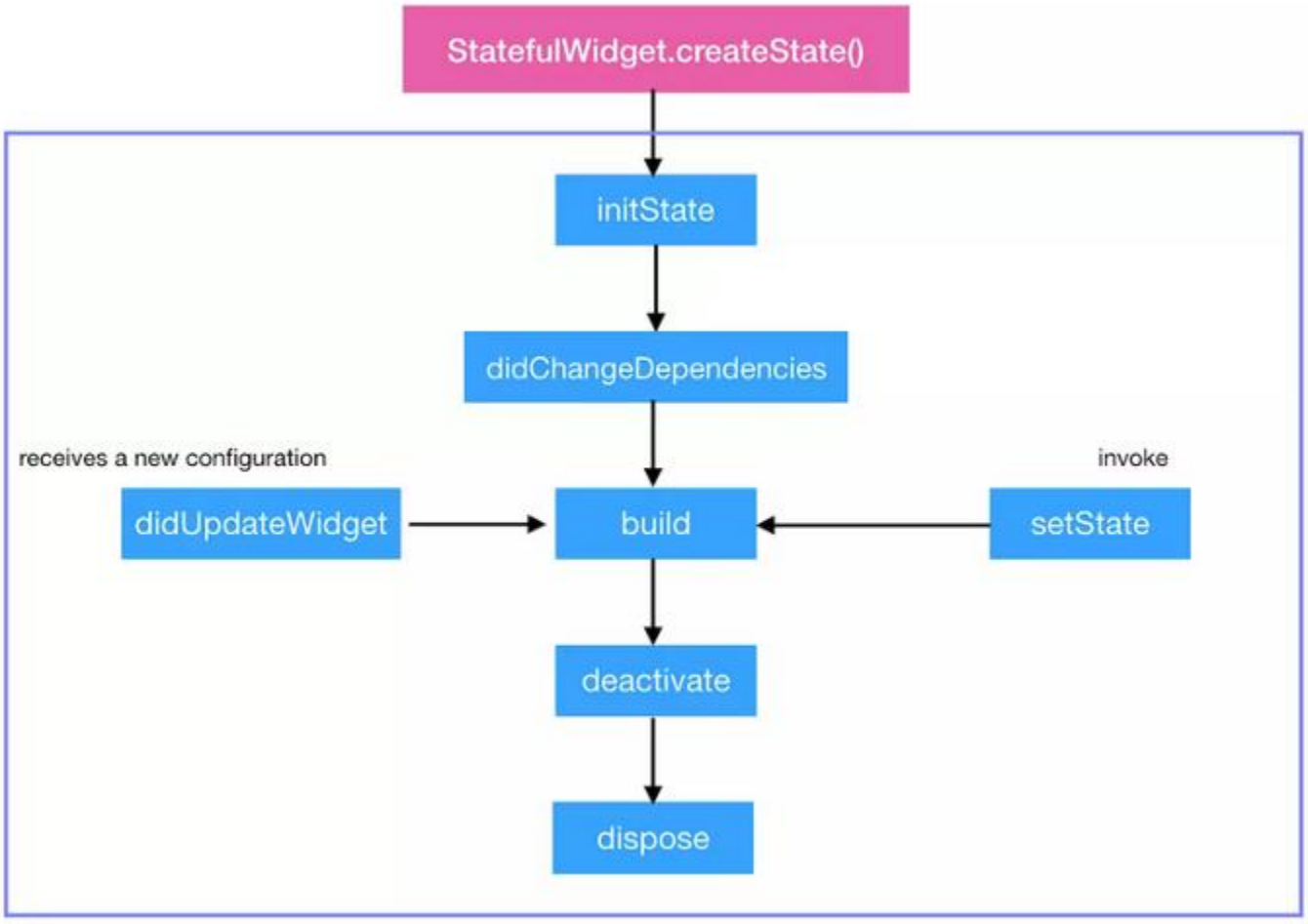
Flutter Developer

Flutter Widgets Life Cycle

Flutter has 2 types of widgets, **Stateless Widget** and **Stateful Widget.**
Stateless Widgets are rendered only once when the widget is loaded. We can't rebuild a Stateless widget based on any user events or changes. (Have a look at different types of trees in flutter!)
However, on the other hand, **Stateful Widgets** can be rebuilt and have their own Widget Lifecycle to create, update and destroy the widgets. Let's have a look!

Stateful Widget Lifecycle

A Stateful Widget goes through the following lifecycle stages:



1. createState()

In stateful widget, the 1st method that is called is `createState()` . The createState() method returns the instate of the state of the Stateful widget. Syntax:

```
class HomePage extends StatefulWidget {HomePage({Key key}) :
super(key: key);@override
  _HomePageState createState() => _HomePageState();
}
```

After this, we create the _HomePageState() class.

2. initState()

This is the first method called after the constructor of the Stateful Widget. It is called whenever the screen or widget is added to the widget tree! This method needs to call `super.initState()` which basically calls the initState of the parent widget (Stateful widget). Here you can initialize your variables, objects, streams, AnimationController, etc. Syntax:

```
@override
initState() {
  super.initState();
    // TO DO
}
```

3. didChangeDependencies()

It is always called for the 1st time after `initState()` . You can include few functionalities like API calls based on parent data changes, variable re-initializations, etc. Syntax:

```
@override
  void didChangeDependencies() {
    super.didChangeDependencies();
  }
```

4. build()

The build method is the one that shows and renders the UI part to the user. Whenever you want to update your UI or if you click hot-reload, the Flutter framework rebuilds the **build()** method! If you want to explicitly rebuild the UI if any data is changed, you can

use **setState()** which instructs the framework to again run the build method!

```
@override
Widget build(BuildContext context) {
      return Container();
}
```

5. didUpdateWidget(Widget oldWidget)

If the parent widget changes its properties or configurations, and the parent wants to rebuild the child widget, **with the same Runtime Type,** then didUpdateWidget is triggered. This unsubscribes to the old widget and subscribes to the configuration changes of the new widget!

```
@override
  void didUpdateWidget(covariant CurrentClass oldWidget) {
    // TODO: implement didUpdateWidget
    super.didUpdateWidget(oldWidget);
  }
```

6. setState()

This method notifies the Flutter framework that the internal state of the widget tree has been modified and the build method needs to be rendered again. This internal state may or may not affect the UI visible to the user and hence it becomes necessary to rebuild the UI.
Note: Using setState in long widget trees is costly as it rebuilds the entire widget tree and not just the changed component. Always break down your widget and use any State Management Techniques.

```
void function(){
  setState(() {});
}
```

7. deactivate()

This method is called when the widget is no longer attached to the Widget Tree but it might be attached in a later stage. The best example of this is when you use **Navigator.push** to move to the next screen, deactivate is called because the user can move back to the previous screen and the widget will again be added to the tree!

```
@override
  void deactivate() {
    super.deactivate();
  }
```

8. dispose()

This is called when the State object or Widget is removed permanently from the Widget Tree. Here you can unsubscribe streams, cancel timers, dispose animation controllers, close files, etc. In other words, you can release all the resources in this method. Now, in future, if the Widget is again added to Widget Tree, the entire lifecycle will again be followed!

```
@override
  void dispose() {
    super.dispose();
  }
```

We have seen the entire widget lifecycle, but you need to be very careful while coding. Your code should know, whether the particular widget is present in Widget Tree or not! For this, we have a keyword called **mounted** which returns true if the current widget is present in Widget Tree!

Tip: Always use if(mounted) with setState() or when you use `context` because we never know the user's interaction with the app.

As setState triggers the build method and we can’t rebuild the Widget which is not present in the Widget Tree.

```
void function() {
    if (mounted) setState(() {});
  }
```

This will run setState only if the widget is present in the widget tree.

Author: https://abhishekdoshi26.medium.com/widget-lifecycle-flutter-3db5d824d033