ScopeTreasure Courses

FLUTTER COURSES · LESSON F.3 · CODE SCOPE SERIES

Flutter Widgets Deep — Stateless vs Stateful

The two widget kinds: Stateless (never changes) vs Stateful (changes with setState) — with a live counter app.

Introduction

Every widget you will ever write belongs to one of two families. A **StatelessWidget** is built once and never changes — a label, an icon, a title. A **StatefulWidget** carries **state** — data that can change while the app runs, like a counter number or a ticked checkbox — and redraws itself when that data changes. Choosing the right family for each piece is *the* daily decision of Flutter work, and today you learn it with a live counter. *(Roman Urdu: Stateless = kabhi nahi badalta; Stateful = state badle to screen khud refresh.)*

Key points

- A **StatelessWidget** has only its `build` method: give it its settings (through the constructor, with named parameters) and it draws the same thing forever. Most widgets on most screens — text, icons, cards, layouts — are stateless.
- A **StatefulWidget** comes in **two classes**: the widget class itself (settings) and a **State class** (the changing data + the `build` method). `createState()` connects them. This split looks odd for one day, then becomes normal.

  (Roman Urdu: Stateful widget do hisson mein aata hai — widget khud, aur us ki State.)
- `setState(() { ... })` is the bell you ring to tell Flutter "the data changed — rebuild this widget". Change the data *inside* the `setState` call, and the screen refreshes itself automatically. Change data *without* `setState` and the screen shows the old value — the number changed in memory, but nobody told the screen.
- The `build` method runs again on every `setState`. That is why `build` must be *fast and pure*: it describes the screen from the current data, nothing more. Heavy work lives outside it.
- **The decision question** for every widget: *"Will anything on this piece change while the user looks at it?"* No → Stateless. Yes → Stateful. A page can be stateful while its children are stateless — state lives only where change lives.
- Both families take settings the same Dart way: constructor + named parameters + `super.key`, e.g. `const MyTitle({super.key})`. The `const` before a widget whose settings never change lets Flutter skip rebuilding it — free speed.

Code examples

Example 1 — A StatelessWidget of your own

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Stateless Demo')),
        body: const Center(child: MyTitle()),
      ),
    );
  }
}

class MyTitle extends StatelessWidget {
  const MyTitle({super.key});

  @override
  Widget build(BuildContext context) {
    return const Text(
      'Yeh text kabhi nahi badlega',
      style: TextStyle(fontSize: 22),
    );
  }
}
```

What it does: `MyTitle` is a class extending `StatelessWidget`. Its whole job is the `build` method, which returns a styled `Text`. `MyApp` uses it exactly like a built-in widget — *your* widgets and Flutter's widgets follow the same rules; that is the deepest idea in Flutter.

What you will see: the title bar "Stateless Demo" and the line "Yeh text kabhi nahi badlega" in the centre. Wait as long as you like — it will never change, because nothing in this app *can* change. That is statelessness you can see.

Example 2 — A StatefulWidget: the counter, understood

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(home: CounterPage());
  }
}

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

  @override
  State<CounterPage> createState() => _CounterPageState();
}

class _CounterPageState extends State<CounterPage> {
  int count = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Stateful Demo')),
      body: Center(
        child: Text('$count', style: const TextStyle(fontSize: 48)),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          setState(() {
            count++;
          });
        },
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

What it does: The State class holds one field — `count`. The button's `onPressed` runs `setState`, inside which `count` grows by 1. Flutter then re-runs `build`, and the `Text` is rebuilt showing the new number. Data changed *inside* setState → screen followed. That loop is the heartbeat of every interactive app.

What you will see: a "0" in the centre and a round **+** button at the bottom corner. Tap: 1. Tap: 2. Tap: 3. Nothing else on the screen moves — only the piece whose data changed is rebuilt. Remove the `setState` wrapper (just `count++;`) and tap again: the screen freezes at 0, the classic beginner bug, now impossible for you to make unknowingly.

Example 3 — Famous widgets, and a stateless one you reuse

```
class SubjectChip extends StatelessWidget {
  const SubjectChip({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: const EdgeInsets.all(12),
      decoration: BoxDecoration(
        color: Colors.teal.shade50,
        borderRadius: BorderRadius.circular(12),
      ),
      child: const Row(
        mainAxisSize: MainAxisSize.min,
        children: [
          Icon(Icons.book, color: Colors.teal),
          SizedBox(width: 8),
          Text('Computer Science'),
        ],
      ),
    );
  }
}
```

What it does: A reusable chip built from famous widgets: `Container` (padding + soft background + rounded corners — F.4 goes deep), a `Row` (side-by-side layout), an `Icon`, a `SizedBox` (a gap of 8), and a `Text`. Add this class to Example 1's file and put `SubjectChip()` in the body: it works instantly, because it is stateless and needs nothing from outside.

What you will see: a small pale-teal rounded chip: a book icon and "Computer Science" side by side. You designed a widget once and can now drop it ten times on ten screens — the reuse that Lesson D.8 promised, delivered.

Quick recap

- Stateless = built once, never changes. Stateful = holds state, rebuilds on `setState`.

- StatefulWidget = widget class + State class, joined by `createState()`.
- Data changes belong **inside** `setState(() { ... })` — outside it, the screen never hears the news.
- `build` re-runs on every setState — keep it fast and purely descriptive.
- Decision: "Will it change while the user looks?" No → Stateless. Yes → Stateful.

Common mistakes

1. **Changing state without setState** — the value changes in memory, the screen shows the old one, and the beginner debugs the wrong code for an hour. Any field that appears in `build` must only ever change inside `setState`.
2. **Making everything Stateful "to be safe"** — state is weight: more classes, more chances to err. Stateless is the default; a widget *earns* statefulness only when something on it truly changes.
3. **Doing heavy work inside build** — reading files, long calculations, network calls inside `build` run again on *every* rebuild and make the app stutter. `build` only describes; work happens in button handlers and lifecycle methods.

Practice

1. Extend Example 2: add a second button (a `FloatingActionButton` is one button — instead, put two `ElevatedButton`s in a `Column` in the body) for **minus** and **reset**, each using setState correctly. The count must never go below 0.
2. Convert Example 1's `MyTitle` to take a setting: `const MyTitle({super.key, required this.text})` with a `final String text;` field, and show two different titles on one screen using a Column.

Practice quiz

1. What is a StatelessWidget?

- A widget that changes constantly
- A widget with only a build method: give it its settings and it draws the same thing forever
- A widget with two classes
- A widget that cannot be reused

Answer: A widget with only a build method: give it its settings and it draws the same thing forever

2. What are the two classes of a StatefulWidget?

- The widget class and its State class, joined by createState()
- Two widget classes with the same name
- A widget and its parent
- A class and its test

Answer: The widget class and its State class, joined by createState()

3. What happens if you change data without setState?

- The screen updates at once
- The value changes in memory, but the screen shows the old value, nobody told the screen
- The app crashes
- The widget becomes stateless

Answer: The value changes in memory, but the screen shows the old value, nobody told the screen

---

Watch the video lesson: https://youtu.be/Rqo4gthRHE4

Read online: https://scopetreasure.com/notes/flutter-widgets-stateless-stateful · More free tools and notes at https://scopetreasure.com