TIERS LIMITED SUMMER INTERNSHIP 2024

MOBILE APP DEVELOPMENT

Stateful Class and Stateful Widgets

CLASS # 5

Overview

In this class, we will cover the fundamental concepts of stateful classes and stateful widgets in Flutter. Understanding state management and the role of stateful widgets is crucial for creating interactive and dynamic user interfaces. We will also explore several important stateful widgets and their usage.

Objectives

- Understand the difference between stateless and stateful widgets.
- Learn how to create and manage stateful widgets.
- Explore common stateful widgets such as Checkbox, Switch, TextField, and TextFormField.
- Implement state management using setState.
- Understand the lifecycle methods of stateful widgets: **`initState`**, **`setState`**, and **`dispose`**.

Understanding Stateful Widgets

Stateful Widgets:

Stateful widgets are widgets that can change their state during the lifetime of the widget. They are dynamic and can be updated in response to user interactions or other events.

Stateless vs. Stateful Widgets:

- **Stateless Widgets**: Immutable, meaning their properties cannot change once they are built. Examples include Text, Icon, and Container.
- **Stateful Widgets**: Mutable, meaning their properties can change during their lifetime. Examples include Checkbox, Switch, TextField, and TextFormField.

Creating a Stateful Widget:

A stateful widget consists of two classes:

1. The stateful widget class, which extends **`StatefulWidget`**.
2. The state class, which extends **`State<T>`** where **`T`** is the type of the stateful widget.

Example:

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

  @override
  State<MyStatefulWidget> createState() => _MyStatefulWidgetState();
}
```

```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Stateful Widget Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text('This is a stateful widget.'),
          ],
        ),
      ),
    );
  }
}
```

- **`class MyStatefulWidget extends StatefulWidget:`** Defines a new stateful widget class named **`MyStatefulWidget`**.
- **`const MyStatefulWidget({super.key}):`** The constructor of the **`MyStatefulWidget`** class. Using **`const`** makes the widget immutable.
- **`State<MyStatefulWidget> createState() => _MyStatefulWidgetState():`** Creates the state for this widget.
- **`class _MyStatefulWidgetState extends State<MyStatefulWidget>:`** The state class associated with **`MyStatefulWidget`**.
- **`Widget build(BuildContext context):`** The build method defines the widget's UI.
- **`Scaffold`**: A top-level container providing a structure for the material design layout, including an app bar and body content.

Lifecycle Methods:

1. **initState**

   The **`initState`** method is called once when the stateful widget is inserted into the widget tree. It is used to initialize any data or state before the widget is built.

   Example:

   ```
   @override
   void initState() {
     super.initState();
     // Initialization code here
   }
   ```

2. **setState:**

   The **`setState`** method is used to update the state of the widget. It triggers a rebuild of the widget subtree, updating the UI with the new state.

Example:

```
void _updateState() {
  setState(() {
    // Update the state here
  });
}
```

3. **dispose**

   The `dispose` method is called when the stateful widget is removed from the widget tree permanently. It is used to clean up any resources (e.g., controllers, listeners) that were created in `initState`.

   Example:

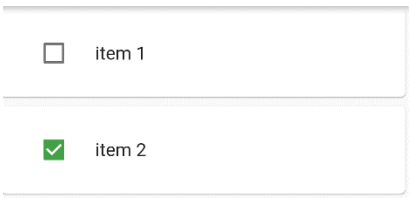
```
@override
void dispose() {
  // Clean up resources here
  super.dispose();
}
```

Common Stateful Widgets

1. **Checkbox:**

   - A checkbox is a material design checkbox. The checkbox itself does not maintain any state; instead, when the state of the checkbox changes, the widget calls the onChanged callback.
   - Example:

```
bool _isChecked = false;

Checkbox(
  value: _isChecked,
  onChanged: (bool? value) {
    setState(() {
      _isChecked = value!;
    });
  },
);
```



2. **Switch:**

   - A switch is a widget that can be toggled on or off. It is used to represent two mutually exclusive states.

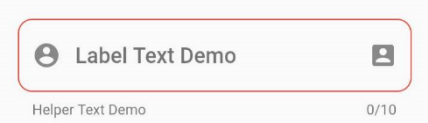
- Example:

```
bool _isSwitched = false;

Switch(
  value: _isSwitched,
  onChanged: (bool value) {
    setState(() {
      _isSwitched = value;
    });
  },
);
```

3. TextField:

- A text field allows the user to enter text into an app. It appears in forms and dialogs.
- Example:

```
String _text = "";

TextField(
  onChanged: (String value) {
    setState(() {
      _text = value;
    });
  },
);
```



4. TextFormField:

- The **`TextFormField`** widget is a convenience widget that wraps a **`TextField`** and integrates it with the `Form` widget. It provides additional functionality for validation and form submission.
- Example:

```
final _formKey = GlobalKey<FormState>();
String _name = '';

TextFormField(
  decoration: InputDecoration(labelText: 'Name'),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your name';
    }
    return null;
  },
  onSaved: (value) {
    _name = value!;
  },
);
```

Difference Between TextField and TextFormField

- **TextField**: A simple text input field. It does not have built-in validation or form submission features.
- **TextFormField**: A more advanced version of TextField that integrates with the Form widget, providing built-in validation and form submission features.

Exercises:

1. **Exercise 1: Create a Stateful Counter**
    - Create a stateful widget that has a counter.
    - Add a button that increments the counter when pressed.
2. **Exercise 2: Implement a Checkbox**
    - Create a stateful widget with a checkbox.
    - Update a text widget based on the checkbox state.
3. **Exercise 3: Create a Switch and Display State**
    - Create a stateful widget with a switch.
    - Display the current switch state in a text widget.
4. **Exercise 4: Implement a TextField**
    - Create a stateful widget with a text field.
    - Display the entered text below the text field.
5. **Exercise 5: Implement a TextFormField with Validation**
    - Create a stateful widget with a TextFormField.
    - Add validation to ensure the input is not empty.
    - Display the input below the form.