## **1010 - Graphical User Interfaces with Swing**

### **Content coming soon!**

Help Make These Materials Better!

I am actively working to complete and revise this eBook and the accompanying videos. Please consider using the following link to provide feedback and notify me of typos, mistakes, and suggestions for either the eBook or videos:

[CIS150AB Course Materials Feedback (Google Form)](https://forms.gle/4173pZ1yPuNX7pku6)

### **What’s the Point?**

* Identify the characteristics of a GUI
* Understand the role of the Swing library
* Create a GUI using Swing widgets
* Create event handlers

Source code examples from this chapter and associated videos are available on [*GitHub*](https://github.com/timmcmichael/EMCCTimFiles/tree/4bf0da6df6f4fe3e3a0ccd477b4455df400cffb6/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs).

### **.1. Graphical User Interfaces**

Until now, we’ve been creating console applications, in which the entire user interface is text-based. Though many utility applications are text-based, those are generally used by "power users," software developers, and system administrators. End users are used to applications that rely graphical elements to interact with the user. This is called a graphical user interface, which is abbreviated as GUI and pronounced "gooey."

GUIs on desktop and laptop computers have elements like windows, buttons, text fields, and checkboxes, and the user interacts using a mouse or touchpad. These GUI elements are called widgets or controls. Mobile applications have similar widgets, but they are designed for touchscreens and are generally more simplified than desktop applications.

Time To Watch!

Intro to GUIs [COMING SOON!]

### **.2. The Swing Library**

The code required to create and display a functioning GUI is complex and far beyond our current skills, but we can use pre-built GUI widgets written by other developers. These are typically bundled into libraries, or frameworks, that we can use in our own programs. The two most common GUI frameworks for Java are called Swing and JavaFX; we’ll use Swing in this course because it’s a little simpler for beginners and does not require any additional installations.

|  |  |
| --- | --- |
| **Note** | JavaFX is newer and more powerful than Swing, but it’s also more complex and has a steeper learning curve. The concepts learned in Swing will transfer to JavaFX, so learning Swing is a great place to start. |

To use Swing classes, we simply need to add an import statement at the top of our Java file:

import javax.swing.\*;

We’ll begin by instantiating a JFrame object, which is the main window of our application. We can use setters on the JFrame object to set attributes like the title and size of the window; we’ll also want to set the attribute that determines what happens when the user closes the window. We can then add other widgets to the frame, like buttons, text fields, and labels. Finally, we’ll set the frame to be visible, which will cause the window to appear on the screen.

*Example of a simple GUI using Swing*

import javax.swing.\*;

public class BasicGUI {

public static void main(String[] args) {

JFrame frame = new JFrame("Hello, World!");

frame.setSize(300, 200);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JLabel lblHello = new JLabel("It's so gooey!");

frame.add(lblHello);

frame.setVisible(true);

}

}

This code creates a window with the title "Hello, World!" and a label that says "It’s so gooey!".

|  |  |
| --- | --- |
| **Note** | The JFrame.EXIT\_ON\_CLOSE attribute tells the program to exit when the user closes the window. If we don’t set this attribute, the program will continue running in the background after the window is closed. There are a few other options for this attribute, but EXIT\_ON\_CLOSE is the most common. |

*Figure 1. Screenshot of*BasicGUI.java*in action on macOS*

As you can see, the code to create a simple GUI is is a little more complex than the single println() statement of an equivalent console application, which is why we’ve waited until now to learn it.

Time To Watch!

Intro to Swing in Java [COMING SOON!]

Files from video:

* Starter code: [SimpleGUI.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/SimpleGUI.java)
* Completed code: [SimpleGUIFinished.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/SimpleGUIFinished.java)

### **.3. Event-Driven Programming with Swing**

A GUI application offers a lot more flexibility than a console application, since the user can interact with the program in many ways. In a console application, the program runs from top to bottom, and the user can only interact by typing text. In a GUI application, the user can click buttons, type in text fields, and select items from drop-down lists. This means that the program must be able to respond to these events; we call this event-driven programming, and it is a key concept in GUI programming.

In Swing, we can add event listeners to widgets, which are objects that respond to events. For example, we can add an event listener to a button that will run a method when the button is clicked. The method that runs in response to an event is called an event handler. Once we’ve added an event listener to a widger, the event handler is like any other method in our program, and we can write it to do whatever we want.

Time To Watch!

Swing Event Handling [COMING SOON!]

Files from video:

* Completed code: [GreetingFrame.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/GreetingFrame.java)

Note: there is no starter code for this video.

### **.4. Processing User Input with Swing**

Once we’ve learned how to work with widgets and add event listeners, we can put everything together to create a GUI application that gets input from the user, performs actions or calculations with that data, and displays the results.

Time To Watch!

Calculations in Swing [COMING SOON!]

Files from video:

* Starter code: [GUICalculations.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/GUICalculations.java)
* Completed code: [GUICalculationsFinished.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/GUICalculationsFinished.java)

### **.5. Widgets**

Using JLabel and JTextField, we’ve been about to create GUI programs that function much like console applications, but that doesn’t really take advantage of the power of GUIs. A well-designed GUI application utilizes specialized widgets that are designed for specific types of user input and output.

Here are some common widgets and their purposes:

**JLabel**

Displays text or an image.

**JTextField**

Allows the user to type in a single line of text.

**JTextArea**

Allows the user to type in multiple lines of text.

**JButton**

A clickable button that can run a method when clicked.

**JCheckBox**

A checkbox that can be checked or unchecked.

**JRadioButton**

A radio button that can be selected or deselected, and can be grouped with other radio buttons to limit the user to selecting only one.

**JComboBox**

A drop-down list that allows the user to select one item from a list.

There are many more widgets in the Swing framework, but these will cover most of what you’ll need for basic GUI applications. There is a great deal of documentation available online for the Swing framework, so you can always look up how to use a specific widget.

|  |  |
| --- | --- |
| **Tip** | Use widgets as they are intended, even if they are able to be used in other ways, such as displaying output in a text field. Users are accustomed to certain behaviors from widgets, and using them in unexpected ways can make the application harder to use. |

#### **.5.1. Widget Naming Conventions**

When naming widgets, it’s a good idea to use a consistent naming convention that makes it clear what type of widget it is. There are a variety of conventions for naming widgets, but two are most common:

* Prefix the name to identify the widget type, like lbl for a label, txt for a text field, or btn for a button.
* Append the type of widget to the end of the name, like helloLabel, nameTextField, or submitButton.

I use the prefix method in my code, mostly because I’m used to that from C# programming (where that’s the preferred style), but you can use either method in my class. The most important thing is to be consistent in your naming so that anyone reading your code can easily understand what each widget is for.

Time To Watch!

More Swing Widgets [COMING SOON!]

Files from video:

* Sample code: [JCheckBoxDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/JCheckBoxDemo.java)
* Sample code: [JRadioButtonDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/JRadioButtonDemo.java)
* Sample code: [JComboBoxDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/JComboBoxDemo.java)
* Sample code: [JComboBoxDemo2.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/JComboBoxDemo2.java)

### **.6. GUI Layouts**

As we add more widgets to our GUI, we’ll need to consider how they are arranged on the screen. While we can set the position of each widget manually, this is tedious and doesn’t work well when the window is resized. Instead, we should use layout managers, which are objects that arrange widgets in a specific way and respond to window and screen sizes in predictable ways. Think of a layout manager as a set of rules that determine how widgets are arranged in a window.

The Swing framework is itself built on top of **another** framework called the Abstract Window Toolkit, or AWT. Swing hides most of that from us by implementing classes that extend the AWT classes. For example, the JLabel class is a Swing widget that extends the Label class from AWT. So we don’t directly use AWT very often, but the exception to that is layout managers. Layout managers are part of AWT, so we’ll have to import them from the java.awt package:

import java.awt.\*;

There are several layout managers available in AWT, each with its own strengths and weaknesses. The most common layout managers are:

**FlowLayout**

Widgets are arranged in a single row or column, and wrap to the next row or column when the window is resized.

**GridLayout**

Widgets are arranged in a grid, with a specified number of rows and columns.

**BorderLayout**

Widgets are arranged in five regions: north, south, east, west, and center.

To create more complex layouts, we can nest layout managers, which means that we can put a layout manager inside another layout manager.

|  |  |
| --- | --- |
| **Note** | There are tools that allow us to create GUIs visually, by dragging and dropping widgets onto a window, and then generating the code that will produce the GUI. But our goal is to learn how layout managers work, so we’ll be creating our GUIs by writing the code ourselves. For GUI designs that don’t have to be turned into an obnoxious professor for a grade, you’re welcome to take advantage of these tools. |

Time To Watch!

Swing Layouts [COMING SOON!]

Files from video:

* Sample code: [FlowLayoutDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/FlowLayoutDemo.java)
* Sample code: [JPanelDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/JPanelDemo.java)
* Sample code: [GridLayoutDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/GridLayoutDemo.java)
* Sample code: [BorderLayoutDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/BorderLayoutDemo.java)
* Sample code: [NestedLayoutDemo.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/NestedLayoutDemo.java)
* Sample code: [NestedLayoutDemoColors.java](https://raw.githubusercontent.com/timmcmichael/EMCCTimFiles/refs/heads/main/OOP%20with%20Java%20(CIS150AB)/10%20Swing%20GUIs/NestedLayoutDemoColors.java)

Last updated 2025-03-08 13:39:39 -0700

# Transcript 1

back in the early days of personal computers the user interface was based entirely on text there were no real graphics and you didn't have a mouse so everything was done by reading words off the screen and typing in commands on your keyboard to tell the computer what to do even the state-of-the-art computer games of the era like zorc were entirely text based the game would describe what was around you in black and white text and then you'd type in the action you'd like to take for example if it told you there was a leaflet in a mailbox you'd type read leaflet or you'd type East to move East it was amazing believe it or not and after a long day of school walking uphill in the snow both ways of course I couldn't wait to get home and play games like Zork in general beginning programmers start out creating text based programs too you use some kind of a print or write statement to display some output you collect some keyboard input and store that in a variable and then you do whatever you need to do before displaying the result using text we start that way because it's really easy to get up and running quickly and then you're just sort of stuck with those text or console based programs until your professor gets off his butt and shows you another way so just like early PC Pioneers we start with text based stuff until something better comes along and that something better that comes along The "first" GUI [Music] is a graphical user interface which we call a GUI but really everybody just calls it a gooey now this particular guey is from a computer called the alto and it was a very early prototype of what a guey could look like it wasn't really a personal computer it was more of a research project I think they determined that if they were really to build it and sell it it would cost something like $10,000 so it didn't have any commercial impact directly but it does have a huge place in History Steve Jobs took a team of Apple employees to see a demonstration of the alto and what he saw inspired them in 1984 to release the Macintosh which is the first kind of widespread personal computer that relied primarily on a gooey for its user interaction and this kind of guey is based on the idea of point and click your monitor displays images instead of text and if you want to interact with it you use your mouse to click on it the Widgets and event-driven programming components of the guey on your screen that you click on and interact with are called Widgets or controls now if you're under the age of 50 you've been using these widgets your entire life because they're things like labels for displaying text buttons to click on text boxes if you need to type something in and drop-down lists which have the fancy name combo boxes and there are of course lots of other Widgets or controls as well but you see these in many of the programs and websites that you use on a regular basis there are some really fundamental differences between how a program with a guey executes compared to a program with a text interface a text based program runs in a specific order that's determined by the programmer and when it needs something from the user it stops and sits there the user can't move on until they give the program what it's asking for a guey program on the other hand can make the interface available and let the user interact however they want basically basically it lets the user dictate the flow of the program by sitting and waiting patiently for us to use a widget when the user does interact with a widget it runs a chunk of code related to that interaction if the user clicks on the save button it runs a function or method to save the work we call this event-driven programming and the idea is that you're going to write your code in what we call event handlers having our program execute using event handlers that respond to the user gives us a lot of flexibility and lets us do things that you just can't do with a text based program so why do we wait until now to learn about goys we should have been doing this all along right the truth is creating a guey can actually get pretty complicated it's not that difficult once you get the hang of it but it's one more thing at the beginning that novices just aren't ready for and we don't want to make them frustrated and Confused just trying to get something simple to display it's easier to start with simple output and input statements still creating all these text-based programs all this time has probably gotten pretty old and you might be a little frustrated with it and as an instructor I want to cash in on a couple of habits I've been trying to instill in my students up to this point "Decoupling" the user interface for example I keep telling you not to put output statements in your functions or methods but to pass data around using arguments and return statements and to do your calculations and save them to variables instead of just sticking them in your output statement and that's been pretty annoying but when you include output in your code that does the work and when you do math inside of your print or write statements you're tying your code to the user interface if you want to use that code in a guey program or a web app you got problems all those output statements won't work anymore if we separate or decouple our code from the user interface it ensures that code is reusable in any kind of user interface I use the analogy of a restaurant since lots of students have done that kind of work in Food Service you refer to the front of house and the back of house the front of house people like servers and hosts interact with the customers and then pass that information off to the backup house where the cooking is done it's then given back to the server to deliver to the customer the customer doesn't need to see what goes on back there Au gooey is front of house and does the work of interacting with the user and calling the event handlers which are the back of house functions needed to get results back to the user so you've paid your dues and written a bunch of boring text-based programs like your professor told you to do now it's time to start creating fancy graphical user interfaces for the rest of the boring programs your instructor makes you do from here on out

# Transcript 2

let's start out by looking at a fairly simple guey like this if we think of these widgets as objects we've already got a bunch of classes to write label text box combo box checkbox and button that's a lot of work before we can even get our goey on the screen so there's a lot library of pre-built widgets we can use called swing actually there are a handful of libraries I do want to mention Java FX which is a newer and really better Library than swing we're not going to use it because it does require a little installation and configuration that I don't want to make students deal with especially if they're new to programming and we're using swing to show fundamental concepts which will transition over to Java FS X if you want to explore that later in swing every widget needs to be a part of a hierarchy of containers before we can put a menu bar at the top and start putting widgets into the content pane we have to create a top level container in most cases that's going to be a jframe jframe will have some additional uses for us later on but for now we can just think of it as something we need to put in place to hold all of our widgets let's take a look at pretty much the most basic swing program we can make we'll need to import the swing package so we have access to all that widgy goodness so don't forget that notice that swing is considered an extension to Java so it's import Java x. swing rather than java. swing inside our main method we'll start by declaring and instantiating a jframe object which we'll just call frame in the jframe Constructor call we can pass in the title of the window and that will be displayed at the top next we'll set the size of the window which here is 300 pixels wide and 150 pixels High last but not least we need to display our amazing window instantiating a jframe object doesn't display it on the screen we can use the set visible method to do that and and it's looking pretty good it's looking pretty boring there's not much to it but it is running notice though that it's actually running as a separate program you can see the little Java icon up here and if I click on the Red X or the Red Dot depending on your operating system it hides the window but it doesn't actually stop the program from running anymore I can take care of that easily Enough by killing the terminal in vs code or by clicking the stop button so this isn't a big deal but we do need to understand that that is the default behavior for a jframe because often a program will open a different window once the user closes the one they're working with we can change that behavior to cause it to close the entire program using something called the default close operation on our jframe object so I'm going to call set default close operation and what I have to pass in is something called an enumeration uh so I'm going to type jframe and get that dot and now it's got a list of all these they're basically like constants that are defined inside that jframe class that we can use here and so I want exit on close that's the behavior I'm looking for let's try that out so once again I've got my guey to appear and this time if I close the Red X it has stopped the entire program from running I can see I don't have those controls up here anymore and that icon is no longer in my taskbar once we have a jframe in place and have set up how we want it to look and behave we can start adding our widgets the obvious place to start is to display some text I know it kind of defeats the purpose here but sometimes a guy still needs to display some text in a goei we usually display text using a label which is implemented with the J label class in swing when we start creating widgets we need to be thoughtful about how we name them goys can end up having a lot of different widgets and when we're writing code that references them it's easy to get confused unlike most other things in Java there's no prescribed naming convention but there are a couple common approaches and what they all have in common is that you need to describe the widget's purpose in the guey one convention is to prefix the name name with an abbreviation that indicates the widget type so lbl hello for a label that says hello the prefix for a text box might be txt BTN for a button and so on I tend to do it this way because it's common in C and I've spent a lot of time in that other people append the widget type on the end but they usually don't abbreviate it so you'd have hello label and some Psychopaths don't indicate the widget type at all hey it's your funeral we'll want to get this J label into our frame before we make the frame visible otherwise we'll need to force the frame to render the window again so I'll make some space here and I'll instantiate A J label I'll call it lbl hello and this is pretty straightforward The Constructor call I make is going to accept the text that I want to display so I'll say hello world because that's how I always start these kinds of things at this point the object exists in memory but it isn't part of our guey we need to put it into our frame using the add method frame. add and just pass in the name of the widget and let's give that a shot okay now I know what you might be thinking this isn't that big of a deal it's not that hard this guy could have taught us this a long time ago instead of making us spend so much time on those boring text based programs not so fast let's go ahead and add another label so J label I'll call this one lbl greeting equals new J label and for this one I'll say welcome to my amazing cooy and don't forget which I often do to add on uh that new label into our frame and let's take a look at at that H it's not working like we hoped if we want to add multiple widgets to a frame I have to tell it how to organize everything it uses something that's called a layout manager and there are a handful of different choices the easiest one is called flow layout so let's go up here where we're kind of first setting up our frame and let's use the uh set layout method now what this accepts when I'm doing my parameters here it has to be something uh a subass of something called layout manager and the one we want like I said is a flow layout so I'm going to put new because we have to make an object um flow layout and parentheses because there's a Constructor call in there now vs code is not recognizing this the the layout managers are not part of Swing they are part of something called uh a WT which is Way Beyond what I want to get into right now but I do have to import that java. awt star and that gives us our layout managers it gives us some colors and some other things that we'll use kind of along the way uh let's go ahead and try running that okay so both labels got displayed but it's just kind of jammed them together let's go back to the code and let's try making that second label a little bit longer that takes up a lot of space and let's try running it now this might make it a little more clear what is happening because you can see that it's put that second label on the next line down incidentally it's also too big to fit completely so it centers it and it kind of crops the edges what a flow layout does is a lot like the way a word processor behaves it goes to the top left and it starts adding widgets from left to right and then when it gets to the right hand side and runs out of room it just goes down to the next line and continues from there this isn't necessarily the behavior we want but it lets it lets us get one more widget on the screen uh for now as I said in a previous video there's a reason we start out using print line statements look at how much code it takes us just to get a couple of lines of text on the screen in a way that doesn't even look great and it doesn't even do anything interactive yet for that we'll need to move on and learn about handling gooey events which we'll do in the next video

# Transcript 3

before we can bring our goys to life and make them respond to the user we need to do a little housekeeping we're going to take our basic understanding of Swing widgets and reorganize our guey in a more objectoriented way instead of dumping all our widgets in our main method we can extend the jframe class and treat our gooey window like an object with this approach our widgets are fields or instance variables of our frame and our Constructor can set up the frame instantiate the widgets and add them to the frames layout this approach gives us the ability to instantiate from our class multiple times giving us multiple windows to use in our application more importantly for the time being it's going to make it a little easier when it's time to start dealing with events here's how a more object-oriented guey this program is going to use a gooey to get the name of the user and then display a simple personalized greeting and before I forget we do need to have those same two import statements that we've uh been using previously so from java X we need the swing package and from java we need a WT which is abstract window toolkit it's not something I go deep into but uh we're going to have a couple references there that will uh will come from awt the whole whole idea here is that we want to start using inheritance and kind of an oish approach so my class is called greeting frame because it is going to extend uh the jframe class so this is going to be the frame that we use as the the basis of our guey and our fields in this kind of objectoriented approach is going to be where we put our widgets so uh I'm going to start out by just declaring what all of my uh widgets are and so I'm going to start with I'm going to need a couple of labels I will need a um a prompt for the user to tell them what they are typing in right so that's going to be a label that says hey put in your name and what we've done in the past is we've sort of just gone right ahead and then instantiated that there is absolutely nothing wrong with doing that um and a lot of people would do it that way I'm going to just try and be consistent with my previous lessons on oop where at the top of my class I just declare my variables and I declare them as private because they are fields and uh I'm going to then you know essentially set values to them or instantiate them in the Constructor just like I do with any other class that I develop uh and so I'm going to put both my my labels here on one line since I'm doing it that way the other label I'm going to need is at the uh at the end when they're done I need a label to display that greeting next up I'm going to need a text field or I call him a text box but the class in swing is J text field uh and I'm now using this prefix naming convention here right so uh the the label that tells the user hey type in your name is called lbl name well that gets paired up with a text field that goes with it right the prompt is referring to what they type in this text field and so this prefix naming convention lets me kind of use the same root name but say lbl for the label and txt for the text field so um that's one of the reasons I like this approach again the the specific naming convention doesn't much matter and the other piece we're going to need is a button for the user to click on when it's done uh we could incidentally make it so they just hit enter or something like that but we're going to keep it simple uh so that's going to button submit so I've got those three Fields or you know instance variables I always call them that are going to be our widgets again in an object-oriented class we are going to use a Constructor to essentially set values to all of those Widgets or or fields and the first thing we did in our old approach was we set the title of our frame in our Constructor and so I guess I could ask for that here but what I'm going to do partly to remind you that you have this option is I'm just going to call the superclass Constructor and I'm going to pass in my title that way now I don't have to do it that way because there is also a the frame jframe excuse me class has a set title method so I could also just do it that way but um I'll do it then with the super super class Constructor call and the next thing I'm going to do is I'm going to kind of configure the way my Frame Works and so the one I always forget is set default close operation for some reason I never remember to get that right and it's really annoying if you forget because I just click the the Red X to close it when I when I test it and then it's still running in the background so jframe do exit on close should give me the behavior I want um and now the next thing I want to do is set the size so remember um before I was saying frame. set size well that was because frame was an object that I had just created but now this is the frame this class is the frame and so that's why I'm referring to this dot uh for that the this dot is um is optional but I like to to use it and that's part of the reason why is to be consistent where before we were saying frame dot now we're going to say this dot uh so I'm going to set size I'm going to set it to uh 300 by 200 and the only reason I know that is because I fiddled with it before I started recording so that I would know roughly what size I want it and then I have to give it a layout man manager set layout and I'm going to use flow layout which I think is the only one I've even talked about at this point so that it knows how to arrange these widgets when I start adding them to the frame with my with my frame kind of set up and ready to go I actually could um could start running it and testing it and so here's where I'm going to break from the way I normally do things so normally when I'm teaching object-oriented programming I would go create a separate file that would be like my program with a main method and it would instantiate a greeting frame object and uh and go from there this video I'm going to be showing you a handful of different things that we haven't really talked much about before so I I'm trying to keep it as simple as possible and only for that reason I am going to put my main method inside this greeting frame class and it um it makes me want to take a shower because this is just not the way that I normally do it I I believe pretty firmly that we should have a separate program for this but um for the sake of your learning here and making it easy I'm going to just do it right here so I'm going to um in declare and instantiate my uh my object here so this is going to be a oops it's not going to be a gooey it's going to be a greeting frame and I'll just call it frame since that was the wording we used before equals new greeting frame and uh if I run this it will execute but I'll save you the suspense I have forgotten haven't forgotten but uh normally up here we would also set this to be visible after we're done adding all of those things I'm going to save that for when I'm done with my Constructor I'm going to say frame do set visible to true and I think I should be able to run this and see my goey so far which is not going to be much of a gooey uh it's just my window set to the size that I wanted it's got a FL a flow layout set up there and it should close when I uh when I click the red X but um as I always say I don't like to go along without testing my program and honestly we're pretty much done with our um with our main method at least for a long time so I'm going to fold that up and get it out of the way and head back to my Constructor so the next thing I'm going to do now is um is configure my widgets so first up is my uh label called lbl name now remember in the in the other way we we sort of combined all into one statement and we said J label um lbl name we're not doing that anymore if I put J label it's going to try and make a new J label with this same name so U now this is already an instance variable so I'm going to say this. lbl name equals new jlabel and then I'm going to put the text that I want which is just a prompt to say enter your name next up for the user is uh that prompt goes along with a text field so this. txt name and again I just like that those kind of go together and sorry for the quick cut there my dog started barking so this is going to be a j text field now in the Constructor call here I could um I could specify some text uh so you know I could put a default value essentially and and that would be fine it would display that and then the user will be able to just type over it but I'm going to leave it just blank for now next up is our button new J button I will just make that say submit not very creative and then last but not least is our uh our outputs label at the bottom where where we're going to um output our message now problem here is I don't know what my message is so I can't put anything there um I could put some placeholder text in there and it would display um I could even put some placeholder text there and then make it invisible uh so that the user doesn't see it but the easiest thing is just to leave that empty another one that I always forget is I have to add these widgets to my frame and previously we would have said frame. add but remember we are in the frame object now because that's what we are what we are uh working on here so frame. add instead is going to be this do add and Order counts right um so uh the flow layout puts them in the correct order for us so first is going to be the label and then is going to be this dot txt name I'm guessing most of you aren't bothering with be this dot nonsense but I don't think it's nonsense and it's my video so I can do whatever I want so then is the button and then this dot lbl greeting I think that is all of our controls let's go ahead and run that Setting the size of a widget again okay we're looking pretty good but we have a couple of problems one of them we can see and one of them we can't uh they are related to each other this little thing there is my text box I think I can type type in there you can't see that but I think it's accepting my input um problem is the user has no idea what they're typing they may not even recognize that as a text box and and know that they can type there uh what happens is the flow layout manager is going to make everything as compact as possible so when I give the text to a label it makes it exactly this size it doesn't leave Extra Space same thing with our text field since I left it empty it says oh it doesn't need any room so I'll squish it down we need to address that so I could like I said I could put some kind of default text in there and I could just make it really wide right that would uh that would do it but then I would have that ugly text up there and I have to make it visible and invisible and stuff like that um what I want to do instead is tell the layout manager the size that I want this text field to be and so um it's similar to what we did up here with the the frame itself but with a layout manager like this we can't specify the exact size what we can do instead is I can say this. txt name. set preferred size and what I can do is say this is what I would like it to be there incidentally I could also say set minimum size and the the layout manager will try to accommodate that but if it can't because there's too much other stuff going on or too much in the way it might not actually respect this request um but the um the set preferred size method doesn't take just a pair of numbers it takes something called a dimension object and so that's where we're going to put our two numbers and the only reason I know to make this 250 by 25 again is I sort of played around with some layout before I hit record and so I think that's going to look about the size that we want now I said there was also a problem we couldn't see this was one we could see uh the J label has the same size issue right when it um when it compressed everything to make it efficient or um as small as possible as compact as possible that J label was empty so it's there's a j label there but it's just teeny tiny and I need that to be there so that when I put a greeting it will display because the behavior here if I change the text of that label it doesn't sort of re uh recalculate and redraw the size unless I tell it to and that's just a pain in the neck so I'm just going to do the same thing I'm going to set a preferred size for my greeting label and I'll just use the same Dimension I did uh up above 250 by 25 and I think that'll look all right let's go ahead and run this and see how it looks now there we go you just have to trust me that the the label is there I guess you don't have to trust me I can I can prove it I can bring receipts right right there we go there's my label um but it looks a lot nicer to the user if it doesn't have that information there all right so this is a a good enough basic guey for the the program that we are creating here and you may be thinking well we haven't really accomplished much right I could have very easily written this same thing in our old way of doing it just put it all in the main method right um so I'm going to remind you that again my purpose here is to teach objectoriented programming so it's important to me that we're uh we're writing this in kind of an oop manner so that's one thing um this does have a couple of other effects one of them is if I wanted to I could come down to my main method which really should be in a different class if you ask me but um but I could make multiple instances I could have a frame two and then I could set them both visible and they'd both be on the same uh on the screen at the same time or I could make one visible and then make the other pop up when the first one closed or something like that it's doesn't help us in this particular program but it could be useful other times more to the point though this is going to help us this um object-oriented approach where we are extending jframe it's going to help us with the next topic we turn our attention to Let's revisit the idea of Event-driven Programming introduction event-driven programming in the context of a guey the program is going to render the window and the widgets and then it will wait until the user does something those some things that the user does are are called events and some of them are really obvious like clicking a button or checking a box but there are all kinds of other events every time the mouse moves that's an event hovering over a widget is an event typing in a text field is an event and so on so these events are happening constantly while the guey is active as developers our job is to decide which events to respond to and exactly what those responses are maybe the program doesn't need to do anything when the user is just moving the cursor around but if they click that button the user expects something to happen the mechanism that recognizes an event and triggers code to run is called an event listener and the code it triggers is called an event The ActionListener interface handler event listeners are way more complicated than what we can do right now but swing actually that awt package I keep glossing over can take care of that part for us with something called action listener so we'll need to use that in our class in the same way we use stuff from jframe but remember that in Java multiple inheritance isn't allowed a cousin who's a close friend can't be both a friend and a family member a class can only have one super class to get around the multiple inheritance problem action listener isn't actually a class it's something called an interface an interface is like a partial class that establishes a set of rules and by rules what I really mean is that it establishes one or more methods that we have to create in any class that uses it and instead of extending a class like we did with jframe we'll implement the action listener interface we can let that take care of the listening and we can focus on writing an event handler which is what the program does in response to the event let's see how that works we'll begin with another import statement we need uh java. aw. event so that we can deal with some of this event handling stuff and we're going to move on to this line of code where we Define our class and we've set it up to extend jframe and we know that we can't uh keep putting extends and and extend other classes Java only let you extend one class what we can do though is we can implement an interface and so we're going to use the implements keyword and the interface I want is called action listening now right off the bat we start getting an error what an interface does in Java is it defines one or more methods but it doesn't write any code for those methods so it it decides on a name and a return type and whatever arguments it needs but there's no curly braces with the code that runs when that method is ex execut it the compiler can't deal with that it needs to have code so if we're going to implement an interface we have to write the code for that method or however many methods there are it's a little bit like overwriting except we're not replacing code that was written in a super class we're implementing it for the first time we're writing the code for the first time the method that action listener defines and needs us to complete is called action performed and that's actually our event handler so it was something we were going to have to write anyway vs code recognizes what the problem is and if I hover over that um the the name of my class it offers a quick fix and the fix is to you know write the code for the the methods that aren't implemented so you don't have to let VSS code do it for you but let's go ahead and click that and see what it comes up with I'll have to scroll down Writing an event handler with actionPerformed() to the bottom to find it and here it is it has the uh override compiler directive because it does need to make sure that it correctly replaces uh the definition in our interface and then it's put a couple lines of code in there that look really unfamiliar and possibly a little intimidating or scary this is just placeholder code what it actually does is it causes an error to happen if we try and run this code because it doesn't want us to forget that we we did that maybe in a hurry and then we go off and work on other stuff and we forget to come back with it to finish this when we run our program it's going to crash when it gets to this point so we are meant to delete this code and replace it with whatever the real code is that we want I'm also going to take care of one housekeeping thing that is just one of my little um Obsession things here I'm going to put this action performed method above my main method I already feel weird enough having this main method in this class as I keep saying I'm going to scoot it down to the bottom because it really is kind of separate now I just want to create a little personalized greeting and display it in that label at the bottom of my form I'm going to write this with three separate lines of code even though we could do it in one and that's just because I want to be clear exactly what I'm doing and what the different pieces of this are intended to do so the first thing I'm going going to do is create a string variable and I'll call it greeting and I'm just going to put kind of hello and the first part of our greeting in there and next what I'm going to do is I'm going to add on to that string by grabbing whatever the user has typed into that text field so I'm going to say plus equals and now I'm going to retrieve this Dot txt name. get text so as the name suggests the get text method goes to that text box text field and retrieves whatever is in there now if I've left a a default string in there when I first created it then it will get that back out but if the user has replaced that with something else get text is going to get that for me and I'm going to add it on to my little greeting string here and finally I want to take this greeting string that I've created and I want to display that in my label that I've created for that purpose so I'm going to say this. lbl greeting. set text and in my set text method I'm going to pass in the greeting that I've set up so that makes sense right in the widget that is intended for the user to put in input I'm going to get that information out and in the widget that is intended to display my result I'm going to set T text back into it let's run this and try it out we'll see what happens when I type in a name and click the button type in Tim and click and nothing happens let's take a look at why Attaching a listener remember that there are two different pieces of code that are involved in this uh event handling uh interaction we've created the event handler but we need an event listener to trigger this code and cause it to execute so what we have to do is we have to find the widget that is related to this event and we have to attach a listener to it so I'm going to move up into my Constructor here where I've done all of my work with these different widgets and the widget that I care about for this event is the button right so I'm going to add a little space down here I'm going to say this button submit. add action listener all right and what this is going to accept as a parameter is an object that has this action performed method that it's going to run now that is set up this way because we could be using a different object we could have a different class where we we've defined our event handler and we would need a reference to that in our case our action performed method is right here so we need to attach this listener uh to the event within this same class so I'm just going to put the keyword this as we've been doing kind of all along now let's try it again again type Tim click submit and sure enough it says hello Tim so now we're getting somewhere right this might not be the most exciting program in the world but this little example has opened up a whole new way for you to write your programs before guies you knew how to display text and get input in the terminal or console with text Fields labels and buttons you can do the same thing but you can do it in a goey that looks a lot nicer for the user and is probably a lot more what they are expecting when they open a program of course there are lots more interesting widgets out there that we can use and we'll explore those in another video

# Transcript 4

we're going to jump right into an example of using swing to create a program that does some kind of a calculation this is going to be a short video because we're not really learning anything new there there might be one little piece of this that you haven't seen before but mostly this is about taking what we've learned about swing so far and applying it in a little more real world kind of an example because when we first start out we're often learning to how to put a message on the screen or something like that so in this program we're going to take some input from the user we're going to perform some calculations with it and we're going to Output the result on the guey let's take a look at the guey I have set up for us it's really simple really straightforward this is going to ask the user to enter the amount of they spent on food the amount they spent on drinks we have a check box for the amount for adding a 20% tip and then we're going to calculate the total price pretty straightforward as always we will begin with our import statements I do need to add that awt event package at the top and now that I have that I can add on my interface to this so I'm going to say implements action listener right and as we've seen once we add that implementation of action listener we're going to start getting an error because the action listener interface requires us to have a method called action performed and so I'm going to not be able to compile until I put that in place I'm going to scroll down I've still got this main method down at the bottom I've talked about that a few times I don't really like that it should be in a different program as far as I'm concerned uh but I want to keep everything on one screen for the the sake of the recording but I do need to create this uh method this action listener uh action performed excuse me and I don't actually remember the exact um method signature here if you remember in a previous video I just let Visual Studio code create this for me but I think I can figure this out I think I remember um it takes an action event uh object as a as an argument here and I think that's yep that got rid of our error I really should these really are overrides when we're doing that so I'm going to add that override uh compiler directive kind of late for that I've already figured out um how to do it correctly and here's where I'm going to do my calculations inside this event handler and so we've got uh two text fields we need to work with we've got the price of the food and the price of the drinks and then we've got a checkbox that we need to deal with and the first Retrieving and converting JTextField values text field or text box that I'm going to work with I'm going to break it up into a couple different lines of code uh to make sure it's really clear exactly what we're doing and on the second one I'll combine them in the way that most people would normally write them so what I'm going to do with that first text field is I need to get whatever text is in there and I'm just going to save that in a string variable called input so I'm going to call this dot text food. gettext and we've seen this before right this um the get text method is going to return a string and so that's why I need a string to store that but as long as it's a string I can't do any math with it I need to convert it to some type of numeric data and this is the part that you may or may not have seen before um I need to convert that string into a double and so I'm going to make a double called food price and I'm going to use the double class and now since I'm talking about the class that's going to be a capital D and then it has a method called parse double and parsing is a is a phrase that is a term that means kind of taking something apart and so this is going to take apart a string and it's going to return a double based on that string if it can and that should get us our uh input and convert it to a double I say it should but I haven't tested it yet and so one of the challenges that comes up with a guey uh program is how do we test it and make sure everything's working correctly I personally probably would use some of the debugging Tools in VSS code but we still do have the oldfashioned way which is to use a system out print line statement and so I'm going to say that the food oops the price of the food is going to be food price and I can still output to the terminal in that way the user if they're running the program kind of as an end user they're not going to see that but I as the developer can still do that and make sure everything's working okay so I'm going to test this and make sure that it prints out my food price correctly okay so I'm just going to put in $999 and I'm going to hit calculate and nothing has happened so I have to figure out what's going on uh it looks like because I have this print line statement it looks like this action performed is not actually executing and the reason is because I I've skipped a step and I warned you in one of my other videos that it's a step that I often forget and you need to in addition to writing your event handler code you have to attach that uh that listener to the the widget that's going to be um producing the event right so in our case I have a button that's called BTN Cal and we need to keep an eye on that for a button click so I'm going to add an action listener and as a reminder it takes um as a parameter it takes the object where it can find this this action performed uh code this essentially this this object that implements action listener and uh We've written it in the same class here so it's going to be this let's run that again and see if that gets us at least executing that event handler okay so I'm going to type 999 click my button and it has output food equals 9.99 so my code's working so far now that I've got that um that action listener attached to my button there and I don't really need that system out print line anymore I'm going to leave it there as long as I'm debugging and then when I'm all done I'll clean all that up so what I've done is I took the text from that text field I saved it as a string and then I used this double class to convert that to a double now this isn't foolproof it's pretty easy to use but it does have a potential flaw and that is um it goes back to my Golden Rule of programming which is that users are dumb and if I ask somebody to enter the price of food somebody's going to come along and type blue that causes a big problem with our code causes essentially a crash that the code in this gooey environment is still running um but I'm going to stop that this has caused an exception because you can't convert blue into a double or at least the parse double method doesn't know how to do that uh we can write uh kind of exception handling to deal with that if you know how to do that that's beyond the scope of what we're doing right now so we're just going to live with this flaw which is that if the user does something dumb it might cause our program to break Streamlining the retrieval but I've got this two-step process right I've gotten the text and I've converted it to a double and for the other text box I'm going to combine those into one step which is the way most people would write this once they have the hang of it so I'm going to make a double called um drink price and I'm going to say it's equal to double um parse double and now the input is going to be the output of the or the return of the get text method method so this. txt drink. get text and I think that's correct but once again I'm going to confirm that with a simple little output statement here and I'm just going to run that and see how it's working and so I need to make sure I put different numbers in these two text boxes here so I'm going to put $999 and I'm going to put $4.99 and I do get output that says that those are converting correctly it's kind of downhill from here I am going to need a double for the total price and that is going to be the food Price Plus the drink Price Right pretty straightforward I haven't dealt yet with the tip but I Outputting results to a JLabel can come back to that and and I can output my result now we've seen this before how do we change the text of a label so this is really the same thing the only difference now is that the what I'm outputting happens to be the result of some math but I've got um this. lbl result is the name of my label and I'm going to call the set text method this time and I'm going to set it to say total price and I'll put a dollar sign there I'm not really formatting this and so um we it's not going to look as pretty as I uh as we might want it to look but um but this should work and it should free us up from needing these um these print line statements so let's go ahead and run this version of the code and I'm just going to go with real simple numbers here I'm going to say five and two and it has output 7.0 so it does seem to be working the only bit that's Getting the status of a JCheckBox left is the tip and like I did before I'm going to deal with the tip kind of in separate steps even though I could combine them so I'm going to I'll just get that right here I'm going to create a Boolean variable and I'm going to call it add tip as in are we going to add a tip and to get a value for that I'm going to go to the the chk tip control which is my uh checkbox and it has a method called is selected and it returns a Boolean if the box is checked it returns true if the box is not checked it returned false there is no other possible outcome uh other than those two things so that should give me a true or false U that indicates whether or not I need to add a tip and now it's just an if statement so if add tip then uh the math here is nothing special I I intentionally made it pretty easy I'm just going to say um times equals in other words I'm going to multiply the total times 1.2 which is um 120% right so it's the full value of the total plus 20% for the tip um I don't think there's anything too fancy about that math and I believe that's going to get us um a correct calculation so on my test cases I didn't take the time to to write out test cases ahead of time uh but I'm going to put in um $2 and $8 which conveniently gives me $10 and now I'm going to add the tip which should be um $2 right 20% of 10 should be two so now the total is $12 so this is working pretty well I guess I don't really have to delete these print line statements but I really should if I'm writing a goey program I don't want things getting dumped into that terminal um depending on the environment it's running in it's not likely to cause a problem but it's sort of sloppy right so with those two things cleaned up this now is complete all I've done is I've uh created my event handler and I've done all of my math in that event handler and I have output the result to a label that's on my guey so this gives you a nice uh simple straightforward example of how you can make your program a little more realistic uh and More in line with the kind of things you're going to be asked to do on assignments and depending on what I've assigned to you in this uh in this semester you probably are ready to go off and do at least some of those assignments with just this knowledge we although we will add on a couple things uh before we wrap up with swing

# Transcript 5

creating a rich and userfriendly experience with a GUI takes more than just labels and text boxes swing like other guei Frameworks provides a wide variety of widgets for making an effective user interface in fact there are far more widgets than we have time to cover but I do want to show you a few of the more common widgets so you can use them in your own programs check boxes radio buttons and combo boxes each of these widgets has a different role within a guey so we'll look at them one at a time a checkbox is a widget that allows you to make a binary selection true false yes no onof and so on you can use a single checkbox like you often see when you have to agree to terms and conditions or you can use multiple check boxes you might use this to give the user options allowing them to to select as many as they'd like on a quiz this would be a multiple answer style question select all of the correct answers in swing checkboxes are implemented using the j checkbox class the J checkbox widget includes both the checkbox graphic and the text that goes next to it here's how to use it we'll just declare our J checkbox objects as Fields along with all our other widgets I'm using chk as a prefix and then the last name of the individual the user can select as always the most important thing about your names is being clear and descriptive don't forget that you have to add your J checkbox instance to the guey just like any other widget to see if the user has toggled on on a j checkbox we can use the is selected method we do need to check every J check box to make sure that we cover all of the options and notice that we're not using any else or else if statements that's because each check boox is independent of the others so we have to check them all obviously you'll end up putting whatever code you need inside each of these if blocks they're just empty here to keep the code clean for the sake of the video I just said that each J check box is independent of the others and a user can select as many or as few as they want right well sometimes we only want the user to pick one selection from all of the choices that's the purpose of radio buttons a radio button is actually the exact same thing as a checkbox but it's generally rendered with a circle and a DOT rather than a square and a check mark other than that the only difference is that radio buttons are placed in a group and the user is only allowed to select one choice from the group if they select another then the first choice is toggled off you could put a bunch of check marks in a group and theyd behave the same way and yes if you added radio buttons without putting them in a group they would behave like independent checkboxes and allow multiple selections but the subtle difference in their appearance lets the user know how they function so we should use them in the convention ways learning where the name radio buttons comes from might help you remember how they work when automakers first started putting radios into their cars they weren't digital they had dials and you had to turn them in order to tune to a different station and in order to remember your favorite stations they had physical buttons that actually moved the dial to a specific spot when pressed the button stayed down until you pressed another button and then your previous button popped back up radio buttons in AU GOI were named after those Old School radio buttons the swing class for a radio button is not surprisingly called j radio button when we implement it we're also going to use a class called button group it's not visible in the guey but it's necessary to make the radio buttons behave like radio buttons and only allow one selection the j radio buttton objects don't even need to be near each each other on the screen for the group to work but to make your guey clear to the user you should keep a group of radio buttons close together and you can have multiple button groups with j radio buttons in them and the user will be able to select one choice from each group here's how j radio buttons work declaring and instantiating our j radio button objects will be exactly the same as our J check boxes which of course is exactly the same as our other widgets I use the pref fix rdb for my radio buttons instead of chk like for checkboxes and you also see that I've declared a button group object with the prefix grp in order to make the radio buttons behave as radio buttons and only allow one selection we have to add all of the j radio button instances to the button group instance in much the same way we add them to the frame the only other place we see a difference between j radio button and J checkbox is when we test to see which option was selected since only one selection is possible I like to use if else if for the checks that way as soon as we find one where is selected returns true we can stop looking that's pretty minor but it is more efficient if we stop once we have a selection the last widget we'll look at here is called called a combo box because it works like a combination of a drop-down list and a text box it allows the user to choose from a selection of items much like a radio button but the user can also begin typing as the developer you can decide how the combo box behaves if the user starts to type a non-editable combo box is probably what you associate with the term drop-down list the user is only allowed to select from the items on the list so typing will basically search through the list and make it easier for the user to complete their selection but you can also make a combo box editable which means that the list of items is sort of like a bunch of suggestions the user can pick one of those options or they can type something that's not on the list this kind of editable combo box is almost like autocorrect the user can accept one of the suggestions or just keep typing and put in whatever they want so that gives us two very different uses for combo boxes but in either case a big advantage to a combo box is how little space it takes on the screen you could give the user a bunch of radio buttons and that would serve the same purpose as a non-editable dropdown but you'd be sacrificing a lot of real estate in your window to do it sometimes that's worth it either because you have plenty of space or because it's just really important to make all those choices obvious to the user like when they're agree to those terms and conditions that are so annoying there is another big advantage to combo boxes that's worth mentioning even though we'll only touch on it a little bit here sometimes the list of choices is pulled from some data source like a web API or a database at the time we're writing the code we might not know what those choices are or even how many of them there are so radio buttons wouldn't be practical it's much easier to just populate the combo box options from that data source when the program runs to give us all of this functionality swing has the J combo box class since combo boxes are all about flexibility J combo box has a feature that's probably new to you it's a generic class now Java generics are a fairly large topic and not really something I'm interested in at the moment so I'm just going to show you how to use it correctly for now if you declare a j combo box in the same way we've been declaring our other widgets you're going to see a compiler warning A J combo box is made up of items and those items can be lots of different things I might want a combo box of integers or doubles for example so I need to specify that data type in our case we're going to use strings so the Syntax for generics in Java is the angle brackets we put the type name in Brackets immediately after J combo box and that makes the compiler happy this doesn't seem like a big deal so why not just count that as our lesson in generics well for one thing I can only put a reference type in those angle brackets Not A Primitive type like int as the Beatles themselves once said you can't do that you have to use what's called a wrapper class like integer and that's only the start of the rabbit hole we can end up in so for now we're just going to specify our data type in those brackets and set aside generics until another time okay so now we can instantiate our J combo box and start adding items to it which works a lot like you'd probably expect oops there's that generic type syntax again but anyway we can just add items to the instance with the add item method this is a lot like a raise where you end up having to write a bunch of separate statements to populate all of your data but there is a shortcut and it actually uses arrays in fact I can pass an array into the Constructor call when I instantiate my J combo box and the Constructor will add all of those elements to the object's item list here's an example that uses two different arrays to populate a pair of J combo box objects whether you add the items individually or use an array to populate the J combo box you can then add that widget to your frame just like any other widget once we've got our J combo box up and running we need to be able to see what the user picked and here's another example of some flexibility offered by J combo box we have multiple ways to retrieve the user selection much like an array the collection of items in the combo box has index numbers starting at zero so one way to find out what the user selected is to retrieve that index number and now I can use that index number in my if statement or however I need to work with that selection you can also retrieve the item itself using the get selected item method as part of that J combo box flexibility it gets returned as an instance of the object class remember that class that every Java class extends from our lesson on inheritance so to use it as a string we'll downcast The Returned object to a string and another little peek at that rabbit hole we're trying to avoid but the syntax is pretty easy and once I've converted that into a string I can work with it just like any other string there are a lot of other widgets available in the swing framework but now you know how to use the most common and useful ones for General programs with these examples you can probably figure out how to use most of the others I encourage you to try them out and see how they look

# Transcript 6

arranging your widgets within a jframe can be pretty frustrating swing provides a number of layout managers to help we started learning swing by using a flow layout which puts the first widget in the top left corner and each additional widget gets placed to the right once it reaches the edge of the container it starts a new row so it works basically like Microsoft Word you're typing just goes across from left to right and when you get to the right Edge it bumps your new word down to the next line flow layout is easy to use but the results can be confusing for the user in this example the text boxes can get separated from the labels that tell the user what to input resizing the window demonstrates how the flow layout adjusts and wraps widgets to the next row and you might be able to dial in a width that kind of makes sense so I guess you could just set the size to that I could also set the preferred size for these different widgets to help keep things looking consistent and also control where those new rows are started it's not really ideal but it can work for simple goys and note that though I didn't show it here you can change the alignment of widgets within the flow layout by adding that as a parameter in your Constructor call one thing that can help is grouping widget wigets together swing provides a class called J panel that is basically a generic container that will hold widgets in this example I've added a j panel as one of my fields and instantiated it in the Constructor I've also set the background color so we can see the panel when we run the program the panel is going to hold widgets related to John next I'm going to add the label and text box to the J panel instead of to the frame and then I'm going to add the panel to the frame you can see the red background showing that the J panel is holding our label and text box and now when I resize the window the J panel is treated as a single widget that keeps them together when we get to the right edge of the frame I can use as many panels as I need so I'd probably want to put each label text box pair in separate panels so they always stay together but there's also life beyond flow layout a really straightforward layout manager is the grid layout which establishes a grid of cells that you can use to arrange your widgets when you set the layout you'll add the number of rows and columns to the grid layout Constructor call in this case I've made a grid with three rows and two columns when you add widgets to a grid layout they are put in from left to right and top to bottom so here's the result the grid will fill the entire container and the cells are going to be the same size by thinking about your guey as being in a grid you can often sketch out the basic layout on a sheet of paper or in a table in Microsoft Word and then write the code to produce that the other layout manager we'll look at here is called border layout this lets you position widgets in different regions of the container like the top or bottom left or right in this example I've created five different J panels and placed them around the frame in the positions indicated by the labels page start and Page end are at the top and the bottom of the screen line start and line end are the sides and Center is well the center to use this approach you'll specify border layout as the layout for the frame you'll specify the position of each widget when you add them to the frame using predefined constants actually enumerations that are available in the Border layout class I've added labels to the panels and now I'll add the panels to the frame border layout. page start puts the first panel up at the top for example there are different constants you can use for example you could specify Border layout. North for the top of the screen and east south and west for the other regions they behave a little differently than page start and Page end for example so you can experiment with the those directional references to see the difference there are a bunch of other layout managers in swing too many for us to cover here but we haven't yet seen the most useful trick for organizing the layout of widgets nesting often there isn't a layout manager that works perfectly for what you're trying to accomplish like in this example this colorcoded version helps visualize the way I've laid out these components the pink color is the frame itself and the label is added directly to that the teal is a panel and I've applied a 3x2 grid layout to that panel before adding the checkboxes the orange color represents a separate panel with the button and it's really just to demonstrate an additional panel so that gives us three elements the label the checkbox panel and the button panel and I've added all of those to the frame within a flow layout using panels with their own layout managers gives you a ton of flexibility now I often use a restaurant analogy when I talk about decoupling code from a user interface the guey is front of house like the servers and bartenders and the classes we write are the back of house like the cooks there's a similar analogy related to guey design itself and it's based on one of my golden rules of coding programmers are terrible at UI design you can always tell when a user interface was created by a programmer because it sucks okay that might not always be true but it brings up this idea of using Specialists the cooks don't have to have great people skills because they can focus on cooking the servers don't have to know how to cook and they can focus on good customer service it makes both of them better at their jobs and most importantly it makes for a better experience for the customers in soft Ware development it's often better if a coder can just focus on writing code and then we can get someone with a great design background to develop the guey to help with this there are tools that allow someone to lay out a guey and the tool will generate the swing code to produce that guey that makes it possible for a non-programmer to lay out a good user interface I mentioned this because I want to acknowledge that those tools exist and can make our lives a lot easier but in this course we're learning how to write code so we won't be using tools to create our guey for us on my assignments you need to write the guei code the old fashion way but I encourage you to explore some of those tools outside of class and as is almost always the case with these videos there's a lot more out there that I haven't had time to cover but this should give you enough flexibility to create the goys you need for your own programs CIS150AB