you want to add a song, I just call the add song method, it goes ahead and adds the song to the store, if it doesn't already exist, and it returns reference that song object. If all I'm doing is adding a song, I don't care I just ignore it. I don't assign it to anything, I just say, yeah, thanks for returning that object, that was fun, whatever, and just get rid of it. Okay. But the reason why I've written it this way is if I'm adding an album, what do I do? I ask for the name of the album, and I check to see if that album's already in the store. If the album's already in the store I'm not going to do anything because the album's already in the store. If the album's not already in the store, then I ask for the band name and I create a new album. And then I put that album in the store. So album is my HashMap. I put in that HashMap the name of the album is going to be the key and the actual album object is the object. So I add, you know, the album "Soul" to my HashMap.

Now, I'm going to add all the songs. So I have a Y-loop that goes through and keeps adding songs until I get a null from add song to indicate that the user wanted to stop adding songs. But here's the funky part, every time the user adds a song, right, it comes along and says, hey, you want to create some new album? So let's say I actually want to create some new album over here when I create the album "Soul," so none of this stuff exists yet. Okay. So to create a new album, I say, hey, I want to create the album "Soul." It says, okay, that's fine, create an object for the album "Soul." It has the name "Soul," it's by Peter Gabriel. And it says, okay, what songs are on going to be in there? And it starts asking me for songs, because it's going to add them to my ArrayList in here. And so the first song I say is "In Your Eyes" in on that album. It goes and says, hey, find that song, it already exists. It returns a reference to that song, as a pointer that reference is what gets added to my ArrayList. Now, I go and ask for another song. Do you have any more songs? I say, yeah, there's another song. The song is called "Red Rain." When I go

to create “Red Rain” it comes up here to add song, add song comes along, asks for the name and the band, it tries to find the song and says, hey, that song isn’t already there, so I’m going to create a new song. It creates a new song called “Red Rain,” by Peter Gabriel, has some price associate with it, and adds it to the list of songs for the store. And then it returns this object, which means it returns a reference to this object, and that reference to the object – oops, sorry this got blocked – all right, this is where it is creating a new song, and it adds the song to the store.

Right, it adds it as a song, which is that ArrayList up there, and then it returns the object. So when it returns the object, I went too far, the add object does not know I add that song to the album. So this album, we’re going to add a song, and the song we’re going to add is that same object. It’s “Red Rain.” Okay. So that’s the important thing to keep in mind. That object we only created once, and we passed around references to it or we can return references to it, and assign them other places. And that’s how you get consistency in a much bigger data structure. Now, there are a bunch of other things we could do in here. I won’t go through all the excruciating details down here. But we can list the songs on an album, we can update the song’s price. And by updating a song’s price, all we do is we ask for the song and the band, we find the song in the data set if it existed. If it doesn’t exist we just say, hey, it’s not in the store, and if it does exist then we read in its price, and then for the songs in the store, we find the song at that index and set its price. And we know that whatever other albums contain that particular song, if we happen to update the price over here to you know, \$6.99, we only update it once and all the places that refer to it automatically will see the updated version, because they point to the same object. Okay.

Any questions about that? So I know it’s a lot of complexity to kind of deal with a big data structure like this. But now it’s one of those things like, now you’re old enough to kind of see the big honking data structure. Because in the real world, when people think of software engineering the large, these are the kinds of things they need to worry about and that’s where the complexity comes in. It’s keeping track of all your objects and thinking about what objects you actually need to design and build, in order to actually build an application that’s kind of successful to keep track of and makes things consistent with all the data you have. So any other questions? Uh huh.

**Student:** [Inaudible].

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

**Student:** Sorry. So in this application, do all the songs and albums, they’re also, I guess, singles, or – cause the albums are never priced, right? It’s just the individual songs

**Instructor (Mehran Sahami):** Right. So the albums don’t have a price. You could imagine the cost for an album is the total of all the songs on the album. Or you could actually do something funky. Like this is one of those places you can make a policy decision and say, an album is 90 percent of the cost of all the songs on it. And then all the individual song prices can change and any time you just say, what’s the price of the

album, total up all the prices of the songs, and take 90 percent of that. So it also allows for very dynamic album pricing.

**Student:** Thank you.

**Instructor (Mehran Sahami):** All righty. If there's any more questions, come on up. Otherwise I'll see you on Wednesday.

[End of Audio]

Duration: 49 minutes

## Programming Methodology-Lecture25

**Instructor (Mehran Sahami):** So welcome back to another fun filled, exciting day of cs106a. A couple quick announcements before we start. First announcement, there is two handouts. One of those handouts, which we'll spend some time talking about today, is your last assignment for the class, which is assignment No. 7. It's worth noting, we sort of talked about it the very first day of class when we talked about late days. The very first handout says no late days can be used on assignment No. 7. So that's important to remember. No late days can be used for assignment No. 7 because it's due the last day of the class.

If you're looking at the syllabus, you might say, I didn't think we had class on Friday, the last day of class. That would still be correct. We don't actually have class, so you only need to do electronic submission for turning in your program, but it's still due at 3:15 on that Friday. We just don't actually have lecture on that Friday. We'll talk about the details of that assignment as we go along.

The other handout that you got is the example of the code that you say last time for the Fly Tunes music store. So you should have all that code, which shows and example of a big data structure. In fact, the data structure for that program is perhaps more complicated than the data structure you're going to need to use for the face pamphlet assignment, No. 7.

So as you know, assignment No. 6 review today. I have a feeling a lot of people may have been taking late days on it because it was, for example, the last assignment to actually be able to use late days on. So we'll just do a quick, painful as to how much time it actually took you. Hopefully it wasn't too painful, but we'll see.

Just wondering, how many people are taking a late day? Wow. So for the six of you who actually are turning in the assignment on time, anyone in the zero to two hour category? I would be frightened if you were. I don't think it's actually possible to do in two hours. Maybe, if you just sat there and wrote all the code. Two to four? Oh, I actually have a couple for two to four. Wow.

Four to six? Good to see. I think all the [inaudible] is just going to be smaller. Six to eight? That's a pretty healthy contingent. Eight to ten? Also good-sized contingent. Ten to 12? Couple of folks. 12 to 14? 14 to 16? 16 to 18? 18 plus? Anyone in 18 plus and still hasn't turned it in, like you're just going for late days. All right. That's good.

So that's good to see. It's a good time. World is still normal, though the normal curve just happens to be a little smaller because there's more people taking late days, but it's a good time. If you haven't done it yet, two things. One is that this should give you an idea about how much time it'll take, perhaps. Two is you don't want to push it. Even if you have late days, don't push it off too late because you really want to give yourself time to do assignment No. 7.

So any questions about anything we've done so far before we kind of delve into today's great topic? All right. So I want to spend the beginning of class actually talking a bit about your next assignment. So if you're still kind of in the mindset of thinking about name surfer, and you're like, oh, I'm just so in the mindset of assignment No. 6, take that little part of your head that's got assignment No. 6. Pop it off the stack. That's my whole head. Yeah, that might be painful. Pop it off the stack for a second, and we're going to think about our friend, the face pamphlet.

So just a quick show of hands, how many people are part of a social network or do some social networking online. That's good to see. So I won't need to spend too much time telling you about what a social network is, but I'll spend a little bit of time in case you're not familiar with social networks.

Basically, a social network is just a way of keeping track of some set of people, or profiles of people, and relationships among people. At the simplest level, that's all it is. So what you can think about is you have some profile for a person. In some sense, you can think of the notion of a profile and a person interchangeably most of the time. But as most of you know, if you've ever read the New Yorker and seen that cartoon that no one on the internet knows you're a dog.

So you don't necessarily have to be a person to have a profile. So when we think about a social network in the abstract sense, we always just talk in terms of profile. But if want to think about it concretely, you can just think of a profile as being synonymous with a person, but it need not be, so we generally refer to a profile.

So what a profile has associated with it, it has some attributes or some things that we keep track of for that profile. For example, it has a name. In the case of your assignment, it's also going to have some status that we care about, keeping track of the person. It might have some other things, like the reason why it might be called face pamphlet is because say you want to keep track of people's faces. You want to actually see what they look like.

So you might actually have some image in there. Then I mentioned you want to keep track of some list of friends. So there's some notion of a list of friends that is also associated with a profile. Real social networks also have other things associated with them, and the assignment talks about extensions you can do, but we're not going to get into extensions right now. But these are things we want to think about.

For a person's profile, there's a name. One of the things based on some of the ideas we talked about in the last lecture, is you can think of name as a unique identifier. What that means is that you will never have two profiles in your social network that have the same name. So once you have a name in your profile, in your social network, that name is yours. That profile will be associated with that name until, at some point we decide, maybe we want to delete the profile to get rid of it. But the whole time the profile's actually live, it will have that unique identifier, which is the name of that profile.

So if you ever try to create another profile with the same name, the application will tell you that you can't. I'll show you some examples of that in a bit. But the name is a unique identifier, and this thing's just some string that we keep track of the name. So it can be like your real name or whatever name you want to have, like Code Dog. That can be your profile name. That's cool, too.

Status is just what you're doing right now. You can think of this as some string which is what you happen to be up to at the time. Like Maron is teaching. I don't know. Ben is contemplating life. It's just some particular string of the thing you happen to be doing. You might want to change this a fair bit.

This image thing is just an image. In our world, that means it's a G image. That means it's some picture. Originally, when you create a profile, you may not have a status, and you may not have an image. You may want to add one. You may want to change one as life goes on. We should be able to allow for that. Then you have a list of friends. When you're born into the world, your list of friends is empty. Over time, you can gradually build up friends.

That's the other part of the social network to consider, besides the fact that we have profiles is that we also have some notion of friendship. The way you can think of friendship is basically some profile or some person just has a list of friends. So an easy way to think of a list of friends is a list of names of friends because names are unique identifiers. So if we keep track of a list of names, we have a list of the unique identifiers that are our friends.

That's an easy way to be able to look them up, for example. The idea behind friendship that's key to us – this is not necessarily true in the real world, sadly enough, but for our intents and purposes, as far as the assignment's concerned, friendship is reciprocal. That means the if your friendship is reciprocal. Not all the time. That's the way it is.

What the really means is if I have two people, like I have Bob over here and I have Alice over here, you can think of those as profiles. If they are friends, we draw a line between their names in our little social network. This is not actually how we keep track of them in the application that you'll see, but this is often how social networks are drawn.

We put in sort of a circle for each profile in the social network. Here's Don over here, and here's Chelsea. Notice there's the ABCD phenomenon. Always a good time. So when we put what we refer to as a link between two people, that means they're friends. Friends are reciprocal. So if Bob is a friend of Alice, that means that Alice is a friend of Bob. That's just the way life is all the time in face pamphlet.

It's just a happier place. Everyone's like, oh, I'll be your friend. Okay. Then you're my friend, too. So that's the way you want to think about it. So if Bob ever adds Alice as a friend, that automatically means Alice becomes Bob's friend. There's never a directionality of friendship. Bob clicks, oh, Alice, be my friend. She's like, no. Talk to the

hand. That happens in real life all the time, but not here because it's just a happier place to be in.

So think of friendships as reciprocal. Again, we don't necessarily have to have anything explicit to model this link. All we need to do is think about just having a list of names. So Bob might have Alice in his list of names of friends that he's keeping track of. That's how I would keep track of the link.

What that also means is that reciprocally, Alice should have Bob in her list of friends because the links are always reciprocal. When we actually draw this out, if we want to draw the stylized picture, we could say Alice is friends with Chelsea and Don, and Don and Chelsea are also friends over here. This is where the name network comes from.

When we draw this thing out, eventually you have millions and millions on these things in there, all over the place. Everyone's holding hands and singing Kum Ba Ya. You have a social effect because you have friendship, and what you actually end up drawing is something referred to as a network.

If you're really the mathematical type, this is really a graph. In this case, it's an undirected graph. If that kind of talk makes you hot and bothered, like, oh, a graph, yeah. Take CS103B, which I'll be teaching next quarter. Graphs are just the coolest, most wonderful thing. But that's a thought that's not important right now.

What is important right now is that this is where the name social network actually comes from. We create this network that keeps track of social relations. So if we have that base idea of social networks, how do we actually think about writing an application to keep track of them. So [inaudible] the computer, I'll show you what, at the end of a week and a half's effort, you will have your own social network application.

So we're going to call it Face Pamphlet because it's not really big enough to be a book. It's a tiny book, so it's a pamphlet. So when it starts up, you can see there's interactors galore all over the place. Over on the north side, we got a name. We got some buttons over on the west side. We got some status, some pictures, some adding friends. What does this all mean?

We'll actually go through an example and sort of create a little social network. So what I'm going to do is create a profile for myself. You're like, oh, that's kind of narcissistic. That's okay. I didn't want to – oh, we're going to create a profile for you right now.

So we add the profile on here. Here's what we get in the display. There's a bunch of display elements. We have a name up there with some big font, and it's blue. We don't have an image right now, so we just have a place holder. That's basically just a rectangle with some text. It says no image. We don't have a current status, but what we've done is created an initial profile with the name, Maron Sohami, which is a unique identifier in the system. So right now, our social network has what we refer to as one [inaudible]. There's one profile in there, and I don't have a list of friends.

You might say, okay, that's not so exciting. Let's actually add some stuff. So once I actually have this active profile, I can say, let me change the status. Right now, at this very moment, I happen to be teaching. So when I click change status, it says Maron Sohami's teaching. Sometimes I could just be, no, I just really want to be doing this.

So I can keep changing the status. That's fine. It just updates here whenever I type it into this field and click the button or hit the enter key. I can also add pictures. So I can say, for example, MaronS.jpeg is a picture of Maron. There's a picture that gets loaded. It gets, unfortunately, scaled to always take up the space for the image, which is a 200 by 200 region space.

It's constants that are given to you in the handout, but for these purposes, it's 200 by 200. So this is kind of the bouncer version of Maron in the days of yore, when I was clean shaven. There was actually a time when I shaved. That doesn't happen very often anymore. If you were ever wondering what I look like without the beard, not that I care. But that's what it was like.

So we've created a profile. Not a big deal. You could say, hey, next quarter, I'm planning on taking 106B, and isn't that taught by Julie Zolanski? Yes, in fact, it is. So let's add a profile for her. So we type in Julie Zolanski in the name field up here and click add. We now get a new profile for Julie Zolanski. Julie is just a fiend. She codes like a fiend, so she's coding like a fiend. That's what she's doing.

So you can see down here, not only is all this stuff getting updated as we typed it in, but down here, there's something we refer to as the application message. The application message always tells you what was the last thing you did or tried to do. So your status was updated to coding like a fiend was the last thing you did. Let's give Julie a little picture over here. JulieZ.jpeg, and there's Julie, also scaled to 200 by 200. So she's no longer coding like a fiend. She's kind of feeling stretched.

So now we have her picture. We have her profile. That's all good and well. Let's add a couple more profiles real quickly. So I'll add some of our section leaders. I'll add Sarah K. I'm just going to abbreviate. Notice these fields over here don't necessarily get reset when I create – they just have their old values in them. That's not a big deal. You could add it as an extension if you wanted to, but they just leave their old values.

Sarah K. Sarah, is it okay if I use your picture? All right. Here we go. There she is, and she's actually scaled reasonably well. So we'll also add Ben. There's Ben. We'll add a picture for him. If you're wondering where all these pictures are coming from, in the starter project that you'll get, you'll get a folder of images that contains the images of all the section leaders from the class as well as Julie and myself.

I'm scaling until the cows come home. If you have your own pictures, you can add them into the folder called images, and you, too, can display your image at a 200 by 200 resolution. Ben, where are you? Here's Ben.

Well, Ben's feeling kind of anonymous. So I gave you two anonymous-ish images as well. There was a Stanford logo, which is just a super low-res version of our logo. Then there's also Stanford Tree. So you can change the picture to one of those if you want to. Those just come along for the ride.

If you do the web version, actually, the web version only has these two Stanford logo and Stanford tree images. I didn't put everyone else's image on the web version because for a little bit of privacy. Was there a question?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** No. If I empty the field, one of the things it says in your handout is anytime the field is empty, it just gets ignored when I try to do something. So any field, whether or not it's name or any other field, just ignore it. But I could have something like the nonexistent picture.jpeg, and I try to change the picture to that, and it says unable to open image file.

So in your handout, it actually explains how you can detect if an image exists or not. So you can actually check the case where the image doesn't exist. If it doesn't exist, we just don't update the image to the new image. Really, all we're keeping track of here is the name, which gets created with the initial profile because it never changes, an image that we display, whatever the current status is. If currently there is no current status, and a list of friends.

So far, we've add a bunch of profiles. What else can we do? Well, there's a lot of things we can do. Let's look up Maron again. So if I type in a name I can click look up, and if that profile exists with that name, it brings it up and says you're displaying the profile name down here. Then we can add some friends. So Sarah is a friend. Add friends. So it says Sarah K, added as a friend. Ben is also a friend. So I add Ben.

I can try to add someone who doesn't exist in the system. Like I can try to add – who should I add? Jenny. 867-5309. I add Jenny. Jenny does not exist. As far as I know in my social network, Jenny's not there. I get this message down there.

Remember, friendships are supposed to be reciprocal, so I have Sarah and Ben as friends. I never went to directly update profiles, but if I now look up Ben's profile – Ben N – Ben has me as a friend. So when I added Ben as my friend, at the same time, I got added to Ben's friend list. That's something that you're going to need to manage in the application. Friendship should always be reciprocal. So when you add a friend to a particular profile, you also make sure to go to that friend and add the current profile to that friend.

Now, another thing we can do, besides just creating profiles and done friends, we can look up profiles, like when we had Maron's profile over here. We can look up a profile. If I try to look up a profile that doesn't exist like Maron Sohami Bob doesn't exist. It clears what profile's in there. It says a profile with the name Maron Sohami Bob does not exist.

When there is no profile in the current application – all this stuff, by the way, is explained in the handout. This is just to kind of show you to get a feel for how it works. You don't need to jot this all down in notes. When there is no current profile displayed, I cannot change any of the things in the profile. It says please select a profile to change status because I can't change the status of a non-existent profile or change its picture or add friend to a non-existent profile. But the way I get a new profile displayed in here is I either add a new profile or I look up an existing profile. Then there's a profile there that I can change to whatever I want to.

Now, let's say I've looked up this profile. I say, that's great, and Ben decides he's had enough. Get me out of your social network. So I say, oh, I'm sorry to hear that, Ben. Here's Ben. I'm going to delete Ben from my social network. Even though I'm displaying my own profile. I click Ben up here, and I delete him. Whatever profile was there, it doesn't matter if it was Ben's or whoever else, it cleared, and it says profile of Ben N deleted.

Now what happens when a profile gets deleted? It gets cleared out of the social network so it's no longer there. If I try to look it up, it says a profile with the name Ben N does not exist because, sorry Ben, you've been nuked. You're out of the social network. But Ben was friends with people. He was friends with me, and now it's a sad day. So I go to look up Maron. I should've just used the first name.

Ben has been removed from my list of friends. Sad day, but that's life in the city. You can't be friends with non-existent people. So when someone gets removed from the network, you go through everyone else in the network and say, did they have them as a friend. If they did, they get popped out. I guess you're just not a friend anymore. That's basically the whole social network. That's the whole idea.

So question?

**Student:** [Inaudible]

**Instructor (Mehran Sahami):** Okay. Names are not case sensitive, so I can have a Maron Sohami, and then I can have a screaming Maron Sohami, and he's just different. He's angry. I don't know if we have an angry image in here. I don't know if any of the section leaders were actually looking angry, so we'll just make him the Stanford logo. This is different than Maron Sohami, lower case. So you don't need to worry about it like you did in the last assignment. You were worrying about case sensitivity. Here, you don't need to worry about making things insensitive. You can just case – everything is case sensitive, so you don't need to worry about making strings equal to each other, case sensitive and stuff like that.

So any questions about this general idea? You got to think about it, right? Any more questions?

If you kind of think about it, this is the basics of the social network. You have all the people. You can have as many people as you want from the social network. You can maintain friendships. You have attributes of the people, which in this case, just happens to be their name, their image and their status and their list of friends, but you could imagine – you could extend this in a whole variety of ways.

Really, what this is about is data management. There's a display component. In part of the assignment, there's a bunch of milestones that are laid out. The actual only milestone that deals with this display is the very last milestone, which involves a bit of work to get all this stuff displayed at the right locations. There's a bunch of constants, which is all the places, you put things at the right locations.

But it's really about managing data. How do you manage profiles, managing the friendship relationships, being able to remove stuff, add stuff, which is what most applications in the world are really about. So you're going to get a chance to really exercise your data structure skills and your larger-scale application skills in this program. So any questions about this program?

There's also a web demo of it, so if you want to go to the web and play around with it, you can play around with it. Again, in the web version, there's only these two images, the Stanford logo and the Stanford tree.

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** No. That's the other thing. It's in big font somewhere, like on page 15 in the assignment. So don't be scared by the fact that the assignment handout is 28 pages like some people are looking at. Oh, my God, this is so big. The code is much shorter than the handout, first of all. Second of all, one of the things – all the details are in there, but one of the important things that comes up in the handout is you don't need to worry about scaling.

So if I shrink or increase, yeah. No worries about size or resizing. I figured you got enough practice with that with name surfer. Either you already got practice with it or you're getting practice with it with name surfer. So we don't need to make you do it again. So you don't need to worry about component listener and resizing and everything. You can just lay this stuff out with respect to the constants that are given to you, and you're okay. But resizing, you don't need to worry about it.

It can be a cool extension if you want to add it as an extension. That's perfectly fine. There's a bunch of extension ideas that are listed at the end of the assignment. As you can kind of imagine, if most of you are on a social network, you probably already have thousands of ideas for extensions, if you want to add them, but you're not required to add anything in particular.

So any other questions about this? All right. So with that said, I just want to give you one last little – before we moved into the advanced topic for today, one last little side note

about social networks. Has anyone ever heard of the phenomenon, six degrees of separation? A few folks. There was a movie years ago about this phenomenon, and the movie actually had nothing to do with the phenomenon. It just happened to have the name, Six Degrees of Separation.

But it's basically this idea that everyone in the world is linked, potentially, to everyone else by at most, six hops in this graph, which is kind of weird to think about. I won't go into all the details an mathematics and stuff, but there's some very interesting stuff behind it. But the biggest idea is that these social networks, you can sort of think of little hops between people.

You can say Bob is one degree separated from Chelsea or two degrees, depending on how you count, from Chelsea because you can get to Chelsea by going Bob, Alice Chelsea or Bob, Don, Chelsea, whatever the case may be. But just in case you're wondering where the six degree of separation idea came from, Stanley Milgram – anyone ever hear of Milgram's experiments like obedience to authority, if you're a psychology kind of person? Same Milgram.

We won't get into that. That's just a whole bunch of other interesting psychological phenomenon that are discussed on the other side of the quad. But what he did was said – this was back in the '60s, I think. '67, actually. He sent a whole bunch of packages to various people in Nebraska. So let's just say that's Nebraska.

He sent a whole bunch of – yeah, my notion of geography – to random people in Nebraska and said, these packages need to get to someone who's over here in Boston. Yeah, I have no idea. It's Boston. That's Massachusetts, and here's Boston. These packages need to get over here, but he didn't give them an address. He just gave them a name of random people in Boston the packages need to get to.

The only instructions he gave were to say you can't try to figure out who this person is and send them the package directly. What you need to do is send this package to someone that you know who you think will be more likely to know this person than you are. Those were the only instructions. Then when the packages was trackable, how many times it got mailed.

He measured how many times the package had to be mailed to various people before it actually got to its destination. What he found was there was an average of about five to six for the packages to get from a random person in Nebraska to a random person in Boston. That's where the six degrees idea comes from, but that's – now when people talk about social networks, they talk about things like degrees of separation and what we refer to as small world phenomenon. If everyone's linked by only six degrees of separation, then it really is a small world after all. We can all hold hands and be mechanical animals. If you've ever been to Disneyland, that makes sense. If you've never been to Disneyland, you don't need to go on the small world ride now.

So time for something completely different. Are there any questions about social networks before we delve into our next great topic? If you're really interested, you're like, Maron, tell me more, take CS103B. It's a good time.

The advanced topic that you're going to get – and I hesitate a little bit to tell you this, but this is something that you don't need to know for the final exam. As a matter of fact, you don't need to know it for assignment No. 7. It's just something that's so cool with the way programs are actually done today, it's something you should actually know. That's a notion called concurrency.

It's also important for you to know if you want to understand, for example, in next class, for example, I'll show you how to take the programs that you've written in this class and turn them into applets or turn them into executables so you can share them with your friends. You'll need to understand this idea.

So the basic notion of concurrency is that if you think about it, your computer, there's actually multiple things happening on your computer at the same time. It looks like there are multiple things happening at the same time. You're like, there's oftentimes – I got my email program open and I'm IMing, and for those of you out in TV land, I'm watching a the lecture at the same time. Most of the time, I'm not actually watching. I'm just IMing and doing email.

But to you, it looks like all these things are happening simultaneously on your computer, right? In fact, they're not. Your computer only has one – most of you. These days, things are changing a little bit, but most of your machines only have one processor in them, which means at any given time, really, only one thing is happening.

How does it look like multiple things are happening at the same time? That's the notion of concurrency. What's really happening is the computer or your operating system is saying there's all these things that want to be happening at the same time. What I'm going to do is give each a little piece of time to go and execute a little bit, and then I'll do the next one and the next one.

So email gets a few milliseconds and IM gets a few milliseconds. Then the video gets a few milliseconds, but that's enough time to update to the next frame of the video. I just keep cycling through these super fast. So to you, it just looks like they're all happening at the same time.

If you think about this, you actually did this already. Which assignment did you do this on? Breakout, right? Breakout was doing this. You had the ball, and you had the paddle, and you had to check for collisions. So what were you doing? You were saying, hey, ball, move yourself. Wait some time. Hey, paddle, update. Maybe wait a little bit of time. Hey, check for collision, and I just keep doing these over and over.

It was happening so fast that it looked like the paddle and the ball are moving at the same time. But you knew they weren't moving at the same time. You knew that when the ball

was moving, the ball was getting the message to move. So paddle wasn't actually moving. Paddle got moved later when your execution flow got to that place and moved the paddle.

So you've already seen a notion of this. There's a concept around this notion, which actually makes it much more concrete, called the thread. The idea of the thread or a thread of execution, is to say that the way you can think about a program is that rather than a program just doing one thing, a program can say, hey, I'm going to have some thread over here and some thread over here and some thread over here.

I'm just going to kick these off. I'm going to say, hey, there's some piece of a program I want to do here. So just start it and start executing. There's some other piece of a program I want to do at the same time. Just kick it off and start executing. Same thing over here. I don't want to have to worry about doing this cycling myself. I want someone else to say, oh, you get a piece of time and a little piece of time and a little piece of time. It just keeps cycling through. This happens so fast, much faster than my moving hand.

It all looks like it's happening at the same time. That's the notion of a thread of execution. If you can think of a program, what we refer to as being multithreaded, which means it actually has multiple threads of execution that are happening at the same time, which is a little mind bending. Actually, it turns out to be a fairly straight-forward kind of thing.

So how do we make this a little bit more concrete? How do we think about threads in JAVA's world. It turns out, interestingly enough, you've been sort of doing this a bit the whole time.

There's something called the runnable interface. What the runnable interface says is that anytime I have a class that implements the runnable interface, that means that the class can be turned into a thread. It can be kicked off as a thread because it has some notion of running. So I can say, you know how to run. Okay. I'm going to create a thread, and you're going to go run in that thread. It's like having a little baby and sending them off into the world. It's sort of like what your parents did when they sent you to college.

The whole time you were part of your parent's process. They were managing you. They were like, eat, sleep, homework. At a certain point, you were like, okay. I'm done with that cycle. When can I go do my own thing. Then you were like, oh, I'm graduating from high school.

What they did was your parents said, now it's time for you to get a thread. You were like, okay. I'll figure that out when I get to college. Really, they said that. Maybe you just weren't in the room at the time.

Then they sort of kicked you off. They said go do your own thing. You're going to be doing your own thing at the same time everyone else is doing their own thing. I'm not going to manage you anymore. You're just going to go do your own thing. Hopefully, that thing will turn out well.

So what is the runnable interface? Let's just look at a simple example. I have public, class, my class, and this is just an interface, so I say implement runnable. We can just make up words when we're computer scientists.

This class is going to have some constructor in it, so allow public my class. It needs not actually have a constructor that's explicit, but let's just say it has a constructor. Then there's only one method that a runnable interface – the class the implements the runnable interface needs to implement. That's public void run. You should see that at this point and go, oh, my God. I've been doing threads the whole time in this class.

That's what you've been doing. Guess what? Program implements the runnable interface. So you wrote a method called run, and what we did somewhere in the bowels of the ACM library was we said here's your program. Go run. So it started executing the run method. That's where all the stuff happened in your program.

Now, when you wrote programs that had interactors with them or looked at mouse clicks or whatever the case may be for events, your program – so this is my program over here, for example, that's running. There was another thing running over here that was called the event thread. The event thread was actually the one that was like, mouse click. Yeah, when that mouse click happens, I know about it because I'm managing the mouse clicks.

When that mouse click happened, it said, hey, I got a mouse click. Who's listening for the mouse click out there? You were like, me. I added mouse listeners. I did. It says, okay, I'll let you know there was a mouse click. Then you did something based on that mouse click. You stopped, and it was like this thread kept running. Your program only got called when some mouse event happened. It sort of said, okay. I'll let you know that this mouse event actually happened.

So when you were dealing with mouse clicks and mouse movements and all this stuff, there were actually multiple threads running. There was a separate thread that was getting events and sort of telling the people who were listening for those events that they actually happened.

So with that kind of idea, how do we actually create a thread? So I've sort of told you, if you want to create a thread, you need to have a class that implements that runnable interface. How do I actually create the thread itself?

So here's how we create the thread. We create something of the class that is runnable. So we might say, I have some object, X, that's [inaudible] my class, and my class implements the runnable interface. So this is just going to be some new my class, just like creating objects of any class. It's just another class. It just happens to implement the runnable interface. So you create one.

Now, once you create one, this is sort of like your birth. You're born/myth up to age 18 or whatever it was. For some of you it's 12. There were actually – I heard that there was someone very young at Stanford this year, but I won't verify that. So let's just say your

life up to age 19 happened. Then at some point, your parents said, hey, I'm going to turn you into your own thread. You can just go an execute independently. You're like, well, what about the tuition checks?

They're like, yeah, we'll handle those. We've got the tuition check listener going on. So it'll actually pick up those events. But we want to spawn you off as a thread. Now, it's kind of the words that we use in the computer science [inaudible]. Spawn a thread.

So what we're going to do is create an object of type thread, which we'll call C. What we'll say is that's going to be a new thread. What you give to thread is the object that you want to execute on a separate thread. So we give it X. That object has to be of some type that implements the runnable interface. So that's how it works.

You create a thread. You say, hey, thread, the thing that you're going to execute is this class, X. And the way – what time you're going to start it is, you say, T. So you tell the thread start. When the thread starts, what it does, it says, hey, object. How are you doing? I'm going to call your run method, and you're just going to start executing in your own thread from your run method.

So that's kind of the basic set up. It's fairly simple, actually. You create an object that implements runnable. You create a new thread that's passed that object, and then you sort of kick off the thread. It starts executing whenever you say start. So let's make this a little bit more concrete. Let me show you an example.

So what we're going to do is create a super simple example that's basically just a square. It's a rectangle. It's a G rect that's going to slide across the screen. So let me show you the code. We have some class slider that extends G rect. So our class that we're going to create that's going to be runnable is just a rectangle. It's going to implement the runnable interface. It's going to have some constructor, and what we're going to say is hey, you're going to actually be a square. I'm going to give you some initial size and some initial color.

So the first thing it does, it says, I need to call the constructor for my super class. My super class is G rect. So I'm actually a G rect that's a size-by-size G rect because I'm a square. But to initialize myself as a G rect, I need to call my super class' constructor.

So I say, super size comma size. I set myself to be filled, and I set my color to be whatever color was passed [inaudible]. Then once I'm created, I'm kind of hanging out as an object until I get put into a thread and run, at which point my run method gets called. When my run method gets called, all I'm going to do is go through a sequence of steps where I'm going to pause for 40 milliseconds and then move myself a small amount in the X direction, over and over.

So I'm basically just going to slide across the screen. That's what I'm going to do. My step size is five. That's my whole object that I want to run. So where's the program that I run? Here's the program that runs it. So the program, because it's a graphics program, it is

going to be a thread that someone else, that you have hither to not seen, is going to create and run. So its run method says, hey, I'm going to add a button that's called slide, to the southern region.

I'm going to add an action listener. I'm going to wait for mouse buttons, or I'm going to wait for you to click that button. If you click that button, so if the command I get is slide, what I'm going to do is create one of these sliders. I'm going to create an object called slider that's of type capital slider.

Its initial size is going to be some constant size that I set to 20. Its initial color is going to be some randomly-generated color. So I have some random generator. I'm just going to give it a random color.

Then what I'm going to do is create a thread that's going to have the slider in it, and I'm going to kick off that thread. So if I run this program, here it is. Nothing's gone on. I click slide. The box gets created of a random color, and it's now in a different thread. It slides across the screen. You're like, huh. Okay, Maron, that was a whole lot of work to create a sliding square.

Now, here's the interesting thing. That square is getting added to this canvas. So any squares I create will get added to the same canvas. As a matter of fact, if I look at the code over here, ever time I get a mouse click, I'm going to come and execute this code. Let me make this little thing go away.

I'm going to come execute this code, which means I'm going to create a new slider of a new random color. I'm going to put it in a new thread. So every time I click the button, I'm going to get a new thread that's executing and parallel with all my other threads. You're like, okay. What does that mean? That means I click this once, and I get a box. I click it again, and while that one's still running, I get another box of another random color. They're all sliding across the screen at the same time. These aren't the droids you were looking for.

So that's the beauty of it, right? I didn't need to manage. Oh, you go move yourself. You go move yourself. No, you wait. You go move yourself. I just say, hey, you go run. I want another box. Yeah, you go run, and you go run, and you go run. They're all running at the same time.

Now here's something that's even slightly cooler than that. What's slightly cooler than that is your threads can interact with each other. These threads were all independent. So one thread was executing here, which was a little box that was moving. Another thread was executing here. There was a little box. Never did the two meet. They were just hey, if I run into you or whatever, that's all cool because we're just all treads here, and we're all executing.

Sometimes, you care about knowing what's going on with your threads or threads care about what's going on in some other thread. So there's actually ways that you can think

about keeping track of data, of having shared data between your threads, which means all your threads are sort of running along. There's some piece of data. So these are all just classes. In these classes that are happily running along, in that class, there's actually some variable that's a reference to some object over here.

Guess what? This guy's also got a reference to that object. This guy's got a reference to that object, which means if anyone of these threads updates this object, the other threads can look at that object and see what its updated value is. This is, for example, how ATM machines work. There's different ATMs that can be updating your bank account, but there's only one bank account that they see. They're not looking at different bank accounts. It's the same principal.

So what are we going to do? We're going to create a race. The way the race is going to work is we're going to create something called the racing square. So it's like, yeah, I'm the racing square. I'm ready to go. It's also going to extend the G rect and implement runnable. What does it do? Well, it's pathed at index to let it know which square in the race are you. There was a bunch of squares. They're all going to run across the screen at the same time.

Which index are you? So there's some number of squares. We're just going to index them for zero up to the number of squares minus one. We're also going to pass you a shared array. When we pass an array, arrays are always passed by reference. So you don't get a copy of the array. You get where the array lives. So we're going to give you this array bullion finished, and what that array is going to be is an array for all of the different squares that are running. You know what your index is in that array.

When you're done, you should say, hey, my finished is true. Until you're done, you say my finished is false. So we'll get to that in just a second. What it's going to do is create a square of size by size, set itself to be filled, keep track of its index, and keep track of the list of finishers. So it needs to have some internal variable to refer to this reference finished.

So those are just variables that are down here, right? I have some private integer and some private reference to the array, and so when I set finishers equal to finish, what I just have is essentially that diagram over there. I have some reference to this place where the array lives in memory. I have a random generator because I care about having random generators, as you'll see in just a second.

So what's this square going to do? That's all it does to get itself set up. When I gets run, what it's going to say is in the list of finishers, my index, because that's me, it's going to be false because I haven't finished the race yet. When you sort of kick off the thread, I'm just starting to run. I haven't finished.

To run the race, I'm going to move 100 steps, and in each one of those steps, what I'm going to do is pause some random amount of time and then make some step forward. So my step size, I have step size that is just some fixed constant. Basically, what I'm going to

do is take a bunch of steps. I'm basically going to take steps of size five. I'm going to take 100 of them. So I'm going to move 500, but the amount of pause between each one of the steps I take varies.

So I actually move it potentially different rates over time. That's why it's a race. Each one of these guys will be moving at different rates. Then I need to watch out for photo finishes. So this code is actually buggy, and I'll show you why in just a second. When I get to the end of the race, so after I've run the race, I've taken my 100 steps over time, what I do is say this stuff's commented out for right now. I say, I'm finished. So my – the list of finishers at my index is true. I'm like, I crossed the finish line. I finished. Did anyone else finish, or did I win?

So what I do is I want to go through up here and count all of the other people that finished. So before I set myself to having finished, it's sort of like you're about to cross the finish line. You kind of look around. You're like, is it me? Did someone else already finish?

So before I finish, I look around, and I say through the entire list of finishers, if anyone else is finished, it finished so I is true, then I increment the count. So I'm basically just counting how many other people finished before I finished. If no one else finished before I finished, I make myself red because I'm the winner.

No one else should be red because if I cross the line first, my finishers should be true. So when someone else does that count, they should get a count of at least one so no one else should cross the line at the same time. That's the basic idea for this racing square.

The way we run the racing square is we're going to have a bunch of racing squares. So we're going to create an array of racers, just of [inaudible] racer size. That's how many racers we're going to have. We want to have some array of finished, so that's just the same or number of racers, and we're going to have each racer – is going to be in a different thread. So we need to have an array of threads, one to keep track of each one of our racers.

We create a finish line in the race, the numbers aren't important. We'll just see it draw on the screen, and we have a little start button to create all the action. So what's going to go on when someone clicks the start button? What I'm going to do when someone clicks the start button is go through all of my racers. If I had an existing racer on the screen, I'm going to remove it. I don't need to worry about that until after the first race.

But if there was already an existing racer, I remove it. Otherwise, I don't worry about it. I create a new racing square at index I because I'm creating all racing squares from index zero up the number of racers. I'm passing them all the same shared array finished. They're all going to be referring to the same shared array, just like the picture over there. Then I'm going to add the racer to the canvas at a particular location. The math basically just has them all sequentially down the screen.

Then what I say is, hey, I've created you, I've put you on the screen. You're revving to go, so I'm going to kick you off. So racer sub I is going to be in a new thread, sub I, and then I'm going to tell thread sub I to sort of go start itself.

So if I run this, I'll show you what it looks like. So if you think about each one of these different threads, you might say, doesn't the first racer have a slight advantage because their thread gets kicked off first? Yeah, but it turns out all the stuff executes so quickly that the random pauses in there actually have a bigger affect.

So here's the thread's example. There's my finish line. I'm ready to start. Yeah, every once in a while, a little bug in JAVA comes up, and it doesn't start. So let me recompile. Not a bug in the program. This was kind of an interesting thing that I just discovered on the side this morning. There's actually a difference in how Windows and the Macintosh happen to deal with this. Sometimes a little error comes up.

Here we go. So here are the two racers. They're just running in different speeds because they're in separate threads. They're running, they're running, they're running. You can see it's just the random pauses between each one of their moves. One crossed the finish line. He's like, oh, I'm the winner. I'm red.

You're like, okay, that's kind of cool, sort of, but how could I potentially extend the program? So instead of two racers, let me make there be ten racers. Instead of ten racers – actually, let me have there be ten racers, and what I'm also going to do is have them take a little victory dance in the end. So when they get to the end of the race, they're going to look around and see who else is around them. Before they cross the line, they're going to be like, I'm done. I'm done. That takes about 50 milliseconds to do that dance. You can just pretend that the square is dancing.

So I go ahead and run this program. Just to show you ten racers because ten racers is kind of cool. You notice I only had to change really one parameter. I just changed the constant from two to ten, and I didn't need to update anything else in the program. Now I got ten racers, and they're all running different. You're like, oh, you can start rooting for you racers. We could put G images in there. You're like, come on, little guy! Oh!

That's way more exciting for me, evidently. So you're like, that's kind of interesting. You just increased the number of racers, but what else can you do? Now, think about this. What if everyone was created equal? So rather than all of them having a random delay between 50 and 150, what if I just said, your pause is always 50. You're going to generate a random number between 50 and 50, so it's always going to be 50.

So that should mean all racers are created equal, right? So if all racers are created equal, all of them run at exactly the same speed. So if you think about everyone running at exactly the same speed, the weirdnesses begin to come up.

So here, we have a race, and we start. Everyone's just cruising. You're like, oh, I can't tell any difference. They're all like, we're all the winner. You look at that, and you get a little

disturbed. You're like, whoa. My code looked when I crossed the finish line who else was crossing. I was supposed to – if no one else had crossed before me, I was red. If someone else crossed before me, I wasn't supposed to turn red. So how come we're all turning red?

Here's where the real trick with multi-threaded programming comes in. What happens is everyone finishes the race at the same time. They all do their count at the same time with the shared rate. Some of them may do their count slightly earlier, but the ones who finish their count slightly earlier, they do a victory dance.

So everyone's at the finish line dancing, and everyone, when they looked around, at the time they looked around, no one else had crossed the finish line. Then they all say, I finished at the same time, and no one else finished before me, so they all turn red. So you might say, okay, Maron. That's kind of funky. What happens if the differences are just small?

So I'll give you one last example, just to show you the full funkiness of threads. So rather than the difference in each time step being between 50 and 150, it's just between 50 and 65. That means a whole bunch of people are going to finish the race at close to the same time. So if they finish the race at close to the same time, here's what happens.

We just tickled that bug again. Let's do it one more time. That's the other problem that you'll see with multi-threaded programming is there's a lot more things that can go wrong, which is why you're not required to know it for this class. Let me just show you the last example. Oh. It's just toying with me now. Green button. Ah. Most of the time, the green button doesn't work, but now it does.

So now, we start the race, and there's very tiny differences between these racers. What happens when there's tiny differences? Let's do it again. Yeah. Some people finish first, and some people don't. This is what makes multi-threaded programming hard. It's how to keep track of stuff like this, but now you know. So I'll see you on Friday.

[End of Audio]

Duration: 50 minutes

## Programming Methodology-Lecture26

**Instructor (Mehran Sahami):** All right. Welcome back to – what kind of day is it going to be in 106a? Anyone want to – fun-filled and exciting. It always is. Thanks for playing along. So a couple quick announcements before we start. One announcement is that there is one handout today, so several people have asked in the past, these programs that I make in these classes are kind of cool. I would like to be able to share them with my friends and relatives and whoever else. We're going to talk a little bit about how you do that today and what that means underneath the hood, but the handout actually explains it all. It's this notion of a jar file. We'll talk more about a jar file as we go along.

The graphics contest, for those of you who are doing it, is due today. Just wondering, quick show of hands, how many people entered the graphics contest. Wow. Not as many as I would've thought. There could be a couple people who are at home, even if you don't win of getting 100 on the final in a random drawing. So that's a good sign.

One thing I do want to check, I just heard a little bit before class that some folks were having some trouble submitting their graphics contest because there actually might have been an issue with the server that takes submissions. So if you submitted to the graphics contest, whether you're in here or you happen to be watching the video, email me, and let me know what the name of your contest entry was. That way, I know for sure that we actually got all the contest entries that we think we had, and if we didn't get one, I can email you back. The thing I would ask you is, if you can't email me any time this weekend because this weekend is actually, when we're going to make the first pass looking over all the contest entries, and then we're going to have a small pool that we'll take to the section leaders. They will vote and give the winner. I'll announce the winner in class next week.

I might show a demo of the winning two as well. We might do the random drawing in class as well to see who actually gets the third coveted random drawing spot, even if you don't win. So please email me if you entered the graphics contest, just to make sure.

One other thing with email for SCPD students, I know it's still a little too early to think about final exams, but if you're an SCPD student, it's not too early. If you're not an SCPD student, it's not too early, either. But for SCPD students, if you're taking the final exam, if you plan of taking it at your sites and you're not going to come on campus to take it, email me by 5:00 p.m. December 5, letting me know that you're taking it at your site and the name and email of your site coordinator, just like the midterm. That way, I can get the information to your site coordinator for the final well before the final.

If you're planning on coming on campus to take the final, you can feel free to send me an email to say you're coming on campus. If I don't hear from you, I will assume you're coming on campus. So you only need to email me if you're taking it at your site, so please do that if you're an SCPD student and you plan on taking it at your site.

Any questions about anything we've done so far before we dive into our next great topic?  
All right.

One of the things that we've done the whole time in this class is we use these things called the ACM libraries. The ACM libraries are a set of libraries that are actually created by a task force of people. The ACM is the Association of Computing Machinery. We talked about them at the very beginning of the class when we talked about these libraries. They put together some nice libraries of stuff that are really useful for teaching, which is why we use them.

Today, what I'm going to do is lift a little bit underneath the hood and talk about standard Java, which is what you would get if you didn't use the ACM libraries and you just used the standard Java libraries. Now, there's no reason why you can't continue to use the ACM libraries after this class. They're just another set of libraries that were written by a group of people that you're certainly welcome to use.

So there's no reason why you should stop using them, but there were a couple important issues related to standard Java. Now it's time for you to know. So the first thing that's related to thinking about standard Java is when you're running your programs, when you go into Eclipse and you click on the little running guy to compile your programs. It give you a list of what classes you actually might want to run.

If you only have one project, you may only get one choice, but one of the things you kind of think about is in the name surfer program, I actually have four or five different classes. How come it always knew which class to run? How come it always knew the name surfer class was the class that I actually should run? Anyone want to venture a guess?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It's the only class with [inaudible] which is very related to an underlying issue. It's the only class that actually was extending programs. So one of the extended programs, what actually was happening in these ACM libraries is you were getting a method called main. Main is actually the place – you're old enough to see main. Main is actually the method at which Java classes actually start running.

So one of the things you should think now, you never wrote a method called main. I never saw a method called main, and you're telling me that's where Java programs actually start running. Yeah, in fact it is. It's because programs provided this main method for you. What this main method did in the program was essentially get the rest of you program running by getting a few things set up and then kicking off your run method. So you didn't actually need to worry about this.

But now you're sort of old enough to actually see what that main method is all about. So if we think about what this main method does, the header for the main method is also kind of weird. This is part of the reason why we never showed you the main method before. The header for the main method is actually public static void main, but we're not

done yet. Main actually has some arguments. It has an array of strings called args and arguments, and then something in here happens inside of me.

If we showed this to you on the first day, we would've had to go through and explain what all these words meant before we explained what main even was, before we explained how you write your first program. That would've been a pain. Now we can just tell you. Public mains is a public method. You know that. You probably recall the other public methods you've written. Static means that this is actually a method that belongs to the class – it's not something that you would actually call on a particular object.

So you never have some object – like, here's my object X, and I call X .main. Main is just something that gets called. It's a class method as opposed to being a method that gets called on an instance. Void means it just returns nothing.

What is getting passed in here is an array of strings. Where is that array of strings coming from? This actually harks back to when computers weren't all nice and graphical and everything. When people wrote programs, they wrote program and were typing on what's called a command line. They wrote the name of the program out. They actually typed it, and then they typed a bunch of things that they wanted to be passed into the program such as initial information to start that program. That was the initial thing, so if you had some program like name surfer, you might actually start off by giving the name of the program.

Then after name surfer, you might give it the name of the data file, like data dot text. You might've given it some other things as well that were separated by spaces. This list of stuff is essentially what gets passed in here as arguments. They're strings, and this is how the program would actually know what came in on the command line when the program was kicked off. Java's not that old of a language. It sort of came around and gaining popularity in 1995. People weren't doing a lot of this in 1995. I already had my mouse and my folders and all this other stuff, even if you were six years old. You probably did.

You're like, I never typed this stuff, so why do I care about it? The reason why Java's derived from another language called C, and there's a variation called C++ that was created when people were writing programs in the days of yore. The whole notion of main and having some arguments to get passed to main kind of came along with the baggage of actually having a program language that matches the same style programming languages when they did do this.

So a lot of the times in real Java programs these days, there aren't really any arguments. If there are arguments, there's some system parameters or something like that. We don't usually worry about them. So when you go and look at some other Java program that isn't using the ACM libraries and you see this main thing, and you're wondering what it's all about, you can think of main analogously to run. It's just where the whole time you've been thinking of run as where you're execution starts, main is really where execution started.

If you think about execution actually started in main, so how did this thing actually kick off my run method? Now you're sort of old enough to see that, too. So what it actually did – let's say this was the main method is something like name surfer. So somewhere inside of a program, inside of the ACM libraries for program, we had this main method that figured out what the name of your class was. Essentially, it had a one-liner in it that would've been equivalent to this.

New name surfer dot start cards. So it's a one liner. Now you know what this means. What was it actually doing? When main started, no objects exist in the world. It's a static method. So there's no object that you're giving the main message to. Main just wakes up and says, hey, I'm main. What am I going to do? Why don't I create some object of this particular type, name surfer, which happens to scan an object, which is your program. Remember your program, as we kind of talked about, implements the run method, and actually a program underneath the hood implements the runnable interface that we talked about last time, with threads. We talked about the runnable interface.

All programs implement the runnable interface. How do you kick off something that's runnable? You say start. So what it basically did was created an object of your class, which was name surfer, and told it to start. It happened to start pass along these arguments, but you never needed to see those arguments. As a matter of fact, you never did see them because when your object was in stand shaded, it didn't expect any arguments. So the arguments actually got passed to this thing called start, which just ignored them, basically, and then started your run methods to kick everything off.

Last time when we talked about start, we talked about this in the context of threads. So we said, oh, you create a new thread, and the object that's you're going to start running, we put inside a thread. We kick the thread off with start. In this case, we're not actually creating a new thread. We're just saying we want to start executing this object I just created. It implements runnable, so you'll start running from the method run, but I'm not creating it in thread. So this thing is going to execute in the same thread of execution as the entire class.

So I don't suddenly kick off something that's running in parallel with this guy. It's actually going to sequentially start name surfer, and that's the last thing this guy does. Run your whole program. Any questions about that? Kind of a funky concept, but that's basically what's going on. You should see it. We were creating an instance of your program and then just kicking it off. That's why, this whole time, we had this thing called the run method that had to be public because it was implementing the runnable interface. But now you've seen main.

We could've just had main in your program to begin with and included all this code. The only reason we didn't put it in there before is because we didn't want to explain any of this stuff. In week two of the class, right after [inaudible] all holding hands and singing Kum Ba Ya. We're like, oh, it's [inaudible] the robot. Let's give Java, public static void main. Straightening args. You're like, what is going on?

We hadn't done arrays. We hadn't done classes worrying about static. We certainly hadn't done methods. We hadn't even done parameter passing. So we just waited until the end. Now you see it.

Now that we have that idea, now we can think about, okay, if this is kind of what the standard Java world is, let me think about taking this idea and using it to help me take my existing programs that I've written and pack them up into a form so I can share them with family and friends. So that's what we're going to do next. The basic concept of doing this is something that's called a jar file. You've actually seen jar files before because you've been working with something this whole time in your projects called ACM dot jar. This was just a jar file that contained all the ACM libraries.

Basically, all a jar file is, where does it get its name. It's not a big mason jar, although you can think of it like that. It stands for Java archive. That's where the name comes from. The basic idea behind the jar file is this can contain a whole bunch of different stuff. Most of the time, what it contains, is a bunch of classes in Java. You can think of this as the compiled version of the classes.

You can actually put source code inside the jar if you want, but most of the time when you get a jar, it doesn't have the source code associated with it. I just has the actual dot class files, which are the compiled version of the files. So you could put source files in here if you wanted. You could put data files in here if you wanted. You could put a bunch of things in here if you want, but we're really going to focus on the case of putting classes in here.

So one of these already existed for you. It was ACM dot jar, and we're going to figure out how to actually create some of these ourselves and use them because you can actually think of them as something that's executable.

So if we come to the computer, here's name surfer. This is an actual working version of name surfer, so I'm not going to show you the rest of the files in case you're taking [inaudible]. All the code's in here, so the first thing I did in the name surfer program is I thought about, hey, I want to think of this in the standard Java world now, even though you're still using the ACM libraries. When I want to build this jar, I want to build it in a way that sort of makes it maximally portable. I can give it to someone who's over here on this PC and someone who's over here on this Mac. They don't need to have a Eclipse or anything like that. They can just run it. That's the whole point.

So the first thing I'm going to do is to introduce my friend, the main method. So basically I put in the code you just saw. I add a method, public static void main has this array of strings called args that sets the parameters. That's just the way main is always defined to be. What it's going to do is create a new name surfer object and kick it off. That's the only thing I need to add to my program. So anywhere you had some class that extended program, you would add these three lines of code to get it compliant with standard Java.

Once we do that, we need to create the jar file, which is the thing we're actually going to be able to execute. One thing you might be saying to yourself is that the ACM libraries already provide this. So why am I putting it explicitly in there? As a matter of fact, it's not super required that it be in there. The real reason why we put it in there is to try and maximize portability. Even though Java's supposed to have this property where we write the Java code once and it can run on PCs and it can run on Macs and it can run on Vista and Tiger and all this other stuff, in reality, there's some little differences between these operating systems. So little problems creep up here than there, every once in a while. You actually saw a few of those in class on occasion. Some of you experienced them in the Lair.

By putting in this line explicitly rather than kind of relying on the code that's in the ACM libraries, this is actually doing a bunch more complicated stuff. It needs to infer what the name of your class was. You never told it explicitly, hey, ACM library, I need a new one of these guys. Java has a facility called reflection where it can go and say, oh, let me take a moment to reflect. I'm going to go and look at the names of your classes and then do something to actually generate some code based on the names of your classes, which is what it was doing.

But that kind of stuff can get a little bit frosty, so we're just going to say, hey, ACM, don't worry about it. I'm just giving it to you. I'm a nice guy. So just put it in to make it explicit. It maximizes portability is the main reason we put it in.

So how do we create the jar file? What we're going to do – these steps are all in excruciating detail in your handout, so you don't need to worry about scribbling down notes quickly. What we're going to do is you first click – that's the most difficult part of the whole thing. You select the project that you want to create the jar file from. So name surfer. Then we go to the file menu, and we pick export. So this whole time you were doing imports when you were bringing stuff in, now it's finally time to give something back in the form of an export.

So what you want to do when you click export is it brings up a little dialogue box. Sometimes this is closed. Sometimes it's open. It's not a big deal if it's closed. Open it. Inside Java, you click on jar file. That's what you want to create. It has this nice little jar icon there to remind you it's a jar file. You click next. I'm just going to take you through all the steps.

We need to specify a couple things here. First we need to specify what's in the jar file. So I'm going to open up name surfer, and what I'm going to put inside the jar file – I don't want to have all of name surfer in the jar file. In terms of if you throw everything in there including this stuff called dot project and dot class pass, it actually gets a little bit confused. Some of those things are not germane to what we want to pack up in our jar. They're just kind of other administrative information.

What we really want to have is everything inside the default package. So I just click on default package. I can double click on this to make sure that all of my Java files were

checked. That's what I want in my jar. I want basically all of my Java files or the compiled version of all of my Java files. I make sure that this is checked. It's checked by default. Just make export generated class files and resources. So what it's going to do is compile those Java files into their corresponding class files. That's what it's going to put in the jar.

There are other options, like I can export the Java source files if I wanted to. Then I give someone the actual source code. Most of the time, you don't want to do this if you don't want people sniffing through your source code. You're like, here, take the compiled – the ACM libraries, we're not going to give you the source code. Just take the ACM libraries. They're good for you.

So once we have this, we need to specify where we want to save this jar file. So that's what select export destination. They should've just said, where do you want to save it. That's the export destination. We can browse around. Basically, I've already created a folder that has all my name surfer code in it over here. I'm going to save the jar file in that same folder. So this is just in the same folder for my project called name surfer that has all of my class files in it. You can put it wherever you want, just don't forget. That's kind of the key.

I'm going to save it here. You don't need to worry about the options down here. They're just fine in the defaults that they're at. You click next. Then you come to the screen that seems like, what's going on. Export files with compile warnings. Yeah, we just want to export everything. So we just click next.

This is the most interesting part of the whole thing. What we want to do when we actually create this jar file, and this is something we only need to do when we have other jar files like the ACM libraries, we need to generate a manifest file. Basically, all a manifest file is, it's a complicated name. It sort of sounds formal. It sounds like they're on a boat, and they're like, oh, where's the passenger manifest? Really, all a manifest is – I'll show you the manifest file. It's two lines long, and then you add one line to it. It's basically just a little bit of administrative information that's kept around with your jar file so it knows what kind of stuff you're using with this jar file. That's all it is.

So what we want to do is generate the manifest file. Make sure you save the manifest in the workspace is checked, which it should be. Then you need to specify where you want to save the manifest file. The place I want to save it is basically in the same folder for my name surfer project in the name manifest. You could browse around if you wanted to, but usually the name you give it is the name of the folder that all your project stuff's in, and then the name manifest, which is going to be the actual name of the file. Any questions about that? Manifest file?

The other thing you need to do when you're specifying a jar is a jar's just a bunch of classes. It needs to know in some sense if someone ends up running this jar – you'll see that in just a second. You can run a jar – where should it start? Which class should I call its main method. That's what you specify here. Select the class of the application entry

point. It's kind of a very formal way of saying, where should I start running? So the place I want to start running, it lists to you all the classes that have a main method. Those are all the ones that can start running. Name surfer is the only one.

So I click okay. So it says the main class is name surfer. Now there's no more next buttons. Now all I can do is click finish, and I just created a jar file. You're like, oh, [inaudible] double click. Double click. We're not there yet.

There's two things that you'll notice if you look over here in the package explorer. You'll notice now we have something called a manifest file because that's where I saved it. I also have name surfer dot jar, the jar file I just created. They were both put in the same folder with my other files. That's where I wanted to save them so they happened to show up here in the package explorer. Here's the funky thing. Even though we created this manifest, and we created this jar, they don't have quite the right information that we want. So what we do is we double click on the manifest file to open it up. Here's the whole manifest file.

It's got a version, which is 1.0, and it's got the main class, which is name surfer. Why does it know the main class is name surfer? Because I told it. That's the application entry point. Just for letting me know that I told it, that's where the application starts.

There's one other thing I need. What I need to do is say, you know what, that's a good time, but you also need to use the ACM libraries. It says, you didn't tell me about the ACM libraries. You say, well, now I'm going to add the coveted third line that was talked about in the manifest file. So I'm actually going to modify the manifest file and add something called a class path.

All a class path is, it actually looks just like this, C-P. Class path just tells basically the application what other stuff were you using. Are there other jar files that you're using? So what you specify here is the name of any jar files that you're going to be using as part of your program, separated by spaces.

So I'm going to use ACM dot jar. Here's ACM dot jar over here. I'm going to use the ACM libraries, and those are all in some jar file. I'm also going to use the jar file I just created, name surfer dot jar. If I had three or more jar files, I'd list them all on the same line. So if you go and write some application some day where you're like, hey, I got this jar file for my friend. Here's this jar file I downloaded from the web, which I wouldn't encourage you to do, and there's this other jar file from somewhere else. You just list them all here with space in between.

Then we save the file. That's probably the most difficult thing that people forget. So you save the file. You save the manifest file. What do you do after you do all this? Create a jar all over again. Why? Just trust me.

Here's how it's going to work. I'm going to do it a little bit different this time. Rather than going to the file menu and picking export, I'm going to take the advanced course. I'm

going to right click on the project name and click export. It brings up exactly the same window. I'm going to export a jar file. Come over here. What do I want to export? Oh, yeah. It's all this [inaudible] again. What I want to export is not all this stuff. I just want to export everything in my default package. Where do I want to save it? I want to save it in the same place I had before. You might say, but aren't you going to replace the one that already exists? Yeah, I need to replace the one that already exists because I updated my manifest.

I need to say it's not just about you anymore. Now it also involved ACM jar. You can replace the old one. So I'm going to put it in the places before. Then I click next. Here's the thing about export warnings, export affairs. I don't care. I'm just going to export. Here is the only place where things are different. The second time I go through this whole thing, I don't want to generate another manifest file. I generated a manifest file the first time, and then I modified it. What I want to do is use that modified manifest file. So I say use existing manifest from workspace, which is again, just a formal way of saying use the manifest file that's already there.

It asks me, where is the manifest file? Strangely enough, if you look, this is exactly the same text as here. No coincidence. You just saved it. That's the one you want to use. It's going to be the same name. Now notice which class of the entry point is grayed out. It's grayed out because I told it before, your entry point is name surfer. That's in the manifest file. If that's already in a manifest file, I'm using it, it doesn't need to know it again. So it doesn't even let me specify it again. Just say that's the difference. Then I click finish.

They give me one last one. Are you sure you want to override that last jar file? He was your friend. I say yes. We didn't know each other that well. Now I've created this jar file. What do I do now? Well, I kind of come over here, and I say, here's the jar file I just made. Now one thing I can do with the jar file is I can double click it. If I double click on the jar file, it just starts kicking off the name surfer application. So I don't need Eclipse anymore to go and run and have a little running dude. I have a little application.

I'm like, oh, yeah. How popular was that? Bob's kind of fallen off in the meantime. There are some other interesting ones. I was looking at these the other night. Verna. After the '60s, done deal. Thanks for playing. So name your children Verna. Make a comeback.

So name surfer's just kind of running here. It's fun. It's just a little stand-alone application. If you wanted to pack this up and send it to a friend of yours, you wouldn't just send name surfer dot jar. What you would do is say, hey, you know what I want to do? I want to create – I'll just do it in here – some new folder. So I'm going to create some new folder, and I'll call this my program or whatever you want to call it. I'm just not going to call it name surfer again because I already have a folder called name surfer. My program. What I'm going to put in my program, that folder, is the jar of my program. I'm also going to put in the jar of the ACM library because I also need that jar. That's separate.

I also need in here any data files like names dot data. That's all the data that gets read when the program starts. It's not like magically that's just going to be there now. It's still

going to go and try to find that file. So I still need to put all that stuff here. Now this folder is something I can zip up and send it to a friend of mind. When my friend gets it, they would be like, oh, name surfer dot jar. Then they get going.

So now you can package up any program that you've written in this class. Remember to put in main. You got to go through this two-part [inaudible] the jar, export it, process, modify the manifest, create another jar, put it out there again. But you're good to go. Any questions about that? Now you can package up and share with friends.

What's even cooler than sharing with friends that you can email stuff to is sharing with friends on the web. You just have millions of friends on the web. Most of them you just don't know about, but they're probably on in the late hours of the night, looking at your web pages and things. You can actually take any files that you create here, like jar files, and make them available in a web browser.

There's one other thing I should mention, and this is kind of a little thing. Before you go and create all this stuff and send it off to your mom and dad and be like, mom, dad, breakout. Go play. It's a good time. In order for them to run your jar files, they need to have the Java run time environment installed. Remember on the second week of class where we said go to the CS106 web site. You need to download a clip. There's this thing called the JRE you also need to download.

If you have a mac, the JRE in most cases is already installed. If you have Windows, it's not installed. They need to go down, and you can send them to the CS106 page and say download the JRE. Here's a copy of handout No. 5. Go ahead and install it, and then they can run your programs. Your program can't run if the computer doesn't have Java, right? It's a bunch of stuff that's going to execute Java byte code, and your computer's like, what's Java byte code? I don't know what to do with it, so it won't run. They need the Java Runtime Environment.

Now assuming someone has the JRE, they can go to a web page. You might put some page on the web that allows you to load your applet. Your applet is just a webified version of your program. So here look. It's running inside a web browser. This isn't the actual application. This is my web browser. I could go to some search engine from here. I'm sitting in the web browser. I'm not just running a regular application on my desktop. This guy's actually running in my web browser.

Here's how I make it happen. I create a web page. If you don't know about HTML and creating web pages, unfortunately I can't explain that to you in five minutes. What I can show you is basically what this file's going to look like. So if you know a little bit of HTML or you just want to essentially copy and paste this idea, this will work for you. So all you do is say – the entire page that allows your applet to run. You say this page is HTML. There's these little things called tags.

The name of that page is name surfer. I want to create a little table. I'm going to create a little border around my application. What's my application? My whole application is right

here. I have an applet. What's the name of the archive, which is a Java archive that contains my applet? It's name surfer dot jar. Code is what's the entry point. This guy no longer has a manifest file available to us. What's my entry point? Name surfer dot class. That's where you start running. So it says I'm going to go name surfer dot class, find its name. That's where I start running.

The space I'm going to give you to run on the screen is 700 by 500. That little snippet of code is what goes on whatever web server this is serving on. It looks for those jar files, slaps them into the page and then they're good to go to run more java program inside their web browser. How many people know HTML? You folks. There's some number of folks that this might be a reasonable thing to do. If not, you just don't need to worry about it. It's not a big deal.

The one other thing you should know if you create a web page is that when something's running in the web page, it doesn't have access to the rest of the file system. What that means is if I actually happen to be over here – notice I have index dot HTML. I have name surfer dot jar, and I have ACM dot jar. What happens in my names dash data file? It doesn't exist. Why doesn't it exist? Because I couldn't read it anyway. Once I'm running on the web browser, for security reasons, it doesn't let you go in and pull stuff out of your file system.

If it did, people could do really bad things to your computer. So what you need to do if you want to find that data, you let us run name surfer on the web. What did you do? Here's the dirty little secret. I actually created a giant array that had all the data in it and made it part of the program. So sometimes you can do stuff like that if you don't actually want to read from a file. The other thing you can do is you can take those files and include them in the jar file because the jar file cannot only have compiled [inaudible]. It can also have data files. That's another way of doing it if you want to do it that way.

We don't have time to go into all the details. It's kind of the same process. When you're exporting stuff to the jar file, you include some data files in there. So what's creating the executable. Now that we know all this funky stuff about executables put on a web page if I want. I'm feeling happy. I'm good to go. It's time to come back to our friend, standard Java.

Standard Java's what allowed us to think about doing this stuff because we learned about main. I want to show you a couple examples of programs that actually don't use the ACM libraries at all to show you why we use the ACM libraries. One of the things you might be wondering is why weren't we just doing the standard Java thing the whole time? Part of the reason is things are just so much easier and cooler when you have the ACM libraries.

So if you go back over to Eclipse, we're kind of done with name surfer and all this manifest stuff. Here's a program that's written in standard Java that writes out hello world on the screen. This is something you kind of did in the first class by saying public class

hello world extends console program in printlin. Hello world out to the screen, and you would've gotten it in the console.

So what's different here? What's different is we have public class hello world, and it doesn't extend anything. It doesn't extend program. It doesn't extend console program. There are no imports for the ACM libraries. We're not using any of the ACM stuff. So we don't have a console program. We don't have a nice console that we write stuff out to that displays in a nice little window. What we do have is something called the system output console. If we want to put stuff on that, it looks real similar to what you had before. We used printlin, which is why we made the method that you used called printlin to match this. But we need to say system dot out dot prinlin and then the text we want to print out.

Again, here I have a main method. My main method can have whatever I want in it. This is just where execution starts. So if I compile this and run it – let me show you why. This world is not all that cool. Here's hello world. It just ran. You're like, I don't see anything. You don't get a cool window that comes up and is like, hello world. You have the system console. The system console if you happen to be using a development environment like Eclipse or something else, it's basically just a window in that development environment that shows messages that you print out. If you happen to be in the bad old days that I think I erased over here where you have command line text where you actually type stuff in, the console would just be that same window where the text would appear where you type stuff.

If you don't get it in a separate window, it's not that cool. A lot of times, this window's close anyway. If you wanted to kick it up a notch, why use the console? Why not do something graphical. So here's graphical hello world. What we want to do is create a window that is going to have some title associated with it. We're going to put the text hello world in that window. What do we need to do? Again, execution starts at main. We need to create something called a J frame, which you never had to worry about before. What's a J frame? It's actually a frame that's going to hold the window.

We're going to start running our application. It's going to create a little window for us that we're going to display stuff in. What are we going to put in that window? We're going to put a label. J labels you've seen before. This is just like you've seen. We're going to create a J label that's called hello world, and we want this label to be center justified as opposed to left justified or right justified.

So I need to say J label dot center to center justify it. If I don't give it a justification, it'll by default be left justified and look kind of ugly because we want it center. Then I add this label to my frame. So similar to the idea of having a canvas and adding stuff to the canvas, it's exactly analogous. We want to make it just as easy. So when you saw standard Java, all the same concepts would apply. Here, we're just adding the J label, which as you know now, is a J component in the big Java hierarchy. J components can be added to J frames.

So we have a J label that gets added to a J frame. I set the size for that J frame, which is 500 by 300. That tells me how big the window's going to be when it starts. Then there's this other stuff that I need to do. You would think why do I need to do that? It doesn't make any sense that I would not otherwise want to have it this way. Here's what I need to do.

If someone clicks close on the window, I need to say, hey, if someone clicked that, then you need to close yourself. Otherwise, the window goes away and the application keeps running. It seems odd, but we need to have that there for this application to stop running. Then we say window, I know I created you and everything. You need to make yourself visible. Otherwise, no one will be able to see you.

You're like, why would I create a window that I wasn't going to make visible? Yeah, that's why we use the ACM libraries. So we need to make sure that this guy's visibility's true. If we run this, here's graphical hello. So after all this stuff, here's graphical hello.

Yeah, would you want all that explained to you so you can get hello world? You're like, I'm writing a social network. I don't need to worry about hello world in the middle of my screen. That's because you have the ACM libraries. So you're like, okay, let's kick it up even one more notch. You're like, [inaudible] all this mouse stuff and interaction, right? That should be something I get some benefit from having standard Java. So we'll have an interactive version of hello.

Center the interactive version of hello. Now some of this stuff should begin to get a little more repetitive in the sense that you have your main. You have your J frame. The frame is called interactive hello. That's the title of that window. What we're going to add to this J frame is a new class that we're going to create called a moving label. I'll show you what a moving label down in just a second, but we need to set the size of the window, set default close operation exit on close. Set visibility to true.

Basically all this does is create this window of a particular size. It's going to add this thing called a moving label to it. Moving label's just another class I create. What's a moving label? A moving label is a J component. It needs to be a J component because I'm going to add it to a frame. To display something on a frame, I can only display components on the frame. This is going to extend component.

It's going to implement our friend, mouse listener. It's going to listen for the mouse. You're like, oh, I remember mouse listener. That where the mouse got clicked and dragged. That stuff's exactly the same. So I have my constructor. What my constructor has, it has some starting text for this label at its starting X and Y location. That should look kind of familiar to you. Basically I just store off the text. I store it at X and Y. This guy wants to listen for mouse events.

So it says add a mouse listener, and it needs to say if you get some mouse event, send them to this. So this is a little bit different than you've written in your programs before

where you just said add mouse listener. You just had an open [inaudible]. That's because we kind of took care of the rest of this stuff for you.

Here's where things get a little bit funky. The difference between this and thinking about having some sort of label that you just display on a canvas, some text that just sits there, is that this guy now has to worry about what's known as painting himself. That means it needs to draw itself on the screen.

Well, when I had labels before, they just knew how to draw themselves. Yeah, that's because we gave you a label that was a little bit smarter and knew that it was a label that should draw itself. This guy needs to be told to draw himself. There's a method called paint component that gets called whenever this guys gets displayed on the screen. There's some other thread that's going to call it for you automatically. Here's the graphics context in which your going to call yourself.

Within that graphics context, I'm going to draw some string. The actual method name and everything is not important here. It just shows you that there is extra stuff you need to worry about, which is why we didn't want to do all this stuff to begin with. This stuff you've seen before. What happens if a mouse is clicked? I get the new XY location. I repaint. What does repaint mean? It means redraw yourself. I'm going to redraw myself at this new XY location. When I call repaint, someone comes along and says, hey, to repaint this area, I'm going to call your paint component method so you can repaint yourself.

Whoa, this is really weird. I'm over here. I get a mouse click, and I know that I want to redraw myself. But rather than telling myself directly to redraw myself, I go and tell the system, I need to get repainted. Then the system says, that's the start of Jabba the Hutt kind of sitting there fat and happy. Some day, if you actually – we won't get into it.

You go and say I need to repaint myself. It says, okay, you need to repaint yourself. Well, when I'm ready for you to repaint yourself, I'll call your paint component method. Until then, you don't repaint yourself. So it delays other stuff in the system to worry about, and then oh, yeah. There was you. You asked to get repainted. I'll call your paint component. You say, okay, well, now I'm going to draw myself at the new XY location.

So it kind of convoluted the whole notion of you're doing something here, and you're asking someone else to do something for you. Then they're going to call you back to do what you originally intended to do. Now it makes a little bit more sense after we talked about threads and after you've seen all this stuff. Third day of class, not so hot.

So if we run this, this is called interactive below. Basically, what it does is it just brings up our particular message, CS106A rocks in the middle of the screen. Now every time I click the mouse button because this is the mouse click event over here. Every time I click the mouse button, the XY location of the mouse becomes the new base point for the text that gets drawn. So it just moves around the screen.

Any questions about that? So this is standard Java. You've seen all the concepts using the ACM libraries. The notion of adding things, having mouse listeners. As a matter of fact, the whole mouse listeners concept, we just took the standard Java ideas and used them sort of in conjunction with the ACM libraries. But there's a lot of things in the ACM libraries that just made it so much easier to, for example, to graphics contest entries or to write a social network.

So you're welcome to continue to use the ACM libraries after this class. Some people were wondering why we use these libraries, and this is the reason why. There's a whole lot of stuff you'd have to worry about otherwise. Questions?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** You'd be using your favorite word processor, like notepad. There's actually some places which I won't name, but it's actually reasonable that some schools in your first programming class, what you do is they say, you need to have notepad, or you need to have some text editor. Then we're going to do command line stuff. You're going to type in name surfer, names dash data dot text to run your program. Then everything is going to be [inaudible].

So one thing I want to leave you with now, in our final few moments together, is sort of a notion – we're still going to meet next week, talk about life after this class, but if you want to go on in terms of learning more about Java, especially standard Java, we sort of just started things off by giving you this book, which talks all about the ACM libraries. This, I think, is a great book to actually learn everything with and use the ACM libraries. But if you want to go on, what are some other resources you can use?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** It's not pretty. If you go to the Java section of any bookstore, just be prepared to stay a while. One book that I would recommend, not that I get any kickbacks from these folks, called Learning Java. It's actually a pretty good time. Some of the examples you actually saw here were based on this book. It's a little big. That's why we don't use it as a textbook in this class. Will we break the 1,000-page mark. It's so close. No, 950. Sorry, close.

There are some other books. Actually, the original specification of the Java programming language. This is an older version of the book. This was when I was a wee tyke in the days of yore and got the old version. I forgot what version they're up to now. This was second edition. I think now they're up to three or four. Something like that. The books are a little bit bigger, but for a book that specifies the language, Java, it's actually very well-written as a reference. So I'd recommend this as well.

If you're into oh, I want it all, Big Java. It's big. I think it might actually cross the 1,000 page – oh, yeah. It's like 1,200. If you really want to get hardcore and you're all about web-based Java, Java Server Programming. Everything you would want and a lot of

things you don't want to know. Everything you want to know and more. That's just a small set.

So these are just a few books I'd recommend if you want to go on beyond this class. But you can go into any bookstore, and you get inundated with that kind of stuff. Now you have a context for putting all the pieces together because you've seen all the things that you actually need to know to be able to work with the huge set of tools that Java actually has. Any questions about any of this stuff?

You're good to go? All right. I'll let you go a couple minutes early because most of the time, I let you go a couple minutes late. Have a good weekend.

[End of Audio]

Duration: 45 minutes

## Programming Methodology-Lecture27

**Instructor (Mehran Sahami):** So welcome back. Wow. That's a little loud. To our last week of cs106a. Of course, it is another fun filled exciting day despite it being our last week. We're getting down to the end. We have class today, there's class on Wednesday, there's no class on Friday. So next time will be our last day. But a few announcements. There's actually just a load of announcements because we're so close to the end of the quarter. First announcement, there's one handout, which is your section handout for this week. There are still sections this week, so despite the fact that we don't have class on Friday, still go to your sections this week.

There's a couple problems on the section handout, as well as the sectional will just be a general review for the final exam in case you have any questions. That's a good place to ask them. Also it would be a good place if you want to ask some questions say, about [inaudible] for example, the last assignment. Just wondering, how many have started assignment number seven? Wow, good to see. Anyone done with assignment number seven? A couple of folks. That's good to know. I might talk to you afterwards as to how much time it actually took you, but hopefully, it wasn't too painful. The graphics contest was due last Friday. The winners will be announced in next class, so Ben and I actually took a first pass already over all the contest entries. There was actually some very impressive entries in the contest. Things you were just kind of jaw dropping, like, go and show them to other faculty in the department because they're just that cool. But this afternoon we are having our staff meeting with all the section leaders and they will actually be the ones voting and deciding on the winners in each category. So we'll give them the short list and they'll make the final determination, and then on Wednesday, I'll announce it to and I'll check with the winners of the contest to see if they're okay demo'ing it. But if they're okay demo'ing it, then I'll show you the winning contest entries as well, plus on Wednesday we'll have the random drawing to give away the last sort of grand prize. But if you didn't happen to win, just for entering you still get an entry into the grand prize or you could just get a free 100 percent on any assignment in the class including the final exam. So assignment number seven, we just talked a little bit about. Since it's due the last day of the quarter on Friday, but we don't have class that day, it's just electronic submission. So if you're wondering about what do I do with the hard copy, you don't need to do anything with the hard copy. We just need electronic submissions.

For the other assignments in this class, we requested that you turn in a hard copy because when you did interactive grading or if you're a section leader to write comments on, we actually had something that they could mark up. For assignment number seven, because it's due the last day of the quarter, there will be no interactive grading for the last assignment so the down side is there's no interactive grading for it, the plus side of that is that you don't need to turn in a hard copy because we can just take a look at your online submission to figure out functionality and other kinds of things. And just as a reminder, even though it wasn't clear that was on the handout, like, right on the front the last couple days, no late days on assignment number seven. Just in case you're wondering. So final exam, it's time to start thinking about the final exam. Finals are next week. You probably

know that, but just in case you didn't, the final exam is just like the mid-term. It's open book and open notes, so you can bring in your text book for the class. Feel free to bring in all print outs of all your programs, all the notes you've taken in the class, all your handouts. That stuffs all open, but just like the mid-term, it is a closed computer exam. So if you have a laptop or a PDA or whatever, you can't use that during the exam. Same rules basically applies to the mid-term. And we'll talk a little bit more about the final when we actually do review for it next class.

The regular final is scheduled for Thursday of finals week. That's December 13th, 12:15 to 3:15 P.M. in Kresge Auditorium, which is the same place we had the mid-term. One of the only rooms large enough to actually accommodate us, and the alternate has been scheduled. So alternate final is December 12th, that's Wednesday of finals week, 3:30 - 6:30 in the afternoon, also in Kresge Aud. So this time seemed like a relatively unpopular time for other final exams. And you're free to take either one. So you don't need to send me an email saying you have a conflict with the regular exam or whatever. If you just want to get done with your finals earlier and you want to take the alternate exam, you're just welcome to take the alternate exam. But only take one exam. So you can pick one, and just one. That's just life in the city. Okay. And if you're an SEPD student, I announced this last time, but I'll announce it again. I've already gotten email from one of you, which is a good thing. To email me by 5:00 P.M. December 5th, that's Wednesday if you're planning on taking the exam at your site. If you're gonna come in for the exam, you don't need to email me. You can, feel free to email and say, "Hey, man, I'm gonna come into campus and take the exam." And you're welcome to take it at either one of these times as well as if you're out in SITN. But if you plan on taking it at your site, send me an email. Also, let me know the name and email address of your site administrator so I can send the site administrator your exam to administer to you. So that was just a load of announcements.

Any questions about anything before we delve into our next great topic? All right. You're feeling okay. Good. So a lot of today's class is actually about life after this class because we're getting pretty close to the end of this class. So one of the things I want you to just kind of know about and so you can think about it, are what are some of the options that are available to you afterwards. Whether or not you're just thinking about declaring a major or if you've already declared a major or you just want to get sort of a lay of the land of what's this whole computer science thing all about. Because probably the biggest thing I would stress, despite the fact that you just spent the last nine weeks programming, is computer science is not computer programming. Okay? A lot of time the two get equated, but if it was called computer programming, this class wouldn't be called programming methodology, we'd just call it something like programs that work, right, and we wouldn't worry about style and all this other stuff, and good software engineering principles, and at the same time, computer science wouldn't be called computer science, it would called something like programming. Right? And it's, like, "Oh, what did you major in?" "Oh, I majored in programming," and that's like, when you say, "Oh, I'm sorry. I think you can get shots for that kind of thing now." Because it's not just about programming. There is programming in computer science, but there's actually a science to the field and there's a lot of things that go on outside of programming and that's what

it's important to, in some sense, appreciate. So if we think about life after this class, let's first kind of deal with some of the short-term logistical kinds of things. Like, you just took this class; you might think, well, there's probably a couple things you think. You think, "Hey, Miron, that was kind of interesting, I might consider taking 106b." You might consider, "Hey, Miron, that was interesting, I might actually considering minoring or majoring in computer science." And you might say, "Hey, Miron, that was interesting, in the same way, for example, that dropping a brick on my head is interesting, and I'm gonna run screaming."

And if you're thinking the third option, I apologize, because that was not the point of this class, but here's a few things that you can potentially think about, even if you're in the third option and definitely some things to think about for the first two options. And I guess there was also that option of the, "Oh, I got the general education requirements out of the way and now I will go on figuring out what to do with the rest of my life," and if that's the case, you should pay attention as well. So what happens after 106a? So here's cs106a, this is where we're all sort of happy, and we're scrappy and we're making social networks, and life after this kind of, you know, your next immediate step is actually pretty clear. There's a class cs106b, that's called Programming Abstractions, which is the next class to take. And that class is on a language called C++, so you'll learn a whole new language, although, you'll realize when you actually CC++, that a whole bunch of things in it are just the same as Java. Whole notions of parameter passing and methods, and decomposition and objects, all those same things exist in here. Okay. But you also will get with this class called Programming Abstractions because, so far, what we've done is used a lot of extractions. There you get into a whole bunch of tradeoffs with how you can make things run more quickly versus perhaps using more memory versus different kinds of programming techniques that actually come up. There's also some really cool ideas that come up in here, which are just sort of mind blowing ideas, which is the notion – for example, one of them is called recursion, which is so far we have methods and methods call other methods and they call other methods. What if a method called itself? That's kind of weird, Miron. Why would a method call itself? Because some functions are defined in terms of themselves. Right?

If you kind of think about the factorial function – anyone remember this function? The "N" function. Right? This is N factorial. And all this really is – sorry if I just shattered your ear drums, is N times N minus 1 times N minus 2 times....times 1. You just multiply everything together. That's where [inaudible] all about. You can define a function in terms of itself. And it turns out, yeah, a factorial, that's kind of a simple way to understand it. It turns out that this is a hugely powerful concept that allows you to do all kinds of things, and this is kind of another cool thing you get in cs106b. Okay. Now, you might say, "Okay, Miron, that's still sounding like programming to me, even though I'm learning these cool concepts, isn't that just a programming class," and in some sense, yeah, this is a programming class. There's other options that are also available to you now that kind of fall into the category of being part of the CS major or the CS minor. A set of classes called cs103. And cs103 come in a different couple different flavors, like, vanilla, grape and pork – no, they come in those in a, b sequence and there's [inaudible] – I can't think of anything in the world that would come in those three flavors.

And this is really a class that in some sense is about discrete math. And you might say, “Oh, gee, Miron, besides your class, I’m taking calculus and that’s about as much fun as sliding down a 50 foot razor blade. Why would I want to do that again? Not on the sharp side, right, just imagine the other side, like, the flat side of the razor blade and it’s been made slick and it’s like a big slide. It’s fun. Wait until all my friends in the math department see that. Anyway, why would I care about this discrete math thing? Well, first of all, this is an operative word here, which means this little symbol that you have grown to know and love, our friend the integral, just nod around, right, this is all discrete, this is, like, “Hey, you know what, what we want to think about our some things that are useful to us in a computer science context,” and computers at the end of day are digital objects. Right. They have ones and zeros, which means there’s a whole bunch of things, like, sets for example and logic that come up in these things. But there’s also interesting ideas that come up in here, like, computability. In these classes, you get exposed to some things like some of the biggest open problems in computer science.

Now, there isn’t time to go into what some of the biggest open problems in computer science are, but there’s a problem called the P = NP problem. Right. And this is a big question mark. Basically, we just don’t know if these two things, one of them named P and the other one named NP are equal to each other or not. And you’ll find out what those are in the class and you might say, “Okay, Miron, why do I care about that?” Because it turns out this little problem here, has a \$1 million prize associated with it. And it’s simple enough to explain that after having had 106a when you take these next two classes you’ll actually get exposed to this problem. It’s one of things that’s, like, a minute to understand, a lifetime to master. And no one’s mastered it yet. But in some sense, this is also a problem that’s only about 35 years old. Maybe just slightly older than 35 years old. So it’s not like this problem that’s existed for, like, hundreds of thousands of years and, you know, cave people were writing does P = NP on stone tablets. This problem actually came to the floor and people realized it was an important problem in the 70s, which means it’s possible that it’ll be solved in your lifetime, and it’s possible that you may be the one, presumably, solving it in your lifetime because it would be difficult to solve it if it wasn’t in your lifetime.

So even if programming, by itself, doesn’t necessarily turn you on, but you think, you know, programming is interesting, is there also some deeper science or some mathematics because for a lot of people, they didn’t necessarily get exposed to computer science earlier on, but they did get exposed to mathematics, this might be the kind of thing that really turns you on. Now, you might say, “Okay, Miron, math doesn’t necessarily turn me on, programming turns me on.” Besides that 106 class, what other options are there? There’s two other classes, cs107 and 108. And these classes, basically, look at building, in some sense, larger scale systems, so this involves object oriented systems and in some sense, building larger applications. So you build some things here which are outside the scope of a one or two-week project, like, you might spend four weeks on a large project in this class by the end and actually build a fairly substantial application, and 107 looks at a whole bunch of issues, that in some sense, we like to think of as lower level kinds of issues, but it involves a lot of programming and it gets into the nuts and bolts as to how does the software sit on top of the hardware of your machine and

how do these things interact and getting into understanding memory better and whole bunch of other things.

And if you think about this set of class, like 106a, b, 103a, b and 107, 108, if you were to take that set of classes and add to it 2cs electives, that's the minor. Okay. So the minor is basically these six core classes. You need to take math up to math51 I should say as a little side note. Just in case you're wondering. That's just something, that, you know, we're not responsible for that, it's just kind of required. And then two cs electives beyond this kind of stuff and then you're getting a minor. Okay. So if you want to kick it up a notch beyond a minor and think about the major – actually, I'll just leave this up. Two cs electives, you sort of add that all together and it equals the cs minor, which is kind of fun. Okay. Now, if you want to think about besides just a cs minor, potentially, actually majoring in cs, you might want to think about, "Okay, first of all, what are some other things that I can do in computer science beyond the introductory classes," and there's a whole bunch of things. There's something that we call artificial intelligence, or just AI, for short. And there's a whole bunch of aspects of artificial intelligence. That's sort of the highest level.

It's the notion of trying to make your computer work more intelligently, and in some sense, appear to be more intelligent, sort of on the order of the intelligence of a person. But really this has a whole bunch of sub fields to it, for example, robotics and various other things such as computational biology, there's a lot of computational biology that's ground in artificial intelligence. Data analysis and I'll show you some examples of these as we go along. And this is today, and there's a whole bunch of people in the world who are wondering what happens tomorrow. And if you can do slightly better than 50 percent predicting what happens tomorrow based on analyzing all the data from today and before, you make tons of money. Okay. And if you wondered is this really the case? Yeah, in fact, anyone heard of a company called D.E. Shaw? Yeah. Anyone. A few folks. Yeah. It's David Shaw. He was actually a grad student at Stanford in computer science. And this whole – I wouldn't say he started this whole thing, this actually existed long before that, but there's whole companies whose entire business is based on the notation of quantitative analysis and guess who are a bunch of people that they employ? Computer scientists who go and do the data analysis and actually figure this out. Okay.

So the application and understanding what are all the variables that you care about and the information that exists in the stock market that you can extract and model with different kinds of algorithms to make your prediction, is all part of what computer science is all about. Besides, AI, there's various other kinds of little areas. I'll show you some more pictures, like, robotics. Anyone heard of Darpa Grand Challenge or a little robot called Junior or Stanley? Yeah. Oh, Junior, he's so cute. Because it's a robot, that in some sense, is a car. Right. And there's no reason why a car can't be a robot. Just think if Carol had wheels on it, and instead of move...you had move at 60 miles an hour, you'd be doing the same thing, except you'd be doing it in a simulation. This is Stanford's car, Junior, and this is a car that's basically a robot. It doesn't have a human driver, at least most of the time. Right? It has things like various kinds of sensors on it, various sorts of radar and other kinds of laser range finding that sense what's going on in the world and

then it makes decisions. Okay. And so let me show you a little example of that. So here's a little video of Junior actually involved – the joy of software.

That's another thing you can do as a computer scientist. You can fix other people's bugs. Here's what's actually going inside Junior when it's actually running along. It's sensing a bunch of things about its environment, and you can actually see it's driving along – this is where it has some uncertainty or some distribution over where it thinks it is, where it thinks different lane markers in the road are and it's doing all this by actually taking pictures of the road, analyzing them in real time and then making various kinds of decisions about where to steer and where to go. And this is all happening in real time. Right. This isn't, like, "Oh, we had to load all this data and figure it out on some super computer," there's just a little bank of computers inside of Junior that is actually figuring this out as he goes along. It figures out certain places to stop or how it's going to re-maneuver itself. Let me show you the set of computers that are actually doing this. They're just sitting in the back of the computer. Yeah, there's a few different machines in here, but it's sort of computational power on par with what you're gonna have in your dorm room by the time you graduate basically. Let me show you one more quick example of Junior actually parking. Okay.

So these little red marks over here are actually cars and it's basically sensing that these areas are blocked and what it wants to do is get to a parking spot that's between two cars right here. So it plans this little path and it looks like it's gonna rear end this other car over here, but really all it's doing is repositioning itself so it can re-plan to be able to back up and then pull into the parking spot. Right. And if you think about all of the dynamics that need to be going on to do this, all the low level stuff to sense where things are, the high level planning to figure out how sharp of a turn it can make and now it's gonna back out and drive off. All of this stuff is basically just software. It's a computer science problem. And that's how the junior team actually views this robotic car. They view it as there's a bunch of sensors in the car and there's some actuators, like, they can hook up computers to the steering wheel to turn it and really the whole problem is solved in software. How to do the planning, when to turn the steering wheel, by how much, when to figure out if lanes are blocked, stuff like that. Okay. So that's a little bit of AI. Let me show you a few other fields. Okay. So besides AI, and there's a class related to this, cs121 or 221, you sort of have your choice.

This is kind of a survey of artificial intelligence and this is kind of, in some sense, modern techniques for artificial intelligence. If you really want to go and build robots, I sort of suggest you take 221. If you want to get a lay of the land, of what's in artificial intelligence, you can take cs121. Okay. Some other things that you take along the way are a class like cs140, Operating Systems. Right. And if you've ever wondered about things, like, "Hey, I have my Mac, how does my Mac actually do all this stuff for me, how does it take care of a file system for me, how does it take care of the fact that there's multiple things running at the same time, how does it deal with the fact that I may actually be running more applications than I actually have real ram in my computer?" There's a notion of virtual memory, for example, where it uses your disk for part of memory. That's all stuff that's covered in Operating Systems class, and if you're interested in

systems kinds of things, there's just a ton of things that you'll find here that you can kind of build on. Right? Graphics is a big area that's in cs, and it turns out, interestingly enough, of our graphics faculty, Pat Hanrahan is one of the faculty here. He actually has, not one, but two academy awards. All right. Interestingly enough, he's actually got Oscars. Right. And you might wonder, "Why does he have Oscars, Miron?"

Well, because guess what, there's all these animated movies these days, there's a system called Render Man that was actually responsible for being able to do a lot of the rendering for original computer graphic movies. He was on the team that built that system. And he's done a bunch of other stuff since then, which is why they gave him a second one in 2004. Okay. There's a guy named Ron Fedkiw, and I'll show you a little animation that his group developed. So here's what looks like a lighthouse and water, and here is basically, a realistic computer animated waveform crashing over the lighthouse. Right. This was all done. This wasn't like scanned over some real lighthouse when there was flooding. This is all basically done as a computer simulation. All right. That's the kind of stuff his group does. And as a matter of fact, for doing stuff like this, it doesn't just show up in little animations to show in 106a, if you happen to see Star Wars 3, he was in the credits for it, if you happen to see – what were some of the other movies he was in – anyone see Terminator 3? Horrible movie. Don't see it. But he was in the screen credits for that as well. Evan Almighty, yeah, so there's serious movies that involve major computer graphics where the stuff that's being done here is actually at the cutting edge of that to be able to figure out new ways of actually doing things with computer graphics and actually doing the animation. But there's other things you can do. Like, here's a mis-focused camera, you just bring the picture into focus automatically. Here's a really blurry one. Awe – pretty hardcore. And here's focusing through a splash of water. So it doesn't just have to be a picture of some solid object. I hope you can actually see that re-focusing while it's happening.

Then we get into the audio part. I won't share the audio part. It's kind of more of the same. But that's the basic idea. They're actually starting a company around this idea of light field photography where you have a camera and just the way the lenses are constructed and the amount of light that you sample at various kinds depths of fields allows you to take this image and then be able to refocus on different parts of it later or clean things up or whatever. That's just another thing that's kind of based on graphics that you wouldn't necessarily think of right, but photography really is taking some sample of the world, turning it into a graphical image and then doing manipulations on that image.

So a lot of the things that happen in graphics, apply directly to photography as well. Okay. So besides graphics and robotics, we talked a little bit about those. There's folks that worry about stuff like databases, like, handling large volumes of data on streaming data, on different kinds of things you could do with data and I was kind of thinking about this and I was, like, what's a demo I could show having to do with large volumes of data because that's not something you can actually draw a picture of real easily. And then I just thought I'd show you this. Because Google came out of Stanford. It came out of a group of folks who did things like understanding data structures and the algorithms

associated with them and who understand how to keep track of large volumes of data and be able to do manipulations on that kind of data. And in the early, early days, most people don't know this now, but if you went to google.stanford.edu was the web address for Google. Okay. And it turned out at some point this was actually eating up so much of the entire bandwidth on campus that some folks said, "You really need to go and move this somewhere else," and then they actually created the company Google, which is based on a misspelling.

Right. The actual – does anyone know what a Google is, which is the correct spelling of Google, is ten to the hundredth power, it's 10 with a hundred zero's after it. And so Larry Page and Sergey Brin were grad students here and they wanted to think of same name that captured the largeness of Google or of the web search that they were doing so they went off and registered Google because that's how they thought it was spelled, or at least one of them, and I won't tell you which one thought that. When they were grad students, and then when the other one of them came back to the room and looked at it he said, "You misspelled it," but two things transpired. One was that this .com was already taken, and the second one was when you're a grad student and at the time it was, like, \$50 or \$70 to actually register the name, that's kind of spendy when you're living on Ramen. So that's what it was. Okay. But it just shows you the kinds of things that get done by taking basic ideas in computer science and building them to a larger scale. Other things that go on. I'll just give you a brief sampling.

Cryptography, which is big for web security. Right? It turns out a lot of the web is actually pretty insecure. Much more less secure than you would actually imagine. Anyone ever had a credit card number stolen? A few folks. Yeah. When you get your credit number stolen, then you think twice about a lot of the transactions you make. I had it happen, actually, a couple times and I still, like, you know, Christmas time rolls around, I'm just like online shopping until the cows come home. But it's important to actually think about what's secure and what's not secure. And there's actually a group that deals with cryptography, especially security in the context of the web.

Other kinds of things that go on. We talked about AI, and sort of a sub field of AI, which is growing into a whole area of its own, is machine learning, and I talked a little bit about things like biology or predictive data analysis. There's actually also machine learning that affects your life on a daily basis, whether or not you know it. How many people have a spam filter on their email? Anyone? Yeah, did you know that chances are probably in all likelihood that your spam filter is actually based on machine learning? It's seen a whole bunch of email, some mail that was spam, some mail that wasn't spam. And it learned, no one told it what was spam and what wasn't spam. It learned to figure out how to distinguish between what's spam and what's not spam. Now, it's not perfect. Right? People aren't perfect either, so sometimes you get messages in your inbox that are spam, and every once in a while, rarely, but it happens, someone sends you a message and you never hear about it, and they're like hey, I sent you this email and the you go check your spam folder and it shows up in there. But spam filtering is another one of these things that in the last oh, ten years or so, is another something we take for granted and don't think

about the fact there's actually a bunch of science under the hood as to how to do this and people continue to do research how to improve it.

And there's a bunch of other things based on this, like, robot and navigation. Some of the stuff you just saw with Junior, is actually based on learning landmarks of the road or learning where lanes are on the road or what obstacles actually are. There's a ton of other things. I'm just giving you a sampling. Now, if any of this has interested you at all, there's a guy you need to go see. One guy you could go see is me, and I'd be happy to talk to you about any of this, but there's another guy you can go see whose name is Dave Koslow. And Dave, is what we refer to as the CS course advisor. I'll just put the CS advisor up here. He's in G160. He's the guy you see when you want to declare a computer science minor or major. Not that I'd be putting in a plug, but he's an interesting guy to talk to about some of the different possibilities in the field, but open invitation. So this class is gonna end, like, after Wednesday or after the end of the week or after you take the final depending on how you look at it. Don't be a stranger. Right. Come on by. If you want to talk computer science, if you want to talk about what's possible to do in the field, come by. My office hours will be on the web or send me an email to set up a time to talk, and I'd be more than happy to take you through a bunch of this stuff. So besides, Dave, there's me.

Now, last but not least, and I shouldn't say, last but not least, you might say, Miron, computer science is kind of interesting, but are there other related majors that I should consider. So in the sense of full disclosure on fair play, there's computer science, there's some other possibilities. There's electrical engineering if you're more interested in the hardware side of things. There's math and computational science. And math and computational science is more if you're interested in the mathematical side of computing. You'll still get a lot of math if you do computer science major, but if you sort of are really kind of immersed on the mathematical side, math computational science is something to consider. And there's also a major called symbolic systems or just sym sis. And sym sis is also a fun major. It's actually a combination of linguistics, computer science, philosophy and psychology. I always forget the last one. Except, it's always different every time which one I forget. And the basic idea here is to think of both humans and machines as symbol processors, right? People are symbol processors, in some sense, because they take in symbols of the world, namely, language or visual [inaudible] that they actually see and they make some sense of it, and then they act in the world.

But now you might say, okay, that's interesting. You've told me about all these fields, but you told me that computer science was more than just programming and so far, it's unclear what I might be doing other than programming all these cool application you're showing me. So let me tell you about a few of them. This is the one I refer to as kind of the peanut butter cup version of computer science, which is you can take computer science and a whole bunch of other things and mix them together and they're just two great tastes that taste great together, and I'll show a lot of examples of that. So there's CS and business. Okay. If you're interested in sort of the business side of thing, product management is a whole field or a whole area that people go into, especially in high tech

product management which are people who don't necessarily program, but they have technical backgrounds to be able to define what products are going to do and how people are gonna interact with them. So if you look at a lot of high tech companies, people who are product managers, who are taking more of a managerial role and defining a role for product, many of them, in some sense, I'd actually say most of them, probably have a technical background. In a lot of cases, it's computer science even though they do know programming. They do product definition.

Beyond that, and this is kind of a popular one around here. Entrepreneurship. Yeah. That's good enough. I always get nervous writing that. And that's the whole notion of you think about people who are doing startup companies. There's been a ton of startup companies. I can't name them all because over the last few years, there's been over 2,500 companies that have come out of Stanford. Some of them are big and you know about, like, Google and Yahoo and Cisco and Sun and HP and all these other ones, and there's a whole bunch of smaller ones out there that also did pretty well. Anyone ever remember Evite? Anyone ever send an Evite? Yeah, that was started by a guy I lived next door to many years ago. And they did pretty well. It got acquired eventually, but life was all good. And the whole notion here of thinking about startup companies – now, one thing that's interesting is a lot of people think, "Oh, well, if I want to do entrepreneurship, I should go do business, right?" Well, what I'd actually challenge you to do, if you think that, is go find out about the backgrounds of people who are things like successful venture capitalists and see what they did when they started.

And one of the things that you'll actually find, which is surprising, is most of these people didn't start as business people, they started as technical people who actually went and did interesting technical work and at a certain point, realized there was a need and then moved on into the business realm. Tons of examples of that. I'll just give you a quick one. Eric Schmidt, who happens to be the CEO of Google, PhD in computer science. Right, now an MBA. And that's not to say an MBA is a bad route. It's just to say that, realistically, if you look at what a lot of people have done, the route to actually getting there, in many cases, actually, flows through a technical area. Okay. There's also finance, in the sense of computational finance. All right. Again, not only in predicting the stock market, but there's a whole bunch of people that what they do is they worry about different kinds of modeling algorithms or managing different kinds of funds, basically, by thinking of financial markets as a computational problem that they model with different kinds of data structures and different sorts of algorithms to potentially make predictions on or just to get insight into. If you're interested in this kind of stuff, there's actually a program called the Mayfield Fellow's Program. If you do a search for Mayfield Fellow's Stanford, in your favorite search engine, you can find out more about it, which is actually a program that you learn about entrepreneurship. You go into an internship with a startup company to learn more about it, but you actually get immersed in thinking about the different issues of starting a company. We'll just leave the CS up here.

And biology. This has become a hugely popular area these days. Okay. So there's a whole bunch of things like bioinformatics, and bioinformatics – there's kind of different flavors of CS and biology, is thinking about the information systems that keep track of

biological data, or they keep track of medical data. Right. So if you think about if there's a whole bunch of medical data that's being kept on, like, your medical records and results for tests and a whole bunch of things that I want to be able to slice and dice in different ways, or understand how, for example, symptoms that you have might be related to some other symptoms or some other diagnosis that happened in the past. These are the kinds of information systems that deal with that, and we have a whole program here called the BMI program, Biomedical Informatics that just deals with that. But beyond that, there's also fun things, like, genomics and proteomics, and doing things like being able to look at gene expression data and DNA and be able to determine what kind of diseases do you hereditarily have more of a disposition to because of your genetic makeup. And if you're interested in this kind of stuff, there's actually a program also on bioengineering. They don't, right now, have an undergraduate program. They have a graduate program. They're gonna form an undergraduate program. That's something you could be interested in, or it's something we actually have sort of a sub areas of the computer science major that you can also do this sort of stuff in.

Now, one thing that's kind of interesting, which also sometimes surprises people is they say, "Oh, I want to be a patent lawyer. I want to go and deal with all these issues like making sure that file copying of music is legal for everyone, so I'm gonna go and be a political science major cause that's what I should do to go to law school." Right. Turns out that if you actually want to be an intellectual property lawyer, you need to have a technical background. There's a list of approved areas that you could've done for your undergraduate degree that allow you to become an intellectual property or copyright lawyer. Computer science got added to that list about 15 years ago. Political science, not on that list. Okay. So it's something you should probably know now. If this is the area that you're thinking about going to, you need to understand the technology to understand how intellectual property and copyright issues apply. You need to understand what an algorithm is. What parts of an algorithm are obvious versus what parts of an algorithm are not obvious? That's what allows people to do this work. Okay.

And then last, but not least, CS plus CS. So you can just do – you don't have to mix computer science with something else. You can just do computer science, and obviously, programming is part of this. There's a lot of people who are very happy being software engineers and there's lots of jobs in software engineering and life is good. But there's also people who go into engineering management. Most managers, in computer science, are not professional managers. They are people who at one time were programmers or engineers, and worked their way up through the ranks and eventually became managers and became senior managers and became VPs and the whole deal. Right. So it started by having a technical background. It didn't start by saying, "Hey, I want to be a manager," and having someone hire you to be a manager. Okay. And there's also, and this is near and dear to my heart, so I'm just gonna sort of wrap up quickly, teaching. Right. So you could think about computer science as a field that you go into because you want to teach it to other people, in addition, to perhaps doing some stuff in it yourself because you find it interesting, but if teaching at all is something that's interesting to you or, like, when you were in your section, you were, like, "Hey, section leading is kind of cool, this is

something I might consider," there's the cs198 program. And this is a program that I've talked about in the past, but I just want to spend a little bit more time talking about.

And what you need to go into cs198 is you need cs106a and b, or you could've taken X, but at this point, it's kind of too late to take X. So what you really need is cs106b, after one more class, you're eligible to become a section leader. And being a section leader, you might say, "Oh, well, what does that involve?" And it turns out to involve a whole bunch of things. One is that you actually teach a section, which is kind of cool in itself because you get to learn the material a whole lot better when you teach it to someone else. There's always some new little nuance about something that you learn somewhere. So you learn the material much better by teaching the section. You also get to know other section leaders. So there's kind of a social aspect to it, and especially if you want to go into computer science, this is a great way to meet project partners and other people who you know are really interested in computer science and motivated. You also get to meet faculty. So when it comes time for getting letters of recommendations, which is something that people don't necessarily think about early on in their program, but then later on how many people are thinking about letters of recommendations now, like, it's getting to that grad school application time, and how many people wish you had thought about it earlier. Yeah. Mostly the same hands.

This is a good way to do that. Is to actually get to know people who are involved in teaching and we have regular staff meetings, and it's a good way to sort of think about that. And at the same time, and one last side point I would put in, is that there's a huge network of people who went through this program who are out there. So the cs198 program is actually a program that's not just known at Stanford, but it's actually known nationally. Like, if I go to other companies or something like that, there's people that come to visit, for example, from Microsoft and they're, like, "Yeah, tell me where the 198 meeting is, and what's going on there," and they come and recruit from that group of people, and this happens for a whole bunch of companies across the board. So, with that kind of said, hopefully this has given you a little bit of a taste for what computer science can be about. Not just thinking about programming per se, which is what we've done a lot of in this class, but programming is really just the first step that opens up a whole bunch of other venues. And hopefully you got a sense of some of the other classes you can take that will broaden your horizons even more and some of the different areas you can go into potentially with a computer science or related major that can open up all these possibilities. Any questions about any of that? You're all set? All right. Then I will see you on Wednesday.

[End of Audio]

Duration: 47 minutes

## Programming Methodology-Lecture28

**Instructor (Mehran Sahami):** All righty. So welcome back to our last fun-filled, exciting class of CS106A. It's always a little sad at the end of the quarter, at least for me. For you, it might be a lot more exciting that you're done with this class. But so we have a bunch of announcements. There are some things to get through today, namely some things also related to your final exam, as well as some unfinished business with the graphics contest.

So there's two handouts today, the last two handouts except for the final exam which are a practice final and the solutions for the practice final. The problems that are on that are actually taken from – most of them. Some of them were written just for that, but most of them were taken from actual final exams in the past. So just like the midterm, it should give you a chance to get a notion of what kind of questions we would ask, what sort of topical coverage there would be, the level of difficulty, and I've left the blank pages out of the exam just to save trees, but we would have space in the exam for you to actually take in.

And we give you the solutions so you can check your exams as well. And again, I would strongly encourage you to do this as a way of studying for the final exam, just to get a sense for how comfortable you feel with the material because there is kind of a range of stuff. As I mentioned today, today is the last class lecture. There is no – to sort of pay homage to what's supposed to be dead week, although most classes don't really have a dead week, our attempt at dead week is a dead day, so there is no lecture on Friday, although sections are still happening this week, so go to your section this week. There are section problems, and there is a bit of a review happening as well, so sections are still very useful to go to this week. Assignment No. 7 you know is due next time. You only need electronic submission for that because we're not gonna do interactive grading, so there's no interactive grading for Assignment 7, but it is due at what would be class time on Friday. You just need to submit electronically. Don't worry about paper version. Just wondering. Anyone – how many more people – or I should say how many people are done with Assignment No. 7 already? Just wondering. Wow. A fair number of folks, so hopefully it wasn't too excruciatingly painful.

Course evaluations, there's actually two sets of evaluations, and I would ask you to please do them. They're actually really important. The two course evaluations there are is one is a university-wide course evaluation. And actually, if you do your course evals for the class that you're in, you get access to your grades earlier in Axess, so sort of provides a strong incentive to actually do the course evaluations. But I would encourage you to do them, and be honest in the course evaluation for whatever you think because that's information that we can also use to try to improve the class in the future, so it's useful for us in the long term, and the university actually uses it for a bunch of stuff for it, so it's very important for them. The university, since they have their own eval – we have a specialized eval that we also do. Sort of like you took the mid-quarter eval that was online, we have an end of quarter evaluation for 106A that also gives us some very specific feedback about this class: how much time you spent on assignments, what

assignments you liked, what assignments you didn't like, stuff like that – that we actually – It's totally anonymous, but we use it for some data analysis to kinda figure out how are we pacing the class, is the class actually achieving the objectives that we're trying to get to, and you'll find out about that second evaluation in your sections this week. And that one's also very important, so I would encourage you to go to section, find out about that evaluation, and take it. But the other one's the university-wide one.

A couple other announcements – the final exam as I mentioned before, but I will mention it again – it is an open book, open note exam. Bring printouts of your other programs if you want to. You can bring the text book for the class. You can bring your Karel reader if you want, although I can tell you right now that Karel will not be on the final exam. And I'll give you a list of topics as to what's actually fair game for the final, and what kind of stuff is covered in the book that's not gonna be on the final just as part of our final review. But you can bring the Karel course reader if it just makes you feel better to look at Karel. You're not gonna need to worry – or maybe there's some algorithm in there that just reminds you of how to write Java. You're welcome to bring it, but you're not actually gonna need to know Karel for the final. It is closed computer, though, just like the midterm exam, so if you have a laptop, don't bring it along, or if you bring it along, don't turn it on.

The regular exam is Thursday of finals week, December 13th. That's just over a week from today, 12:15 to 3:15 p.m. in Kresge auditorium, and these dates, and times, and places are all given to you on the front of the practice final anyway, so you have them as a reference. And the alternate final is the day before, Wednesday, December 12th, 3:30 to 6:00 p.m. in Kresge auditorium. You are welcome to take either exam. You don't need to e-mail me. You don't need to make prior arrangements. All you need to do actually in terms of making prior arrangements, you need to figure out which exam you wanna show up for, so the only prior arrangements you need to make are with yourself. Pick which exam you wanna go to, and go to it, and it's all good. It just provides you with a little bit of flexibility if you wanna go home a little bit earlier or if you had a conflicting class because I know some people are watching the class on TV. SBD students, I'll say it one more time, and it's the last time I'll say it because it's our last class. E-mail me by 5:00 p.m. today if you plan on taking the exam at your site. Hopefully, I've already heard from all of you that this applies to, but if not, you have your last chance for like another hour after class today.

And last but not least, anyone find a cell phone in here? It's not mine, but there are someone who's in this class who unfortunately has lost their cell phone, so if you do happen to find a cell phone in this class, just come and bring it up to me after class, and I will make arrangements with the student to get it back to them. So any questions about any of the administrative kind of stuff before we sort of delve into some unfinished business? So we have a little bit of unfinished business in this class, and the unfinished business is the graphics contest – well, besides our final review. And I've gotta say that the graphics contest – last time, I told you they were just impressive. I think they were more than just impressive. Some of them were just jaw dropping, so I'm gonna show you actually, and announce the winners of the graphics contest, as well as show you the

winning entries. And we'll also do the random drawing to see of all the people who entered who gets the bonus prize.

So our first winner, and if we have a little fanfare, I can kinda slide this to the side. Actually, I'll slide both these to the side. So we have two categories. We have algorithmic and aesthetic. In the algorithms category – algorithms, I can't even spell these days anymore. Is David Tobin here? David, come on down. So good job. As part of winning, there's a couple things you get. One is you get a bag of candy, a whole bag, well, not the whole thing. You get one of these. You can pick which one you want, though, so rummage through. A Milky Way fan?

**Student:** [Inaudible] Milky Way.

**Instructor (Mehran Sahami):** All right, that's always good. And we'll show you – we'll demo your program and show what's actually going on, so the program that David did was a program to do fractals. And so I will show you that program, and what is particularly cool about the program. So if we do fractal graphics, it kinda starts off, and it shows you a fractal.

And you're like oh, that's kinda cool. You need to do some math. You need to figure out some stuff with fractals. But as many of you know, fractals you can kind of zoom into at any range, so we can pick a range over here, and it will automatically zoom in and expand out, and we can just keep zooming into the fractal if we wanna keep going further. Now besides just that, that's kinda cool in itself, there's also some built-in fractals. So there's a few that David has lovingly provided for us that we can kind of look at, and that's kind of cool in itself. Now the thing that really made me go, "Wow, that is really cool," is not only are there these built-in fractals, there's a little text box down here. And in that text box, you can type in an equation for a fractal that includes exponents, and you type in the equation, it actually figures out – it parses your equation, runs the math on it, and then generates the fractal.

And you can do this – it's sort of freeform. I was kinda playing with this other day, minus X raised to the fourth, so this X plus X squared minus X to the fourth down here, and it takes a little while because it's rendering every pixel on the screen how it should look. And so you can get these really cool effects, and then we can always zoom down. So it takes a little while to redraw, but it goes ahead and redraws it for you. So not only is it doing all the graphics, but it also is providing the opportunity for you to just type in an equation for it to understand the equation and render it, so very nice work. Thanks very much. And so as you know, for that effort, you also get 100 percent on anything in the class, which can include the final exam, which means at this point you can take your practice final and be like recycle bin if you choose to take the hundred on the final, which means you just don't show up, and when I see that little glaring gap I know oh yeah, that was 100. That doesn't apply for everyone else, okay. Also, in the aesthetics category, is Sally Hudson here? Come on down. It takes a little while. You also get your choice of the random candy. And you might wonder why there's random candy because well, I just buy a lot of candy at Safeway. Kit Kat fan? Good times. And what Sally did for her

program – this is also pretty – well, a lot of the programs actually have very good algorithmic and aesthetic qualities, so it's hard to just pick one for the other, but it's a program called the Tessellator.

And I'll show you what the Tessellator does. It starts with a little square. Anyone a fan of M. C. Escher? A tessellation is just where you put a bunch of little things on the screen, but they all have to fit together. So you can click on any part of this square, and to guarantee that it'll tessellate, the part of it that should allow for the tessellation to happen, basically sort of the corresponding piece on the other side, also kind of expands out. We don't want the tessellation to overwrite itself, so we can just kind of pick random areas, and that creates little places – or we can go this way – that create the tessellation. You might say, "That's kinda cool," but now we click set tile to set what the tile is. We've now defined what the tessellation tile is. We can change its size. So let's say I make it a little bit smaller because I like small tessellations, and that's kinda cool. But now I can pick multiple tiles of multiple colors, so let's say I want three different colors. Let's see. I have little my red, blue, green sliders over here, and notice the color changes as I slide, so I slide over here, so that's kind of a puce. I don't know. Well, we'll set the color. So it indicates your color over there. There's color No. 1. Let's just go – as the big Stanford Cardinal fans, we'll just go for red, although cardinal is slightly off from red, and I don't know where it's actually, so maybe it's got a little blue in it? Maybe a little green, too? Let's just say that's cardinal, although you would look at that and say, "But Mehran, that's really salmon." That's cool, too.

And then we'll just pick another color where we'll go, "We won't go white." That's where you have all the – you could even just teach red, green, and blue with these three sliders by themselves. What happens if we put red and blue together? Oh, that's too close to the other one. Let's just go for some blue, so we set the color. Now at this point we go tessellate. And you'll notice because of the way the piece is drawn, you can tessellate everything together of the multiple colors. And you might think that's kind of cool, and that's kind of cool in itself, but it's kinda even cooler if the tessellating pieces decide to pop out and bounce around, multiple of them at the same time. So very nice piece of work. Thank you very much. And this keeps going until they're all done, and you can actually just reset and do the whole thing all over again, and that's pretty cool. Now both of those programs are pretty cool, and you both get hundreds for your work. There was one program that was entered that when I showed it to the – well, first when I saw it, I went, "Oh my god," because it made you weep in a really good way. And then I showed it to the section leaders. Ben and I were demo'ing these because the section leaders all picked the winning entries, and they saw it. And as we went more and more into the different pieces, every time we showed a different piece, everyone just went, "Ah," and it was just unbelievable. So Chris Miel, are you here? Come on down. This is an additional award we're giving called the peoples' choice award because the entries were so good that we figured why stop at two. Because when it comes down to it, in this class there is some level of stuff that I want you to know, right? And if you demonstrate that you know that stuff, we can have some more awards. It's not a problem. So Chris, first of all you get to pick a bag of candy.

And now I will show you Chris' program which is Zelda. All right? So it begins with this, and you're like, "Oh, that's very unassuming." So you start with the beginning graphics screen. If you look at this one thing, you kind of look at this for a while, and then you realize there's a little flash of light that goes across. Someone had to animate that, and they had to think about it. And so you can load a game, or you can start a new game, so I'll show you a new game. It has some nice graphics in there where again if you actually read everything – anyone in here Zelda fans? Yeah, I was when I was much younger. It's actually a very funny sort of take on the Zelda concept. Notice, if I turn different ways he's – oh, getting a little feedback there. There's a shield that I can use in different directions, and there are different screens I can go into, like I can go into this house over here, and here I can buy stuff. Like I've – I'm a little bit here so my hearts are down here, so I can say, "Hey, I wanna buy a couple more hearts," and I'll buy a potion, too. And here's my funds over here. Here's my magic potions and my emeralds that I buy stuff with. And I'll exit.

And then I just kinda cruise along, and there's stuff that goes along in the world including things that like fade. Notice the fade effect there. And as you continue to cruise along in the world, you see various kinds of things that maybe we wanna kill, and just kind of cruise along. And you get to places where there's just animation going on everywhere, like the water's animated. I'm getting beat by the octopus. Never let yourself get beat by an octopus. And you can't do things like run into the river. It actually is aware of what – aw, I died. That laugh goes on a little too long. There's a whole world you can explore. You can save the games and come back. I should be using my shield, but I'm kinda ignoramus with the shield. Oh, yeah. That's right, buddy. Sometimes when you kill something, it turns into an object. Sometimes it doesn't. And it was like every time you enter a different screen, it was like this. It was just astounding. So – oh man, I haven't even seen this screen – [inaudible]. I know what I'm gonna be doing over winter break, so I will save you all the rest of the time.

We'll just go in here, so you can see some stuff that's going on. Music changes. We probably can't actually show this on the video because I think are actually sampled from the real game, perhaps? Yeah, that's live from the city. But thank you. That's just an astounding piece of work. So sort of what I refer to as the peoples' choice award goes to Chris Miel. Am I pronouncing that right? Is that Miel?

**Student:** Yeah, it's Miel.

**Instructor (Mehran Sahami):** Miel. All right. Good times. The other thing that came up is these, you get 100 percent on anything which is nontrivial, right? You're getting 100 percent on the final, and I looked at this, and I said, "I could give Chris 100 on the final, but I think he's pretty much demonstrated anything that I could possibly have wanted him to know in this class, so I'll just give him an A plus." So you're done. That's your grade. [Inaudible]. Thanks. And if any of the contest winners if at any point in your career you should need any recommendations, let me know. I'll be happy to write it. Now there were all sorts of honorable mentions. So these are all the winners of the contest. There were some honorable mentions, and I wanna recognize a couple of the honorable mentions that

really stood out. Unfortunately, we don't have time to demo them all. There were a lot of cool programs, but a couple honorable mentions as well. Jasmine Mann? Yeah, you wanna come on down? And also Paris Georgegoudis? Am I pronouncing that right? Are you here? Oh, just too good to come to class, but we'll sort of give it to him anyway, Paris G. So Jasmine, you are also entitled to candy.

**Student:** Yay.

**Instructor (Mehran Sahami):** And so one of the things you also get, as well as Paris who's not here, just for the very strong effort that you put in, is you know there's a participation grade in the class. So you automatically get 100 percent on your participation grade. So thanks very much for a nice piece of work. Jasmine did a musical piano that displayed some stars and had music, and Paris actually did a backgammon program with a computer player as well. So now there's finally a little more unfinished business to do which is that besides all the winners, there is also a random drawing for everyone who put a serious entry into the contest, and both Jasmine and Paris are both eligible for that potential 100 on the final. If you get the 100 on the final, though, you don't get the 100 on the participation because you only get one prize. It's your choice because if you happen to win, you get that 100 on whatever you want, so you can still use it for participation. But in terms of everyone else who's in the contest besides the winners, there was 14 total people who were in the contest. So I figured why don't we just pick the winner collectively in a random drawing, and so I wrote a little program, which just pales in comparison to Zelda. Actually, I even took this program from someone and modified it. That's how low budget this effort was because it was just a cool program. It was like, "Oh, that's cool. Let me just use that rather than rewriting it all myself."

So what this does is it takes each of the entrants of the graphics contest and puts their name on a card. And those cards are shuffled and displayed on the screen. So what we're gonna do is pick one of those cards, and then this will gradually go through and reveal all the other cards who are the non-winners unfortunately. So you can sort of – I know it's kind of brutal, but that way you've got a sense if you're still kind of in the running as we go along. So now, I need to pick a card. These are all random, so it doesn't really matter which one I pick, although afterwards it'll be like why didn't you pick the one to the left. Oh, let's just pick this one. This one good?

**Student:** Yeah. [Inaudible].

**Instructor (Mehran Sahami):** Yeah. All right. See? Here's everyone else. They get revealed slowly. I don't know if you can hear the sound effect of the cards flipping over. Sorry, Jasmine. You don't win again, but you still got a prize up here. Yeah, some of the cards once they're all covered up – well, Paris doesn't win again.

And the winning entrant is – [inaudible] actually did a very nice painting program. I don't remember all the entrants. And you have your choice of the Three Musketeers or the [inaudible] bar. No one likes the Three Musketeers. I should write that down. I'll

remember it. Send me e-mail – also gets 100. Very nice work on the graphics contest. Thanks very much. And now with that bit of business taken care of, we can move on to reviewing for the final exam. So a little bit of final review, what you need to know, what you don't need to know, and maybe a couple examples of the level of difficulty we expect.

So in terms of what's actually on the final, this is kinda what's on. And at a highest level, what's on the final is Chapters 2 through 13 of the book. Okay? So Karel is not on there, and Chapter 1 on history of Java not on there either. Now 2 through 13 covers a bunch of stuff, so in relation to that, let me start off by telling you what's not on the final so you get a sense of what are some of the things you don't have to spend your time on. Then I'll spend some time emphasizing some of the things you do wanna worry about studying. So history, the Chapter 1 stuff as I just mentioned, not on the final. Karel the robot not on the final. He's fun, but we're sorta done with Karel at this point. Zelda might be on the final. No, just kidding. Something people have been wondering about, stack heap diagrams not on the final. So stack heap, the diagrams are not on the final, but you might still be expected to know what is the difference between a stack and a heap, what gets allocated on the heap versus what gets allocated on the stack, but in terms of actually drawing the diagram with the memory addresses, that's not something that we'll ask you a question on.

In terms of the chapter that has to do with images and graphics, there's a whole bunch of stuff if you read the whole chapter about bit operations on images. Bit operations not on the final, so if you sort of skipped that question, we didn't talk about it much in class except where we did a simple example where we took a picture that was color and turned it into black and white. Remember that? On Halloween, no less. I remember the lecture. Bit operations not on the final exam. You don't need to worry about that. Something we sort of stressed the whole time in the class which you didn't need to worry about was polar coordinates, so polar coordinates it's not like hey, we're gonna whip them out in the final exam, now you need to know them. You didn't have to worry about them before. You don't need to worry about polar coordinates with respect to the graphics, right? And that's the only place they actually get used. Somewhere else in your life if you go exploring the poles or something, you might care about polar coordinates, but not in this class.

So other things you don't need to worry about: layouts. We talked a bit about layouts when we talked about things like the border layout, and the table layout, and all this kind of stuff. And I showed you could lay things out where you have a program that has a text area for the console, and a canvas, and all that. You just don't need to worry about layouts at all. That's not something we'll ask about.

Sorting – so Ben gave a nice lecture on sorting. Sorting's kinda useful. These days, sorting is mostly a solved problem. We're not gonna ask you any sorting questions on the final in the sense of you don't need to know any of the special sorting algorithms. We may ask you to keep something sorted, but if you think about keeping something sorted, that's different from sorting it from scratch. So we may or may not ask you to do this, but

in terms of being able to sort something from scratch, like knowing selection sort or insertion sort, those algorithms you don't actually need to worry about them. The stuff we did on advanced topics where we talked about threads, you don't need to know. It was an advanced topic. It was for your own edification. Some of you used it in graphics projects, which was – or graphics contest entries, which was nice to see, but you don't need to know it. We're not gonna do any kind of testing on threads. Okay? And after threads, there was also a lecture on standard Java where we talked about the main method, and generating jar files, and all this nuts and bolts sort of stuff. That's not something we're gonna test you on either in the sense of standard – we'll still test you on Java. That doesn't mean you don't need to know any of Java, but just that specific stuff we talked about in relation to standard Java versus the ACM libraries, that's stuff you don't need to know about either. So what do you need to emphasize when you actually do your studying? Here are the topics, so if you pop all these topics out of Chapters 2 through 13, the rest of the chapters are pretty much fair game, and here are some of the topics that generally come up which you wanna make sure you're good with.

So the most basic thing is you need to be able to write programs, so you need to be able to build a program. That means knowing all the basic syntax that we talked about, what a class is, extending console program, if while loops, for loops, all that kind of stuff, you're just gonna need to know that. It's not like the exam will not have while loops in it, like do the exam without while loops. It's not gonna be like that. You should know also about objects, classes, and interfaces. What does it mean to implement an interface? What does it mean to define an object? What's a constructor for an object? What goes in a constructor for an object? Stuff like that. So you should be able to – if somewhere on the exam I asked you to define a class that does something, or perhaps define a class that implements a particular interface and show you what that interface is, you should be able to do that. Note on your sample – on your practice final, there actually is exactly a question of this form where you write a class that implements an interface so you get a sense for what that actually is.

You should know about the mechanics of method calls. And part of the mechanics of method calls is what kind of things get passed by reference, meaning they get changed once you actually pass them, what kinds of things get passed by value, which means you get a copy and they don't change, and how those interact with what's actually going on when you make a method call. And so related to that is the notion of primitives versus objects. What are the primitive types? What does it mean for something to be an object? The general generic type object that all things that are objects are a subclass of the type object, so if you wanna write something that is entirely generic, you could write it in terms of being able to keep track of an object for example. And you did this to some extent when you did for example breakout. There instead of an object you had a GObject that you kept track of. What was the thing on the screen for example that you collided with? You wanted to see if that was a brick, or the paddle, or whatever? Hopefully, you did that for breakout but that's kind of the same idea there.

So objects, classes, all that happy stuff – this is now in the what is in the final rather than not on the final, so I'll erase all the not on the final stuff. Other things that are also on the

final, strings and characters. Know your strings and characters. Especially in relation to strings and characters, know your string operations. This is just a good thing either to have some tab say in your book to where the list of string operations is, or to have it on a separate piece of paper, or to print out for example lecture notes that are on the class website that contains a list of the standard string operations. It's just good to have them as a reference because you will invariably be doing at least one problem perhaps more that involves some kind of string operation manipulation, and unless you have them all memorized, if you have some handy reference, that's probably a good thing to have. And related to string operations is also – and dealing with characters, so the fact that you can pull individual characters out of a string, you can concatenate a series of characters together to build up a string which is a common way of actually building up a resulting string. That's also kind of related to that, so you need to know the interplay between characters and strings. Graphics, yeah, there's probably gonna be at least one question on there that involves graphics, and that's in terms of the ACM library, so all your happy friends like GRect, and GOval, and GCompound, and all that kind of stuff could potentially show up on the final exam. Again, this is probably somewhere in your book where you have all the different methods listed out for the ACM graphics library, it might be worth putting a little post-it or something in there so you can flip to it quickly if you need to.

And related to graphics is event-driven programs, and by event-driven programs – we've looked at two kinds of event-driven programs. One is event-driven programs having to do with mouse inputs, so what we refer to as a mouse listener, like was the mouse moved, was the mouse clicked, all that happy stuff that you did for example in breakout. And then the stuff that you've been doing more recently with name surfer and also with face pamphlet which are action listeners which are various kinds of buttons, or text fields, or whatever the case may be. So you might have some program that actually for example uses both of these things that may take some mouse input or whatever and does some modification on graphics as a result. Uh huh?

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Ah, excellent. I was just about to say that, and you raised your hand first. No, you don't need to worry about key listeners. So anything we give you will be in terms of mouse listeners and action listeners. You don't need to worry about key listeners.

So other kinds of stuff – then we got into the world of data structures, and we had arrays and array lists. These are also fair game for the final, so some things you should know is the differences for example between arrays and array lists. When is appropriate to use one of them versus the other, so the use of them, or use them. They're your friends. In the sense that if I give you a problem that involves – I give you kinda the setup for that problem, you should be able to choose is that appropriate for an array or an array list. I might tell you which one you should use, but if I don't, you should be able to determine what's appropriate.

In terms of arrays, you also do need to worry about multidimensional arrays. You did some of this presumably already on the Yahtzee program, but multidimensional arrays – at least within reason. I won't give you anything like a 16-dimensional array, but probably two dimensions is fair game. Three is the upper limit. I wouldn't give you anything more than three dimensions. Likely if there's something with multidimensional arrays, it'll probably be two at most. Maybe it'll just be single dimension arrays. And also with array lists, everything that we've done in this class, which is also everything you need to worry about for the final, is sort of the Java 5.0 or later version of array lists. So in the book there's always sort of two versions. There's the here's how you use array lists before Java 5.0 and here's how you use array lists after Java 5.0, or 5.0 and later. You don't need to worry about the before 5.0 stuff. Just the 5.0 and later is all you're gonna need because that's all we've emphasized in the class, and that's these days what everyone's using anyway.

And then after arrays and array lists, which was kind of the introduction to the whole notion of Java collections, was thinking about collections. So besides just array lists you also saw hashmap, which hopefully at this point you are just the master of the hashmap, but you should know the hashmap. And things that come along with hashmaps and collections in general, or the collection interface – I should say collection instead of collections – is for example an iterator. What is an iterator? How do you get an iterator from a collection? How do you use an iterator? That's something that you just wanna be facile with because chances are for some problem to solve the problem, especially if you're doing something that involved a hashmap, you probably wanna be able to get an iterator. And then hashmap you've gotta know, "Oh, hashmap, I need to get the set of keys." Right? The keyset to be able to get the iterator on that. So these are also good places to have tabs in your book for quick reference if you need them.

And last but not least, files. So the basics of files. And you've done a lot of the basics of files already. Know how to open a file and read from a file. Know how to be able to write to a file if you need to write to a file. Chances are we probably won't have you write to a file since you didn't have to do it on any of the assignments, but at least reading from a file is fair game. So topically, that's kinda what's going on there. Now given the topics, let me show you some examples of the kind of questions we might ask besides what's on the practice final which you already have sort of a whole practice final that you can do, and I would encourage you to do that under a timed condition, so you can figure out what kind of things you're maybe a little rusty on, and what kinds of things you're feeling good about.

Here's an example of the kind of question we might give that involves a bunch of these concepts, which is given a hashmap – so you're given some hashmap whose type is hashmap string to string, and you know that these are keys and values. Find essentially all key values are the same. So what we want you to do is find matching key value pairs. What do I mean by that? What I mean is let's say we have some hashmap where here are the keys over here, and here are the values over here. And I might have some keys like Alice, and Bob, and Cathy. And I might have some values like Alice, Jeff, and Cathy. And so what I wanna find out for example to print on the screen is I'd wanna print Alice,

and I'd wanna print Cathy. I wanna write out any of the keys whose value is basically as their value in the hashmap. So that's a very simple sort of set up, but it stresses a bunch of these concepts that you need to know to be able to do it. And that's kind of hopefully the way a lot of the questions are gonna be on the final is simple concepts but stresses the ideas. So if we wanna think about solving that problem, we might be given some header like public void match keys, and this has some parameter that it's passed like a hashmap from string to string that we'll call map. And all this syntax should hopefully be stuff that's just second nature to you at this point. You really wanna figure out how to solve the problem.

If you think about this problem, if we wanna find out which keys have the same value as their value, what's the first thing we wanna do? Run iterate over the keys, right? We wanna have some way of saying, "Okay, I need to go through all the keys and look at them to see if they have the same value as their value." So I'd need to have some iterator. The iterator I know would have to be over strings because it's gonna be an iterator over the type of whatever the key is for the hashmap, and I'll call it it. And how do I get an iterator from the hashmap or over the keyset of the hashmap? So I need to say map dot keyset – these are good things to know in the back of your head – iterator – we sort of string these all together, and so what I now get is an iterator of strings which are all my keys in the hashmap. And now at this point it's just using my iterator but in conjunction with some other functions, like while my iterator has a next – and if you wanna use sort of the funky for each notation that's also in the book where it allows you to do iteration over an iterator in kind of a stylized form, that's fine, too, but we didn't stress that in class, so I'm not gonna stress it here.

While the iterator has a next element, I just get that next element – equals it dot next. And then I wanna see is this key equal to the value. So now it's gonna stress another hashmap concept which is to go look something up in the hashmap, and so I have string value equals map dot get, and what I'm gonna get from the map is whatever matches that key.

So if my key happened to be Alice over here, I'd go and get the value which is Alice out of the hashmap, and I'll just keep writing the rest of it over here. We're almost done. So over here we just say if the particular value that I got happens to match the key. Now it's gonna stress the concept of strings. So all these kinds of things fit together. If value dot equals key, which means that value equals the key value, then I print out to the screen to the key. I could print out the value, too, because they're both the same at this point, but that's the whole deal. That's the end of my while loop. That's the end of my method. So that's kinda the idea, right? This was not particularly earth shattering, but it stresses a bunch of concepts. Unless you remember the little things like, "Oh yeah, I need to get the iterator over the keyset," which is kind of a very standard pattern with hashmaps, it could take you a lot longer than it otherwise needs to take you. Okay, so any questions about that?

Let me show you one other simple example that's the sort of thing that would be fair game which is a little program called target seeker. Basically the idea here is there's a little target that's a red square, and there's a little target seeker that comes after it which is

the black square. And the black square's trying to get its center to match up directly with the center of the red square, fairly straightforward. And every time I click the mouse, I move the target somewhere else, so I'm just keeping the seeker – it's like, "Oh look, I'm so confused." Every time I click the mouse, the center of the target moves to the current mouse location, and the seeker just keeps seeking to wherever that target is. This problem won't be on the final, 1.) because I'm showing it to you now, and 2.) because it's really difficult to actually put that in a handout. Like take these set of pages and just flip through them really quickly and you can see what the seeker is doing. So how do you turn this into code? And the code for this is actually fairly straightforward. It's kind of you need to think about what you were doing in breakout, right? So some of the problems that you're gonna solve leverage concepts that you've seen in your programs, so you've had experience with these kinds of things before.

What we do is we initialize the target. Let me just line up these lines so it looks a little nicer. The target is basically just a rectangle whose size is some constant target size that's defined down here. I'll just show it to you. Target size happens to be ten, seeker size is twenty, and there's a pause time of ten, just so you know. So target square is gonna be the target. Target square is gonna be some new GRect that we set up, and that's a local – or a private instance variable that I wanna keep track of. My target square is just a private GRect. My seeker is also gonna be a private GRect, and I'm gonna keep track of the target that I wanna get to in a private X and Y location known as target X and Y.

So when I start off, I just set my target to be a square of this size, its color is red, and it's filled to be true, so it just initializes the target. And now I say, "Hey, the target starts in the middle of the screen." Presumably that would be in the instructions for the assignment, and so I get the middle of the screen. Know your standard ways of computing the centers of something. That's always good to know. Then I add my target to essentially the center of the screen, which means if I remember that my target square, the place I add it to a canvas is actually its upper left hand corner, I need to take the target X which is the middle of the screen, and subtract off half the width of the target in both the X direction and the Y direction to get its upper left hand corner. And then I add it to the screen, so this just sets up the target.

Then I set up the seeker, and the seeker is also a square of size seeker size by seeker size, and it just starts at zero, zero. That's presumably something that would be given to you in the problem, that the seeker just starts at a different location. And once I've set up the target and the seeker, I add my mouse listeners because I need to know where my mouse clicks, and then I just have this loop that's just always seeking. It's just always trying to find wherever the target is. And the target starts off at this target X Y location. That's where it always – how we're always gonna keep track of it. So what happens is on every step of seek we pause just so the seeker doesn't just jump there. It's not very exciting if the seeker just jumps there every time. And we say is the midpoint of the seeker – let's get that. That's just the seeker's current X location, which is upper left plus half the size of the seeker to come in. And we would do the same thing on the Y, which means we would go down half the size of the seeker in the Y direction. If my target X is greater

than my current X, then I need to move positively in the X direction. Otherwise, I wanna move negatively in the X direction.

And similarly with the Y coordinate, right? If my Y – so I find the seeker's midpoint Y. If my Y is less than the target Y, I need to move toward the target Y, so I'm gonna increase Y. If my target Y's less than the seeker's Y, I'm gonna decrease Y. So I just find these offsets. Are you gonna move in the positive or negative X direction, or are you gonna move in the positive or negative Y direction? And then I'm gonna have the seeker move one step in that direction. Uh huh?

**Student:** Why'd you do that instead of just finding the location [inaudible]?

**Instructor (Mehran Sahami):** Having it move consistently with –

**Student:** [Inaudible].

**Instructor (Mehran Sahami):** Well, so if you actually wanna do that, if you wanna have it look more nicely so that rather than trying to match up the Xs and then the Ys, if you wanna move toward it, you need to compute angles, and I just didn't wanna deal with polar coordinates, but you actually can do it with polar coordinates. We just don't do it because we're not – that's another thing that you know. You don't need polar coordinates, which is why you don't need to know that.

And the only time the target gets updated is when the mouse gets clicked. We set the target X and the target Y to be the new location of where we clicked. We removed the old target from wherever its location was, and we add it back to the new location. And that's the whole program. That's kinda the level of difficulty. Now this is a program that when you have the code, I can explain it all in like five minutes or so, but when you're writing this from scratch, I would probably expect that writing this program would take probably on the order of about 20 to 30 minutes to write it from scratch with some false starts, et cetera, et cetera. But this is kind of the level of difficulty you could expect to see on the final exam, and a notion of about how much time we would actually give you for it. So any questions about this or about our little hashmap example? That kind of gives you a sense of a set of topics or the richness in some of the questions. And you sort of got the explicit list of topics as well for the class. So any questions about the final? You're feeling okay. So I wanted to give you a bit of wrap up if there's no more questions, given that this is our last class. So if you kind of think about where we've been, we sort of had ten weeks – a little less than ten weeks if you count some – oh, we had a day off after the midterm, and we had a day off on Friday, and whatever the case may be.

But I would imagine for a lot of people in this class, there wasn't necessarily – there was some previous programming experience because I went through all your Assignment No. 1 e-mails. Remember those? That was so long ago. There was some prior programming experience, but there wasn't a lot of programming experience. So the real hope in this class was to get you to a point where basically you could take this journey over here from not necessarily doing any programming to getting to a point over here where now a

bunch of the stuff that you actually see, like games that you actually play either online or in an arcade if any arcades even exist anymore, like on your Xbox or whatever, say like Zelda.

For other kinds of things that you interact with like a data management system or social network, when it comes down to it, hopefully this class has helped demystify some of those things, that really underneath the hood, yeah, there's a bunch of listeners that are looking for things, and there are some graphics that get updated, and there's little pause loops, and we need to look for mouse events, and keyboard events. There's a bunch of big data structures like arrays, and hashmaps, and stuff like that. They keep track of stuff like your list of friends in the social network, or the communities you belong in. When you use a search engine, this is just doing the same thing. There's just some really huge repository of data that you now need to go look up. And the interesting problems that come up are things like this huge repository of data doesn't fit on one computer anymore. So what do you do? You break it up onto a bunch of smaller computers, and each of them gets some chunk of the data. And now you go from just simple data management problem to something worrying about breaking up data into bigger pieces.

And this is how computer science progresses in the sense that the problems get bigger in terms of what we actually deal with, and the interactions with the user get more complicated. But hopefully along the way what you learned in this journey was the science of what's involved in doing this stuff. So it's just a bunch of science as to how you actually figure out how all these pieces of a game interact, or how data is kept track of in some larger system, especially when you have smaller components that need to be kept track of in arrays and hashmaps that build up into larger components. And so those are the scientific parts of it.

But hopefully as you also saw today when we showed some of the graphics contest examples and when you wrote your programs there was obviously a variety of ways of doing the programs. That was the thing that at the beginning of the class isn't always clear is there's a huge art in computer science, which is why the textbook for the class is called the Art and Science of Computer Science – is this class, we taught you a lot of this stuff, and we hopefully taught you along the way some of the skills to be a good artist and to be a good software engineer. And the real key in programming is you'll learn more of this stuff as you go along, as you become a computer scientist because there is computer science after all in computer science, and so you'll learn more of this stuff. But what you'll also develop as you go along the way is really being an artist.

There's a famous quote by someone named Don Knuth who's considered the father of modern computer science, which interestingly enough he's still alive and he's in this department, so it's kind of like you're in the same department – you should go look him up, actually – with the father of modern computer science. And one of the things he said is that programmers tend to better if they subconsciously think of themselves as artists. When you're putting something together, don't think of it like you're doing a proof in math. It's important to figure out all the logical steps, but at the end of the day, you're putting something together that has an inherent beauty to it. And that's part of the real

beauty of computer science. So I hope you enjoyed the journey. I hope you learned some stuff along the way. And I just wanna thank you for taking part in the little journey we had together. So good luck on the final exam.

[End of Audio]

Duration: 47 minutes