In this video, we are going to continue the implementation of the view class by adding the buttons to our simple calculator example.

0:00-0:06

Recall that in the first video, we created an empty JFrame to serve as the top level window for the calculator application.

0:06-0:13

We now want to add the number and operator buttons to the calculator.

To do this, we will first create a component to act as a container for the buttons. The reason for creating a container for the buttons is to make it easier to arrange the buttons in a grid-like fashion separated from the other controls of the calculator. We will use a JPanel instance to act as the button container.

The individual buttons are all instances of the JButton class. A JButton is a representation of a push button.

0:13-0:51

This UML diagram shows the relationship between the View and its JPanel, and the JPanel and its JButtons. The View has one JPanel to contain the buttons, and the JPanel has 16 JButtons that represent the various buttons on the calculator.

0:51-1:15

This UML diagram shows the inheritance relationship between the various classes that we are using.

The JPanel is a child class of JComponent, and a JComponent is a child class of Container.

A Container is a Component that can contain zero or more other Components. It provides public mutator and accessor methods to add and get the Components that are in the Container. The add method is the method that we will be using to add the JPanel to the View, and add the JButtons to the JPanel.

We can create the JPanel by using the default constructor. There are other JPanel constructors that we could use, which you can look up in the JPanel API, but the default constructor is good enough for our purposes.

2:05-2:22

ECLIPSE

In the View constructor, we can create the JPanel. Once we have a JPanel, we can add the panel to the View by using the Container method add. Remember that the keyword "this" is a reference to the View that is being constructed, so to add the panel to this view we write this.add(buttons)

2:22-2:57

To create a JButton, we can use the JButton constructor that has a string as its parameter. The string is the label that will appear on the button. It is possible to use images for the button labels, but for now we will use only a text label.

ECLIPSE

Our calculator has 16 buttons. We could create the 16 buttons individually, but it would be better style to use a loop. To use a loop, we need to put the button labels into a list or array. Let's create an array of button labels with the order of the labels matching the order that we want the buttons to appear on the calculator. Next, let's write a loop that makes each button and adds it to the panel. When we run the main app, everything looks good. But if we resize the calculator, the buttons don't remain arranged in a grid, and they don't all have the same size. To maintain the grid-like layout of the buttons, we need to use a layout manager.

3:15-5:31

A layout manager is an object that is responsible for determining the size and position of components in a container. There are many different types of layout managers each designed for a particular purpose. You can find a visual guide to the Swing layout managers by following the link shown here. GridLayout is an appropriate layout manager for the calculator buttons.

5:31-6:00

To create a GridLayout, we need to specify the number of rows and columns in the grid. We can also specify the gap between the columns and the gap between the rows. HGAP and VGAP are the horizontal gap and vertical gap, respectively. In our calculator, we have 4 rows and 4 columns of buttons.

ECLIPSE

Let's create a GridLayout for the panel using the code we just saw. Once we have the GridLayout, we can set the layout manager of the panel. Now when we run the Main application we can resize the calculator and the buttons remain arranged in a grid.

That brings us to the end of this video. In the next video, we'll see how to create the text display for the calculator.