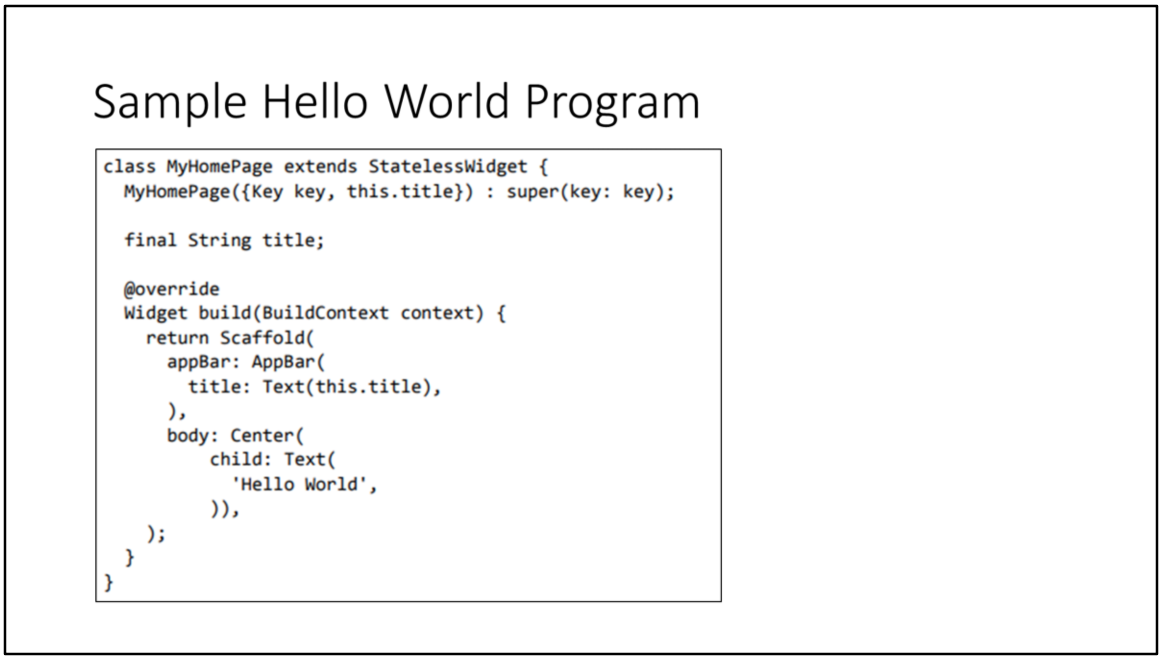
Introduction to Flutter Widgets

Widgets

- **In Flutter, Everything is a widget**.
- In Flutter, the application is itself a widget
- The application is the top- level widget and its UI is build using one or more children (widgets), which again build using its children widgets.
- This **composability** feature helps us to create a user interface of any complexity.



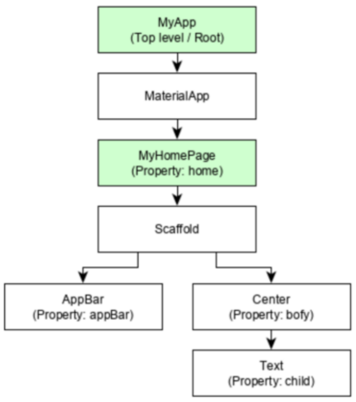
Sample Hello World Program

```
class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(this.title),
      ),
      body: Center(
          child: Text(
            'Hello World',
          )),
    );
  }
}
```

Important Points

- StatelessWidget only requires a single method build to be implemented in its derived class
- The build method gets the context environment necessary to build the widgets through BuildContext parameter and returns the widget it builds
- In the code, we have used title as one of the constructor argument and also used Key as another argument. The title is used to display the title and Key is used to identify the widget in the build environment.
- Here, the build method calls the build method of Scaffold, which in turn calls the build method of AppBar and Center to build its user interface.

Widget Hierarchy of Hello World Program

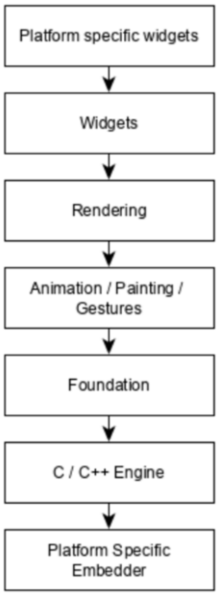


- **MyApp** is the user created widget and it is build using the Flutter native widget, MaterialApp.
- **MaterialApp** has a home property to specify the user interface of the home page, which is again a user created widget, MyHomePage.
- **MyHomePage** is build using another flutter native widget, Scaffold.
- **Scaffold** has two properties – body and appBar.
- **body** is used to specify its main user interface and appBar is used to specify its header user interface.
- Header UI is build using flutter native widget, **AppBar** and **Body** UI is build using Center widget.
- The **Center** widget has a property, **Child**, which refers the actual content and it is build using Text widget.

Concept of State

- Flutter widgets support State maintenance by providing a special widget, StatefulWidget.
- Widget needs to be derived from StatefulWidget widget to support state maintenance and all other widget should be derived from StatelessWidget.
- StatefulWidget will be auto re- rendered whenever its internal state is changed

Layers

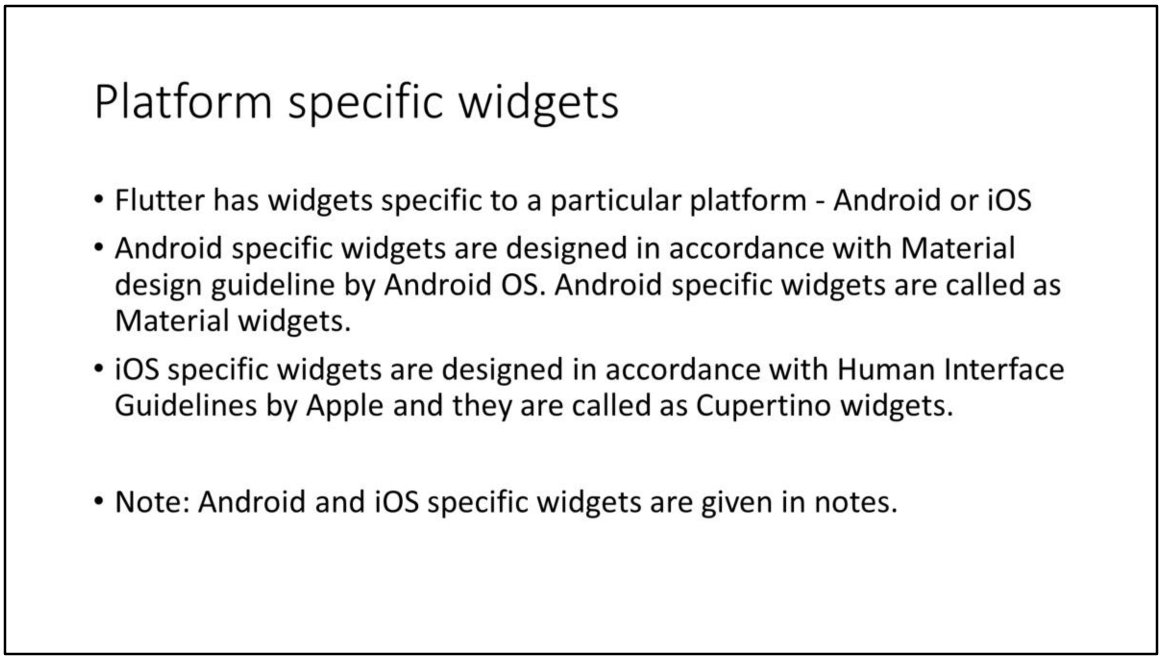

Platform specific widgets
Widgets
Rendering
Animation / Painting / Gestures
Foundation
C / C++ Engine
Platform Specific Embedder

Summary

 In Flutter, everything is a widget and a complex widget is composed of already existing widgets.
 Interactive features can be incorporated whenever necessary using GestureDetector widget.
 The state of a widget can be maintained whenever necessary using StatefulWidget widget.
 Flutter offers layered design so that any layer can be programmed depending on the complexity of the task

Widget Build Visualization

- In Flutter, widgets can be grouped into multiple categories based on their features, as listed below:
  - Platform specific widgets
  - Layout widgets
  - State maintenance widgets
  - Platform independent / basic widgets


Platform specific widgets

- Flutter has widgets specific to a particular platform - Android or iOS
- Android specific widgets are designed in accordance with Material design guideline by Android OS. Android specific widgets are called as Material widgets.
- iOS specific widgets are designed in accordance with Human Interface Guidelines by Apple and they are called as Cupertino widgets.
- Note: Android and iOS specific widgets are given in notes.

Some of the most used material widgets are as follows: (Android)

- Scaffold
- AppBar
- BottomNavigationBar
- TabBar
- TabBarView
- ListTile
- RaisedButton
- FloatingActionButton
- FlatButton
- IconButton
- DropdownButton
- PopupMenuButton
- ButtonBar
- TextField
- Checkbox
- Radio
- Switch
- Slider

- Date & Time Pickers
- SimpleDialog
- AlertDialog

Some of the most used Cuppertino widgets are as follows: (iOS)

- CupertinoButton
- CupertinoPicker
- CupertinoDatePicker
- CupertinoTimerPicker
- CupertinoNavigationBar
- CupertinoTabBar
- CupertinoTabScaffold
- CupertinoTabView
- CupertinoTextField
- CupertinoDialog
- CupertinoDialogAction
- CupertinoFullscreenDialogTransition
- CupertinoPageScaffold
- CupertinoPageTransition
- CupertinoActionSheet
- CupertinoActivityIndicator
- CupertinoAlertDialog
- CupertinoPopupSurface

Layout widgets

- In Flutter, a widget can be created by composing one or more widgets
- Some of the popular layout widgets are as follows:
  - **Container**: A rectangular box decorated using **BoxDecoration** widgets with **background**, **border** and **shadow**.
  - **Center**: Center its child widget
  - **Row**: Arrange its children in the horizontal direction.
  - **Column**: Arrange its children in the vertical direction.
  - **Stack**: Arrange one above the another.

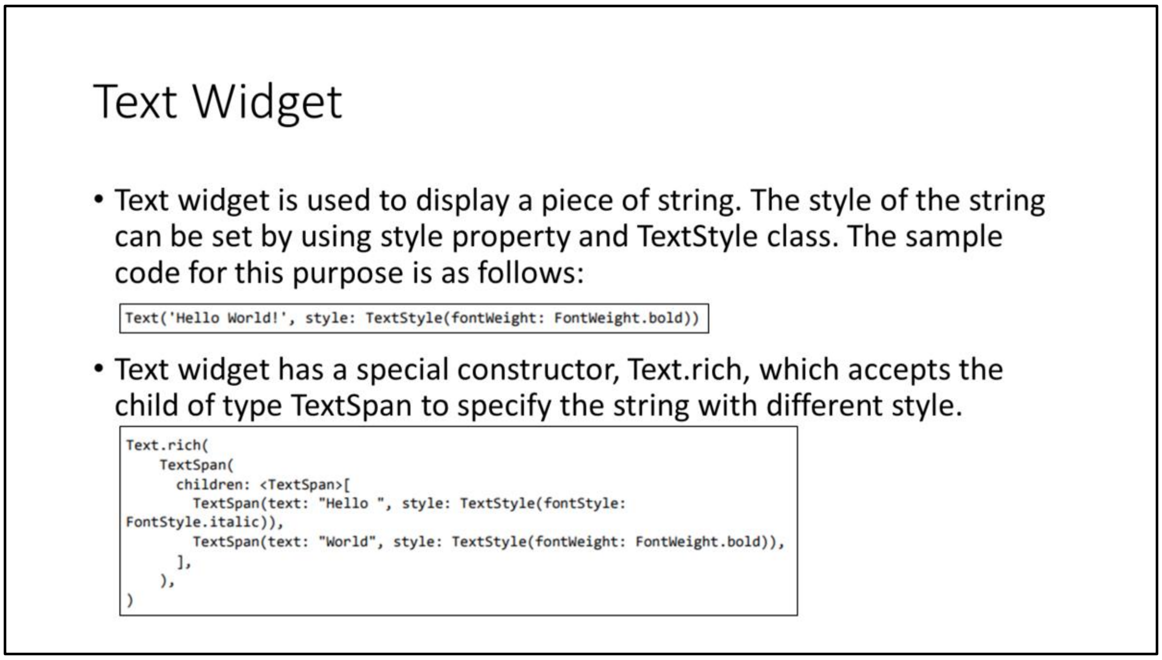
State maintenance widgets

- In Flutter, all widgets are either derived from **StatelessWidget** or **StatefulWidget**.
- Widget derived from **StatelessWidget** does not have any state information but it may contain widget derived from **StatefulWidget**.

Platform independent / basic widgets

- Flutter provides large number of basic widgets to create simple as well as complex user interface in a platform independent manner.
  - Text
  - Image
  - Icon

  etc



Text Widget

- Text widget is used to display a piece of string. The style of the string can be set by using style property and TextStyle class. The sample code for this purpose is as follows:

```
Text('Hello World!', style: TextStyle(fontWeight: FontWeight.bold))
```

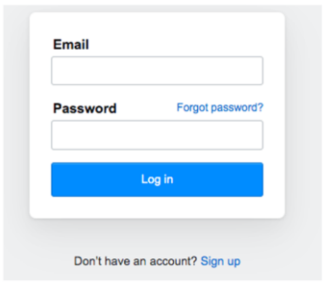
- Text widget has a special constructor, Text.rich, which accepts the child of type TextSpan to specify the string with different style.

```
Text.rich(
    TextSpan(
      children: <TextSpan>[
        TextSpan(text: "Hello ", style: TextStyle(fontStyle:
FontStyle.italic)),
        TextSpan(text: "World", style: TextStyle(fontWeight: FontWeight.bold)),
      ],
    ),
)
```

The most important properties of the Text widget are as follows:

 maxLines, int: Maximum number of lines to show
 overflow, TextOverFlow: Specify how visual overflow is handled using TextOverFlow class
 style, TextStyle: Specify the style of the string using TextStyle class
 textAlign, TextAlign: Alignment of the text like right, left, justify, etc., using TextAlign class
 textDirection, TextDirection: Direction of text to flow, either left-to-right or rightto-left

RichText - Example



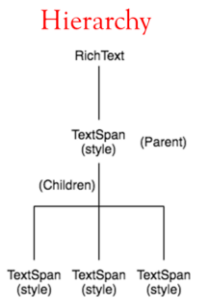


Image Widget

- Image widget is used to display an image in the application.
- Image widget provides different constructors to load images from multiple sources and they are as follows:
  - Image - Generic image loader using ImageProvider
  - Image.asset - Load image from flutter project's assets
  - Image.file - Load image from system folder
  - Image.memory - Load image from memory
  - Image.Network - Load image from network

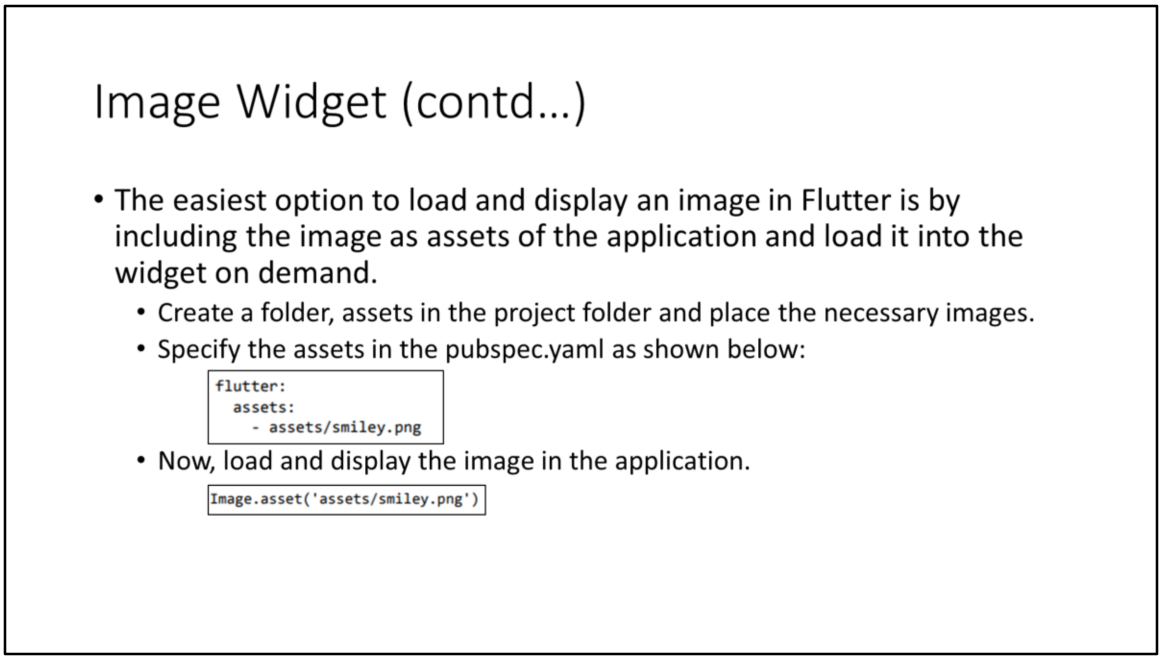


Image Widget (contd…)

- The easiest option to load and display an image in Flutter is by including the image as assets of the application and load it into the widget on demand.
  - Create a folder, assets in the project folder and place the necessary images.
  - Specify the assets in the pubspec.yaml as shown below:

```
flutter:
  assets:
    - assets/smiley.png
```

  - Now, load and display the image in the application.

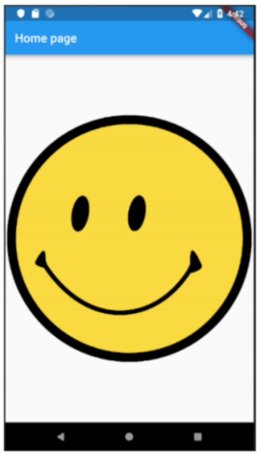
```
Image.asset('assets/smiley.png')
```

The most important properties of the Image widget are as follows:

 image, ImageProvider: Actual image to load
 width, double - Width of the image
 height, double - Height of the image
 alignment, AlignmentGeometry - How to align the image within its bounds

Sample complete code to show image

```
class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(this.title),
      ),
      body: Center(
          child: Image.asset("assets/smiley.png")
      ),
    );
  }
```



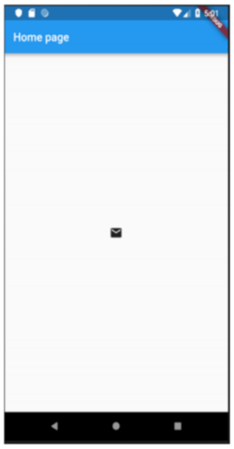
Icon Widget

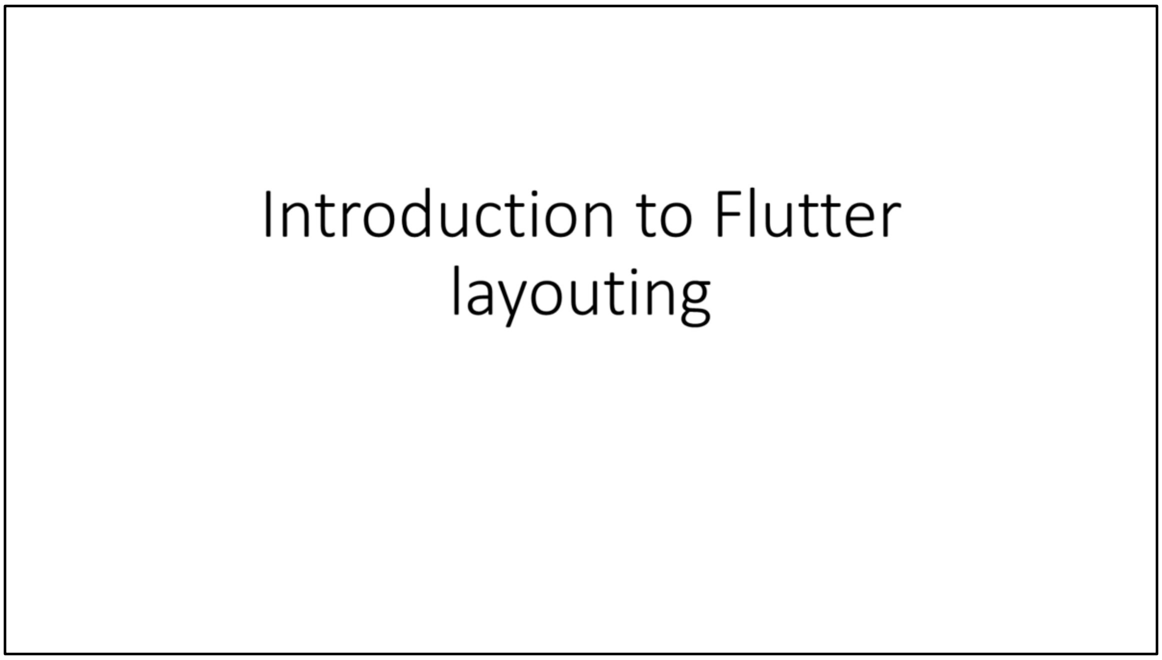
- Icon widget is used to display a glyph from a font described in IconData class.
- The code to load a simple email icon is as follows:

```
Icon(Icons.email)
```

Sample complete code to show Icon

```
class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(this.title),
      ),
      body: Center(
          child: Icon(Icons.email)
      ),
    );
  }
}
```





Introduction to Flutter layouting

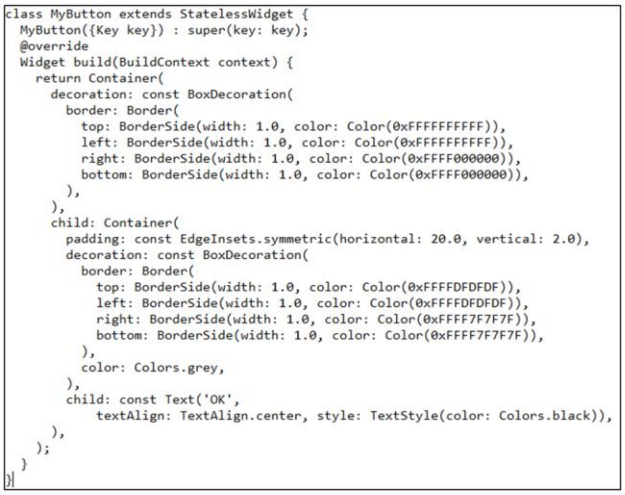
Layout Widgets

- Layout widgets can be grouped into two distinct category based on its child:
  - Widget supporting a single child
  - Widget supporting multiple child

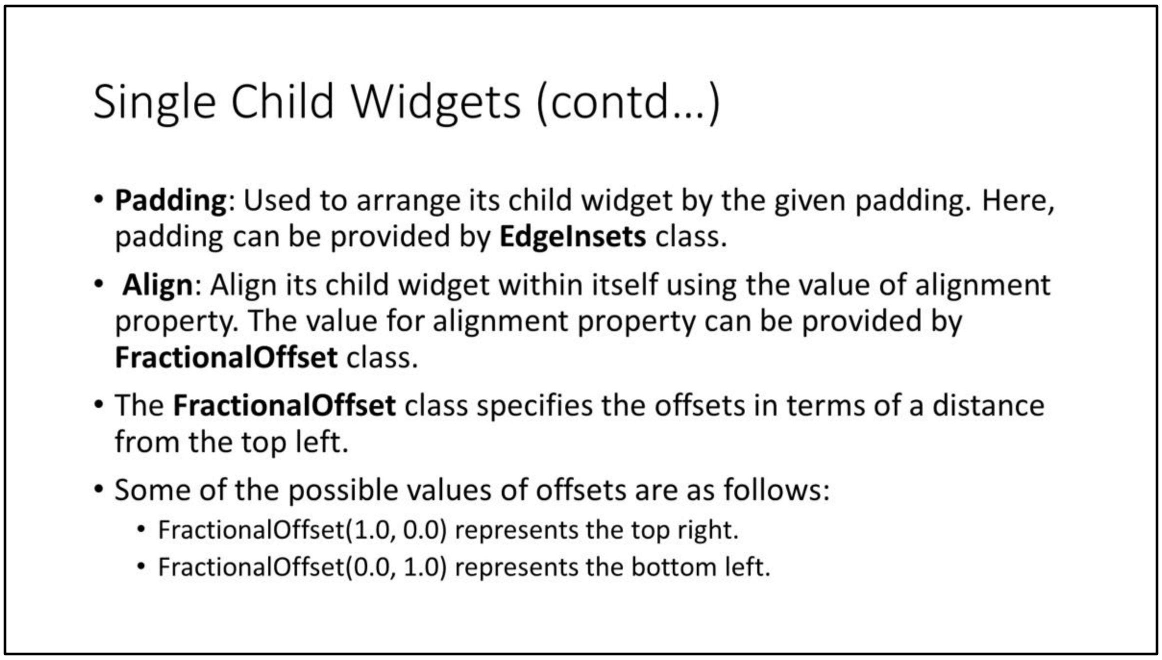
Single Child Widgets

- In this category, widgets will have only one widget as its child and every widget will have a special layout functionality.
- For example, **Center** widget just centers it child widget with respect to its parent widget and **Container** widget provides complete flexibility to place it child at any given place inside it using different option like padding, decoration, etc.
- Single child widgets are great options to create high quality widget having single functionality such as button, label, etc.,

Sample code to create single child widget

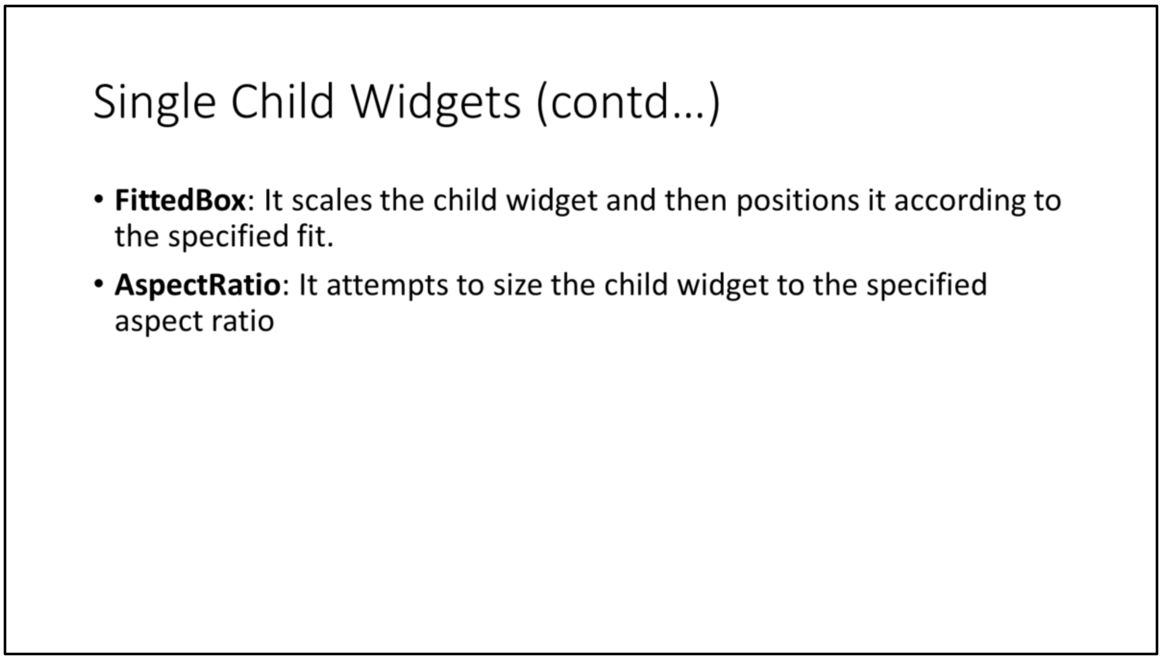


```
class MyButton extends StatelessWidget {
  MyButton({Key key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: const BoxDecoration(
        border: Border(
          top: BorderSide(width: 1.0, color: Color(0xFFFFFFFFFF)),
          left: BorderSide(width: 1.0, color: Color(0xFFFFFFFFFF)),
          right: BorderSide(width: 1.0, color: Color(0xFFFF000000)),
          bottom: BorderSide(width: 1.0, color: Color(0xFFFF000000)),
        ),
      ),
      child: Container(
        padding: const EdgeInsets.symmetric(horizontal: 20.0, vertical: 2.0),
        decoration: const BoxDecoration(
          border: Border(
            top: BorderSide(width: 1.0, color: Color(0xFFFFDFDFDF)),
            left: BorderSide(width: 1.0, color: Color(0xFFFFDFDFDF)),
            right: BorderSide(width: 1.0, color: Color(0xFFFF7F7F7F)),
            bottom: BorderSide(width: 1.0, color: Color(0xFFFF7F7F7F)),
          ),
          color: Colors.grey,
        ),
        child: const Text('OK',
            textAlign: TextAlign.center, style: TextStyle(color: Colors.black)),
      ),
    );
  }
}
```



Single Child Widgets (contd…)

- **Padding**: Used to arrange its child widget by the given padding. Here, padding can be provided by **EdgeInsets** class.
- **Align**: Align its child widget within itself using the value of alignment property. The value for alignment property can be provided by **FractionalOffset** class.
- The **FractionalOffset** class specifies the offsets in terms of a distance from the top left.
- Some of the possible values of offsets are as follows:
  - FractionalOffset(1.0, 0.0) represents the top right.
  - FractionalOffset(0.0, 1.0) represents the bottom left.



Single Child Widgets (contd…)

- **FittedBox**: It scales the child widget and then positions it according to the specified fit.
- **AspectRatio**: It attempts to size the child widget to the specified aspect ratio

Some more Single Child Widgets

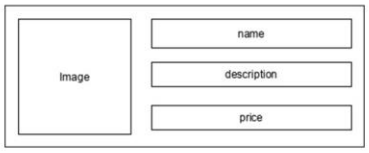
- ConstrainedBox
- Baseline
- FractinallySizedBox
- IntrinsicHeight
- IntrinsicWidth
- LiimitedBox
- OffStage
- OverflowBox
- SizedBox
- SizedOverflowBox
- Transform
- CustomSingleChildLayout

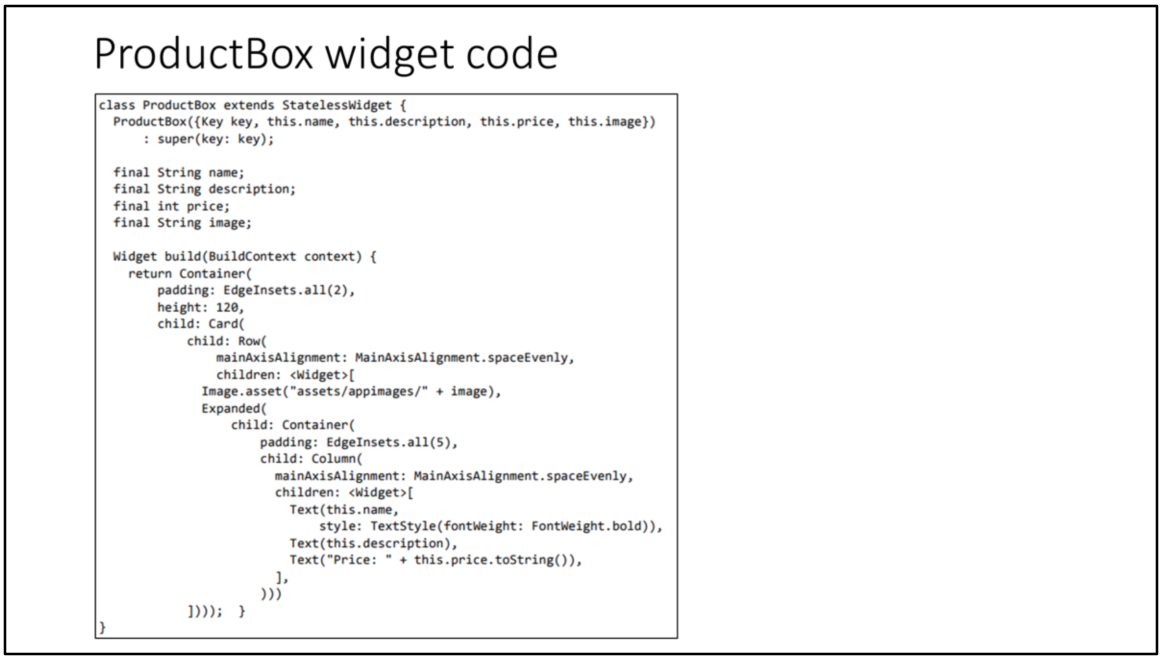
Multiple Child Widgets

- In this category, a given widget will have more than one child widgets and the layout of each widget is unique.
  - **Row** - Allows to arrange its children in a horizontal manner.
  - **Column** - Allows to arrange its children in a vertical manner.
  - **ListView** - Allows to arrange its children as list.
  - **GridView** - Allows to arrange its children as gallery.
  - **Expanded** - Used to make the children of Row and Column widget to occupy the maximum possible area.
  - **Table** - Table based widget.
  - **Flow** - Flow based widget.
  - **Stack** - Stack based widget

Advanced Layouting

- Lets learn how to create advanced layouts.
- Open the hello world application code.
- Now, create a new widget, *ProductBox* according to the specified design as shown below





ProductBox widget code

```
class ProductBox extends StatelessWidget {
  ProductBox({Key key, this.name, this.description, this.price, this.image})
      : super(key: key);

  final String name;
  final String description;
  final int price;
  final String image;

  Widget build(BuildContext context) {
    return Container(
        padding: EdgeInsets.all(2),
        height: 120,
        child: Card(
            child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                children: <Widget>[
              Image.asset("assets/appimages/" + image),
              Expanded(
                  child: Container(
                      padding: EdgeInsets.all(5),
                      child: Column(
                        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                        children: <Widget>[
                          Text(this.name,
                              style: TextStyle(fontWeight: FontWeight.bold)),
                          Text(this.description),
                          Text("Price: " + this.price.toString()),
                        ],
                      )))
            ])));  }
}
```

Observations: Please observe the following in the code:

 ProductBox has used four arguments as specified below:

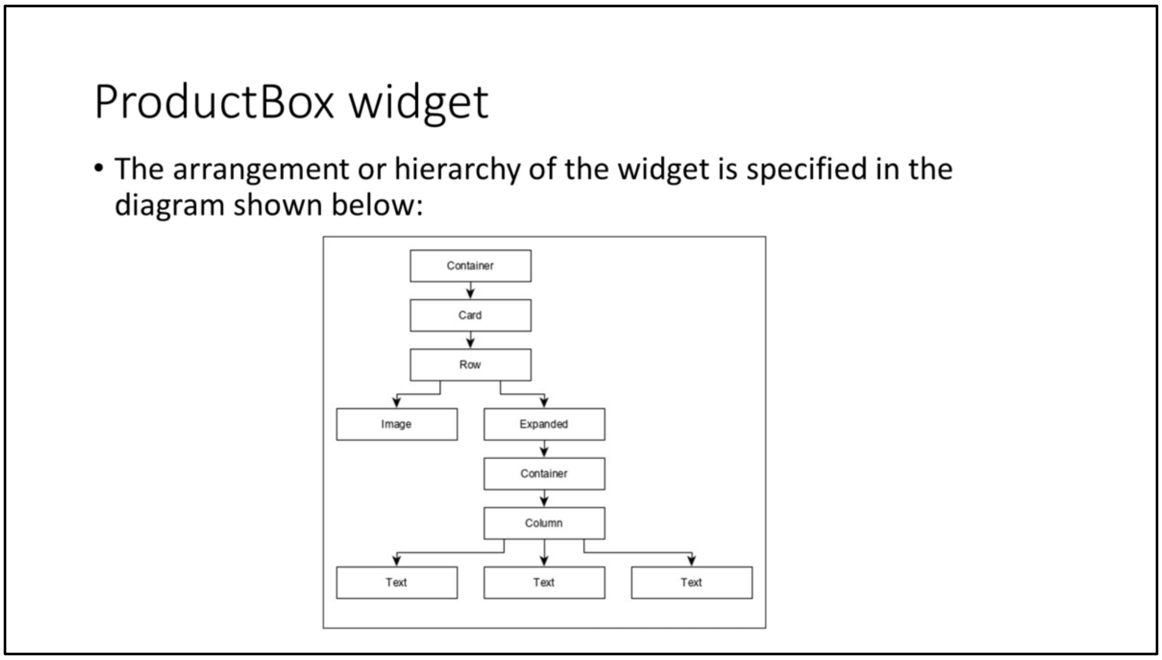
o name - Product name
o description - Product description
o price - Price of the product
o image - Image of the product

 ProductBox uses seven build-in widgets as specified below:

o Container
o Expanded
o Row
o Column
o Card
o Text
o Image

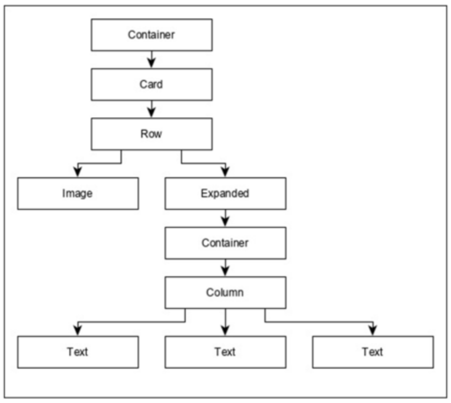
Code:

```
class ProductBox extends StatelessWidget { ProductBox({Key key, this.name,
this.description, this.price, this.image}) : super(key: key); final String name; final
String description; final int price; final String image; Widget build(BuildContext
context) { return Container( padding: EdgeInsets.all(2), height: 120, child: Card( child:
Row( mainAxisAlignment: MainAxisAlignment.spaceEvenly, children: [
Image.asset("assets/appimages/" + image), Expanded( child: Container( padding:
EdgeInsets.all(5), child: Column( mainAxisAlignment:
MainAxisAlignment.spaceEvenly, children: [ Text(this.name, style:
TextStyle(fontWeight: FontWeight.bold)), Text(this.description), Text("Price: " +
this.price.toString()), ], ))) ]))); } }
```



ProductBox widget

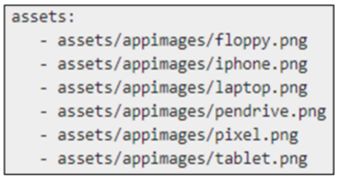
- The arrangement or hierarchy of the widget is specified in the diagram shown below:



$$\begin{tikzcd}
 & \text{Container} \arrow[d] & \\
 & \text{Card} \arrow[d] & \\
 & \text{Row} \arrow[dl] \arrow[d] & \\
\text{Image} & \text{Expanded} \arrow[d] & \\
 & \text{Container} \arrow[d] & \\
 & \text{Column} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Text} & \text{Text} & \text{Text}
\end{tikzcd}$$

ProductBox widget (contd…)

- Now, place some dummy image (see below) for product information in the assets folder of the application and configure the assets folder in the pubspec.yaml file as shown below –

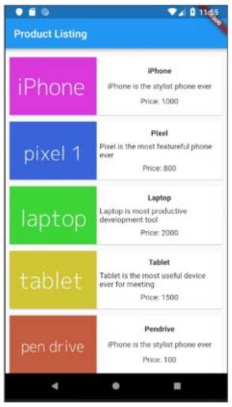


```
assets:
   - assets/appimages/floppy.png
   - assets/appimages/iphone.png
   - assets/appimages/laptop.png
   - assets/appimages/pendrive.png
   - assets/appimages/pixel.png
   - assets/appimages/tablet.png
```

ProductBox widget (contd…)

- Finally use the ProductBox widget in already completed code.

```
body: ListView(
   shrinkWrap: true, padding: const EdgeInsets.fromLTRB(2.0, 10.0, 2.0, 10.0)
   children: <Widget> [
      ProductBox(
         name: "iPhone",
         description: "iPhone is the stylist phone ever",
         price: 1000,
         image: "iphone.png"
      ),
      ProductBox(
         name: "Pixel",
         description: "Pixel is the most featureful phone ever",
         price: 800,
         image: "pixel.png"
      ),
      ProductBox(
         name: "Laptop",
         description: "Laptop is most productive development tool",
         price: 2000,
         image: "laptop.png"
      ),
```



iPhone
iPhone is the stylist phone ever
Price: 1000

Pixel
Pixel is the most featureful phone ever
Price: 800

Laptop
Laptop is most productive development tool
Price: 2000

Tablet
Tablet is the most useful device ever for meeting
Price: 1500

Pendrive
iPhone is the stylist phone ever
Price: 100

With shrinkWrap: true, you can change this behavior so that the ListView only occupies the space it needs

Gestures

- Flutter widgets support interaction through a special widget, GestureDetector.
- GestureDetector is an invisible widget having the ability to capture user interactions such as tapping, dragging, etc., of its child widgets.