# Stateful Widgets

Written by Thapanapong Rukkanchanunt

# Outline

- Recap: Creating Simple Layout
- ListView vs Column
- Stateful Widget
- Managing States
- Benefit
- Summary

# Exercise: Creating Simple Layout

- How many widgets?
- How should we arrange them?
- Factorize your code!

**Oeschinen Lake Campground**

Kandersteg, Switzerland

41

CALL ROUTE SHARE

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.

```
return MaterialApp(
  title: 'Flutter layout demo',
  home: Scaffold(
    body: ListView(
      children: [
        Image.asset(...),  // Image.asset
        titleSection,
        buttonSection,
        textSection,
      ],
    ),  // ListView
  ),  // Scaffold
);  // MaterialApp
}
```

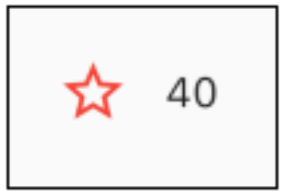
# ListView vs Column

- Listview Widget shows the unlimited number of children inside it, but the main advantage of using ListView is it renders only visible items on the screen.
- The column is used when we must list widgets vertically on the screen and SingleChildScrollView widget provides scroll functionality for Column widgets.

# Interactive Button

- Click to favorited or unfavorited
- Which part corresponds to favorite button?


Favorited




Not favorited

```
Widget titleSection = Container(
  padding: const EdgeInsets.all(32),
  child: Row(
    children: [
      Expanded(...),  // Expanded
      Icon(...),  // Icon
      const Text('41'),
    ],
  ),  // Row
);  // Container
```

**Oeschinen Lake Campground**

Kandersteg, Switzerland

41

CALL ROUTE SHARE

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.

# Creating Stateful Widget

- When you click a button, what change?
- All things that change should be in the same widget
- 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.

# FavoriteWidget class

- 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.

```
class FavoriteWidget extends StatefulWidget {
  const FavoriteWidget({super.key});

  @override
  State<FavoriteWidget> createState() => _FavoriteWidgetState();
}
```

# _FavoriteWidgetState Class

- 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

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

  @override
  Widget build(BuildContext context) {...}
```

# Implement build()

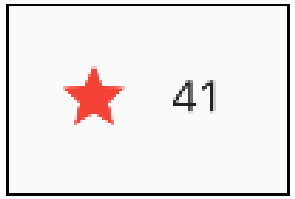
- We can simply mimic original widget to the new one
- Change Icon to IconButton so we can attach onPressed action
- Notice that Icon parameter is conditional on isFavorited variable

```
Widget titleSection = Container(
  padding: const EdgeInsets.all(32),
  child: Row(
    children: [
      Expanded(...),  // Expanded
      const FavoriteWidget(),
    ],
  ),  // Row
);  // Container
```

```
@override
Widget build(BuildContext context) {
  return Row(
    children: [
      IconButton(
        onPressed: () {  },
        icon: (_isFavorited
            ? const Icon(Icons.star)
            : const Icon(Icons.star_border)),
        color: Colors.red,
      ),  // IconButton
      Text('$_favoriteCount')
    ],
  );  // Row
}
```

# Implement onPressed action

- Call setState() and write your code inside it
- You can write a separate function (within the class) for readability.
- Test your code!




Favorited


Not favorited

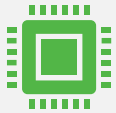
```
onPressed: () {
  setState(() {
    if (_isFavorited) {
      _favoriteCount -= 1;
      _isFavorited = false;
    } else {
      _favoriteCount += 1;
      _isFavorited = true;
    }
  });
},
```

```
onPressed: () {
  setState(_toggleFavorite);
},
```

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

# Managing States

Just now we create a stateful widget that manages its own state



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:

The widget manages its own state

The parent manages the widget's state

A mix-and-match approach

# Managed by Parent

- The ParentWidgetState class:
  - Manages the _isFavorite state and _favoriteCount for FavoriteWidget.
  - Implements _handleFavoritedChanged(), method called when the button is tapped.
  - When called FavoriteWidget, attach the function to onChanged parameter.
  - When the state changes, calls setState() to update the UI.

```
FavoriteWidget(
  isFavorited: _isFavorited,
  favoriteCount: _favoriteCount,
  onChanged: _handleFavoritedChanged,
),  // FavoriteWidget
```

```
void main() {
  runApp(const ParentWidget());
}

class ParentWidget extends StatefulWidget {
  const ParentWidget({super.key});

  @override
  State<StatefulWidget> createState() => _ParentWidgetState();
}

class _ParentWidgetState extends State<ParentWidget> {
  bool _isFavorited = false;
  int _favoriteCount = 0;

  @override
  Widget build(BuildContext context) {...}

  void _handleFavoritedChanged(bool newValue) {...}
```

# Managed by Parent

- The FavoriteWidget class:
  - Extends StatelessWidget because all state is handled by its parent.
  - Need onChanged variable, a signature for callbacks that report that an underlying value has changed.
  - When a button is pressed, it notifies the parent via onChanged.

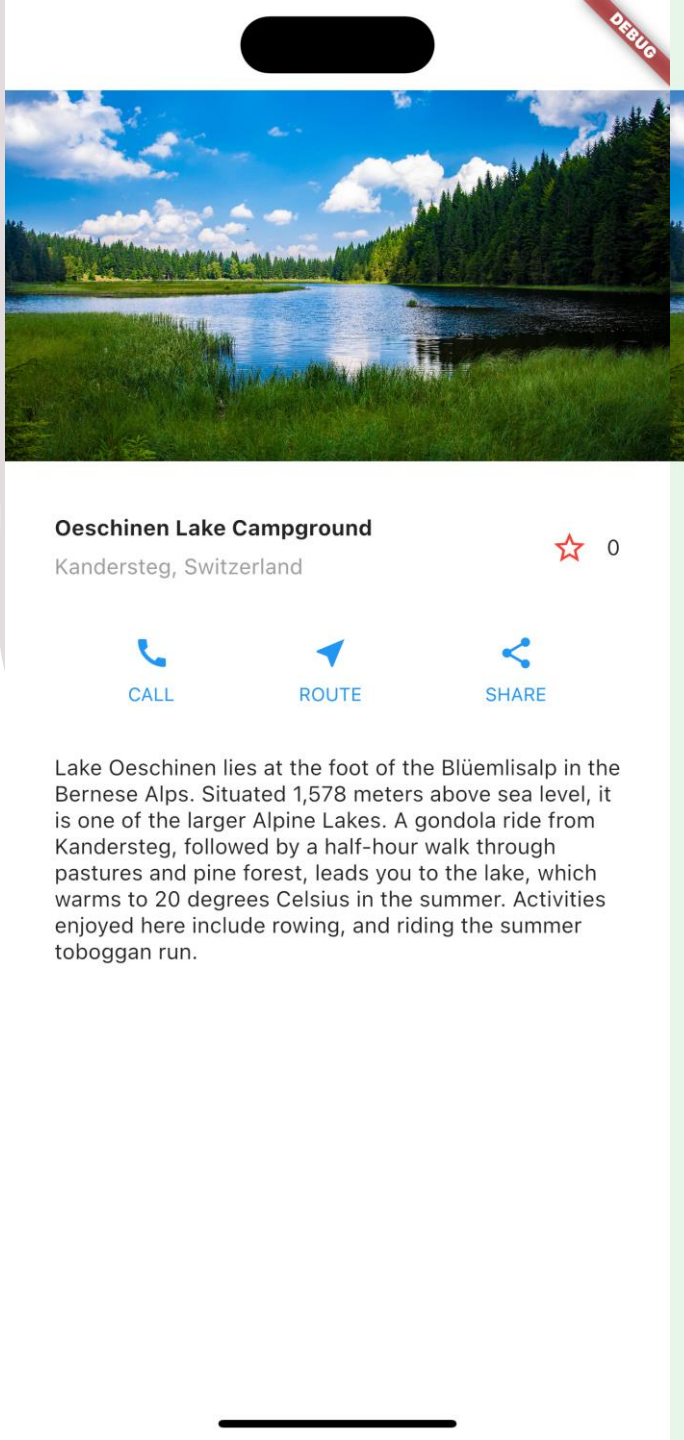
```
IconButton(
  onPressed: _handleTap,
  icon: isFavorited
      ? const Icon(Icons.star)
      : const Icon(Icons.star_border),
  color: Colors.red,
),  // IconButton
```

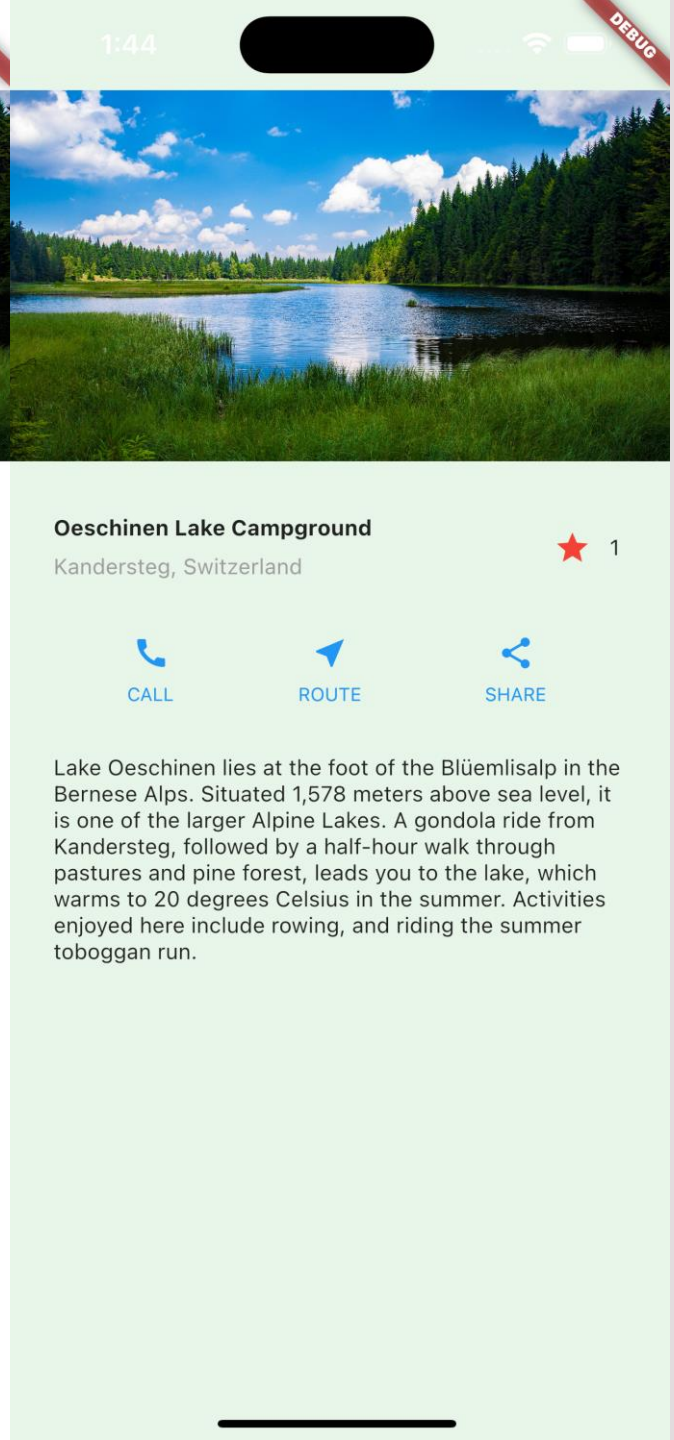
```
class FavoriteWidget extends StatelessWidget {
  const FavoriteWidget({
    super.key,
    this.isFavorited = false,
    this.favoriteCount = 0,
    required this.onChanged});

  final bool isFavorited;
  final ValueChanged<bool> onChanged;
  final int favoriteCount;

  @override
  Widget build(BuildContext context) {...}

  void _handleTap() {
    onChanged(!isFavorited);
  }
}
```

# Benefit

- Supposed we want to change color theme when favorited.
- Because parent manages its children state, we can reuse the variable for other children (or for parent itself).

```
backgroundColor: _isFavorited
    ? Colors.green[50]
    : Colors.white,
```



# Summary

- A widget is either stateful or stateless. If a widget can change upon user interactions, it is stateful.
- A stateless widget never changes. Icon, IconButton, and Text are examples of stateless widgets. Stateless widgets subclass StatelessWidget.
- 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.
- 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 favorite status. When the widget's state changes, the state object calls setState(), telling the framework to redraw the widget.

# Things to Explore

- Advanced UI: Gesture
  - https://docs.flutter.dev/development/ui/advanced/gestures
  - https://docs.flutter.dev/development/ui/widgets-intro#handling-gestures
- Advanced UI: Splash Screen
  - https://docs.flutter.dev/development/ui/advanced/splash-screen