State Management

Stateful Widgets

Jumail Bin Taliba
School of Computing, UTM
April 2020

Outline

- What is state
- What is state management
- Why state management
- How to manage state
- Stateful widgets
- Demo

What are states?

- States are everything that exist in memory when the app is running, e.g., app's assets, all the variables that the app keeps, animation states, texture, etc.
- In Flutter, states can be:
  - Local (wrapped in a widget)
  - App-wide (states shared across different parts of the app code)

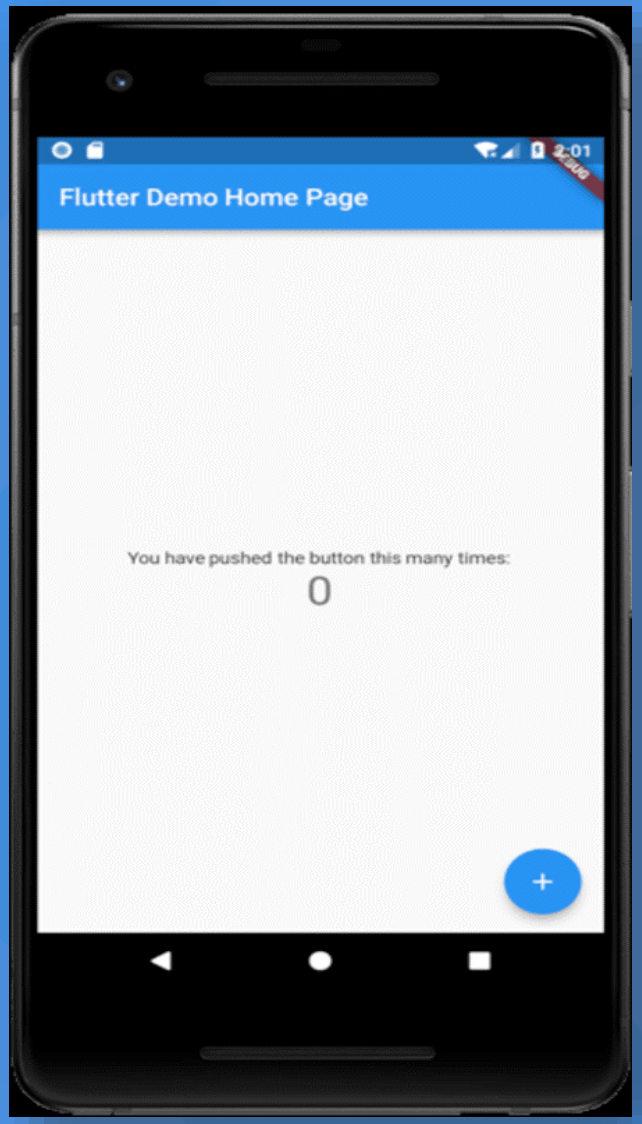
Why state management?

- Application states change over time:
  - user interactions
  - backend interactions
- State management:
  - To make sure the App's UI reflects to the state changes.

Why state management?

Example: Flutter counter app

- *when the user taps on the + button, this action will change the state of the counter (i.e. the counter increases).*
- *The state change occurs in memory*
- *However, we also need to reflect the state change to the UI*



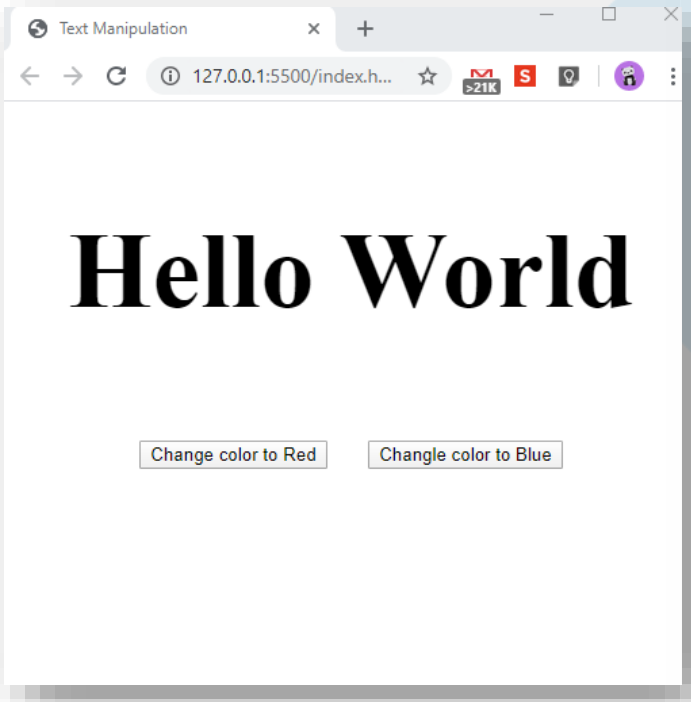
State management approaches

- Two main approaches :
  - Imperative
  - Declarative

Imperative state management

- This is the approach you are most familiar with.
- Examples:
  - Windows Programming
  - Web Programming
  - Android and iOS Programming.
- Need to consider how to change UI ‘manually’
- UI components are mutable
- Key principles:
  - determine the UI component that needs update,
  - and invoke mutations on it.

Example case: Changing text color on a web page



index.html > html

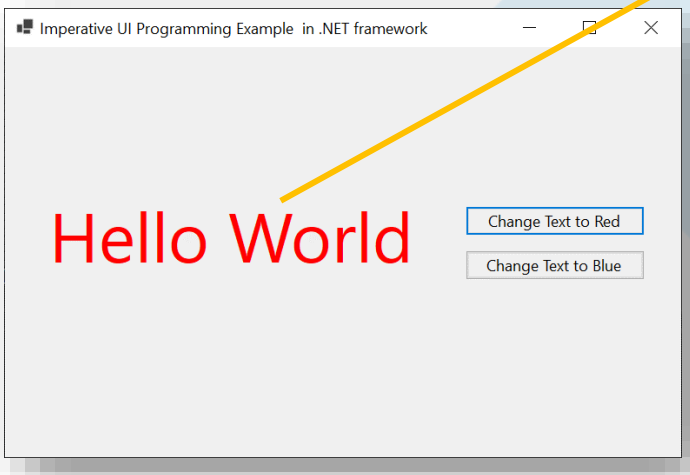
```
1   <!DOCTYPE html>
2   <html lang="en">
3
4   <head>
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Text Manipulation</title>
7       <script src="script.js"></script>
8   </head>
9
10  <body>
11      <div style="text-align: center;">
12          <p id="hello" style="font-size: 5em; font-weight: bold;">Hello World</p>
13          <p>
14              <button style="margin-right: 2em;" onclick="changeTextColor('red')">
15                  Change color to Red
16              </button>
17              <button onclick="changeTextColor('blue')">
18                  Changle color to Blue
19              </button>
20          </p>
21      </div>
22  </body>
23
24  </html>
```

- To update the text color, our code needs do it explicitly.
- First, retrieve the node (from DOM) that holds the text
- Then, mutate (or change) the desired attributes

JS script.js > ...

```
1   function changeTextColor(color) {
2       var node = document.getElementById('hello');
3       node.style.color = color;
4   }
5
```

Example case: Changing text color on a desktop window

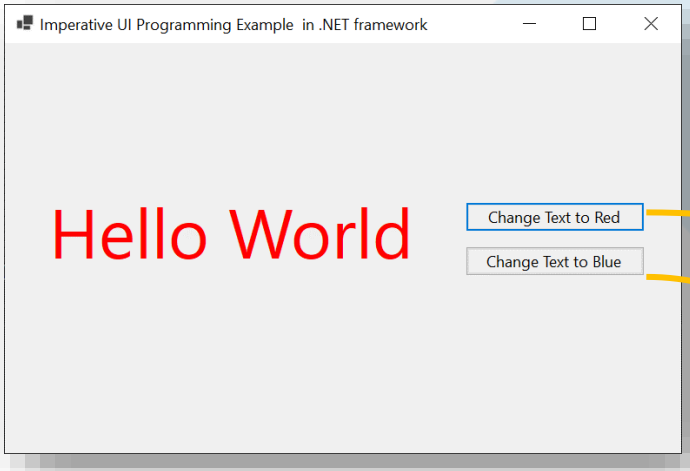


design view

Properties

txtDisplay System.Windows.Forms.Label

(DataBindings)	
(Name)	**txtDisplay**
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
AutoEllipsis	False
AutoSize	**True**
BackColor	Control
BorderStyle	None
CausesValidation	True
ContextMenuStrip	
Cursor	Default
Dock	None
Enabled	True
FlatStyle	Standard
Font	**Segoe UI, 40pt**
ForeColor	ControlText
GenerateMember	True
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
LiveSetting	Off



frmMain.cs

```
HelloWorld                                   HelloWorld.formMain
10
11      namespace HelloWorld
12      {
            3 references
13          public partial class formMain : Form
14          {
                1 reference
15              public formMain()
16              {
17                  InitializeComponent();
18              }
19
                1 reference
20              private void btnRed_Click(object sender, EventArgs e)
21              {
22                  txtDisplay.ForeColor = Color.Red;
23              }
24
                1 reference
25              private void btnBlue_Click(object sender, EventArgs e)
26              {
27                  txtDisplay.ForeColor = Color.Blue;
28              }
29          }
30      }
```

Declarative state management

- User code focuses on what UI to achieve.
- The framework will handle how to achieve that.
- Used by Reactive UI frameworks such as React.js, Vue.js, Flutter.

Declarative state management in Flutter

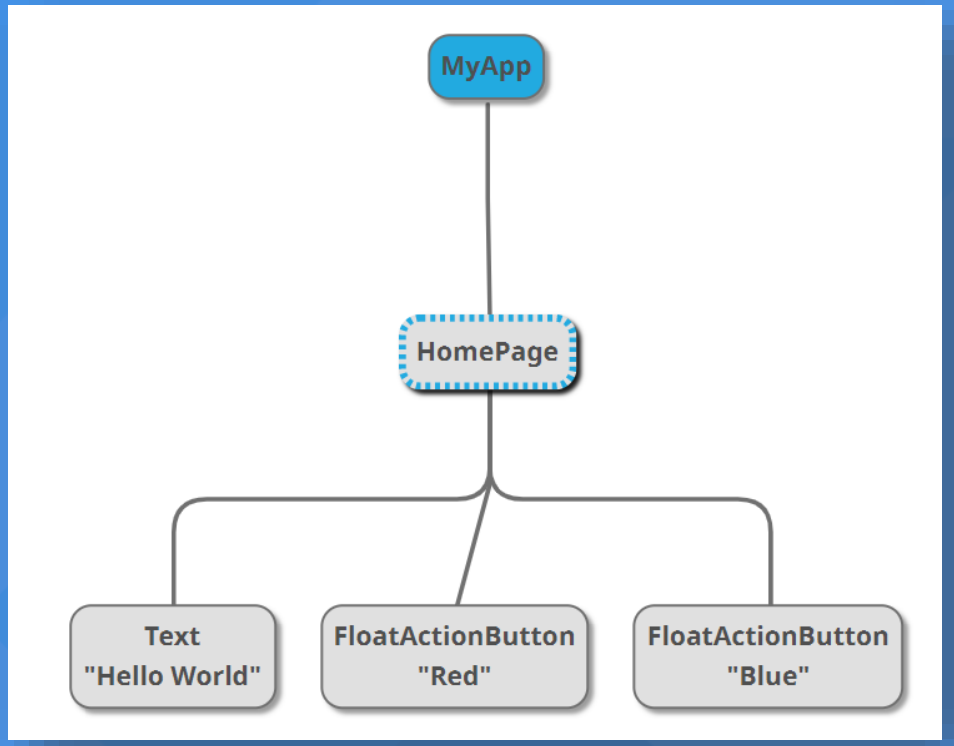
- Flutter uses declarative approach
- In Flutter, widgets are immutable
- So, how does Flutter make UI able to change?
  - It rebuilds new widgets (and replaces the existing ones)

Several approaches to manage states in Flutter

- Stateful Widgets
- Inherited Widgets
- Stream Builder
- Scoped Model
- Provider
- BLoC
- Redux, Mobx, etc.

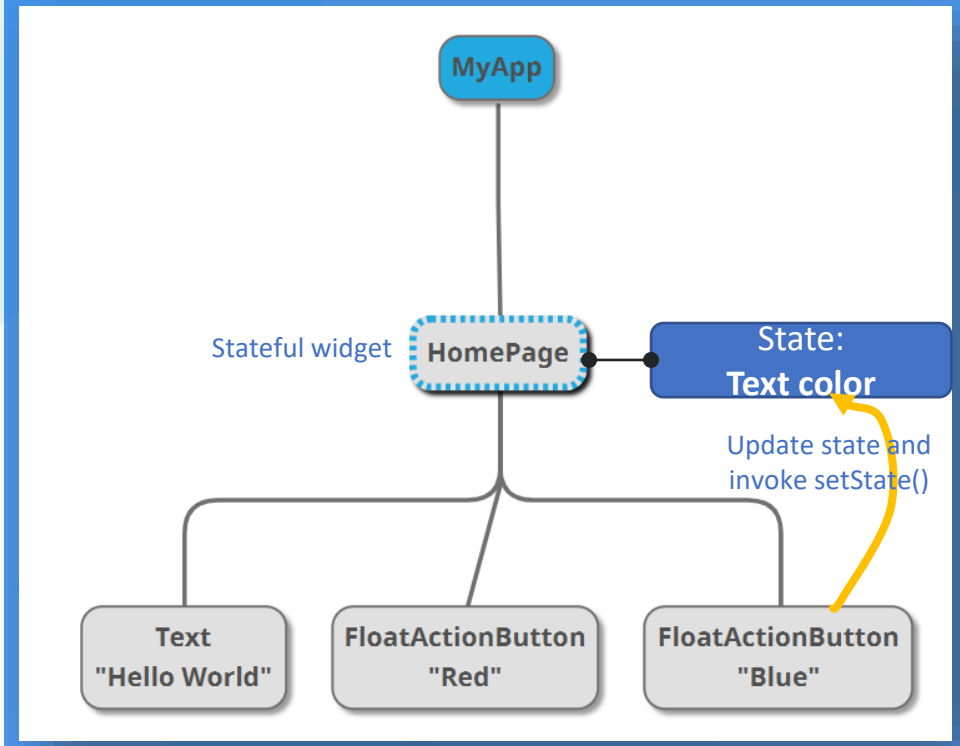
How do widgets get rebuilt?

when the parent gets rebuilt, it also rebuilds the children



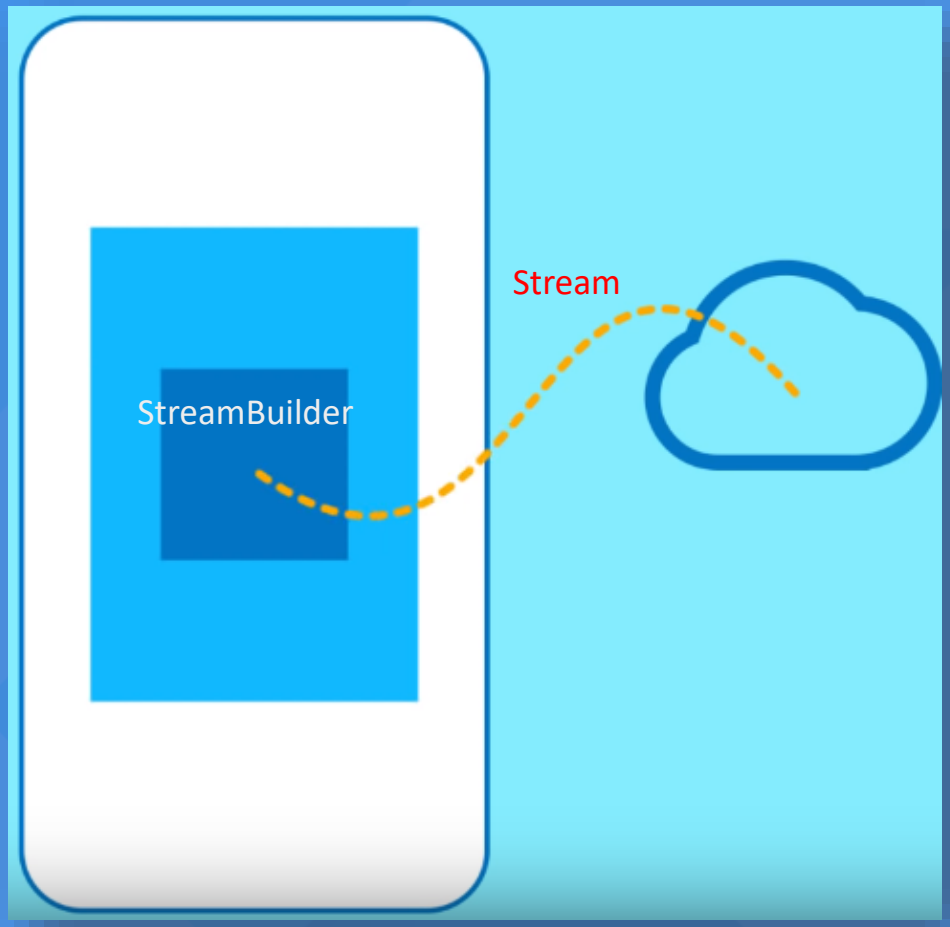
How do widgets get rebuilt?

when stateful widgets get notified via setState()



How do widgets get rebuilt?

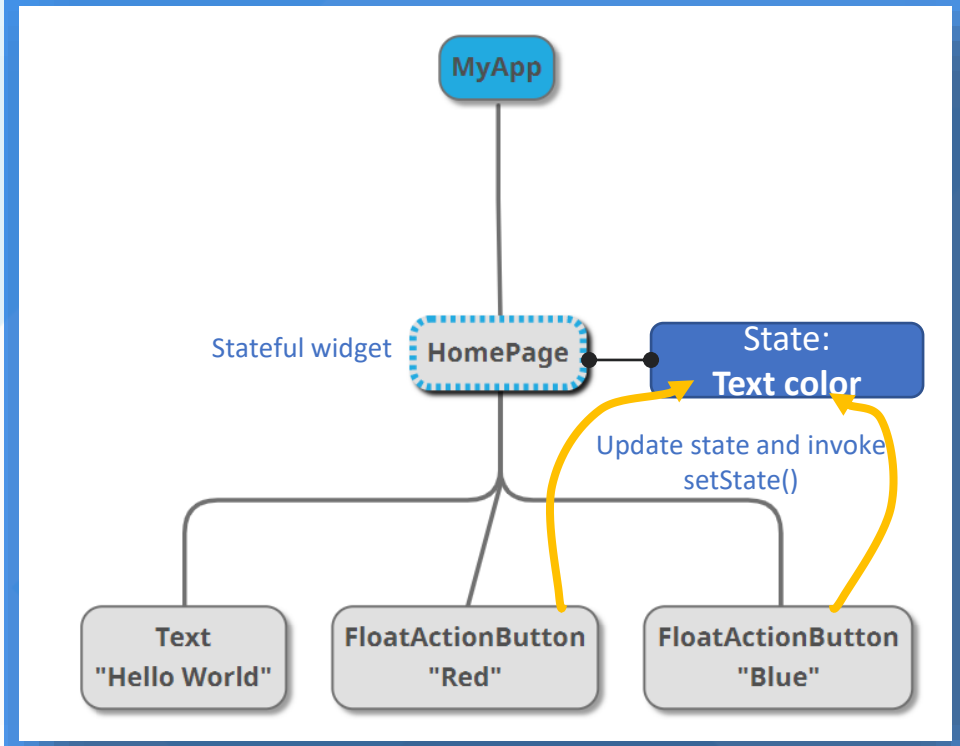
when StreamBuilders receive streams



Stateful Widgets

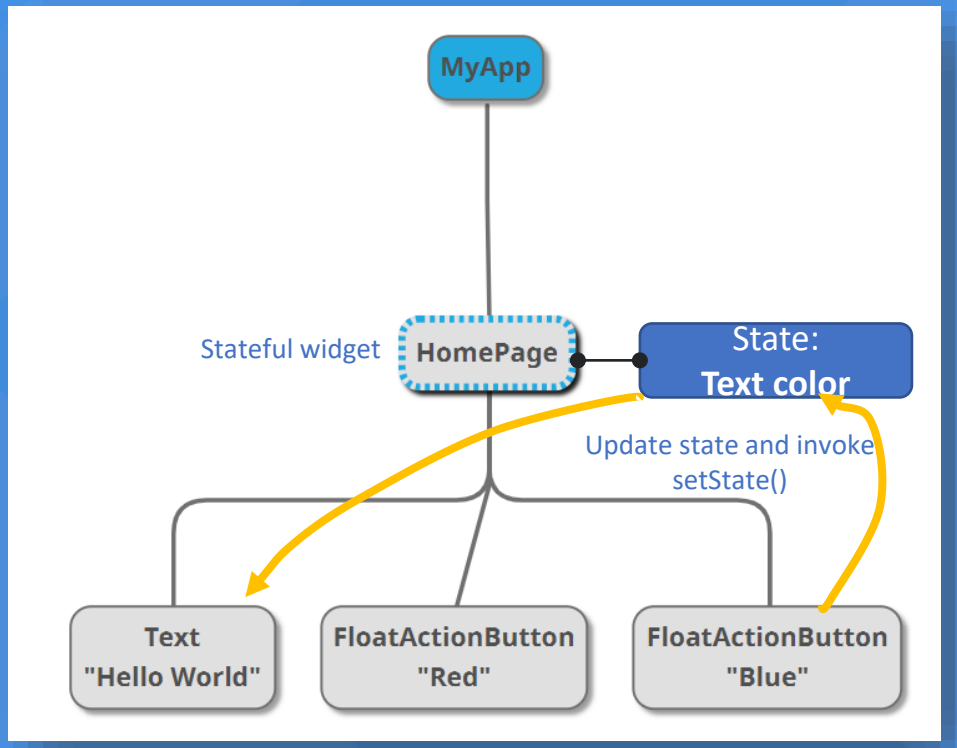
What is a stateful widget?

- A stateful widget is attached with a State object.
- The state object is mutable



What is a stateful widget?

- Each state object has an essential method called setState()
- Invoking this method causes the stateful widget gets rebuilt (so do the children)
- During the rebuilding, widgets use current state stored in the state object



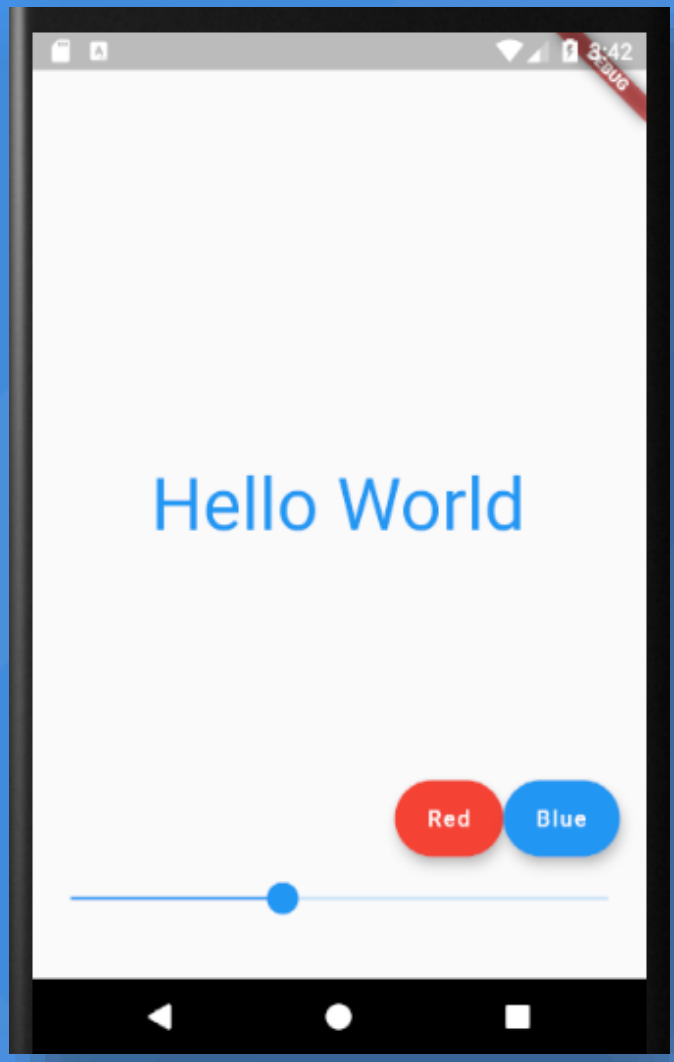
Attendance

Sign in on elearning

The keyword is given in the video

Demo

Changing text color and font size



Preparing Starter Code

1. Open `Git Bash`

2. Clone my github repo *(command below should be in one-line)*

```
git clone https://github.com/jumail-utm/stateful_widget_text
```

3. Move to the project directory

```
cd stateful_widget_text
```

4. Check what inside the repo

```
git log --oneline
```

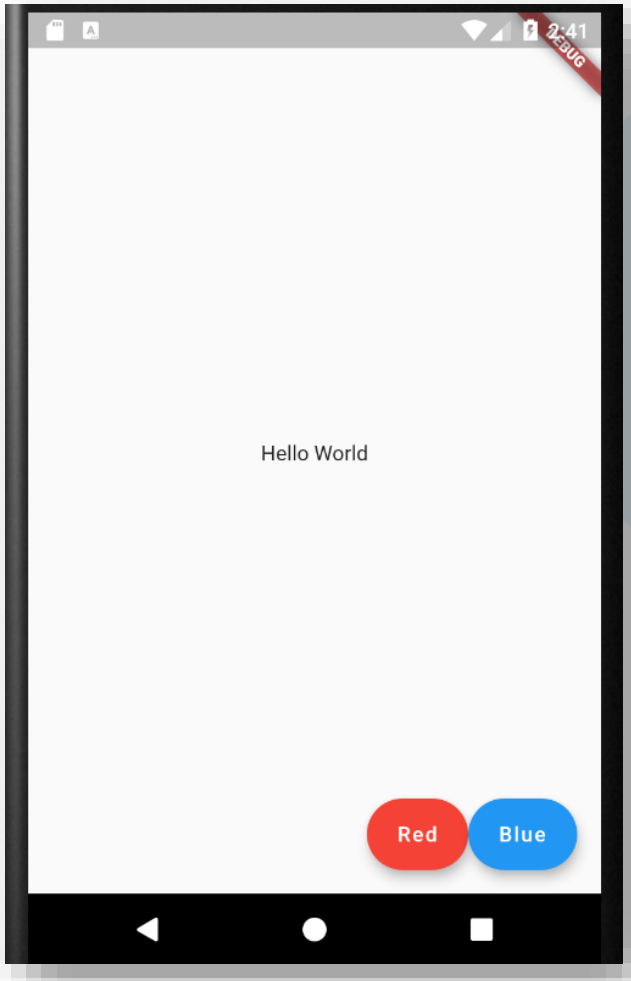
5. Pick the first commit ('initial project – stateless widget') and create a new branch from there.

```
git checkout 3f63864 -b playground
```

6. Open the project into VS Code

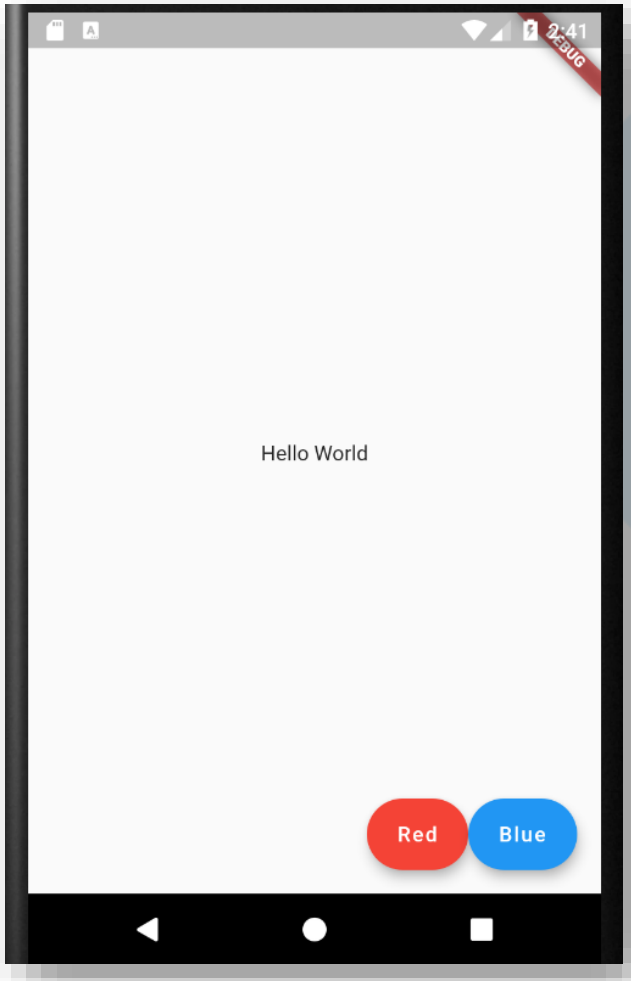
```
code .
```

Stateless Home Screen



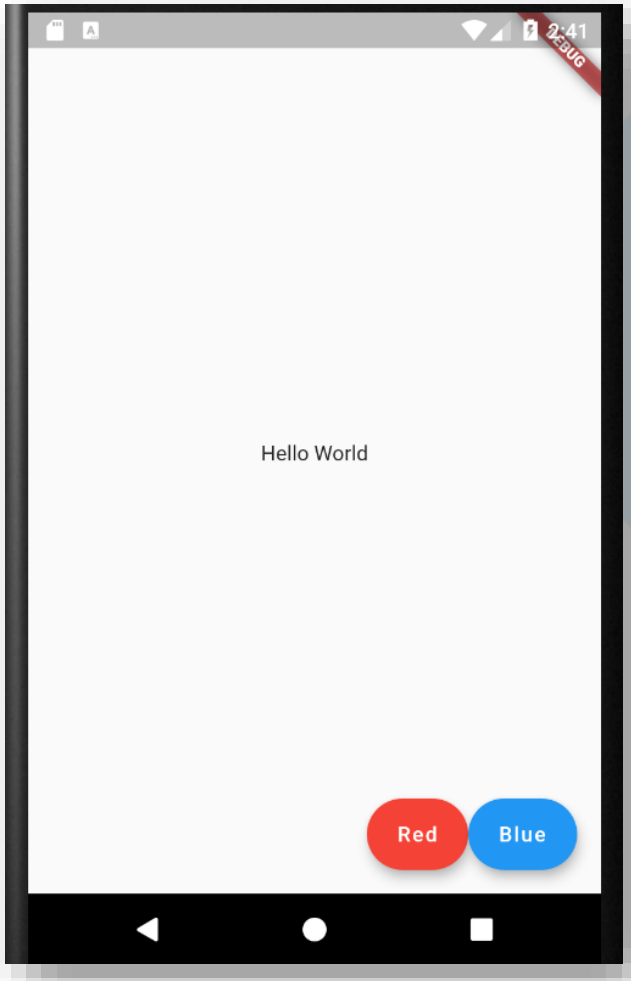

```dart
import 'package:flutter/material.dart';

void main() => runApp(MaterialApp(
      title: 'hello world',
      home: Home(),
    ));

class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text('Hello World'),
      ),
      floatingActionButton: Row(
        mainAxisAlignment: MainAxisAlignment.end,
        children: <Widget>[
          FloatingActionButton.extended(
            onPressed: () {},
            label: Text('Red'),
            backgroundColor: Colors.red,
          ),
          FloatingActionButton.extended(
            onPressed: () {},
            label: Text('Blue'),
            backgroundColor: Colors.blue,
          ),
        ],
      ),
    );
  }
}
```

Convert Home Screen to Stateful Widget



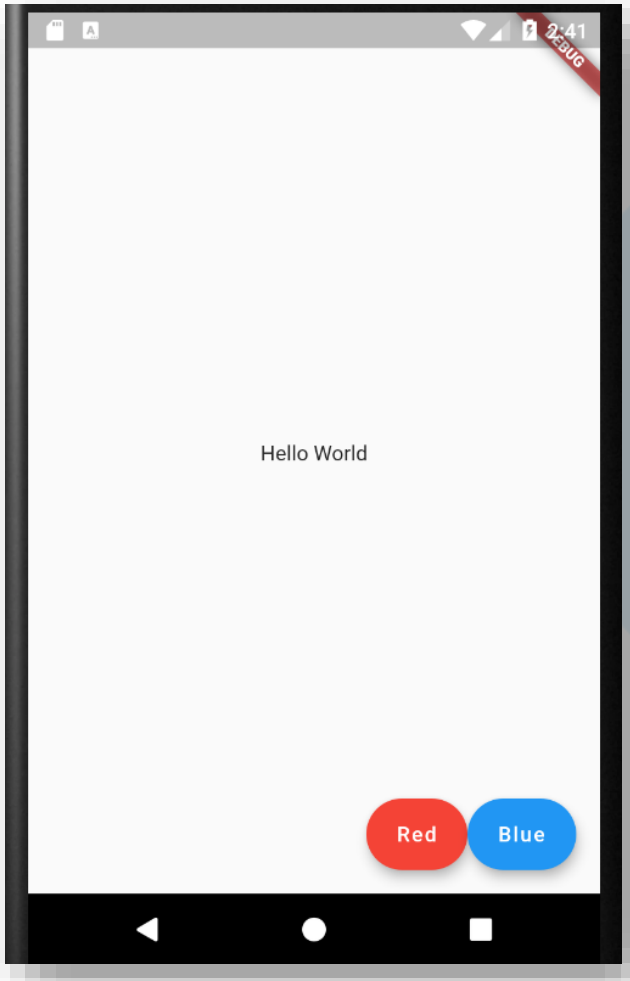

```dart
import 'package:flutter/material.dart';

void main() => runApp(MaterialApp(
      title: 'hello world',
      home: Home(),
    ));

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}

class _HomeState extends State<Home> {
  Color _color = Colors.black;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(
          'Hello World',
          style: TextStyle(color: _color),
        ),
      ),
      floatingActionButton: Row(
        mainAxisAlignment: MainAxisAlignment.end,
        children: <Widget>[
          FloatingActionButton.extended(
            onPressed: () {},
            label: Text('Red'),
            backgroundColor: Colors.red,
          ),
          FloatingActionButton.extended(
            onPressed: () {},
            label: Text('Blue'),
            backgroundColor: Colors.blue,
          ),
        ],
      ),
    );
  }
}
```

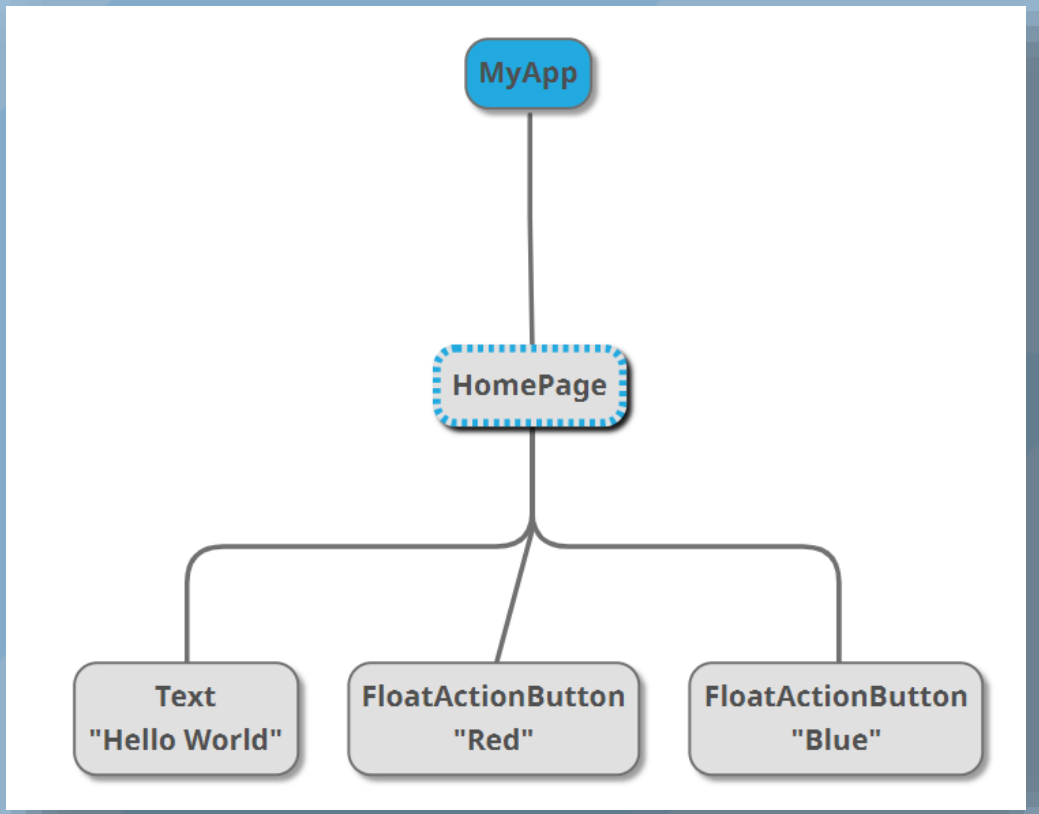
Notify to rebuild Home Screen via setState()




```dart
import 'package:flutter/material.dart';

void main() => runApp(MaterialApp(···

class Home extends StatefulWidget {···

class _HomeState extends State<Home> {
  Color _color = Colors.black;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(···
      floatingActionButton: Row(
        mainAxisAlignment: MainAxisAlignment.end,
        children: <Widget>[
          FloatingActionButton.extended(
            onPressed: () {
              _color = Colors.red;
              setState(() {});
            },
            label: Text('Red'),
            backgroundColor: Colors.red,
          ),
          FloatingActionButton.extended(
            onPressed: () {},
            label: Text('Blue'),
            backgroundColor: Colors.blue,
          ),
        ],
      ),
    );
  }
}
```

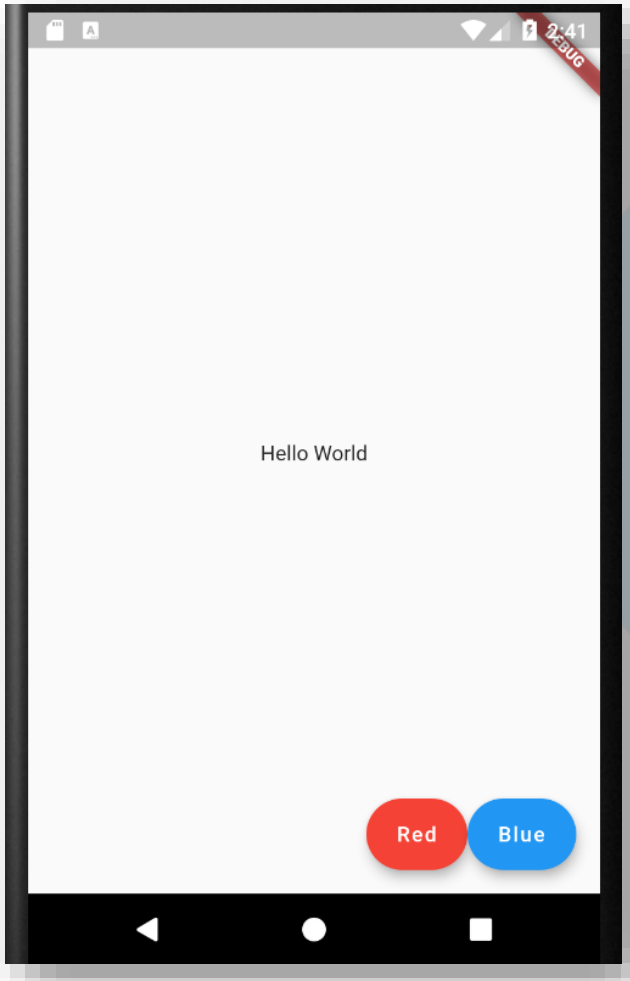
Which widget to be stateful?

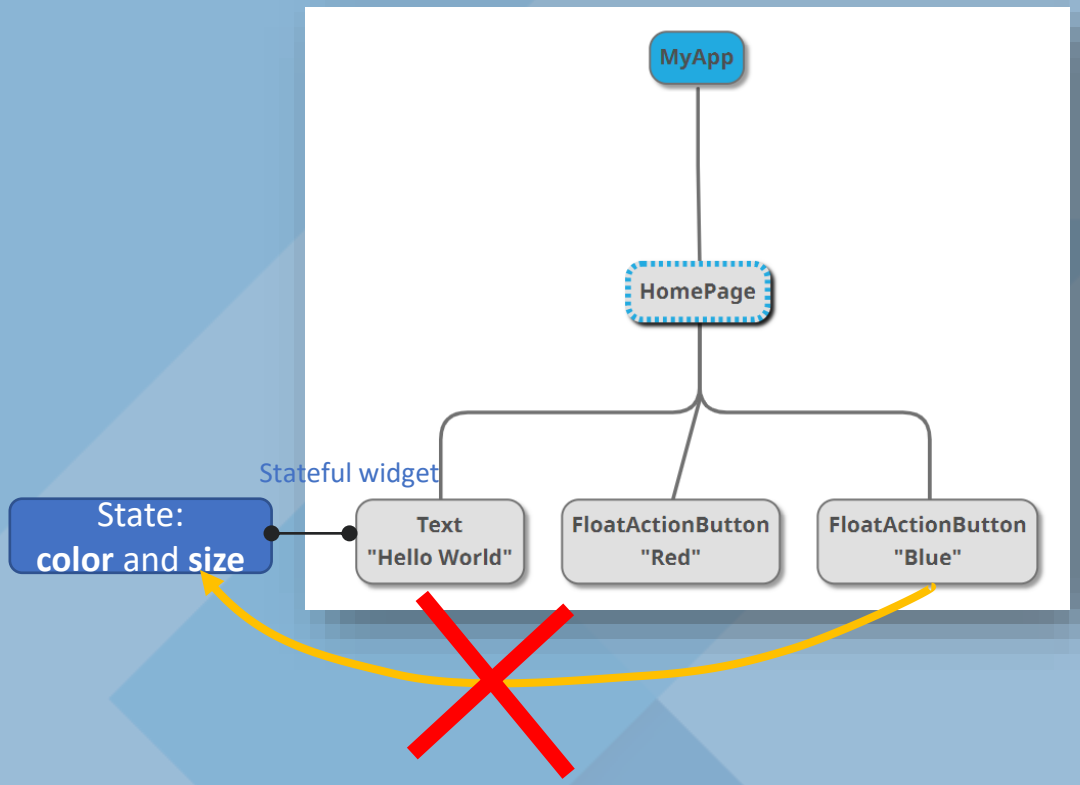




- Determine which part of the code is going to use the state
  - e.g., TextStyle(color:…) in the Text widget
- Determine which part of the code is going to update the state
  - e.g. in onPressed of the buttons

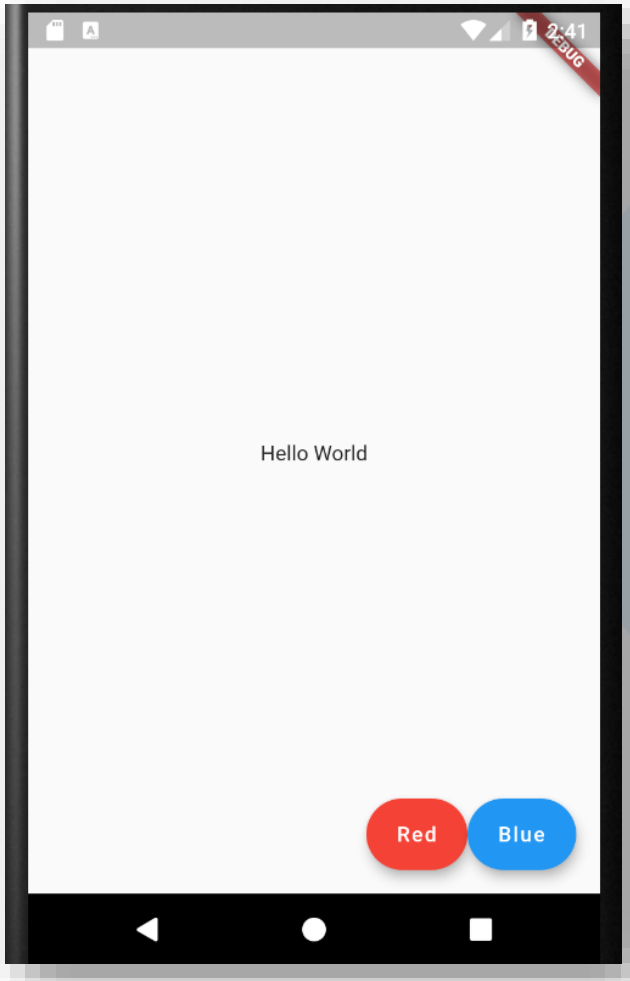
Which widget to be stateful?

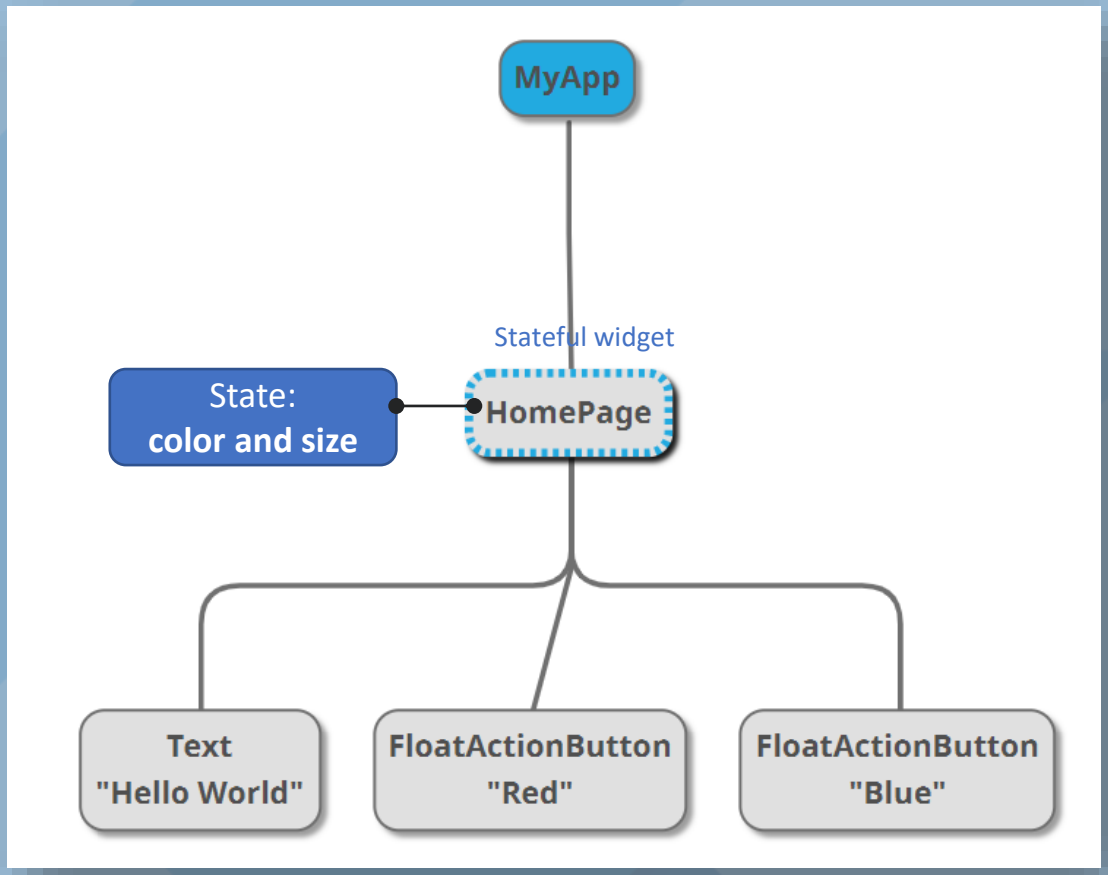




- Theoretically, the Text widget should be stateful as UI update done only on it
- However, the buttons (to update the state) cannot access to setState() if state object are put in the Text widget

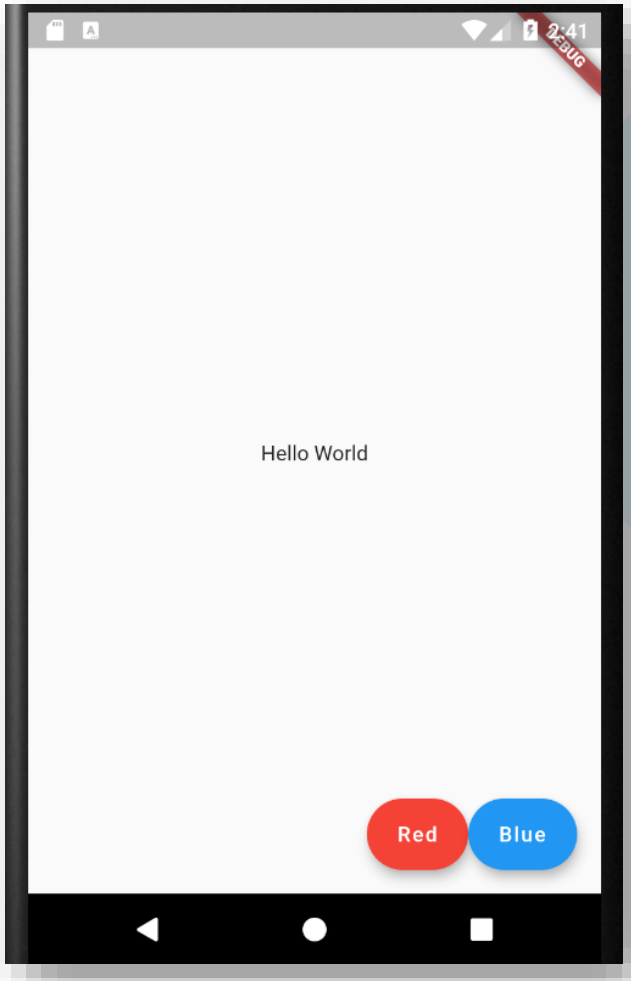
Which widget to be stateful?





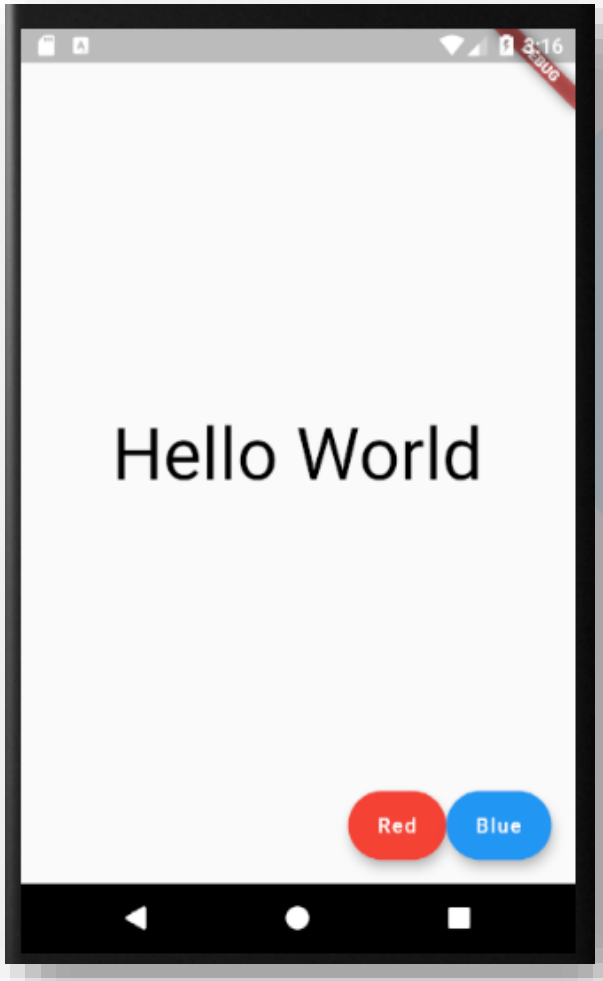
- A common technique: Lift State Up
  - Put the state to the common ancestor in the widget tree
  - HomePage screen is the best place to put the state, thus make it Stateful widget
  - When it gets rebuilt, so do its children

Write setState() in setter for convenience



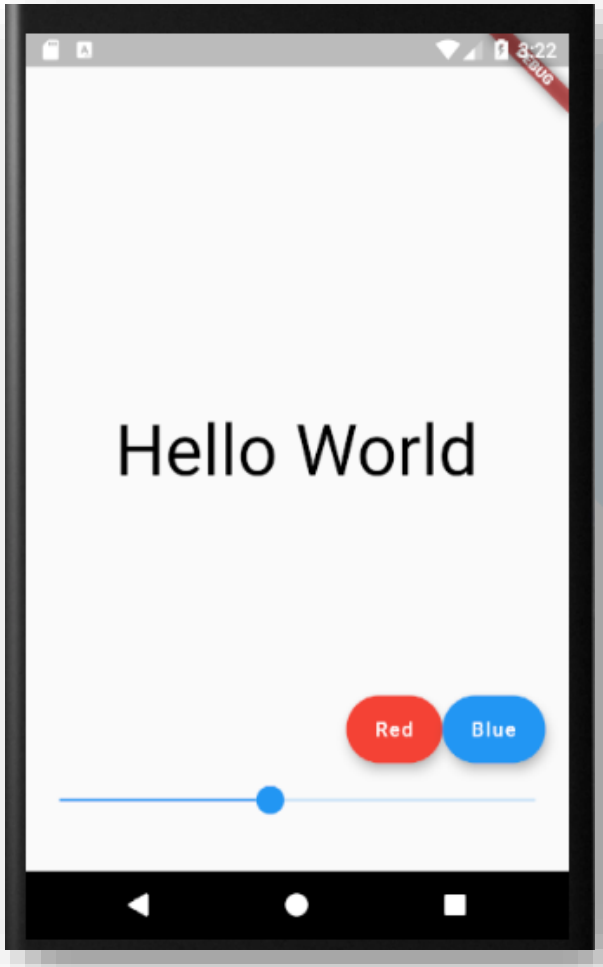

```dart
import 'package:flutter/material.dart';

void main() => runApp(MaterialApp(···

class Home extends StatefulWidget {···

class _HomeState extends State<Home> {
  Color _color = Colors.black;
  get color => _color;
  set color(value) => setState(() => _color = value);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(···
      floatingActionButton: Row(
        mainAxisAlignment: MainAxisAlignment.end,
        children: <Widget>[
          FloatingActionButton.extended(
            onPressed: () => color = Colors.red,
            label: Text('Red'),
            backgroundColor: Colors.red,
          ),
          FloatingActionButton.extended(
            onPressed: () => color = Colors.blue,
            label: Text('Blue'),
            backgroundColor: Colors.blue,
          ),
        ],
      ),
    );
  }
}
```

Add another state, the font size



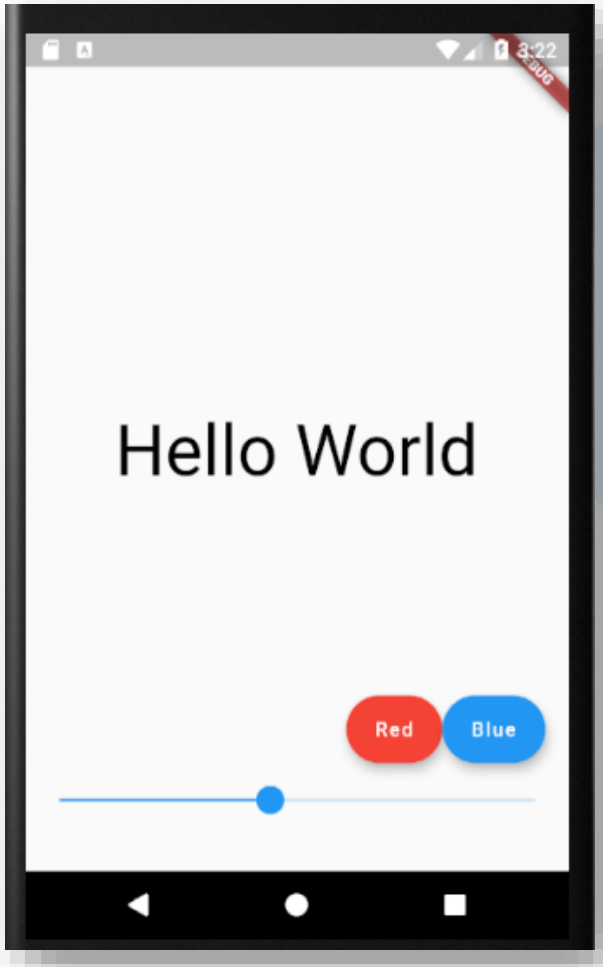

```dart
import 'package:flutter/material.dart';

void main() => runApp(MaterialApp(···

class Home extends StatefulWidget {···

class _HomeState extends State<Home> {
  Color _color = Colors.black;
  get color => _color;
  set color(value) => setState(() => _color = value);

  double _size = 50.0;
  get size => _size;
  set size(value) => setState(() => _size = value);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(
          'Hello World',
          style: TextStyle(
            color: _color,
            fontSize: size,
          ),
        ),
      ),
      floatingActionButton: Row(···
    );
  }
}
```

Add a Slider to control the font size



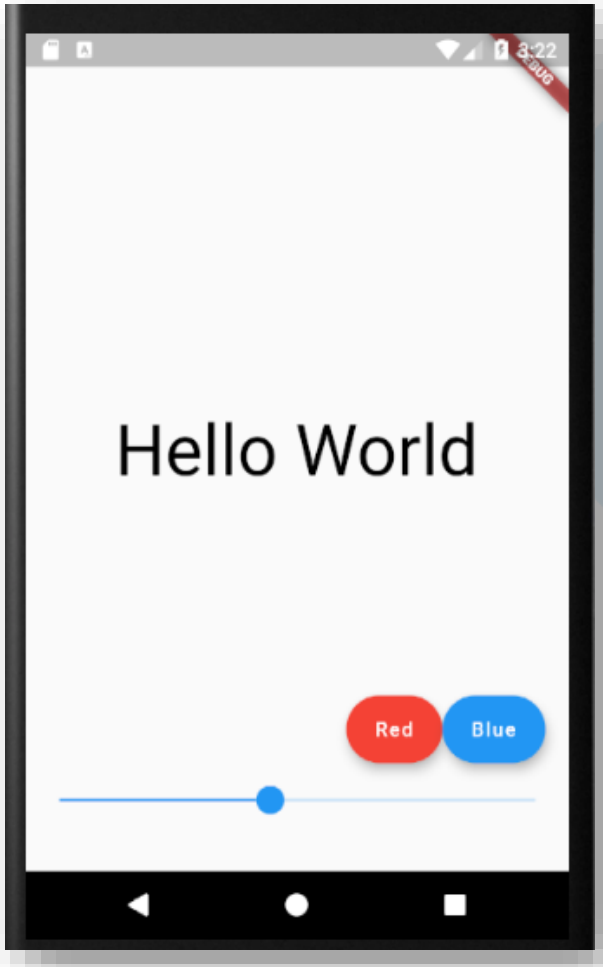

```dart
import 'package:flutter/material.dart';

void main() => runApp(MaterialApp(···

class Home extends StatefulWidget {···

class _HomeState extends State<Home> {
  Color _color = Colors.black;
  get color => _color;
  set color(value) => setState(() => _color = value);

  double _size = 50.0;
  get size => _size;
  set size(value) => setState(() => _size = value);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(···
      floatingActionButton: Row(···
      bottomSheet: SizedBox(
        height: 100,
        child: Slider(
          value: size,
          min: 10,
          max: 100,
          onChanged: (value) => size = value,
        ),
      ),
    );
  }
}
```

Refactor the buttons using a new method




```
23    Widget build(BuildContext context) {
24      return Scaffold(
25  >     ├─body: Center(···
34        ├─floatingActionButton: Row(
35        │ │ mainAxisAlignment: MainAxisAlignment.end,
36        │ │ children: <Widget>[
37        │ ├── _buildButton('Red', Colors.red),
38        │ └── _buildButton('Blue', Colors.blue),
39        │   ],
40        │ ),
41  >     └─bottomSheet: SizedBox(···
50      );
51    }
52
53    FloatingActionButton _buildButton(String title, Color col) {
54      return FloatingActionButton.extended(
55      │ onPressed: () => color = col,
56      └─label: Text(title),
57        backgroundColor: col,
58      );
59    }
60  }
```

Refactor the buttons using a widget class




```
21
22    @override
23    Widget build(BuildContext context) {
24      return Scaffold(
25 >      body: Center(···
34        floatingActionButton: Row(
35          mainAxisAlignment: MainAxisAlignment.end,
36          children: <Widget>[
37            Button(this, 'Red', Colors.red),
38            Button(this, 'Blue', Colors.blue),
39          ],
40        ),
41 >      bottomSheet: SizedBox(···
50      );
51    }
52  }
53
54  class Button extends StatelessWidget {
55    final state;
56    final color;
57    final title;
58
59    const Button(this.state, this.title, this.color);
60
61    @override
62    Widget build(BuildContext context) {
63      return FloatingActionButton.extended(
64        onPressed: () => state.color = color,
65        label: Text(title),
66        backgroundColor: color,
67      );
68    }
69  }
```

Summary

- States are App's data - running
- State management – app's data and UI in-sync
- Imperative vs declarative UI
- Several approaches to manage states
- Stateful widgets – setState()