

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 – Ether, 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 set 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 want to 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 JApplet, 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 want to 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 GCanvas, and gee, guess what? A GCanvas 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, whose 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 album's 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 `parseInt`, and so if you say `Integer.parseInt`, 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 `parseLine`. 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 centered 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