React Native

PanResponder

Responding to gestures

- There are two levels of monitoring gestures
  - Low level: gesture responder system
  - High level: panresponder system

gesture responder system

- The gesture responder system manages the lifecycle of gestures in your app.
- A touch can go through several phases as the app determines what the user's intention is.
  - For example, the app needs to determine if the touch is scrolling, sliding on a widget, or tapping.
  - This can even change during the duration of a touch.
  - There can also be multiple simultaneous touches.
- The touch responder system allows components to negotiate these touch interactions without any additional knowledge about their parent or child components.

PanResponder

- PanResponder reconciles several touches into a single gesture.
- It makes single-touch gestures resilient to extra touches.
- It can be used to recognize simple multi-touch gestures.

PanResponder

- By default, PanResponder holds an InteractionManager handle to block long-running JS events from interrupting active gestures.
- It provides a predictable wrapper of the responder handlers provided by the gesture responder system.
- For each handler, it provides a new gestureState object alongside the native event object:

```
onPanResponderMove: (event, gestureState) => {}
```

PanResponder

- A *native event* is a synthetic touch event supplied by the basic response system with the following form:
- **`nativeEvent`**
  - **`changedTouches`** - Array of all touch events that have changed since the last event
  - **`identifier`** - The ID of the touch
  - **`locationX`** - The X position of the touch, relative to the element
  - **`locationY`** - The Y position of the touch, relative to the element
  - **`pageX`** - The X position of the touch, relative to the root element
  - **`pageY`** - The Y position of the touch, relative to the root element
  - **`target`** - The node id of the element receiving the touch event
  - **`timestamp`** - A time identifier for the touch, useful for velocity calculation
  - **`touches`** - Array of all current touches on the screen

PanResponder

- A gestureState object is a PanResponder object and has the following props:
  - **`stateID`** - ID of the gestureState- persisted as long as there at least one touch on screen
  - **`moveX`** - the latest screen coordinates of the recently-moved touch
  - **`moveY`** - the latest screen coordinates of the recently-moved touch
  - **`x0`** - the screen coordinates of the responder grant
  - **`y0`** - the screen coordinates of the responder grant
  - **`dx`** - accumulated distance of the gesture since the touch started
  - **`dy`** - accumulated distance of the gesture since the touch started
  - **`vx`** - current velocity of the gesture
  - **`vy`** - current velocity of the gesture
  - **`numberActiveTouches`** - Number of touches currently on screen

PanResponder

- Basic usage:
  - Create an instance of a PanResponder object by calling PanResponder.create
  - for each method in the PanResponder object either
    - Link the callback method to methods that you have defined in your code.
  - or
    - Provide bodies for the callback methods
- The next few slides show how to provide bodies for the callback functions
- The following example shows how to provide links to methods that you have defined separately.

Basic usage

```
class ExampleComponent extends Component {
    constructor(props) {
      super(props)
      this._panResponder = PanResponder.create({
        // Ask to be the responder:
        onStartShouldSetPanResponder: (evt, gestureState) => true,
        // start capturing gestures
        onStartShouldSetPanResponderCapture: (evt, gestureState) => true,
        // do respond to gestures:
        onMoveShouldSetPanResponder: (evt, gestureState) => true,
        // do capture gestures
        onMoveShouldSetPanResponderCapture: (evt, gestureState) => true,
```

Must create a panResponder object

Basic usage

```
onPanResponderGrant: (evt, gestureState) => {
  // This method is called when the gesture has starts.
  //  you should show visual feedback so the user knows what is happening!

  // gestureState.dx, and gestureState.dy will be set to zero now
},
onPanResponderMove: (evt, gestureState) => {
  // this method is called when the user moves their finger
  // The most recent move distance is gestureState.moveX and gestureState.moveY

  // The accumulated gesture distance since becoming responder is
  // gestureState.dx and gestureState.dy
},
```

Basic usage

```
onPanResponderTerminationRequest: (evt, gestureState) => true,
    // the PanResponder has gotten a termination request.  Should it honor it?
    // By default you should say yes

onPanResponderRelease: (evt, gestureState) => {
  // The user has released all touches while this view is the
  // responder. This typically means a gesture has succeeded
  // If you have highlighted a component, you should unhighlight it here
},
onPanResponderTerminate: (evt, gestureState) => {
  // Another component has become the responder, so this gesture
  // has been cancelled
},
```

Basic usage

```
    onShouldBlockNativeResponder: (evt, gestureState) => {
      // Returns whether this component should block native components from becoming the JS
      // responder. Returns true by default. Is currently only supported on android.
      return true;
    },
  });
}

render() {
  return (
    <View {...this._panResponder.panHandlers} />
  );
}
}
```

This associates the panHandlers with this component

Ellipsis are a shortcut; React Native replaces this statement with all the items in the PanResponder object. panResponder is a variable that contains an instance of the PanResponder object

Example (defining PanResponder methods separately)

```
import React, { Component } from 'react';
import {
  AppRegistry,
  PanResponder,
  StyleSheet,
  View,
  processColor,
} from 'react-native';


var CIRCLE_SIZE = 80;
```

Import PanResponder

Constant for the circle size

Example

Exported class

```
export default class PanResponderExample extends Component {

  constructor(props) {
     super(props);
    this._panResponder = {};
    this._previousLeft = 0;
    this._previousTop = 0;
    this._circleStyles = {};
    this.circle = null;
    }
```

Local variables and local functions.
Function bodies implemented later
Creating them here makes them class variables.
We will initialize variables in **`componentWillMount`**

```
componentWillMount() {
  this._panResponder = PanResponder.create({
    onStartShouldSetPanResponder: this._handleStartShouldSetPanResponder,
    onMoveShouldSetPanResponder: this._handleMoveShouldSetPanResponder,
    onPanResponderGrant: this._handlePanResponderGrant,
    onPanResponderMove: this._handlePanResponderMove,
    onPanResponderRelease: this._handlePanResponderEnd,
    onPanResponderTerminate: this._handlePanResponderEnd,
  });
  this._previousLeft = 20;
  this._previousTop = 84;
  this._circleStyles = {
    style: {
      left: this._previousLeft,
      top: this._previousTop,
      backgroundColor: 'green',
    }
  };
}
```

This creates the **`PanResponder`** object

These are the functions that will be called by the PanResponder when a touch moves on the screen.
They are defined later in the code.

Initialize circle position

Object that contains the style of the circle.
We will modify this and use it to change the properties of the view that is the circle.

```
_highlight = () => {
  this._circleStyles.style.backgroundColor = 'blue';
  this._updateNativeStyles();
}
```

This function changes the circle backgroundColor when it is touched

This function changes the circle backgroundColor back to default when circle is untouched.

```
_unHighlight = () => {
  this._circleStyles.style.backgroundColor = 'green';
  this._updateNativeStyles();
}
```

Function that is called to reset the style of the circle

```
_updateNativeStyles() {
  this.circle && this.circle.setNativeProps(this._circleStyles);
}
```

See next slide for an explanation of setNativeProps

setNativeProps

- It is sometimes necessary to make changes directly to a component without using state/props to trigger a re-render of the entire screen.
- setNativeProps is the React Native equivalent to setting properties directly on a component.
- Use setNativeProps when frequent re-rendering creates a performance bottleneck
- Direct manipulation will not be a tool that you reach for frequently;
  - only use it for creating continuous animations to avoid the overhead of rendering the component hierarchy and reconciling many views.
  - setNativeProps is imperative and stores state in the native layer (DOM, UIView, etc.) and not within your React components,
  - this makes your code more difficult to reason about.

```
componentDidMount() {
  this._updateNativeStyles();
}

  render() {
  return (
    <View style={styles.container}>
      <View style={styles.circle}
        ref={(circle) => {
          this.circle = circle;
        }}
        { ...this._panResponder.panHandlers }
      />
    </View>
  );
}
```

Set up default styles

Create the circle. This is done via styles. See https://codedaily.io/tutorials/22/The-Shapes-of-React-Native

Install the panHandlers for the PanResponder

Ellipsis are a shortcut; replaced with all the panhandler object variables. **`panResponder`** is a variable of type **`PanResponder`**

This is a callback function. When the component is rendered, it calls this function and passes a pointer to itself. This pointer is assigned to the local variable this.circle. This enables us to manipulate the circle. See https://reactnatve.wordpress.com/2016/05/24/refs-to-components/

```
_handleStartShouldSetPanResponder() {
  return true;
}


 _handleMoveShouldSetPanResponder() {
  return true;
}


_handlePanResponderGrant = (e, gestureState) => {
  this._highlight();
}

  _handlePanResponderMove = (e, gestureState) => {
  this._circleStyles.style.left = this._previousLeft + gestureState.dx;
  this._circleStyles.style.top = this._previousTop + gestureState.dy;
  this._updateNativeStyles();
}
```

When the circle is touched, highlight it (turn blue)

When the circle is moved, move it's position to match the position of the touch

```
  _handlePanResponderEnd = (e, gestureState) => {
    this._unHighlight();
    this._previousLeft += gestureState.dx;
    this._previousTop += gestureState.dy;
  }


} // end of the default class
```

When the touch is no longer in the circle, put in the final position and change the highlight back to green.

```
var styles = StyleSheet.create({
  circle: {
    width: CIRCLE_SIZE,
    height: CIRCLE_SIZE,
    borderRadius: CIRCLE_SIZE / 2
    position: 'absolute',
    left: 0,
    top: 0,
  },
  container: {
    flex: 1,
    paddingTop: 64,
  },
});
```

The circle is actually just a View component. By restricting the width, height, and boarderRadius we make it appear to be a circle. We could color it here, but we color it via the this._circleStyles local variable.

PanResponder for multiple variables

- To make two objects respond *the same* to touches
  - Add a new View with appropriate style
  - In the PanResponder methods, also modify the style of the second component
  - You do not need a second PanResponder object.
- To make two objects respond *independently*
  - Add a new View with appropriate style
  - Create a second PanResponder object.
  - Add new methods for the second PanResponder object.

PanResponder for multiple variables

- To make other components respond to gestures
  - Just need to create a pointer to the component as we did with the ref callback method for the circle.
  - Then create a PanResponder object and methods for the new component.