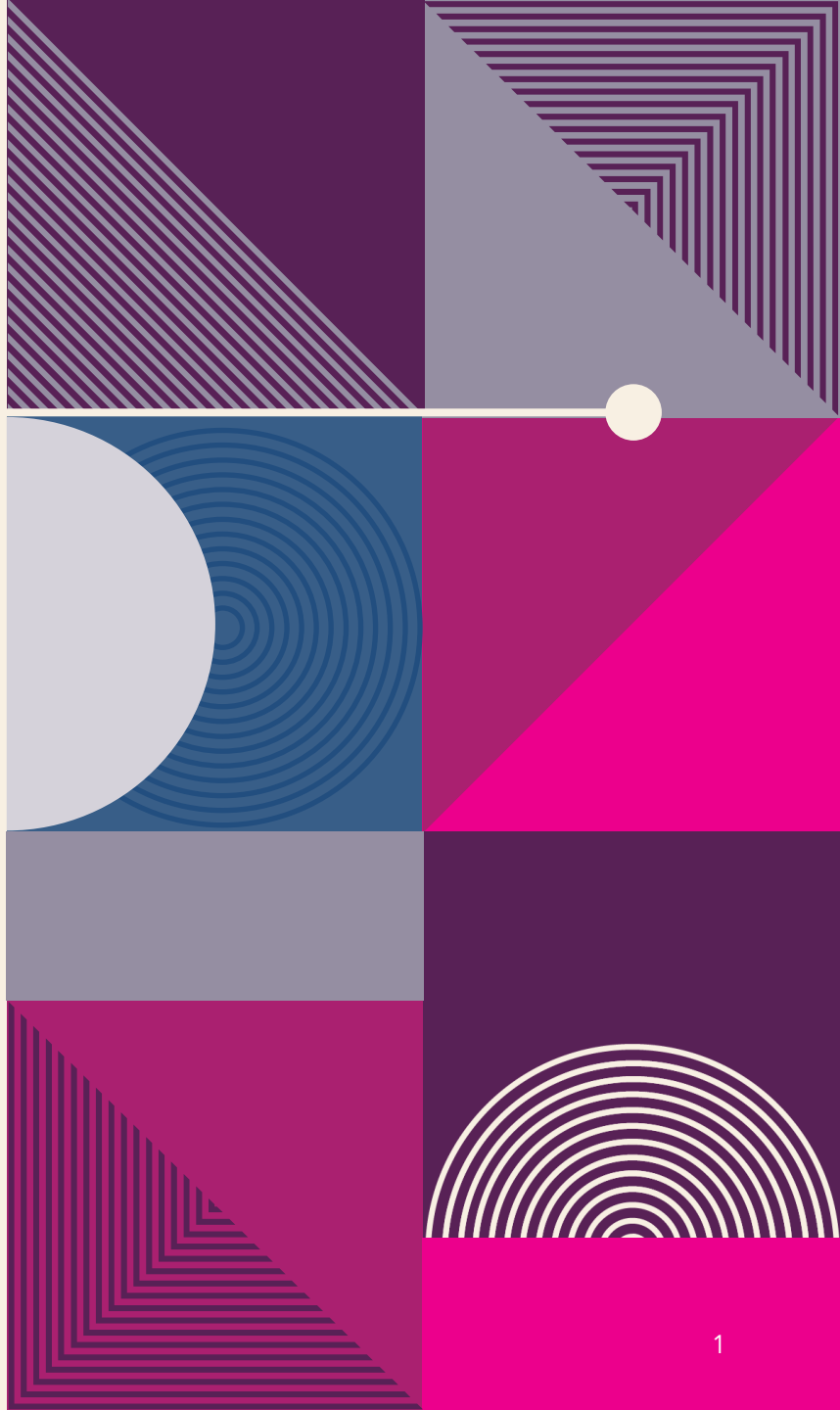
Stateless and Stateful widgets



Flutter has two core types of widgets: Stateless and Stateful. These two widgets are the building blocks of every widget that flutter provides.

Stateless widgets

- A stateless widget cannot change its state during the runtime of a Flutter application. That means a stateless widget cannot be redrawn while the app is in action. For that reason, the appearance and properties remain unchanged throughout the lifetime of the widget.

Example of a stateless widget

```
class StatelessScreen extends StatelessWidget{
@override
   Widget build(BuildContext context) {
       return MaterialApp(
         home: Scaffold(
           appBar: AppBar(
              title: Text('LogRockets'),
              backgroundColor: Colors.blueGrey[600],
             ),
       backgroundColor: Colors.white,
       body: Container(),
),
);
```

Example of a stateless widget

- We can see that in the code, the name of the stateless widget is StatelessScreen and it is overriding the build method. The build method takes the BuildContext as a parameter and returns a widget.

- We use a stateless widget when we create an application that isn't required to redraw a widget again and again.
  For example, when we are creating an AppBar, a stateless widget can be scaffolding or icons that do not need to be changed.

- A stateless widget class is called once, only when it is initialized. Even if an external force acts on it, it won't be updated.

- Whenever the StatelessScreen widget is initialized, the build method is called. After that, the widget will be printed on the screen

Stateful widgets

- A stateful widget is used when some part of the UI must change dynamically during runtime. Stateful widgets can redraw themselves multiple times while the app is running

- . Stateful widgets are useful when the part of the UI we are describing changes dynamically. If we create a button widget that updates itself each time a user clicks that button, that is a stateful widget.

Example of a stateful widget

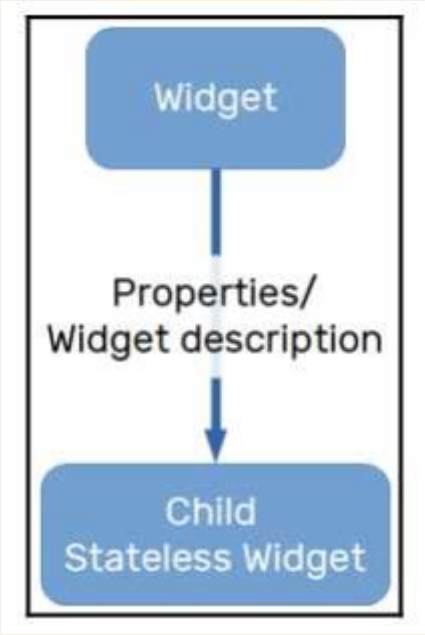
```
class StatefulScreen extends StatefulWidget{
@override
_StatefulScreenState createState() => _StatefulScreenState();
class _StatefulScreenState extends State<StatefulScreen> {
String title = 'Original title';
@override
Widget build(BuildContext context) {
return Column(children: [ Text(title),
RaisedButton(
child: Text('Click'),
onPressed: () {
setState(() {
title = 'Changed title';
});
})
]);
```

Example of a stateful widget

- We created a text field and a button widget.
- Once we call this widget and press the button, we let the value of the text field change automatically.
- In this type of application, we can do that by the implementation of setState(). setState() is a method that is called within stateful widget classes. This method changes the value of a stateful widget each time it is called.

Stateless widgets

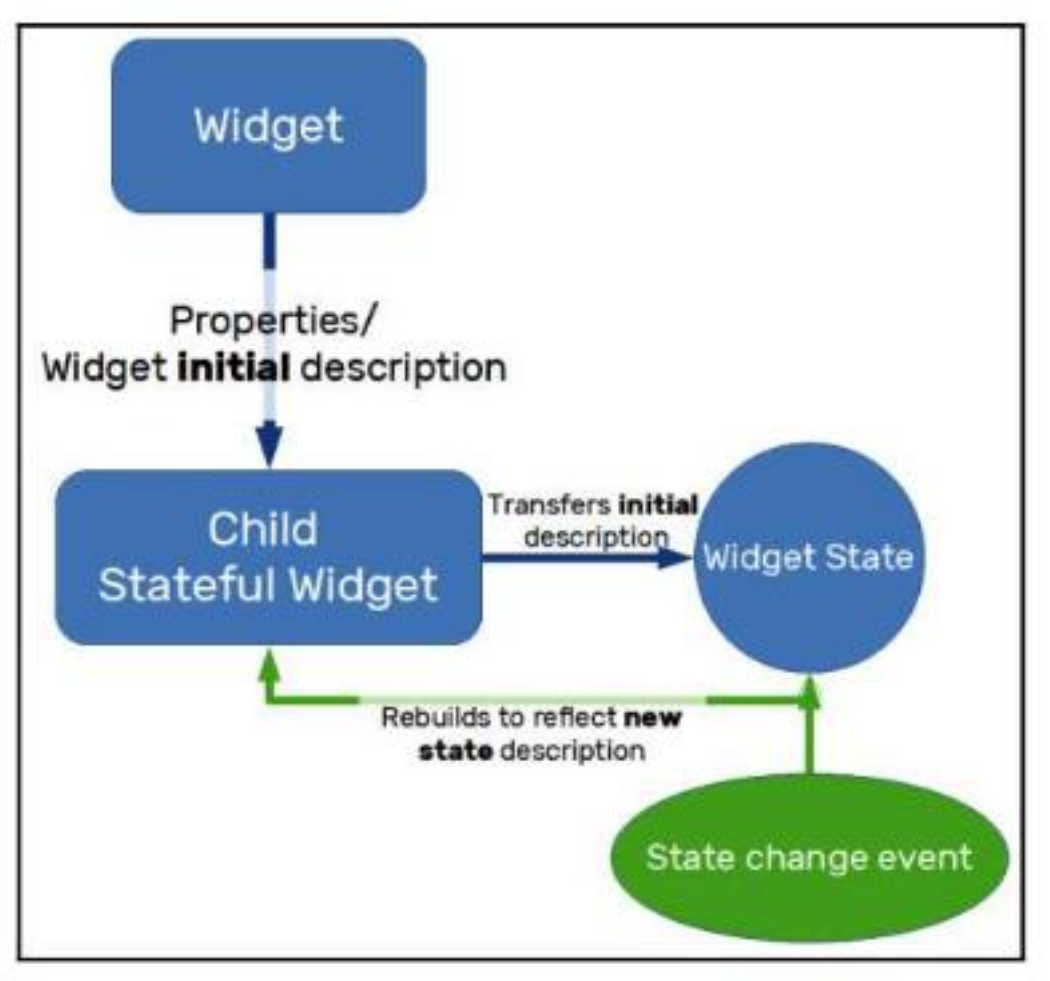
- A typical UI will be composed of many widgets, and some of them will never change their properties after being instantiated. They do not have a state; that is, they do not change by themselves through some internal action or behavior. Instead, they are changed by external events on parent widgets in the widgets tree. So, it's safe to say that stateless widgets give control of how they are built to some parent widget in the tree. The following is a representation of a stateless widget:



The child widget will receive its description from the parent widget and will not change it by itself. In terms of code, this means that stateless widgets have only final properties defined during construction, and that's the only thing that needs to be built on the device screen.

Stateful widgets

- Unlike stateless widgets, which receive a description from their parents that persist during the widgets' lifetime, stateful widgets are meant to change their descriptions dynamically during their lifetimes. By definition, stateful widgets are also immutable, but they have a company State class that represents the current state of the widget.



Widgets have methods

A function that's defined inside of a class declaration is called a *method.* Every class has a build method. That's good because there's some fine print in the code for StatelessWidget. According to that fine print, every class that extends StatelessWidget must contain the declaration of a build method.

A stateless widget's build method tells Flutter how to build the widget. Among other things, the method describes the widget's look and behavior. Whenever you launch the previous program in Flutter calls class's build method.

That build method constructs a MaterialApp instance, which, in turn, constructs a MyHomePage instance. And so on. From that point onward, the MaterialApp instance doesn't change. Yes, things inside the MaterialApp instance change, but the instance itself doesn't change.

How often does your town build a new traffic light assembly? may see one going up every two years or so. The metal part of a traffic light isn't designed to change regularly. The town planners call the traffic light assembly's build method only when they construct a new light. The same is true of stateless widgets in Flutter. A stateless widget isn't designed to be changed. When a state- less widget requires changing, Flutter replaces the widget.

What about stateful widgets? Do they have build methods? Well, they do and they don’t. Every stateful widget has to have a createState method. The createState method makes an instance of Flutter’s State class, and every State class has its own build method. In other words, a stateful widget doesn’t build itself. Instead, a stateful widget creates a state, and the state builds itself.

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

void main() => runApp(App1());
class App1 extends StatelessWidget {
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyHomePage(),  );  } }
class MyHomePage extends StatefulWidget {
  _MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State {
  String _pressedOrNot = "You haven't pressed the button.";
  void _changeText() {
    setState(_getNewText);  }
  void _getNewText() {
    _pressedOrNot = "You've pressed the button.";   }
  Widget build(BuildContext context) {
    return Scaffold(
        body: Center(
          child: Text(
            _pressedOrNot,     ),         ),
        floatingActionButton: FloatingActionButton(
          onPressed: _changeText,  ));
  } }
```

A stateless widget builds itself

A stateful widget creates a state, and ….

…. the state builds itself

Here's a description:

» The Dart language calls the main function when the code starts running.

The main function constructs an instance of App1 and calls runApp to get things going. Then . . .

» The Flutter framework calls the App1 instance's build function.

The build function constructs an instance of MyHomePage. Then . . .

» The Flutter framework calls the MyHomePage instance's createState function.

The createState function constructs an instance of _myHomePageState. Then . . .

» The Flutter framework calls the _myHomePageState instance's build function.

The build function constructs a Scaffold containing a Center with a Text widget and a FloatingActionButton widget.