# Breaking into Flutter

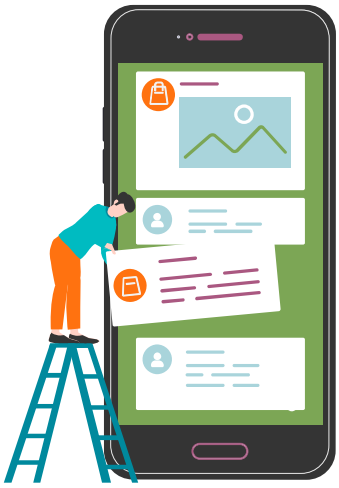
Mobile Application development

1

## Agenda

- Widgets:
  - widget tree, widget types, State object.
- Build context.
- Composition in Flutter.
- Intro to Layout in Flutter.
- The element tree.

2



# Widgets

Widget tree, widget types, State object

3

## Widgets

- Everything is a widget, and widgets are just Dart classes.
- A widget can define any aspect of an application's view.
  - ✓ Some widgets, such as `Row`, define aspects of the layout.
  - ✓ Some are less abstract and define structural elements, like `Button` and `TextField`.
- These are some of the most common widgets:
  - *Layout*—`Row`, `Column`, `Scaffold`, `Stack`
  - *Structures*—`Button`, `Toast`, `MenuDrawer`
  - *Styles*—`TextStyle`, `Color`
  - *Animations*—`FadeInPhoto`, transformations
  - *Positioning and alignment*—`Center`, `Padding`

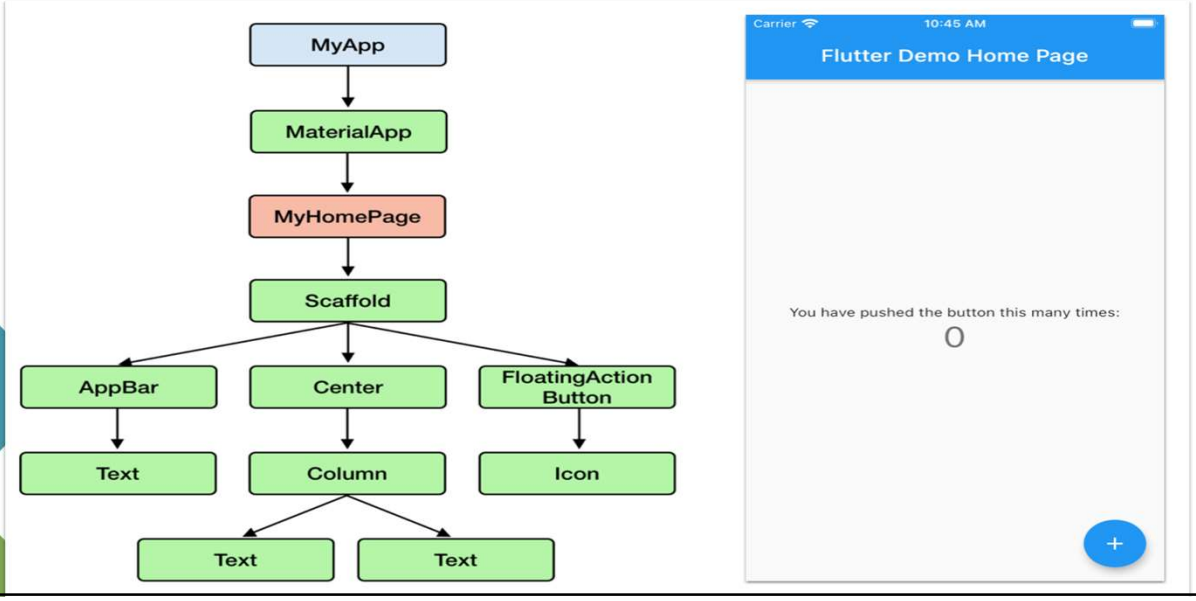
4

# Widgets Tree

- A Flutter app is represented by a widget tree.
- The widget tree is an actual tree data structure in code built by Flutter,
- In short, the tree is a collection of nodes, where each node is a widget.
- Every time we add a widget in a `build` method, we're adding a new node in the tree.
- The nodes are connected by their parent-child relationship.

5

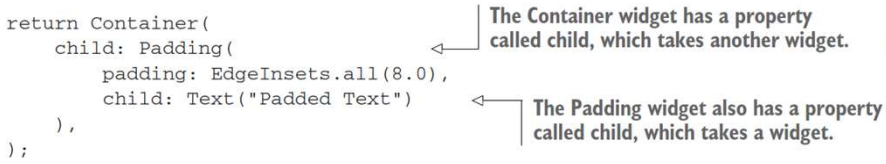
# Widgets Tree




6

- A simple example;

In the widget tree, `Container` is the parent of `Padding`, which is the parent of the `Text` widget.




- Other common properties in Flutter that allow to pass widgets into widgets are `children` and `builder`.

## Widgets Tree

7

## Widgets Types

- Most widgets fall under two categories: `StatelessWidget` and `StatefulWidget`.
- A `StatelessWidget` is a widget that its only job is to display information and UI.
- A stateless widget is destroyed entirely when Flutter removes it from the widget tree.
- A `StatefulWidget` object has an associated `State` object.
- The `State` object has special methods such as `setState` that tell Flutter when it needs to repainting.
- `State` objects are long-lived. They can tell Flutter to repaint, but it can also be told to repaint because the associated stateful widget has been updated by outside forces.

8

# Widgets Types

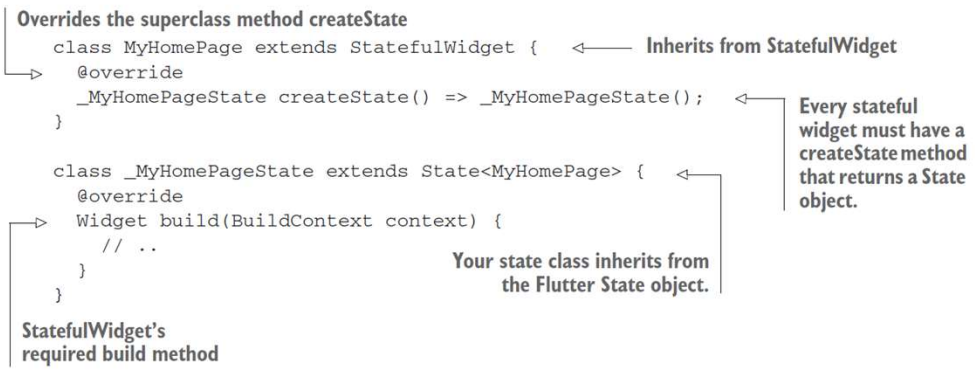
The difference between `StatefulWidget` and a `StatelessWidget`:

- A `StatefulWidget` tracks its own internal state.
- A `StatelessWidget` doesn't have any internal state that changes during the lifetime of the widget. It could be passed configuration from its parent, or the configuration could be defined within the widget, but it *cannot change its own configuration*. A stateless widget is immutable.
- It's important to understand that a stateless widget shouldn't be responsible for any data a developer want to keep.

9

# Widgets – *Stateful widgets*

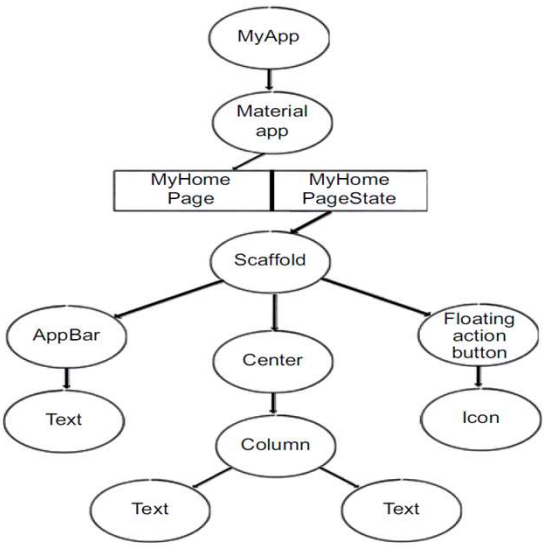
- A stateful widget has internal state and can manage that state.
- Notice the `MyHomePage` tree node is connected to the `MyHomePageState` tree node.
- All `StatefulWidget` instances actually have two classes involved.




10

# Widgets – *Stateful widgets*

- All stateful widgets have corresponding state objects.
- Notice the `MyHomePage` tree node is connected to the `MyHomePageState` tree node.




11

# Widgets – *Stateful widgets*

- The `StatefulWidget` class doesn't have a `build` method. But every stateful widget has an associated state object, which does have a `build` method.
- You can think of the pair of `StatefulWidget` and `State` as the same entity.
  - ✓ In fact, stateful widgets are immutable (just like stateless widgets), but their associated state objects are smart, mutable.
- The class name is `_MyHomePageState` with an underscore, which is used to mark the class as *private*. That means it is only available within the current file.
  - ✓ If a class member, such as a variable or function, is marked private, it's only available to use within that class itself.

12

- Flutter widgets are reactive. They respond to new information from an outside source (or `setState`).
- This is the high-level process:
  1. A user taps a button.
  2. The app calls `setState` in the `Button.onPressed` callback.
  3. Flutter knows that it needs to rebuild, because the `Button` state is marked `dirty`.
  4. The new widget replaces the old one in the tree.
  5. Flutter renders the new tree.

## State Object

13

## Stateful Widget – *setState & initState*

- `setState` is one of the most important Flutter method we have to know.
- It exists only for the state object.
- `setState` is called by the developer, whenever we want Flutter to re-render.

- The state object also has a method called `initState`, which is called as soon as the widget is mounted in the tree.
- `State.initState` is the method in which we initialize any data needed *before* Flutter tries to paint it the screen.
- `initState` is called once every time a state object is built.

14

# BuildContext

- `BuildContext` is another concept in Flutter that's crucial to building apps.
- The `build()` method *must* exist in every Flutter widget. This is the method in which we actually describe the view by returning widgets.
- Every `build` method in a widget takes one argument, `BuildContext`, which is a reference to a widget's location in the widget tree.
- The build context is used in various ways to tell Flutter exactly where and how to render certain widgets.
  - For example, Flutter uses the build context to display modals and routes. If you wanted to display a new route, Flutter needs to know where in the tree that route should be inserted. This is accomplished by passing in `BuildContext` to a method that creates routes.
- It contains information about a widget's place *in the widget tree*, not about the widget itself.

15

# Favor Composition in Flutter

- Flutter builds a mobile UI by composing together a bunch of smaller components called *widgets*.
- Structure is defined with widgets, styles are defined with widgets, and so are animations and anything else that makes up a UI.
- There are two ways to create relationships between classes.
  1. Inheritance, establishes an "is a" relationship.
  2. Composition establishes a "has a" relationship.
- Inheritance tends to have you designing objects around what they are, and composition around what they do.
- In Flutter, always favor composition (over inheritance) to create reusable and decoupled widgets.

16

# Favor Composition in Flutter

- Flutter favors composition over class inheritance. A majority of widgets are combinations of smaller widgets.
- In practice, that means that in Flutter we aren't subclassing other widgets in order to build custom widget. This is wrong:

```
class AddToCartButton
  extends Button {}
```

- Rather, we *compose* a button by wrapping the `Button` widget in other widgets:

```
class AddToCartButton extends StatelessWidget {
  // ... class members
  @override
  build() {
    return Center(
      child: Button(
        child: Text('Add to Cart'),
      ),
    );
  }
}
```

This widget will center this AddToCartButton forever.

Pass text in, and now you have a custom component.

17