Lifecycle of a Stateful Widget

A stateful widget in Flutter is a component that can maintain state and update its appearance in response to changes. The lifecycle of a stateful widget consists of a series of methods that are invoked at different stages of its existence.

In this article, we'll explore stateful widgets in Flutter, diving into their lifecycle and how they impact state management. Through practical examples and code snippets, we'll learn how to effectively handle and update state in Flutter applications.

Overview

First, let's have an overview of the stages in the lifecycle of a stateful widget, and which method is invoked at which stage of the lifecycle.

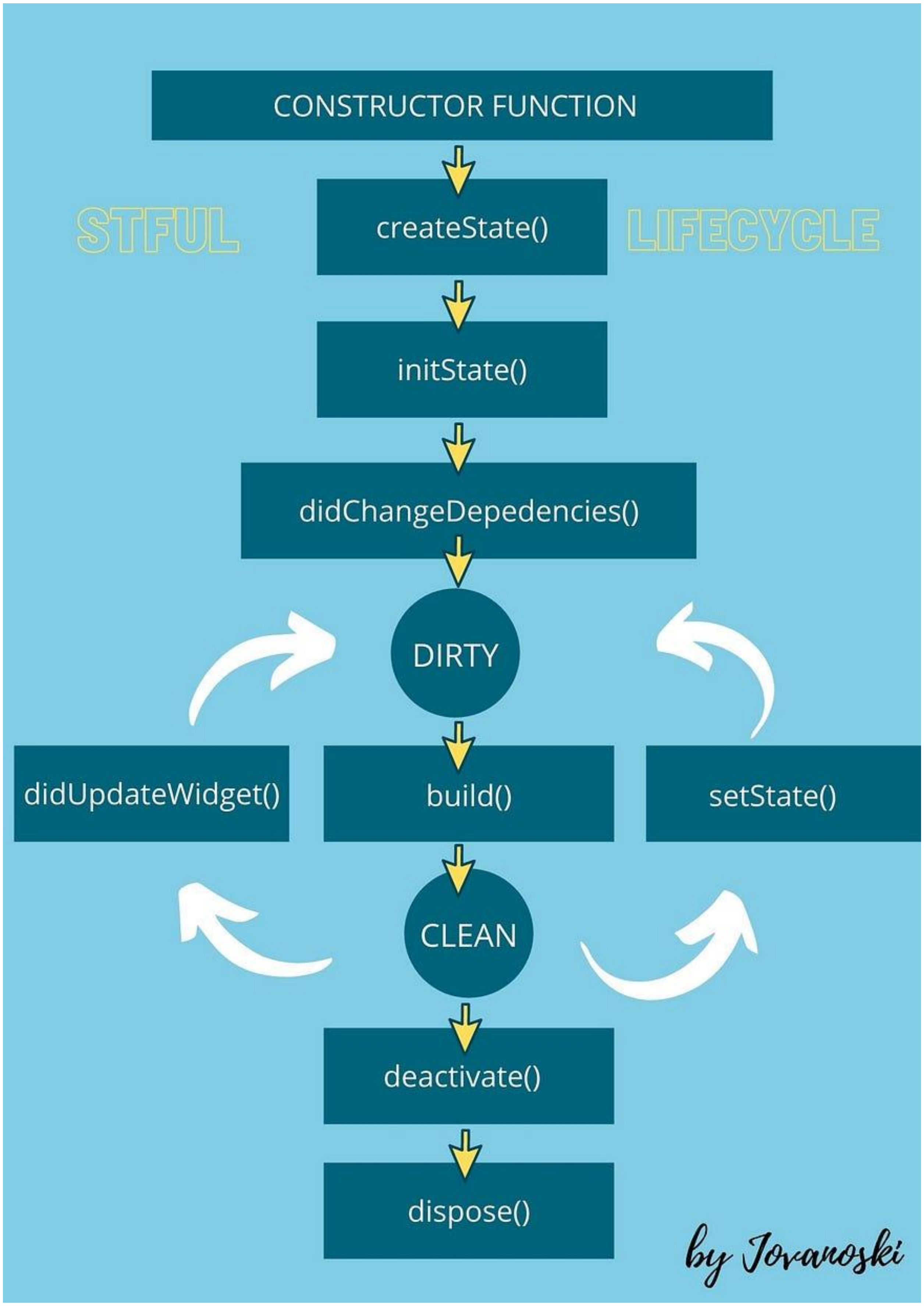
1. **createState():** Upon creation of the stateful widget, its constructor is called, which initializes the widget. The createState() method is then invoked, which creates the state object for the widget.

2. **initState:** This method is called after the widget is inserted into the widget tree, when the object is created for the first time.

3. **didChangeDependencies:** This method is called when a dependency of this widget changes. It is useful when the widget

depends on some external data or inherits data from its parent widget. It is also called immediately after initState().

4. **build:** Builds the widget's user interface based on its current state. This method is called initially when the widget is first inserted into the widget tree, and may be called repeatedly during the lifecycle whenever the widget needs to be rebuilt.
5. **setState:** Triggers a rebuild of the widget when the state changes and the associated methods are called again.
6. **didUpdateWidget:** Triggered when the widget is rebuilt with updated properties. It is called after build() and allows you to compare the previous and current widget properties.
7. **deactivate:** Called when the widget is removed from the tree but might be inserted again.
8. **dispose:** Called when the widget is removed from the widget tree permanently, allowing you to release resources held by the widget.
9. **reassemble:** Called when the application is reassembled during hot reload.

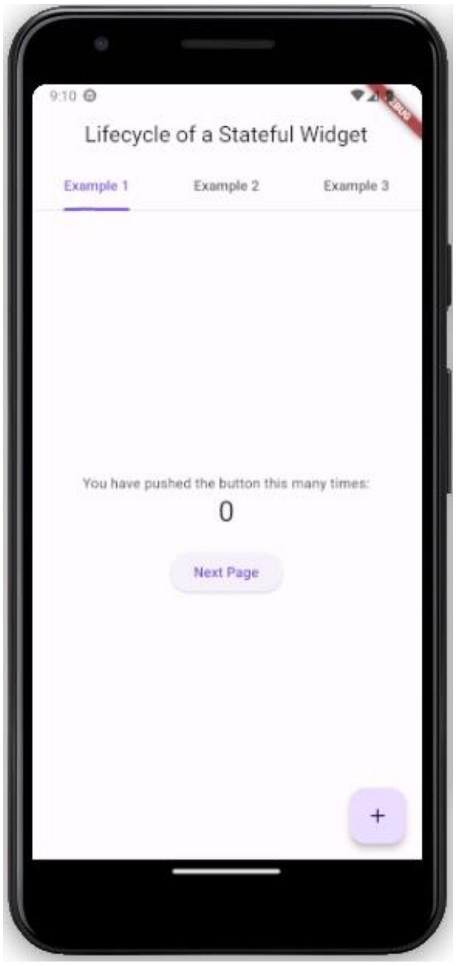

CONSTRUCTOR FUNCTION
STFUL
createState()
LIFECYCLE
initState()
didChangeDepedencies()
DIRTY
didUpdateWidget()
build()
setState()
CLEAN
deactivate()
dispose()
by Jovanoski

Example 1: Simple Implementation Using a Single Widget

First of all, let’s start with a very basic example, involving only a single Stateful Widget.

This example utilises the code of the default Flutter counter application; a Scaffold consisting of a Column and a Floating Action Button.

The Column contains a constant Text widget, a number signifying the amount of times the counter has been pressed, and additionally, an Elevated Button that allows the user to navigate to another page. The Floating Action Button with a + sign increments the counter value.

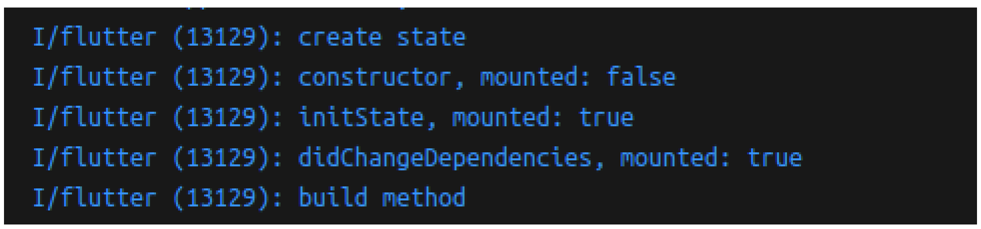

9:10
DEBUG
Lifecycle of a Stateful Widget
Example 1
Example 2
Example 3
You have pushed the button this many times:
0
Next Page
+

Below is the code for the Stateful Widget used in this example:

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

class Example1 extends StatefulWidget {
  const Example1({super.key});

  @override
  State<Example1> createState() {
    print('create state');
    return _Example1State();
  }
}

class _Example1State extends State<Example1> {
  int _counter = 0;

  _Example1State() {
    print('constructor, mounted: $mounted');
  }

  @override
  void initState() {
    super.initState();
    print('initState, mounted: $mounted');
  }

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    print('didChangeDependencies, mounted: $mounted');
  }

  @override
  void setState(VoidCallback fn) {
    print('setState');
    super.setState(fn);
  }

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    print('build method');

    return Scaffold(
      body: Center(
        child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text('You have pushed the button this many times:'),

            // counter text
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ),

            // navigation button
            Padding(
              padding: const EdgeInsets.only(top: 20),
              child: ElevatedButton(
                onPressed: () => Navigator.of(context).pushReplacement(
                  MaterialPageRoute(builder: (context) => const NewScreen()),
                ),
                child: const Text('Next Page'),
              ),
            ),
          ],
        ),
      ),

      // increment floating action button
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }

  @override
  void didUpdateWidget(covariant Example1 oldWidget) {
    super.didUpdateWidget(oldWidget);
    print('didUpdateWidget, mounted: $mounted');
  }

  @override
  void deactivate() {
    super.deactivate();
    print('deactivate, mounted: $mounted');
  }

  @override
  void dispose() {
    super.dispose();
    print('dispose, mounted: $mounted');
  }

  @override
  void reassemble() {
    super.reassemble();
    print('reassemble, mounted: $mounted');
  }
}
```

App is Started

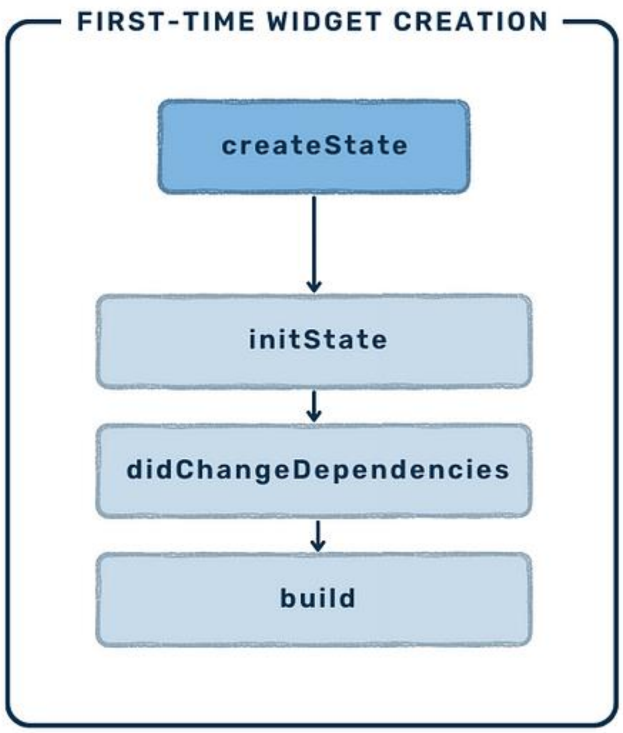
When the app is started for the first time, these are the methods that are called.



```
I/flutter (13129): create state
I/flutter (13129): constructor, mounted: false
I/flutter (13129): initState, mounted: true
I/flutter (13129): didChangeDependencies, mounted: true
I/flutter (13129): build method
```

First, as expected, the CreateState method is called as the widget is initialised. Then, the constructor is called. At this point, the “mounted” property is false, which means that this widget has not yet been inserted in the widget tree.

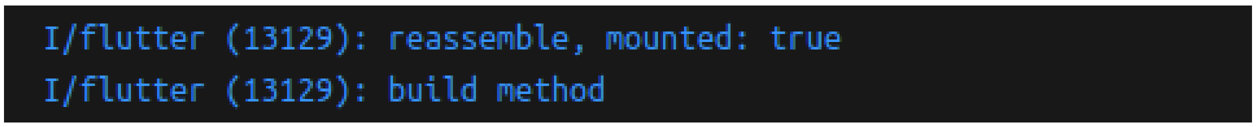
The object is then created as initState is called, and inserted into the widget tree, as can be seen through the true mounted value. As discussed earlier, didChangedDependencies is called immediately after initState, which can also be seen here. Then, the build method is called, and the Widget is built.



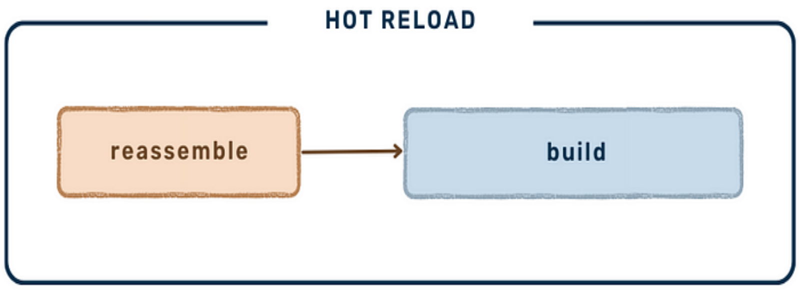
Hot reload

When the hot reload command is called, the reassemble() method is invoked. Hot reload is a mechanism that allows you to see changes made in your running code without having to restart the application.

On hot reload, the widget is rebuilt with the updated values.

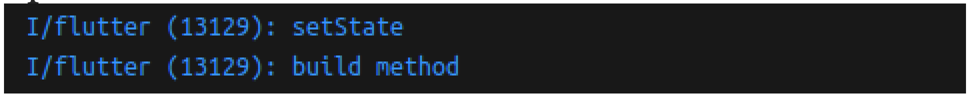


```
I/flutter (13129): reassemble, mounted: true
I/flutter (13129): build method
```

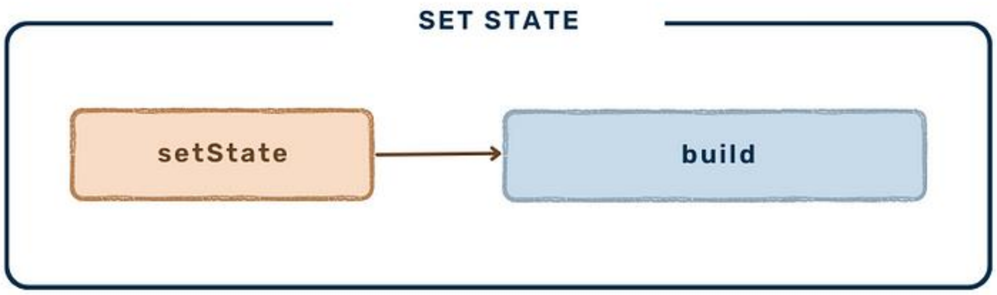


Increment (value update)

Clicking the Floating Action Button increments the counter value, which causes the state of the Widget to change. The setState method is called to trigger a rebuild, and then the widget is rebuilt to reflect this update.

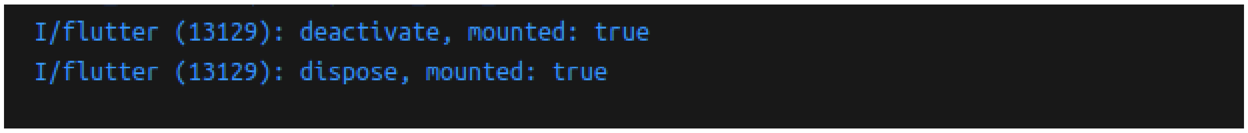


```
I/flutter (13129): setState
I/flutter (13129): build method
```

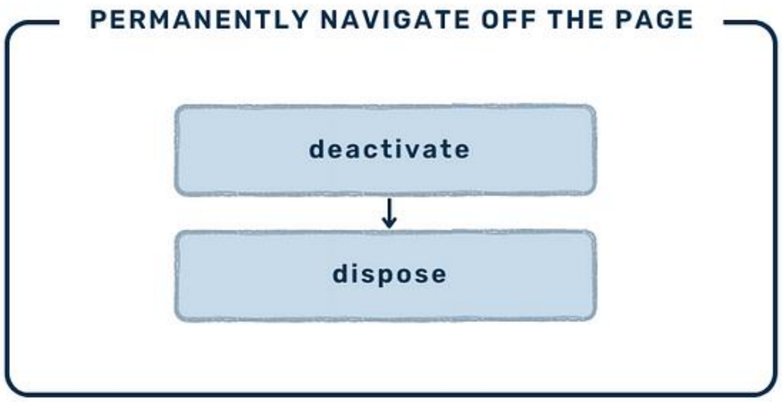


Next Page — PushReplacement

The next page button pushes a new page as a replacement. This invokes the deactivate() and dispose() methods.



```
I/flutter (13129): deactivate, mounted: true
I/flutter (13129): dispose, mounted: true
```



If we were to simply navigate to the next page, these methods would not be called, since the Widget would not have been removed from the widget tree.