This is the Brandi's Bagel House example, and you can see it's running right here we have some radio buttons for the coffee and for the bagel, and then we have some checkboxes for the toppings, and we have a calculate button.

So let's talk a little bit about how this works. Interfaces are pretty important. Most desktop applications have some sort of graphical user interface, and because of that, there are a lot of different graphical user interface libraries that you might have to learn and work with in developing software. And there's all basically the same and they all have differences. So some of the similarities are of course the kinds of components that you see because graphical user interfaces have similar components, and also there can sometimes be a fairly deep inheritance hierarchy and we see that, in Java, that you have to navigate in order to figure out how the different components work and fit together. Deep inheritance hierarchies can be difficult to learn.

And then of course there's the event model, whereby you actually get the messages about what the user has done, what they've clicked on, etc., and how you handle that.

So all of these concepts are going to be similar across all different development environments, but how they're handled will be different.

So, one of the things that's hard to follow is the flow of control, particularly in an object-oriented system. So here we create an order calculator gui. And here's the order calculator gui. So this includes the different panels that you see up here which are separate classes that are, you know, subclassed from classes that are part of Java's framework. And then a panel for the buttons, the buttons, and then a constant. So let's see what happens when this is created -- there -- so you should have access to this code, it's in your book, you have access to it from the website, but you can always pause this and look it over.

So we create a greeting panel, a bagel panel, a topping panel, and a coffee panel. So each of these, we'll look at those. And then the button panel is just part of this order calculator, so then it's got its own button panel. And these are then all just put into the layout, packed, and it's set to visible. And then here's the building the button panel, which just puts those buttons into the panel and creates listeners for them. And then here are the listeners which are a private class. So because they're within the order calculator gui, it means they have access to all of these private fields.

So that's a lesson to learn from this particular design. Is that one way that you can handle where a listener needs to be able to access information that's contained in another class, another piece of the interface -- if you make the listener a subclass of a class that has access to that piece, then the listener can access that piece.

So these listeners can talk to the bagel panel, the toppings panel, and the coffee panel to get the information that it needs in order to do the work of, in this case, the button which indicates that the user is finishd with the order, you know, it's going to calculate the cost of the total order.

So then for example if we look at one of the other panels, this is the bagel panel, and remember that was two radio buttons over on the left, and so its got its private fields, and if you wanted to listen and do something as a result of anything happening within this panel, you could have the listener as a private class within the bagel panel class.

But as it is, it has this getBagelCost message, which the listener here, CalcButtonListener, can, it can send that message to the bagel panel, and of course this gets the information from itself, and then can send back whatever information is needed.

Now in designing modifications to this program, one thing that isn't done here is the coffee panel, for example, doesn't get any references to any other components of the interface in its constructor. But it could. So if the coffee panel needed to know another panel in order to send a message to that panel, you could simply make it a field of the coffee panel.

So say for example we wanted the coffee panel to know its sort of parent here, the order calculator gui. You could just pass that in as a reference to the coffee panel, and then if the coffee panel needed to talk to the order calculator it could.

There are different ways to design interfaces. In older editions of the text, the author actually put everything all into one class. All the interface components, that is -- all into one class. So everything pretty much knew every other thing. Now that's a particular design decision. There are cases where that would be a poor decision, for information hiding and security reasons. But in other cases, that's a much simpler way to implement the interface. And you're welcome to redesign this, when you do assignment 3, so that everything is contained within one class, and so that everything knows, or can send messages to, everything else.

But then another way to handle objects knowing each other, or having a reference to each other so that they can send messages, is in the constructors, giving them references to the objects that they need to know in order to send messages and accomplish the work they need to accomplish.

I hope this is helpful in understanding how this interface works and in figuring out how to modify as you're supposed to in assignment 3. If you have specific questions about this interface or about any of the components -- just general questions about the components I've asked you to include in assignment 3, do feel free to email me those questions.