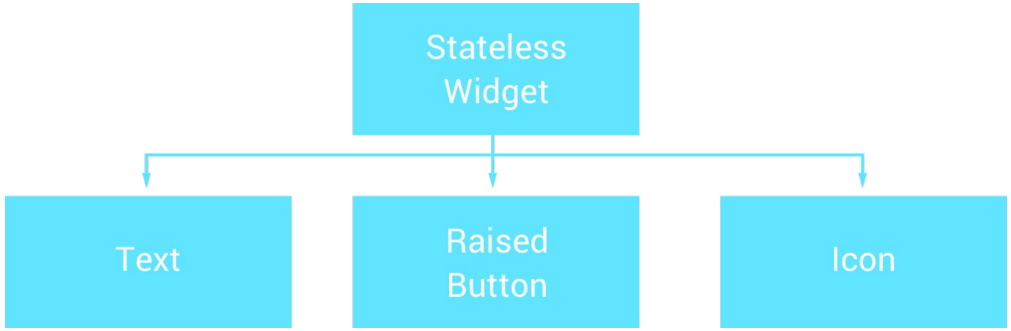
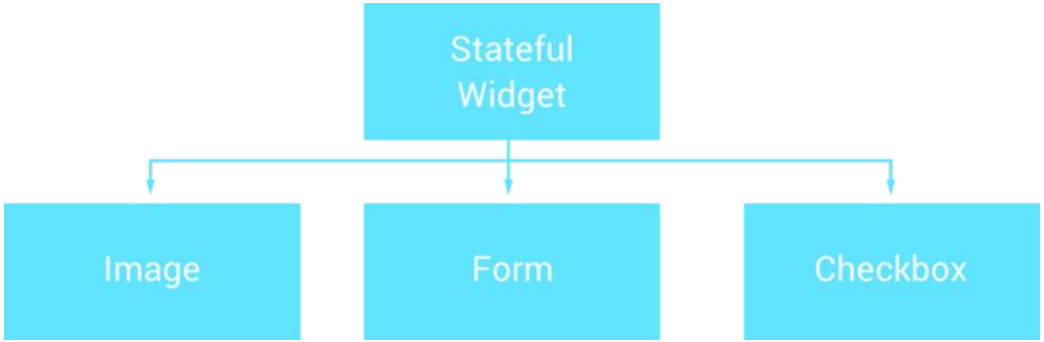
- If you create widgets that don't need to manage any form of internal state, this is where you'll want to make use of the StatelessWidget.



- Stateless widget is **not dynamic.**
- Stateless widget have no internal state which means once built, can never be modified until and unless they're initialized again.

- Stateful widgets are dynamic and have a state, which means that they can be modified easily throughout their lifecycle without reinitiation.



- State is the information that can be read synchronously when the widget is built and might change during the lifetime of the widget.
- State objects are created by the framework.
- In order to change your widget, you need to update the state object which can be done using setState()function available for Stateful widgets. setState() sets properties of state object which in turn triggers the update to the UI.

- **createState():** When the Framework is instructed to build a StatefulWidget, it immediately calls createState()
- **mounted is true:** When createState creates your state class, a buildContext is assigned to that state. BuildContext is, overly simplified, the place in the widget tree in which this widget is placed. Here's a longer explanation. All widgets have a bool this.mounted property. It is turned true when the buildContext is assigned. It is an error to call setState when a widget is unmounted.
- **initState():** This is the first method called when the widget is created (after the class constructor, of course.) initState is called once and only once. It must called super.initState().

- **didChangeDependencies():** This method is called immediately after initState on the first time the widget is built.
- **build():** This method is called often. It is required, and it must return a Widget.
- **didUpdateWidget(Widget oldWidget):** If the parent widget changes and has to rebuild this widget (because it needs to give it different data), but it's being rebuilt with the same runtimeType, then this method is called. This is because Flutter is re-using the state, which is long lived. In this case, you may want to initialize some data again, as you would in initState.
- **setState():** This method is called often from the framework itself and from the developer. Its used to notify the framework that data has changed

- **deactivate():** Deactivate is called when State is removed from the tree, but it might be reinserted before the current frame change is finished. This method exists basically because State objects can be moved from one point in a tree to another.
- **dispose():** Dispose is called when the State object is removed, which is permanent. This method is where you should unsubscribe and cancel all animations, streams, etc.
- **mounted is false:** The state object can never remount, and an error is thrown is setState is called.

- Tree Shaking is used during compilation to remove unused code in packages.
- Dart also has a “tree shaking” compiler, which only includes code that you need in your app. You can feel free to use a large library of widgets even if you only need one or two of them.

The Spacer widget can help you control how much space appears between widgets in a Row or Column. Just add it between two widgets and set its flex property to customize.

A box with a specified size.