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 folks and 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 – of 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 printlin 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 and 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 printlin, 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 printlin 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