

## Flutter: Stateless widget and Stateful Widget

### ❖ What is State?

- The State is information that can read synchronously when the widget is built and might change during the lifetime of the widget.
- Classes that inherit “Stateful Widget” are immutable. But the state is mutable.

As you know in Flutter all the UI components are known as widgets. The widget which contains the code for a single screen of the app can be just of two types —

1. *Stateful*
2. *Stateless*

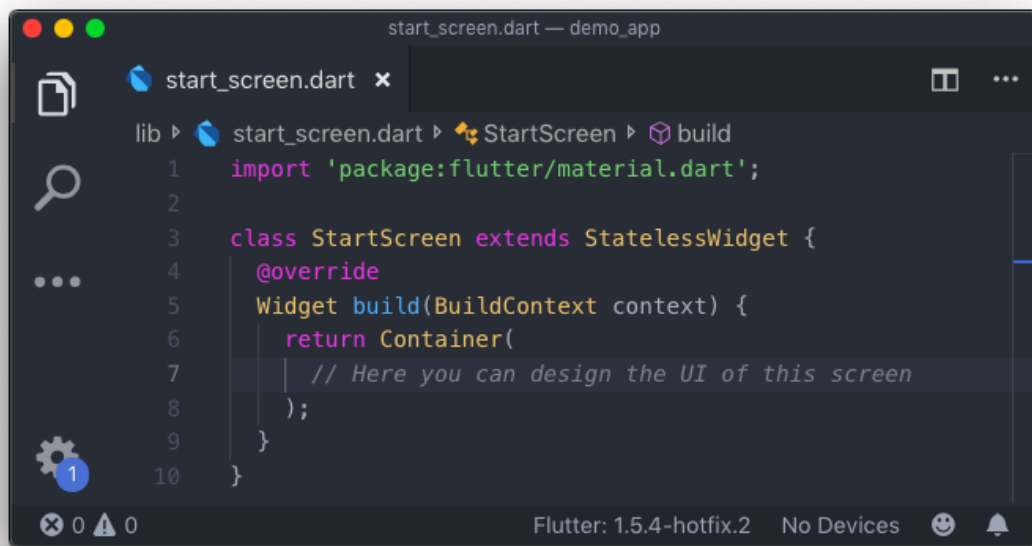
Let's discuss how do they differ.

### Stateless

Stateless widgets do not require mutable state, i.e., it is **immutable**.

In simple words, Stateless widgets cannot change their state during the runtime of the app, which means the widgets cannot be redrawn while the app is in action.

The structure of a Stateless widget looks like this:



## Stateless Widget

So, let's understand what is there in this small code snippet.

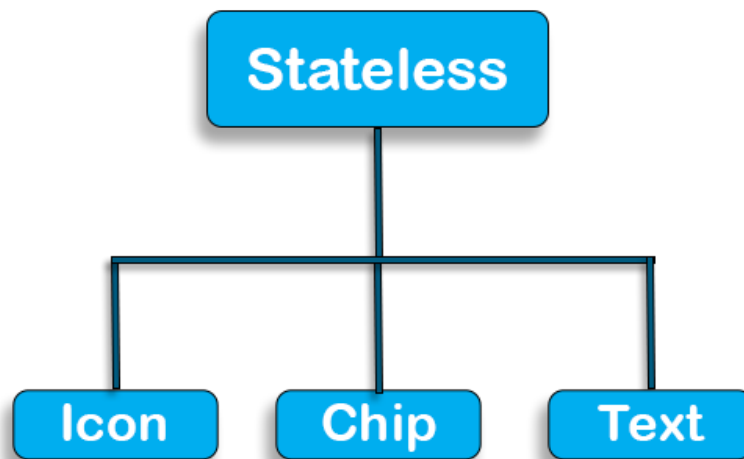
The name of this Stateless Widget is “**StartScreen**”, inside which we have to override the “**build**” method. This build method takes in a “**BuildContext**” as the parameter and returns a widget. That's why you can see that the return type of the build method is a widget. And this the place where you can design the UI of this screen, which is Stateless.

In Stateless widget, The “**build**” method can be called only **ONCE** while the app is in action, which is responsible for drawing the widgets on to the device screen.

*If you want to redraw the Stateless Widget, then you will need to create a new instance of the widget.*

TIPS: You can quickly build a Stateless Widget in VS Code or Android Studio by using the shortcut “*stl*”.

Some examples of Stateless widgets are as follows:



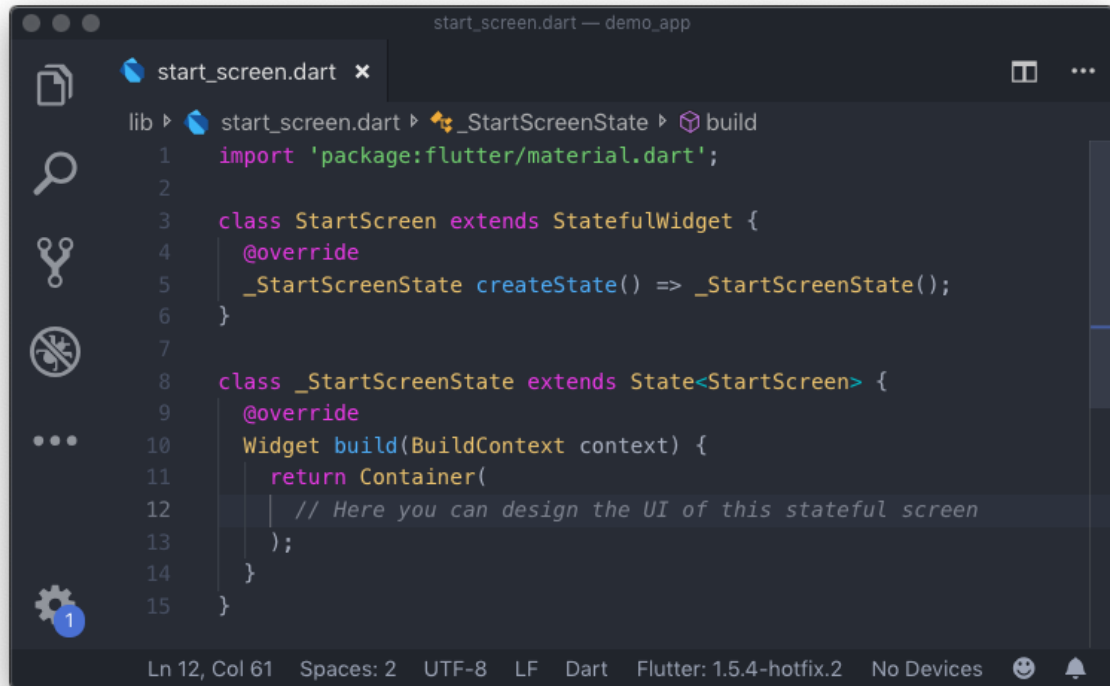
Now, let's move on to the Stateful Widget.

## Stateful

Stateful widgets have a mutable state, i.e., they are **mutable** and can be drawn multiple times within its lifetime.

They are the widgets which can change their state multiple times and can be redrawn on to the screen any number of times while the app is in action.

The structure of a Stateful widget looks like this:



```
start_screen.dart — demo_app

lib ▶ start_screen.dart ▶ _StartScreenState ▶ build
1  import 'package:flutter/material.dart';
2
3  class StartScreen extends StatefulWidget {
4    @override
5    _StartScreenState createState() => _StartScreenState();
6  }
7
8  class _StartScreenState extends State<StartScreen> {
9    @override
10   Widget build(BuildContext context) {
11     return Container(
12       // Here you can design the UI of this stateful screen
13     );
14   }
15 }
```

Ln 12, Col 61 Spaces: 2 UTF-8 LF Dart Flutter: 1.5.4-hotfix.2 No Devices

## Stateful Widget

The name of the widget is again “**StartScreen**”, but now it overrides the “**createState**” method, instead of the “**build**” method, which returns the instance of the class “**\_StartScreenState**”.

The class “**\_StartScreenState**” extends from **State<>** which takes “**StartScreen**” as a template input.

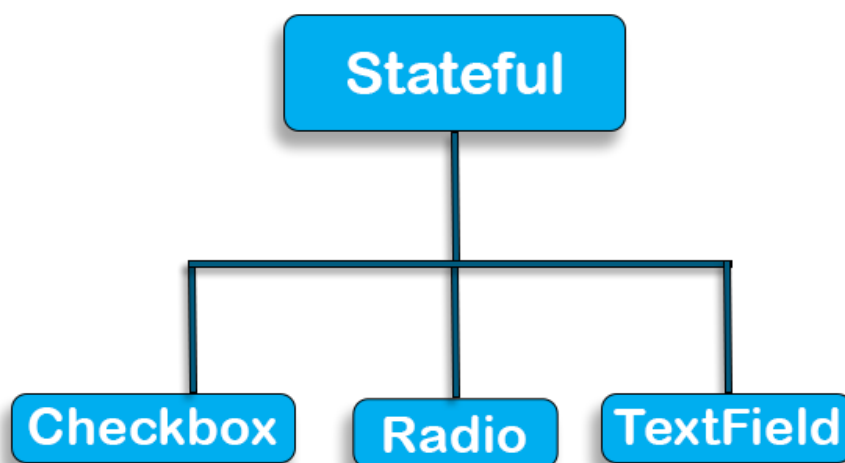
Now, this “**\_StartScreenState**” overrides the “**build**” method and returns a widget. This is where you can define the UI of the app, which is Stateful. As it is a Stateful widget you can call the build method any number of times, which will redraw the widgets on the screen.

## So, how can you call the build method?

It's really easy, you can use the "***setState***" method to call the build method, which will, in turn, redraw the widgets. This is the most important method you will need to use with any Stateful widget, to really use the statefulness of the widget.

TIPS: You can quickly build a Stateful Widget in VS Code or Android Studio by using the shortcut "stf".

Some examples of Stateful widgets are as follows:



## Conclusion

We have come to the end of this short article. I hope you all have received a basic idea of Stateful and Stateless widgets and how do they differ from each other. These concepts would be more clear if you do some projects on your own and get the feel of how the app handles the state.

Stateless Widget	Stateful Widget
<b>These widgets never changed. Their appearance remained constant, so they are Stateless Widgets.</b>	When a widget changes. Their appearance does not remain constant, so they are Stateful widget.
<b>No direct user interaction.</b>	User interaction with the widget.
<b>Overrides the build () method and returns a widget.</b>	Overrides the createState () method and return a state.
<b>Example- Text, Icon, IconButton etc.</b>	Example- CheckBox, RadioButton, Form, TextField.

When the widget's state changes, the state object calls setState(), telling the framework to redraw the widget.

### Using Stateful Widgets:

1. Create a class that extends a "StatefulWidget", that returns a state in "createState()".
2. Create a "State" class, with properties that may change.
3. Within "State" class, implement the "build" method.
4. Call the setState() to make the changes. Calling setState() tells framework to redraw widget.

## Building first 'StatefulWidget' App:

1. Create new Fresh project.
  - a. Import 'package:flutter/material.dart'
  - b. Declare main function.

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2
Run | Debug | Profile
3 void main() {
4   runApp(
5     MaterialApp(
6
7   ),
8 );
9 }
```

2. Now to create a stateful widget just press stf (short-cut key). As we press, it will generate the stateful widget where you can define your name for the widget.

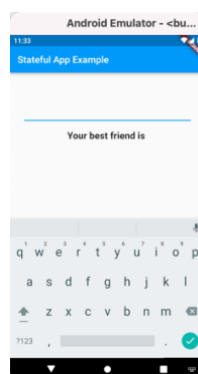
```
class MyFriend extends StatefulWidget {
  const MyFriend({ Key? key }) : super(key: key);

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

class _MyFriendState extends State<MyFriend> {
  @override
  Widget build(BuildContext context) {
    return Container(

  );
}
```

3. Now design the interface.



```

class _MyFriendState extends State<MyFriend> {
  String friendName = "";
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Stateful App Example"),
      ), // AppBar
      body: Container(
        margin: EdgeInsets.all(30),
        child: Column(
          children: [
            TextField(
              style: TextStyle(fontSize: 40),
            ), // TextField
            Padding(
              padding: const EdgeInsets.all(20.0),
              child: Text(
                "Your best friend is $friendName",
                style: TextStyle(
                  fontSize: 20.0,
                  fontWeight: FontWeight.bold,
                ), // TextStyle
              ), // Text
            ), // Padding
          ],
        ), // Column
      ), // Container
    ); // Scaffold
  }
}

```

Run it and provide value in 'TextField' it will do nothing, just because we need to implement last step:

Call the 'setState()' method to make the changes. Calling 'setState()' tells framework to redraw widget.

Add code:

```

      TextField(
        onSubmitted: (String userInput) {
          setState(() {
            friendName = userInput;
          });
        },
        style: TextStyle(fontSize: 40),
      ), // TextField

```



As we run the code, we will give the value in 'TextField' and hit enter, the 'setState()' will redraw the state of the widget.