FLUTTER TUTORIALS

BY: **TECMAN**

TECMAN Lesson 6

Adding interactivity to your Flutter app

What you'll learn

- How to respond to taps.
- How to create a custom widget.
- The difference between stateless and stateful widgets.

TECMAN LIMITED 2019

Adding interactivity to your Flutter app

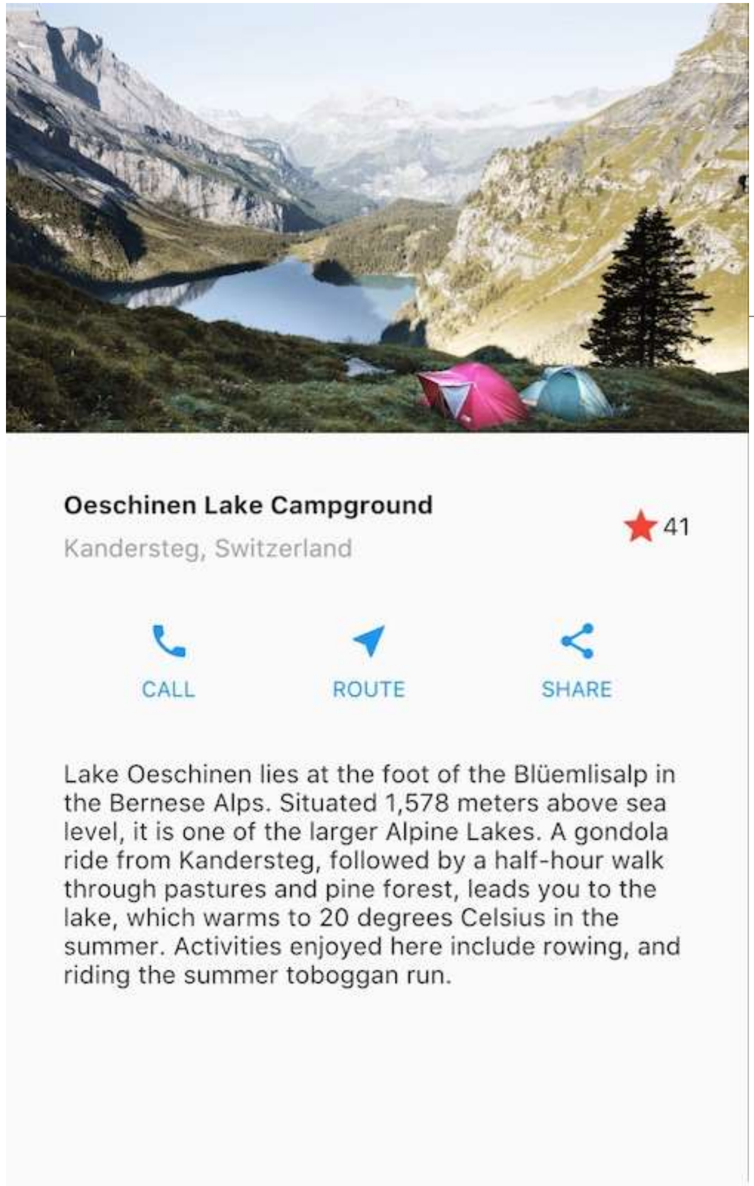
How do you modify your app to make it react to user input? In this tutorial, you’ll add interactivity to an app that contains only non-interactive widgets. Specifically, you’ll modify an icon to make it tappable by creating a custom stateful widget that manages two stateless widgets.

TECMAN LIMITED 2019

Adding interactivity to your Flutter app

Layout tutorial showed you how to create the layout for the following screenshot.

TECMAN LIMITED 2019



Oeschinen Lake Campground

Kandersteg, Switzerland

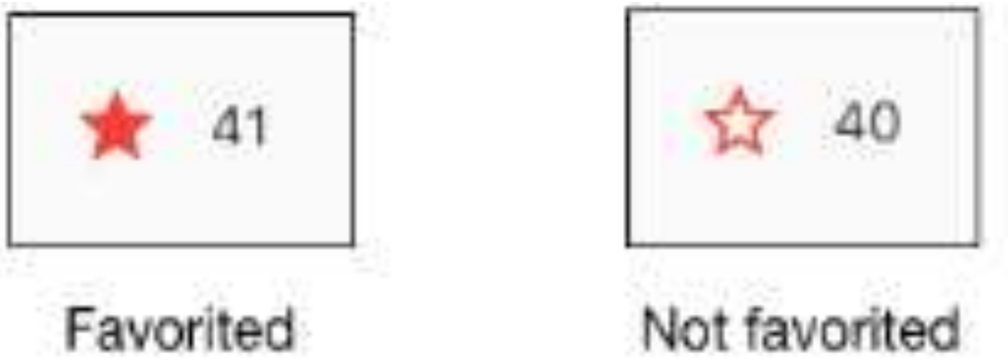
Lake Oeschinen lies at the foot of the Blüemlisalp in the Bernese Alps. Situated 1,578 meters above sea level, it is one of the larger Alpine Lakes. A gondola ride from Kandersteg, followed by a half-hour walk through pastures and pine forest, leads you to the lake, which warms to 20 degrees Celsius in the summer. Activities enjoyed here include rowing, and riding the summer toboggan run.

TECMAN LIMITED 2019

Adding interactivity to your Flutter app

When the app first launches, the star is solid red, indicating that this lake has previously been favorited. The number next to the star indicates that 41 people have favorited this lake. After completing this tutorial, tapping the star removes its favorited status, replacing the solid star with an outline and decreasing the count. Tapping again favorites the lake, drawing a solid star and increasing the count.

Adding interactivity to your Flutter app



Favorited

Not favorited

Adding interactivity to your Flutter app

To accomplish this, you’ll create a single custom widget that includes both the star and the count, which are themselves widgets. Tapping the star changes state for both widgets, so the same widget should manage both.

You can get right to touching the code in Step 2: Subclass StatefulWidget. If you want to try different ways of managing state, skip to Managing state

TECMAN LIMITED 2019

Stateful and stateless widgets

A widget is either stateful or stateless. If a widget can change—when a user interacts with it, for example—it's stateful.

A *stateless* widget never changes. Icon, IconButton, and Text are examples of stateless widgets. Stateless widgets subclass StatelessWidget.

TECMAN LIMITED 2019

Stateful and stateless widgets

A *stateful* widget is dynamic: for example, it can change its appearance in response to events triggered by user interactions or when it receives data. Checkbox, Radio, Slider, InkWell, Form, and TextField are examples of stateful widgets. Stateful widgets subclass StatefulWidget.

TECMAN LIMITED 2019

Stateful and stateless widgets

A widget’s state is stored in a State object, separating the widget’s state from its appearance. The state consists of values that can change, like a slider’s current value or whether a checkbox is checked. When the widget’s state changes, the state object calls setState(), telling the framework to redraw the widget

TECMAN LIMITED 2019

Creating a stateful widget

What's about?

- A stateful widget is implemented by two classes: a subclass of StatefulWidget and a subclass of State.
- The state class contains the widget’s mutable state and the widget’s build() method.
- When the widget’s state changes, the state object calls setState(), telling the framework to redraw the widget.

TECMAN LIMITED 2019

Creating a stateful widget

In this section, you’ll create a custom stateful widget. You’ll replace two stateless widgets—the solid red star and the numeric count next to it—with a single custom stateful widget that manages a row with two children widgets: an IconButton and Text.

- Implementing a custom stateful widget requires creating two classes:
- A subclass of StatefulWidget that defines the widget.
- A subclass of State that contains the state for that widget and defines the widget’s build() method.

TECMAN LIMITED 2019

Creating a stateful widget

This section shows you how to build a stateful widget, called FavoriteWidget, for the lakes app. After setting up, your first step is choosing how state is managed for FavoriteWidget.

Step 0: Get ready

If you've already built the app in Layout tutorial (step 6), skip to the next section.

1. Make sure you've set up your environment.
2. Create a basic "Hello World" Flutter app.
3. Replace the lib/main.dart file with main.dart.
4. Replace the pubspec.yaml file with pubspec.yaml.
5. Create an images directory in your project, and add lake.jpg.

Once you have a connected and enabled device, or you've launched the iOS simulator (part of the Flutter install), you are good to go!

TECMAN LIMITED 2019

Step 1: Decide which object manages the widget’s state

A widget’s state can be managed in several ways, but in our example the widget itself, FavoriteWidget, will manage its own state. In this example, toggling the star is an isolated action that doesn’t affect the parent widget or the rest of the UI, so the widget can handle its state internally.

Learn more about the separation of widget and state, and how state might be managed, in Managing state

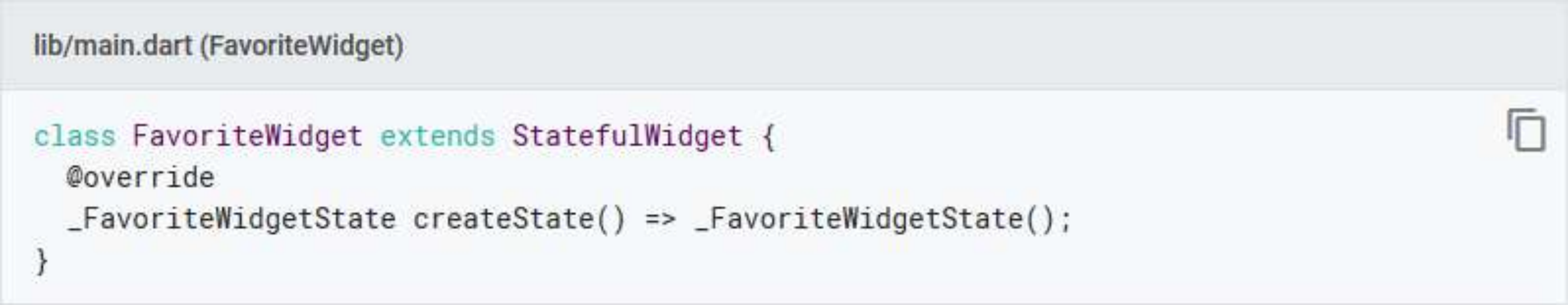
TECMAN LIMITED 2019

Step 2: Subclass StatefulWidget

The FavoriteWidget class manages its own state, so it overrides createState() to create a State object. The framework calls createState() when it wants to build the widget. In this example, createState() returns an instance of _FavoriteWidgetState, which you’ll implement in the next step.

TECMAN LIMITED 2019

Step 2: Subclass StatefulWidget



lib/main.dart (FavoriteWidget)

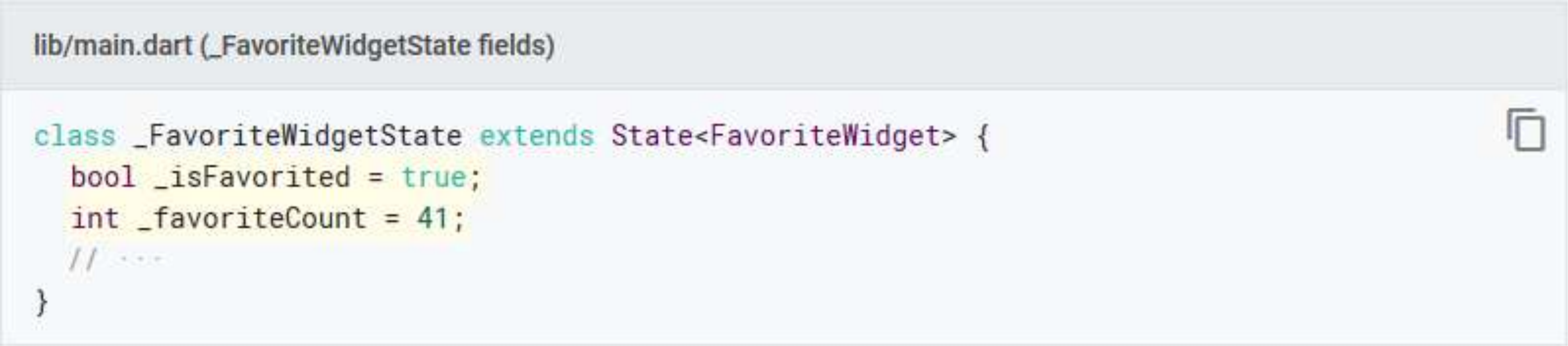
```
class FavoriteWidget extends StatefulWidget {
  @override
  _FavoriteWidgetState createState() => _FavoriteWidgetState();
}
```

Step 3: Subclass State

The _FavoriteWidgetState class stores the mutable data that can change over the lifetime of the widget. When the app first launches, the UI displays a solid red star, indicating that the lake has “favorite” status, along with 41 likes. These values are stored in the _isFavorited and _favoriteCount fields:

TECMAN LIMITED 2019

Step 3: Subclass State



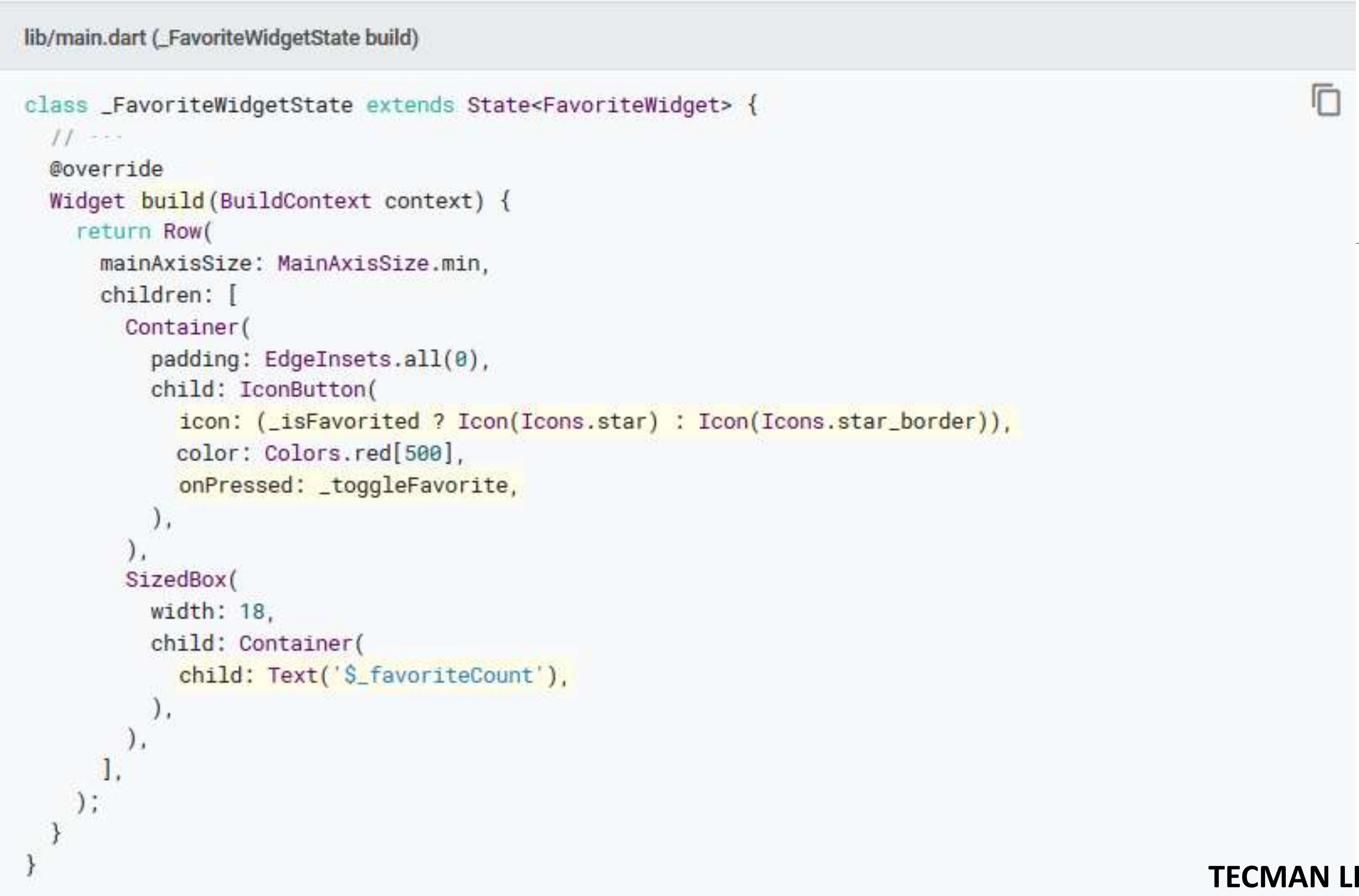
lib/main.dart (_FavoriteWidgetState fields)

```
class _FavoriteWidgetState extends State<FavoriteWidget> {
  bool _isFavorited = true;
  int _favoriteCount = 41;
  // ···
}
```

Step 3: Subclass State

The class also defines a build() method, which creates a row containing a red IconButton, and Text. You use IconButton (instead of Icon) because it has an onPressed property that defines the callback function (_toggleFavorite) for handling a tap. You'll define the callback function next.

TECMAN LIMITED 2019



lib/main.dart (_FavoriteWidgetState build)

```dart
class _FavoriteWidgetState extends State<FavoriteWidget> {
  // ···
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisSize: MainAxisSize.min,
      children: [
        Container(
          padding: EdgeInsets.all(0),
          child: IconButton(
            icon: (_isFavorited ? Icon(Icons.star) : Icon(Icons.star_border)),
            color: Colors.red[500],
            onPressed: _toggleFavorite,
          ),
        ),
        SizedBox(
          width: 18,
          child: Container(
            child: Text('$_favoriteCount'),
          ),
        ),
      ],
    );
  }
}
```

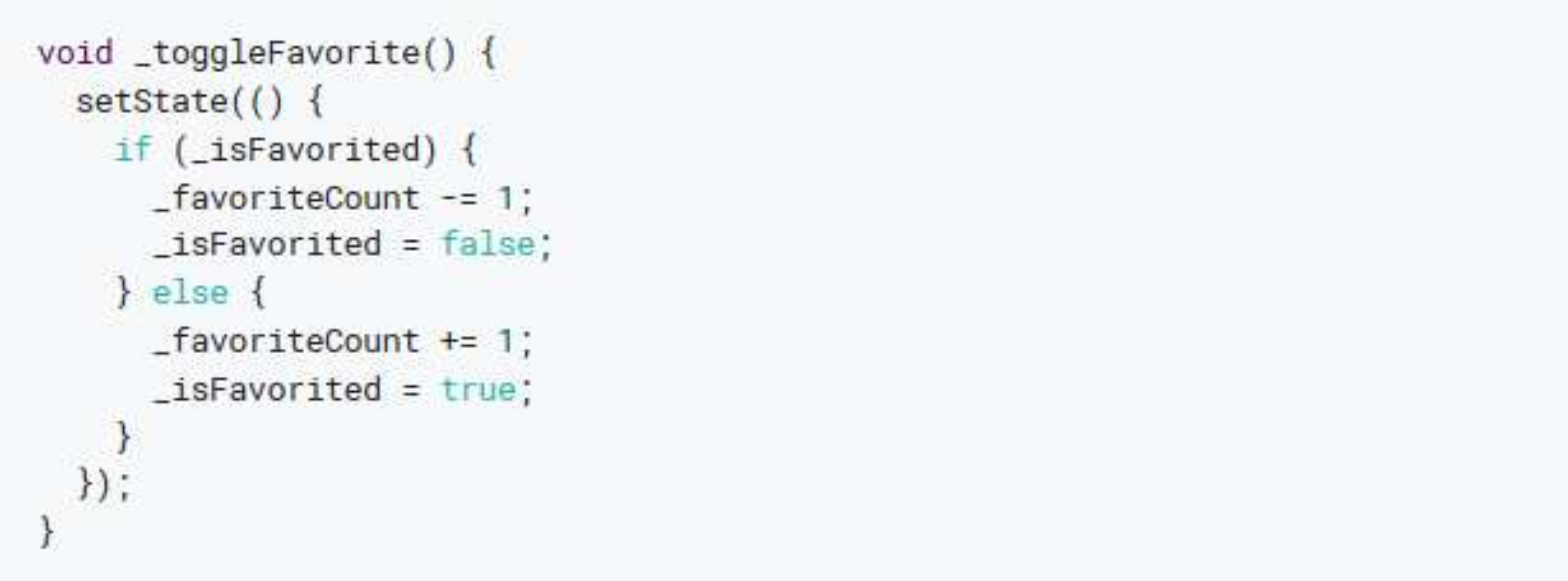
Step 3: Subclass State

The _toggleFavorite() method, which is called when the IconButton is pressed, calls setState(). Calling setState() is critical, because this tells the framework that the widget’s state has changed and that the widget should be redraw. The function argument to setState() toggles the UI between these two states:

- A star icon and the number 41
- A star_border icon and the number 40

TECMAN LIMITED 2019

Step 3: Subclass State

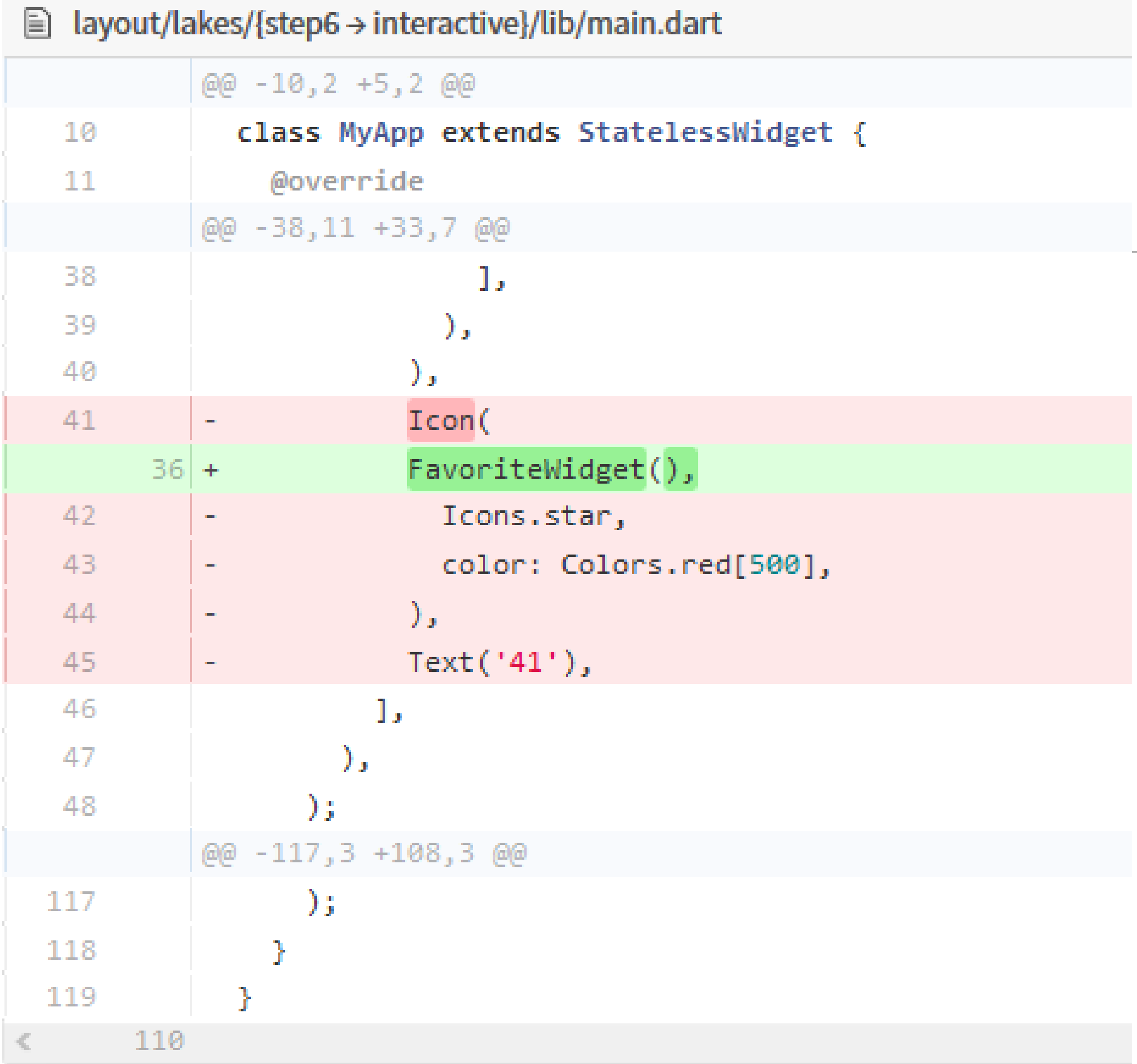


```
void _toggleFavorite() {
  setState(() {
    if (_isFavorited) {
      _favoriteCount -= 1;
      _isFavorited = false;
    } else {
      _favoriteCount += 1;
      _isFavorited = true;
    }
  });
}
```

Step 4: Plug the stateful widget into the widget tree

Add your custom stateful widget to the widget tree in the app’s build() method. First, locate the code that creates the Icon and Text, and delete it. In the same location, create the stateful widget:

TECMAN LIMITED 2019



layout/lakes/{step6 → interactive}/lib/main.dart

```
@@ -10,2 +5,2 @@
 class MyApp extends StatelessWidget {
   @override
@@ -38,11 +33,7 @@
               ],
             ),
           ),
-          Icon(
+          FavoriteWidget(),
-            Icons.star,
-            color: Colors.red[500],
-          ),
-          Text('41'),
         ],
       ),
     );
@@ -117,3 +108,3 @@
     );
   }
 }
```

TECMAN LIMITED 2019

Step 4: Plug the stateful widget into the widget tree

That’s it! When you hot reload the app, the star icon should now respond to taps.

TECMAN LIMITED 2019

Step 4: Plug the stateful widget into the widget tree

Problems?

If you can’t get your code to run, look in your IDE for possible errors. Debugging Flutter Apps might help. If you still can’t find the problem, check your code against the interactive lakes example on GitHub.

- lib/main.dart
- pubspec.yaml
- lakes.jpg

If you still have questions, refer to any one of the developer community channels.

TECMAN LIMITED 2019

Step 4: Plug the stateful widget into the widget tree

The rest of this page covers several ways a widget's state can be managed, and lists other available interactive widgets.

TECMAN LIMITED 2019

Managing state

What's the point?

- There are different approaches for managing state.
- You, as the widget designer, choose which approach to use.
- If in doubt, start by managing state in the parent widget.

TECMAN LIMITED 2019

Managing state

Who manages the stateful widget’s state?

- The widget itself?
- The parent widget?
- Both?
- Another object?

TECMAN LIMITED 2019

Managing state

- The answer is… it depends. There are several valid ways to make your widget interactive. You, as the widget designer, make the decision based on how you expect your widget to be used. Here are the most common ways to manage state:

TECMAN LIMITED 2019

Managing state

- The widget manages its own state
- The parent manages the widget's state
- A mix-and-match approach

TECMAN LIMITED 2019

Managing state

How do you decide which approach to use? The following principles should help you decide:

- If the state in question is user data, for example the checked or unchecked mode of a checkbox, or the position of a slider, then the state is best managed by the parent widget.
- If the state in question is aesthetic, for example an animation, then the state is best managed by the widget itself.

If in doubt, start by managing state in the parent widget.

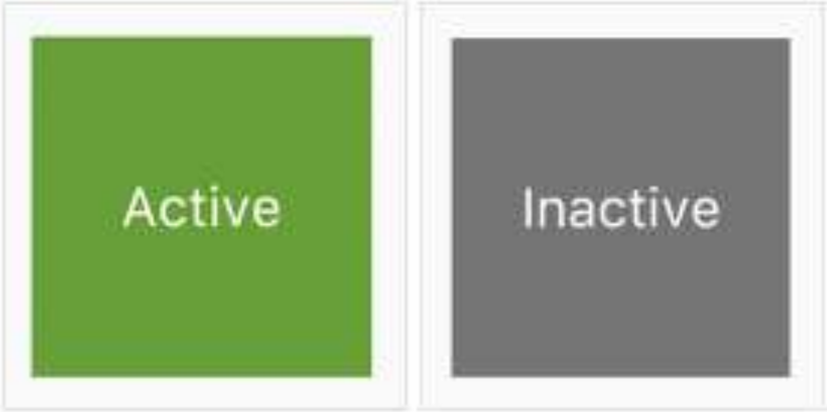
TECMAN LIMITED 2019

Managing state

We’ll give examples of the different ways of managing state by creating three simple examples: TapboxA, TapboxB, and TapboxC. The examples all work similarly—each creates a container that, when tapped, toggles between a green or grey box. The _active boolean determines the color: green for active or grey for inactive.

TECMAN LIMITED 2019

Managing state



Managing state

These examples use GestureDetector to capture activity on the Container.

TECMAN LIMITED 2019

The widget manages its own state

Sometimes it makes the most sense for the widget to manage its state internally. For example, ListView automatically scrolls when its content exceeds the render box. Most developers using ListView don’t want to manage ListView’s scrolling behavior, so ListView itself manages its scroll offset.

TECMAN LIMITED 2019

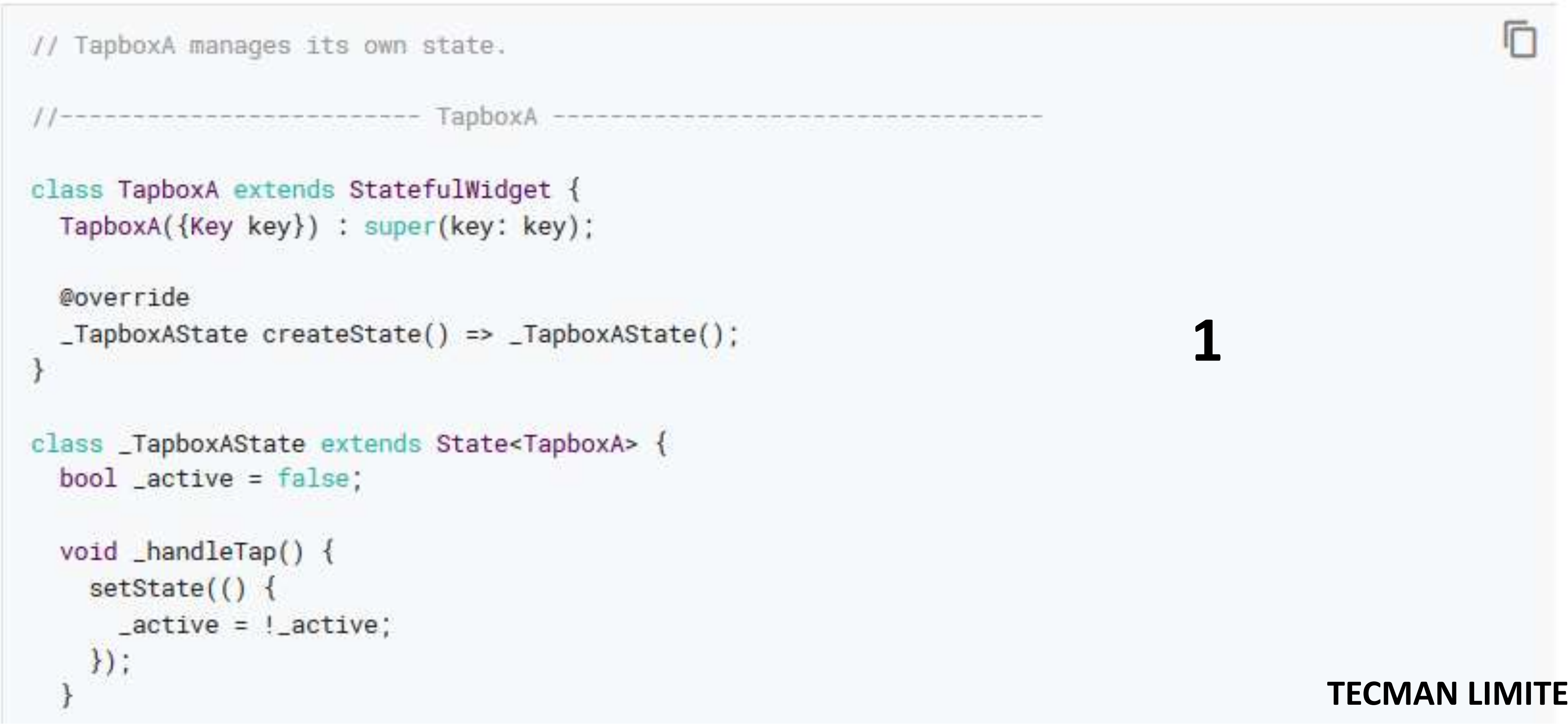
The widget manages its own state

The _TapboxAState class:

- Manages state for TapboxA.
- Defines the _active boolean which determines the box’s current color.
- Defines the _handleTap() function, which updates _active when the box is tapped and calls the setState() function to update the UI.

TECMAN LIMITED 2019

Implements all interactive behavior for the widget.

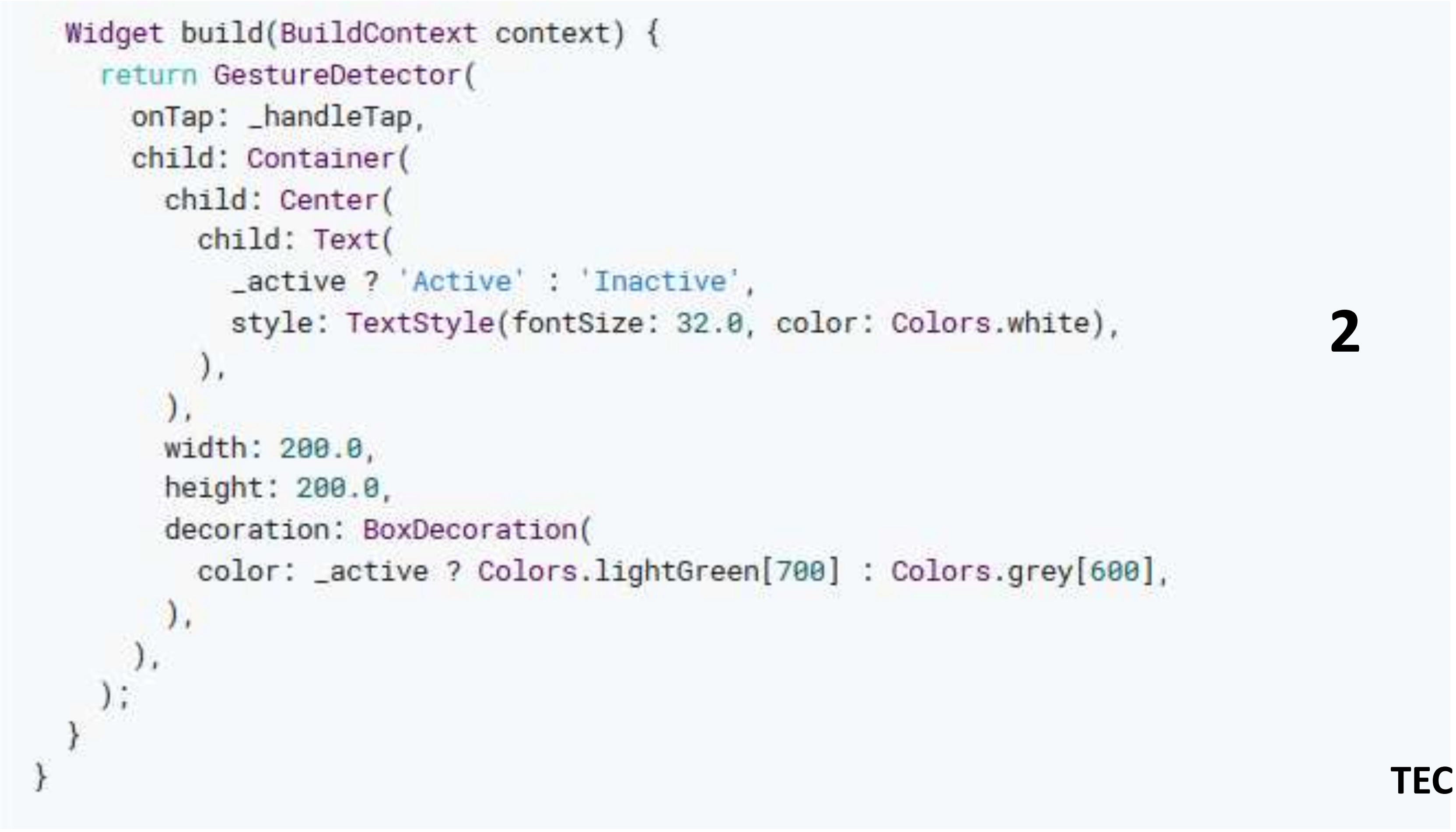


```
// TapboxA manages its own state.

//------------------------- TapboxA ----------------------------------

class TapboxA extends StatefulWidget {
  TapboxA({Key key}) : super(key: key);

  @override
  _TapboxAState createState() => _TapboxAState();
}

class _TapboxAState extends State<TapboxA> {
  bool _active = false;

  void _handleTap() {
    setState(() {
      _active = !_active;
    });
  }
```

1

TECMAN LIMITED 2019

Implements all interactive behavior for the widget.

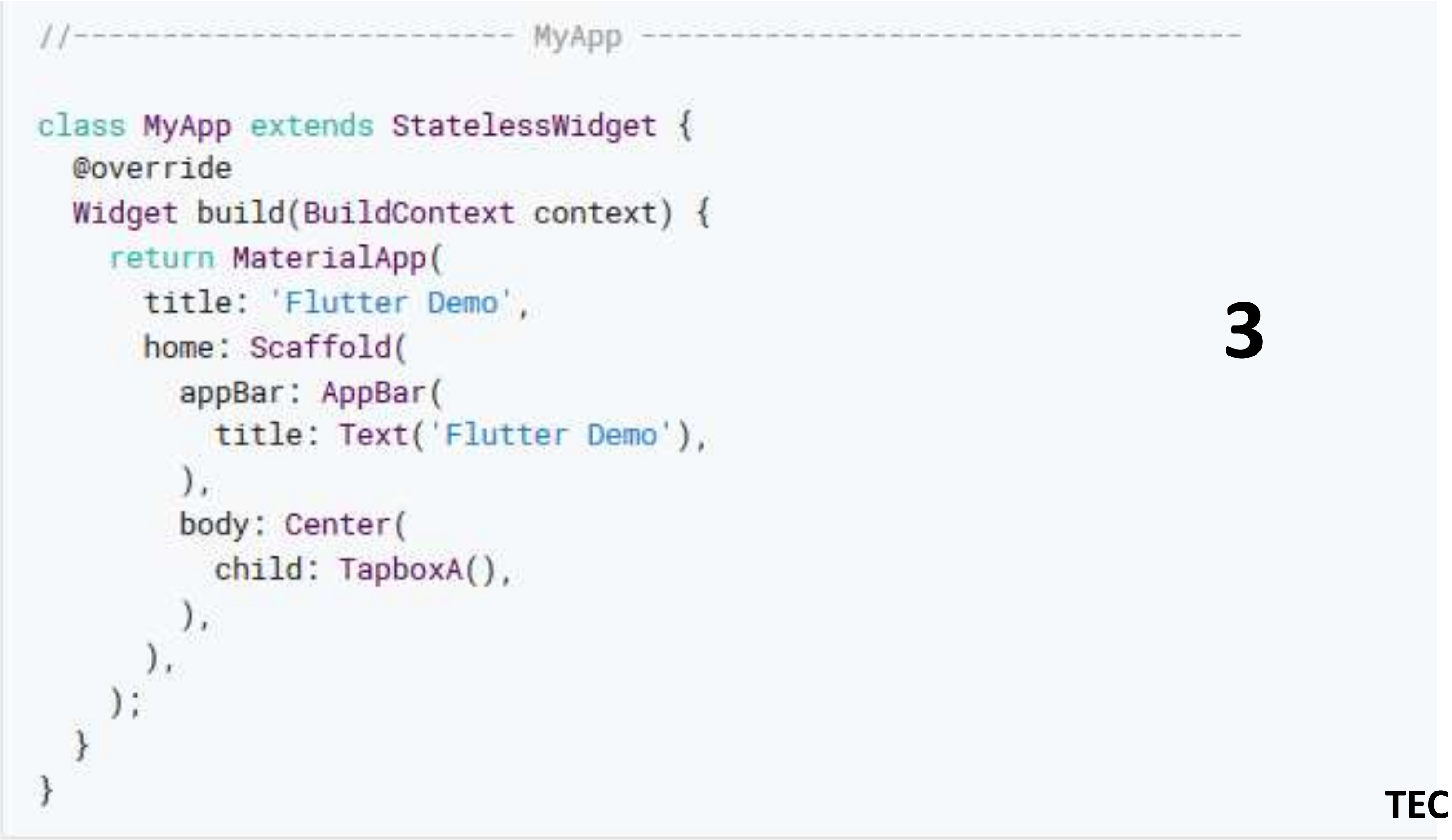


```
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: _handleTap,
      child: Container(
        child: Center(
          child: Text(
            _active ? 'Active' : 'Inactive',
            style: TextStyle(fontSize: 32.0, color: Colors.white),
          ),
        ),
        width: 200.0,
        height: 200.0,
        decoration: BoxDecoration(
          color: _active ? Colors.lightGreen[700] : Colors.grey[600],
        ),
      ),
    );
  }
}
```

2

TECMAN LIMITED 2019

Implements all interactive behavior for the widget.



```
//------------------------ MyApp ----------------------------------

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Demo'),
        ),
        body: Center(
          child: TapboxA(),
        ),
      ),
    );
  }
}
```

3

TECMAN LIMITED 2019

The parent widget manages the widget’s state

Often it makes the most sense for the parent widget to manage the state and tell its child widget when to update. For example, IconButton allows you to treat an icon as a tappable button. IconButton is a stateless widget because we decided that the parent widget needs to know whether the button has been tapped, so it can take appropriate action.

TECMAN LIMITED 2019

The parent widget manages the widget’s state

- In the following example, TapboxB exports its state to its parent through a callback. Because TapboxB doesn’t manage any state, it subclasses StatelessWidget.

TECMAN LIMITED 2019

The parent widget manages the widget’s state

The ParentWidgetState class:

- Manages the _active state for TapboxB.
- Implements _handleTapboxChanged(), the method called when the box is tapped.
- When the state changes, calls setState() to update the UI.

TECMAN LIMITED 2019

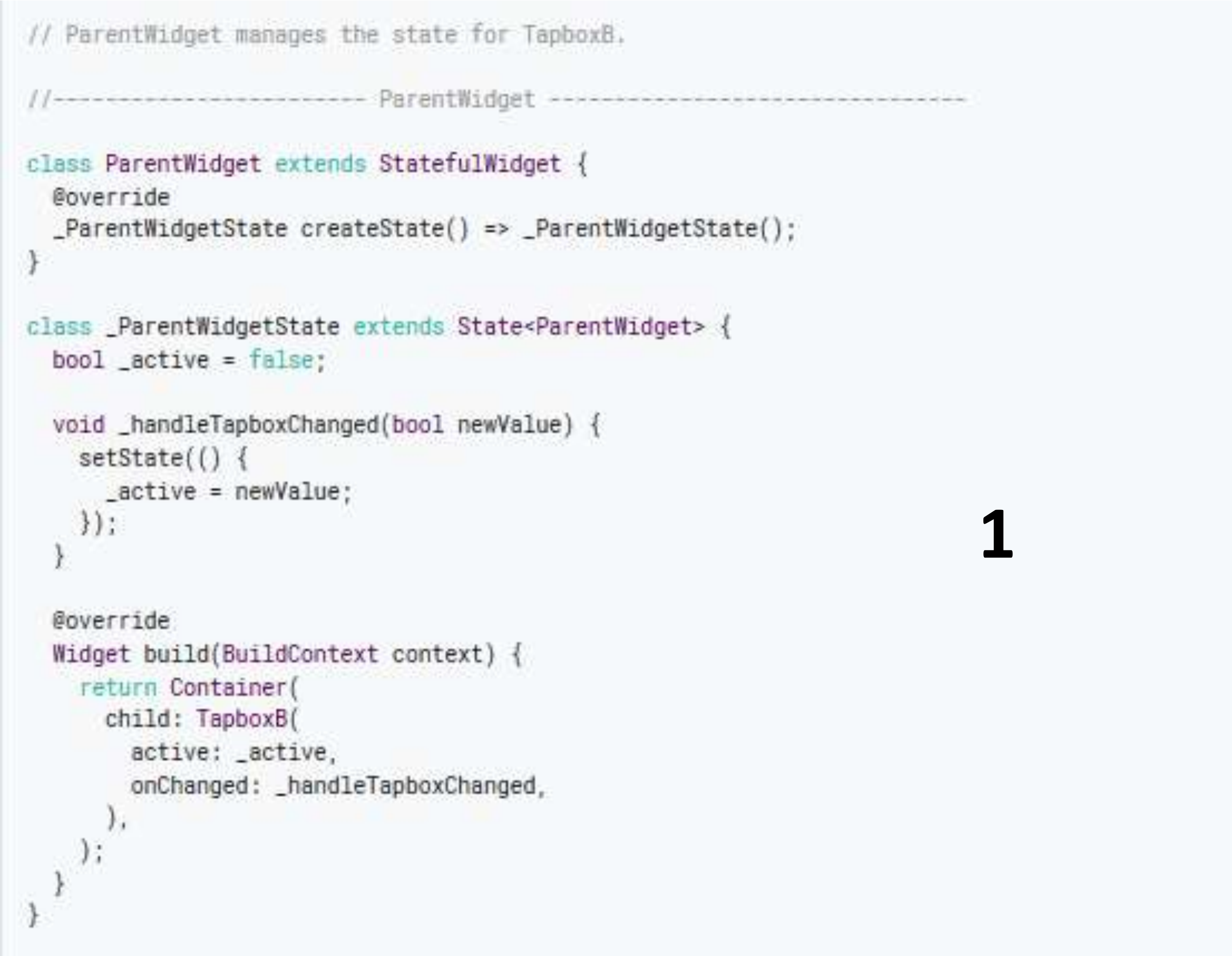
The parent widget manages the widget’s state

The TapboxB class:

- Extends StatelessWidget because all state is handled by its parent.
- When a tap is detected, it notifies the parent.

TECMAN LIMITED 2019

The widget manages its own state

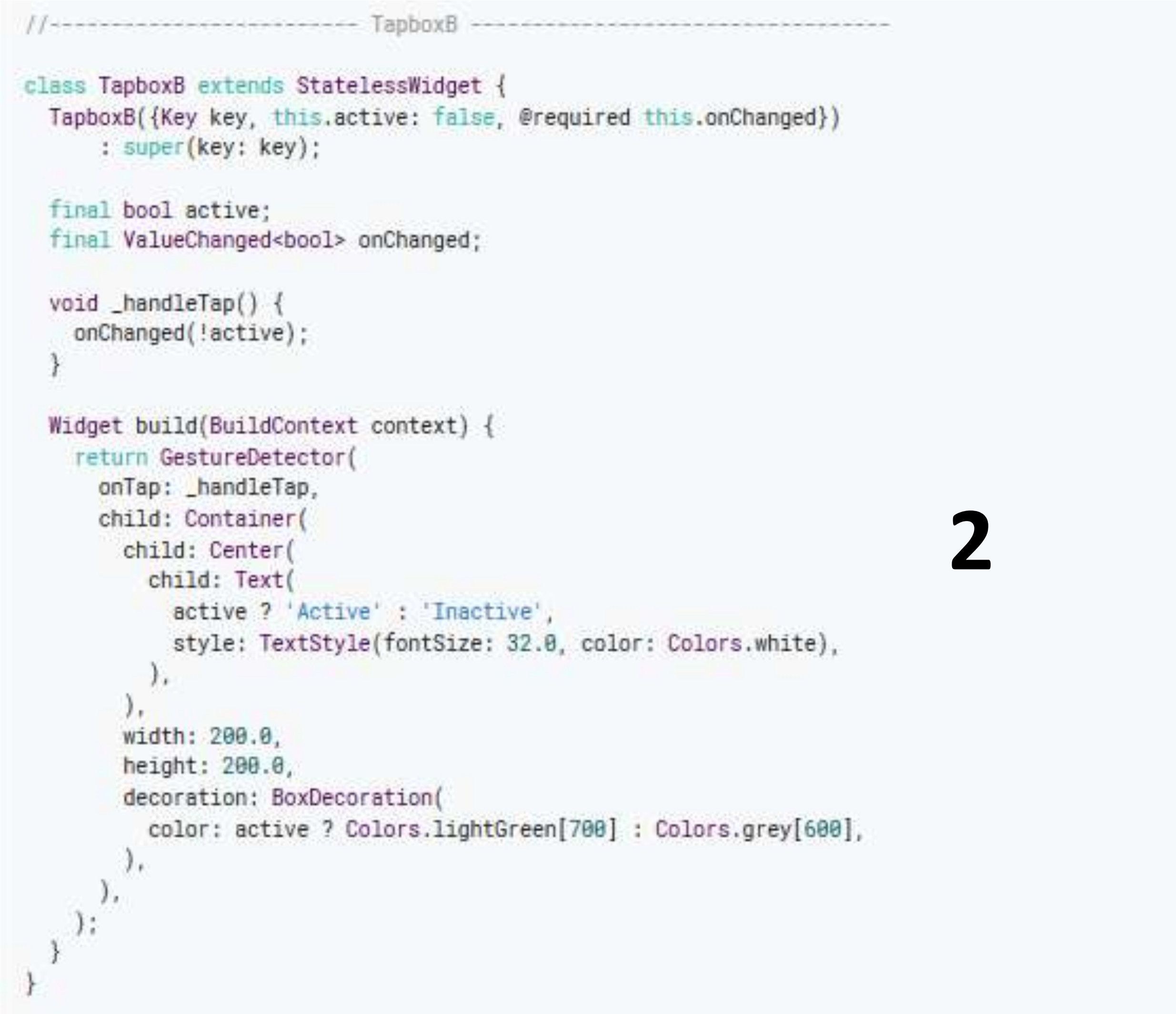


```
// ParentWidget manages the state for TapboxB.

//------------------------ ParentWidget --------------------------------

class ParentWidget extends StatefulWidget {
  @override
  _ParentWidgetState createState() => _ParentWidgetState();
}

class _ParentWidgetState extends State<ParentWidget> {
  bool _active = false;

  void _handleTapboxChanged(bool newValue) {
    setState(() {
      _active = newValue;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Container(
      child: TapboxB(
        active: _active,
        onChanged: _handleTapboxChanged,
      ),
    );
  }
}
```

1

TECMAN LIMITED 2019

The widget manages its own state

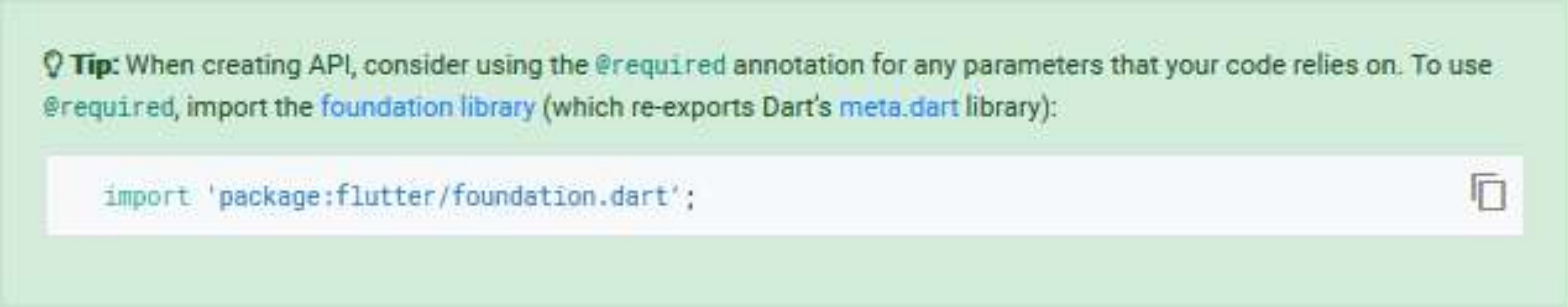


```
//------------------------ TapboxB ---------------------------------

class TapboxB extends StatelessWidget {
  TapboxB({Key key, this.active: false, @required this.onChanged})
      : super(key: key);

  final bool active;
  final ValueChanged<bool> onChanged;

  void _handleTap() {
    onChanged(!active);
  }

  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: _handleTap,
      child: Container(
        child: Center(
          child: Text(
            active ? 'Active' : 'Inactive',
            style: TextStyle(fontSize: 32.0, color: Colors.white),
          ),
        ),
        width: 200.0,
        height: 200.0,
        decoration: BoxDecoration(
          color: active ? Colors.lightGreen[700] : Colors.grey[600],
        ),
      ),
    );
  }
}
```

2

TECMAN LIMITED 2019

The widget manages its own state



💡 **Tip:** When creating API, consider using the `@required` annotation for any parameters that your code relies on. To use `@required`, import the foundation library (which re-exports Dart's meta.dart library):

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

TECMAN LIMITED 2019

A mix-and-match approach

For some widgets, a mix-and-match approach makes the most sense. In this scenario, the stateful widget manages some of the state, and the parent widget manages other aspects of the state.

In the TapboxC example, on tap down, a dark green border appears around the box. On tap up, the border disappears and the box’s color changes. TapboxC exports its _active state to its parent but manages its _highlight state internally. This example has two State objects, _ParentWidgetState and _TapboxCState.

TECMAN LIMITED 2019

A mix-and-match approach

The _ParentWidgetState object:

- Manages the _active state.
- Implements _handleTapboxChanged(), the method called when the box is tapped.
- Calls setState() to update the UI when a tap occurs and the _active state changes.

TECMAN LIMITED 2019

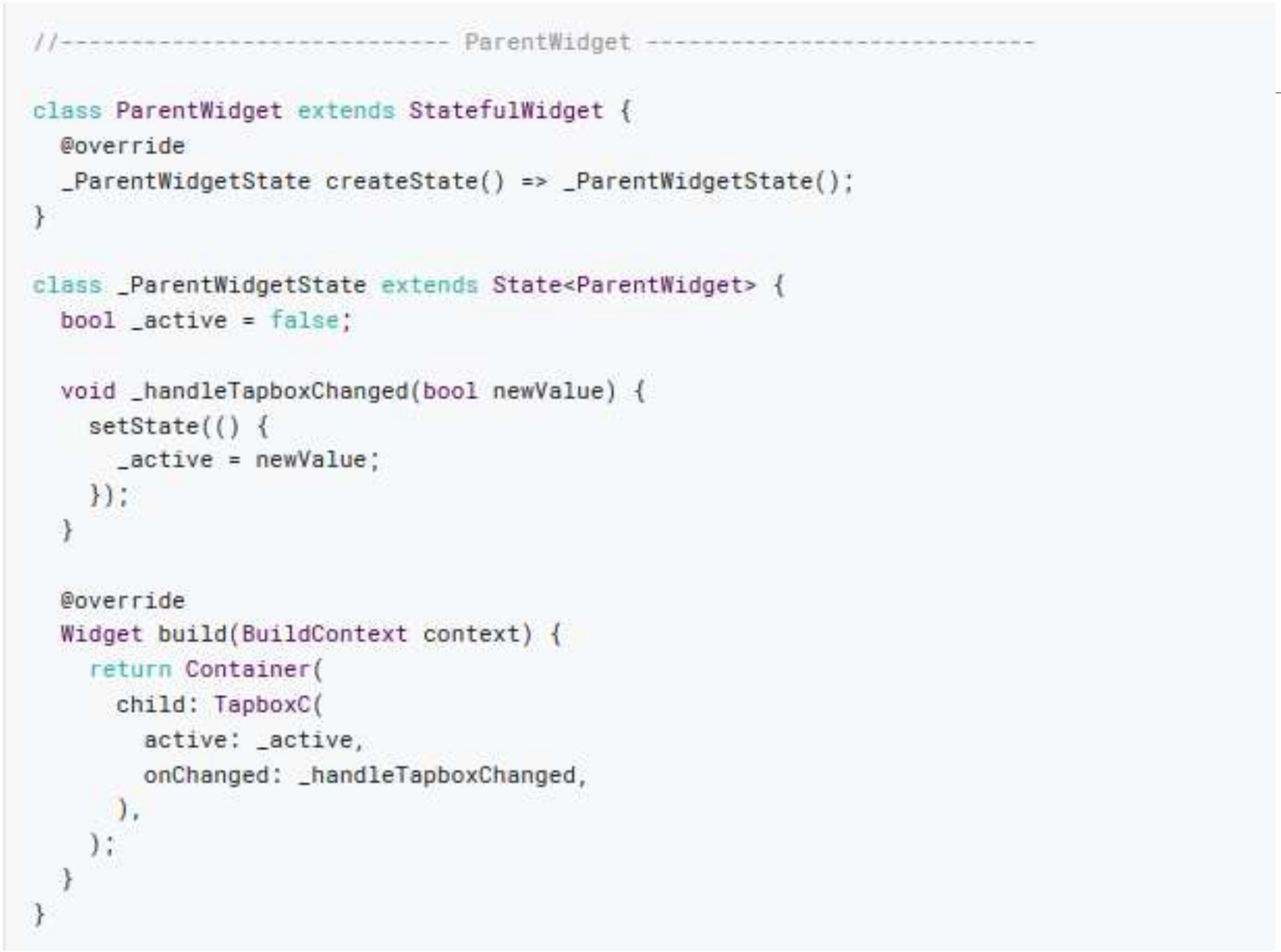
A mix-and-match approach

The _TapboxCState object:

- Manages the _highlight state.
- The GestureDetector listens to all tap events. As the user taps down, it adds the highlight (implemented as a dark green border). As the user releases the tap, it removes the highlight.
- Calls setState() to update the UI on tap down, tap up, or tap cancel, and the _highlight state changes.
- On a tap event, passes that state change to the parent widget to take appropriate action using the widget property.

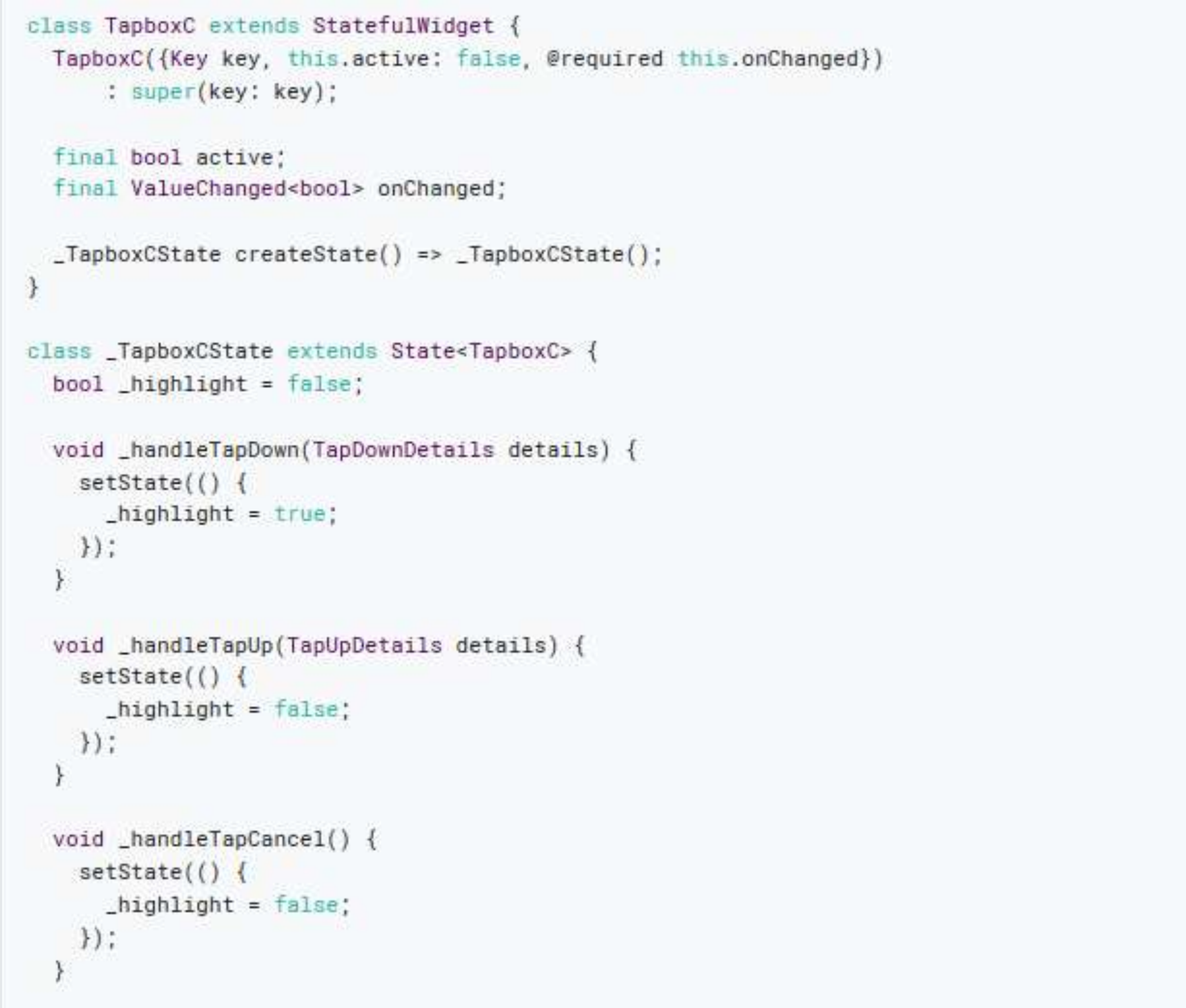
TECMAN LIMITED 2019

A mix-and-match approach



```
//--------------------------- ParentWidget ---------------------------

class ParentWidget extends StatefulWidget {
  @override
  _ParentWidgetState createState() => _ParentWidgetState();
}

class _ParentWidgetState extends State<ParentWidget> {
  bool _active = false;

  void _handleTapboxChanged(bool newValue) {
    setState(() {
      _active = newValue;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Container(
      child: TapboxC(
        active: _active,
        onChanged: _handleTapboxChanged,
      ),
    );
  }
}
```

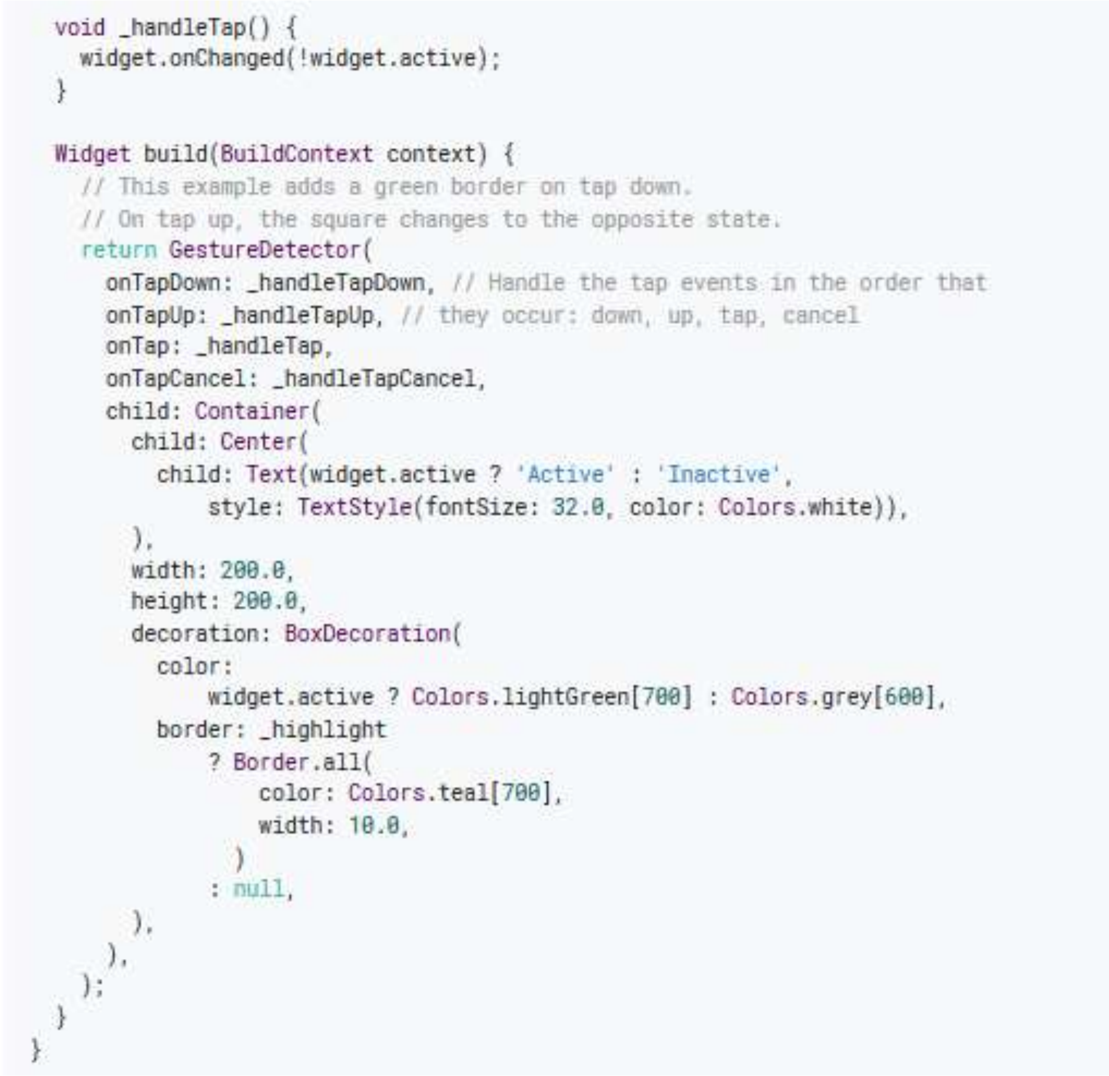
A mix-and-match approach



```
class TapboxC extends StatefulWidget {
  TapboxC({Key key, this.active: false, @required this.onChanged})
      : super(key: key);

  final bool active;
  final ValueChanged<bool> onChanged;

  _TapboxCState createState() => _TapboxCState();
}

class _TapboxCState extends State<TapboxC> {
  bool _highlight = false;

  void _handleTapDown(TapDownDetails details) {
    setState(() {
      _highlight = true;
    });
  }

  void _handleTapUp(TapUpDetails details) {
    setState(() {
      _highlight = false;
    });
  }

  void _handleTapCancel() {
    setState(() {
      _highlight = false;
    });
  }
```

TECMAN LIMITED 2019

A mix-and-match approach



```
  void _handleTap() {
    widget.onChanged(!widget.active);
  }

  Widget build(BuildContext context) {
    // This example adds a green border on tap down.
    // On tap up, the square changes to the opposite state.
    return GestureDetector(
      onTapDown: _handleTapDown, // Handle the tap events in the order that
      onTapUp: _handleTapUp, // they occur: down, up, tap, cancel
      onTap: _handleTap,
      onTapCancel: _handleTapCancel,
      child: Container(
        child: Center(
          child: Text(widget.active ? 'Active' : 'Inactive',
              style: TextStyle(fontSize: 32.0, color: Colors.white)),
        ),
        width: 200.0,
        height: 200.0,
        decoration: BoxDecoration(
          color:
              widget.active ? Colors.lightGreen[700] : Colors.grey[600],
          border: _highlight
              ? Border.all(
                  color: Colors.teal[700],
                  width: 10.0,
                )
              : null,
        ),
      ),
    );
  }
}
```

TECMAN LIMITED 2019

A mix-and-match approach

An alternate implementation might have exported the highlight state to the parent while keeping the active state internal, but if you asked someone to use that tap box, they'd probably complain that it doesn't make much sense. The developer cares whether the box is active. The developer probably doesn't care how the highlighting is managed, and prefers that the tap box handles those details.

TECMAN LIMITED 2019

Other interactive widgets

Flutter offers a variety of buttons and similar interactive widgets. Most of these widgets implement the Material Design guidelines, which define a set of components with an opinionated UI.

If you prefer, you can use GestureDetector to build interactivity into any custom widget. You can find examples of GestureDetector in Managing state, and in the Flutter Gallery.

Other interactive widgets

When you need interactivity, it’s easiest to use one of the prefabricated widgets. Here’s a partial list:

TECMAN LIMITED 2019

Standard widgets

Form

FormField

TECMAN LIMITED 2019

Material Components

- Checkbox
- DropdownButton
- FlatButton
- FloatingActionButton
- IconButton
- Radio
- RaisedButton
- Slider
- Switch
- TextField

TECMAN LIMITED 2019

THANK YOU

TECMAN LIMITED 2019