DIY Sliders

Objective

- After class today you should be able:
- Implement your own slider using a CustomPainter.
- Finish project 2 (due this week).

Basic architecture for versions of the project.

- State
  - What do you need to know in order to…
  - Draw things on the screen?
  - Return values for your widget?
- Listener Class
  - Update the state based on what's happening.
  - onPointerDown
  - onPointerMove
  - onPointerUp
  - Don't forget to call setState(() {…}) to force a redraw
- CustomPainter
  - Draw things on the screen (aka canvas) based on the state.
  - Circles, lines etc.
- API
  - How does a programmer use your slider?
  - Initialization
  - Values.

First step:

- Make a circle move to where the finger goes down.
- State:
  - Where the finger when down.
- Listener
  - onPointerDown: update state with finger down location.
  - (and setState eventually)
- CustomPainter
  - Put the circle where the finger is.
- **That’s about it for project 1 due at the end of the week.**

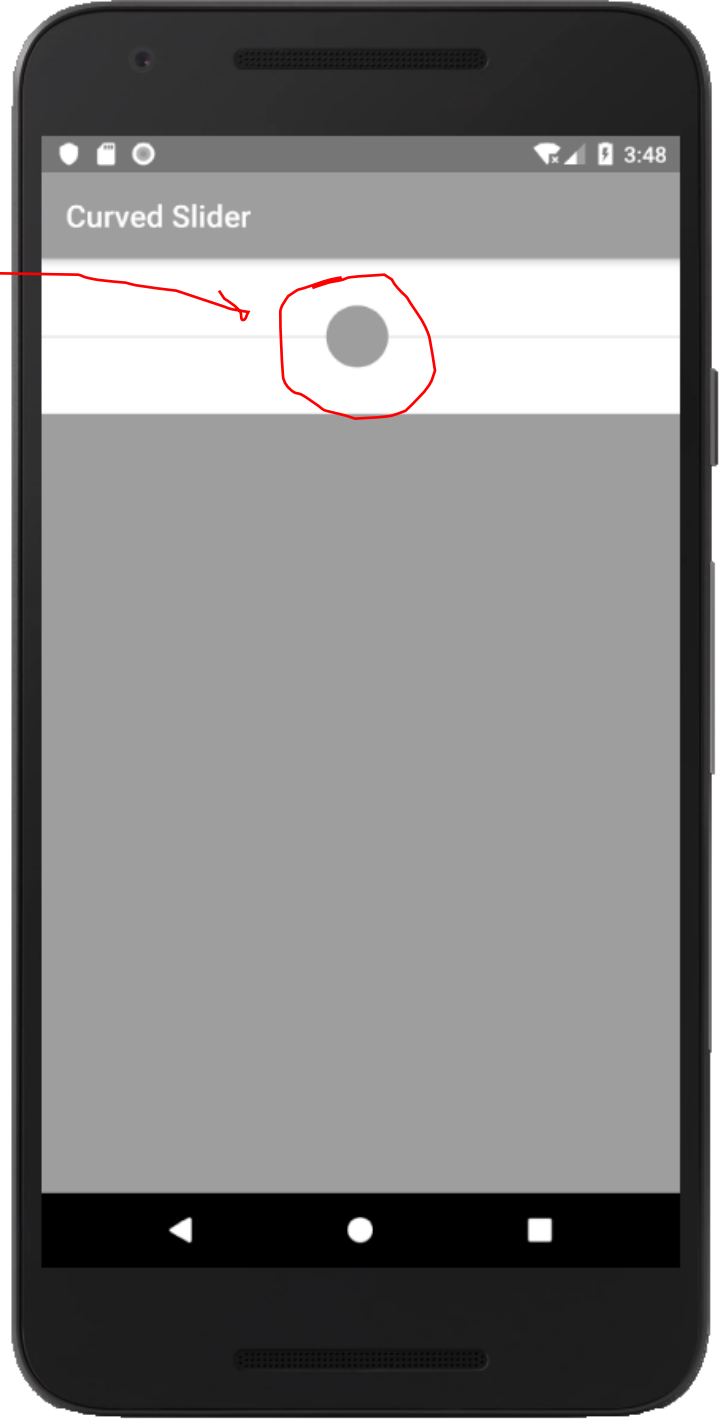
Next step:

- Circle follows the finger.
- State
  - Same as before: where is the finger.
- Listener
  - onPointerDown: same as before.
  - onPointerMove: update state with finger location.
- CustomPainter
  - Same as before.
- So what? The point here is that the architecture decomposed the problem for us and made it easier to think about.

Now for sliders.

- What should the behavior of a slider be?
- How should it look?
- What should we have in the API?
  - For initialization?
  - Return values?

- In the beginning, let's just do a simple one thumb vanilla slider.



Vanilla slider version 0.1

- State
  - Where the thumb is located.
  - Value (0 on the left, 100 on the right). Calculated from thumb position. Need to think carefully about where to update this. Not in CustomPainter. Probably in a method called from Listener
- Listener
  - onPointerDown: did they hit the thumb? Close enough?
  - onPointerMove: what to do if they slipped off the thumb?
  - Update thumb position.
- CustomPainter
  - Update thumb position.
  - Draw the line on which the thumb travels. Drawing a diagram helps get the positions down.
- API
  - A method for getting the slider value.
  - No events yet.
- Pretty close to done for project 2 (Just need a way to display the value as it slides along)

Vanilla slider version 0.1.

- How will someone know when the value changes?
- We could expose a getSliderValue method.
- So they just have to call that repeatedly
  - This is called “polling”--*generally a bad idea* for a widget to force polling.
- The appearance and functionality are also brittle.
  - What if someone wants to use the range 0 to 200 instead?
  - Smaller thumb? Only ½ the screen width?
- We’ll deal with getting the value first.

Vanilla slider version 1.0

- State:
  - No changes.
- Listener
  - No changes
- CustomPainter
  - No changes
- API
  - Expose an interface, use the Flutter convention for this. A listener must be passed to the constructor for the slider.

Exposing an interface

- Getting the right API is critical.
  - Most of the time, for vanilla widgets, just follow the convention.
  - For other situations, be thoughtful, adapt as needed.
- In flutter, the convention is to pass a callback in the constructor.

```
final double value;
final ValueChanged<double> onChanged;
…
MySlider ({
  @required this.value,
  @required this.onChanged,
  this.xPosition,
  this.thumbColor,
  this.thumbInactiveColor,
  this.thumbActiveColor,
  this.radius
});
```

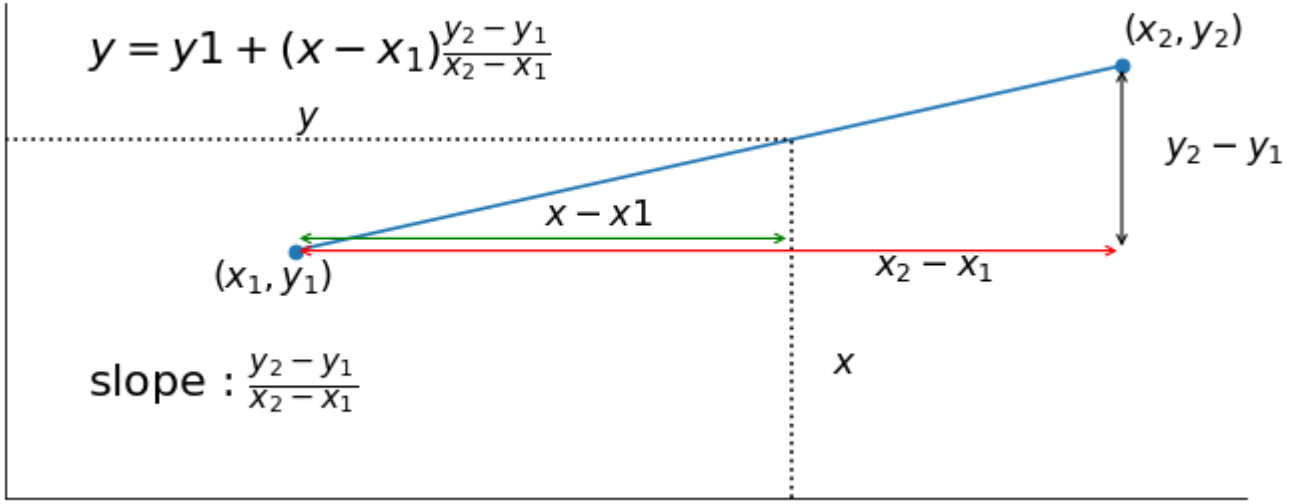
Vanilla Slider version 1.0 wrapup.

- This is the version due 1/24/20
- Slider constructor takes a callback as a parameter, stores the callback and calls the callback when the value changes.
- Slider calls the callback with the new value as a parameter.
- The callback updates some state inside setState.
- Everything that uses that state is redrawn.

- Slider constructor also takes a left and right range.

What about linear interpolation?

- y is a component of the color. x is just x.



Linear interpolation — Tutorials on imaging, computing and mathematics (matthew-brett.github.io)

Exposing an interface—Android Java version

- Inside the widget class, declare an interface.
  - Let's have just 1 method: onValueChanged.
  - When the value changes, the seekBar calls this method in the listener.
  - The listener does whatever it wants.
- In the widget, make a "register to listen" method.
- In the listener,
  - implement the interface
  - Instantiate a widget and then register to listen to that widget.

Reflecting on version 1.0

- Pretty complete for next project.
  - Major ideas for a widget are in place, except generalization (see below)
- We won't go through all the project at this level of detail, but this should get you started.
- For project 2,
  - generalize to *n* thumbs.
  - Generalize to a (reasonable) range of values.
    - Big on right, small on the left.
  - Generalize to different screen widths.
    - We'll talk about this one in some detail.
- For project 3, curved slider (but for a specific curve).

Generalizing the slider: thumbs

- State
  - Keep track of multiple thumbs. I like Lists.
- onTouchEvent
  - Which thumb did they hit, if any?
  - What happens if they slide a thumb past another thumb?
- onDraw
  - Draw all the thumbs.
- API
  - Number of thumbs as a parameter.
  - Return a list of thumb values rather than a single value
    - Or some other way of doing this which you dream up.

Generalizing appearance: padding, thumb diameter etc. (not required for project)

- State:
  - All kinds of parameters about size need to be kept.
- onTouchEvent.
  - Nothing.
- onDraw
  - Nothing.
- API
  - Add the parameters.
  - Maybe include a default constructor with default values.

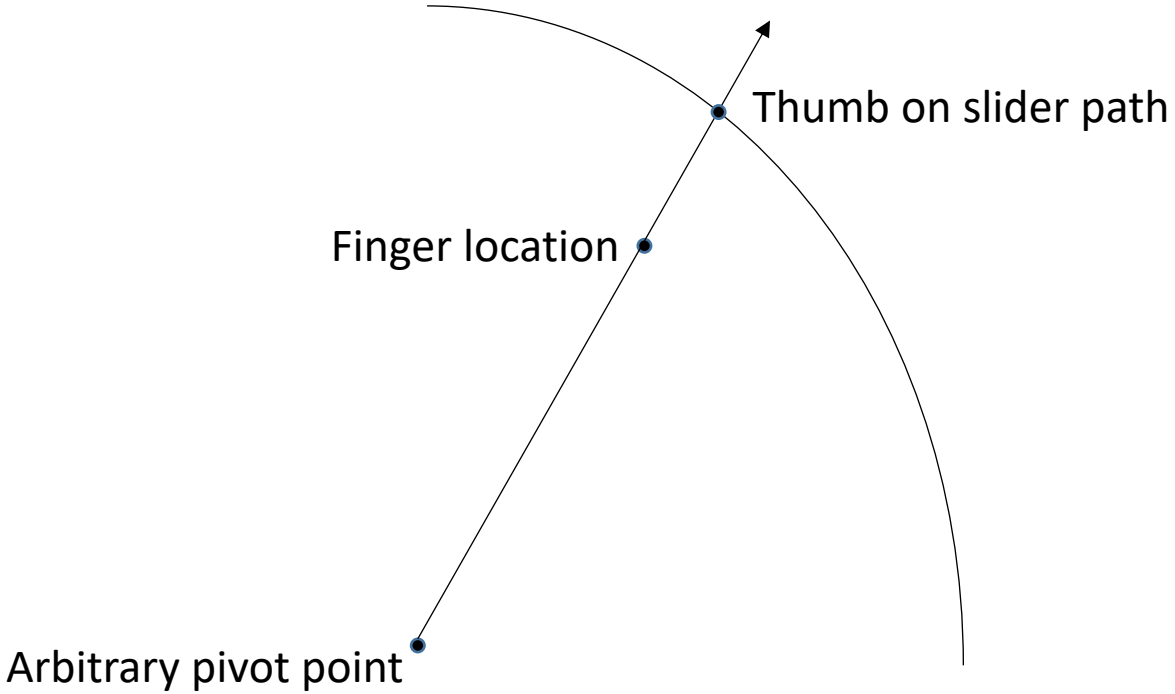
Curved sliders

- State
  - Interpolate the value of the slider differently. Was just linear interpolation on the x value of the thumb. You can decide how to do this.
- CustomPainter
  - Draw a curve instead of a line. (A bit of a pain to specify, especially since there isn't a "draw part of an ellipse using this implicit representation" method.)
  - Don't forget to mind screen space and ellipse space issues.
- Listener
  - In OnPointerMove, calculate the new thumb position differently. (not difficult to code up once you know what to code up).
  - Initial finger press on the thumb is about the same.
- API
  - No change.

Drawing an arc (in flutter)

- Slightly painful, but not that painful. See
- https://api.flutter.dev/flutter/dart-ui/Canvas/drawArc.html

Where is the thumb on the arc?



~~This can be a ray tracing problem, see~~ ~~https://www.dropbox.com/s/x0w5t5w0sjerg5p/Curved%20slider.pdf?dl=0~~

Notes from our multi-day discussion in class

- See https://www.dropbox.com/s/t0vn9esjli80tbz/Curved%20slider%20v2.pdf?dl=0

Programming tips

- Have a piece of paper on which you sketch this out.
- Don’t forget to normalize your direction vector!
- Make sure you can see what your code is doing. If you can’t see it, you can’t debug it.
  - I draw stuff on the screen for debugging.
  - Then turn it off when I’m done.
- Check you equations in your code.
- I found this ellipse viewer handy
  - https://www.desmos.com/calculator/huko7itjso