Widgets

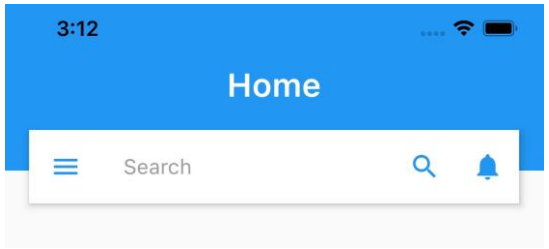
Mobile App Development

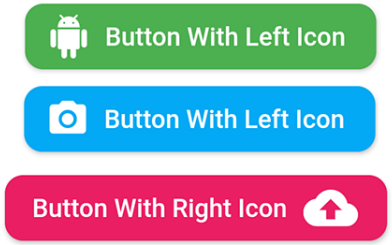
Lecture Set – 04

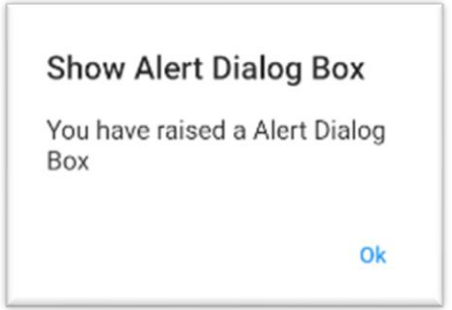
Engr. Ali Asghar Manjotho, Assistant Professor, CSE-MUET

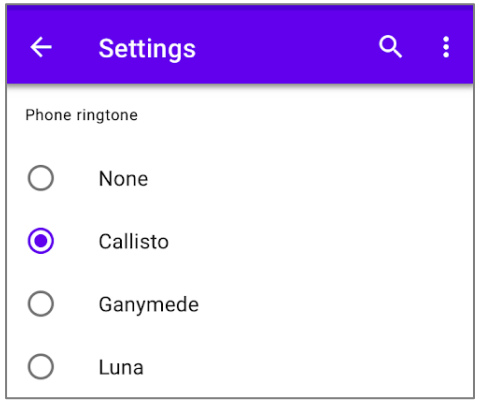
Widgets

Widgets are the building blocks of a Flutter app, and each widget is an immutable declaration of the user interface.



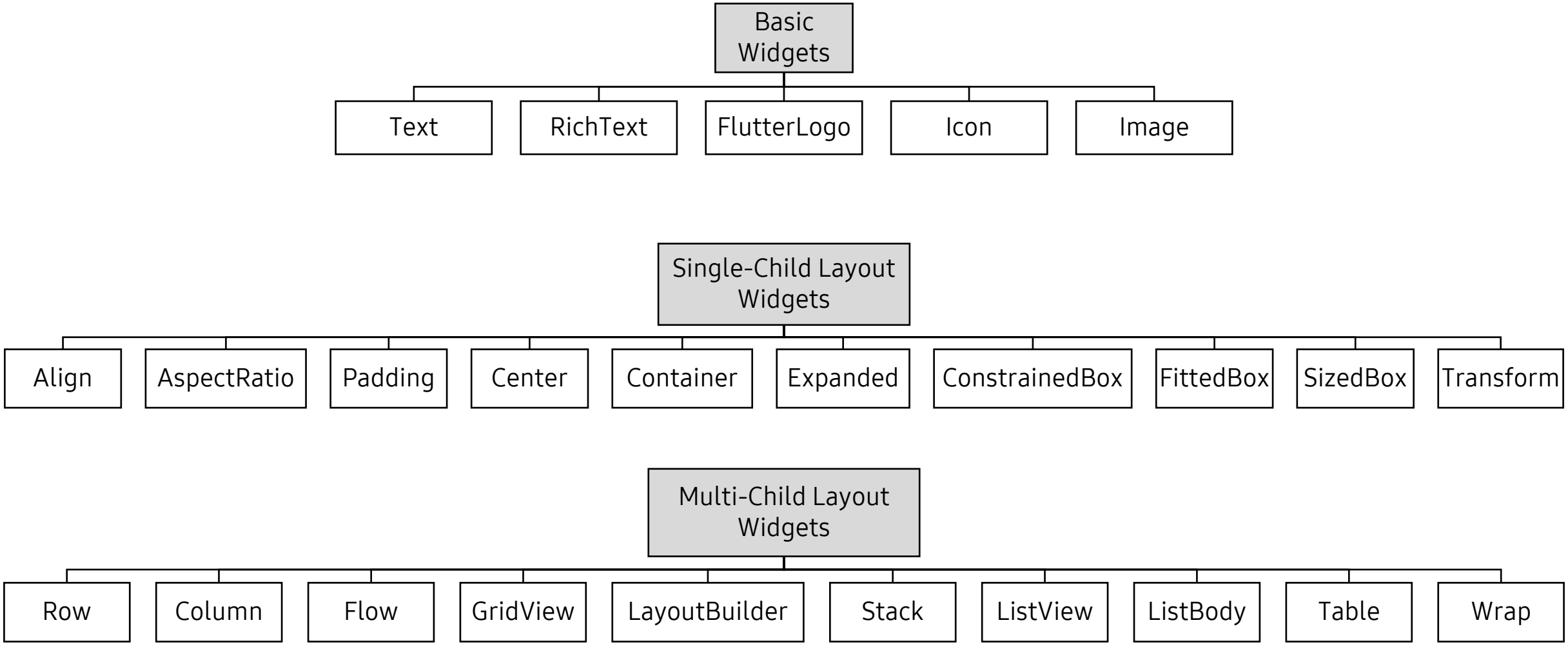




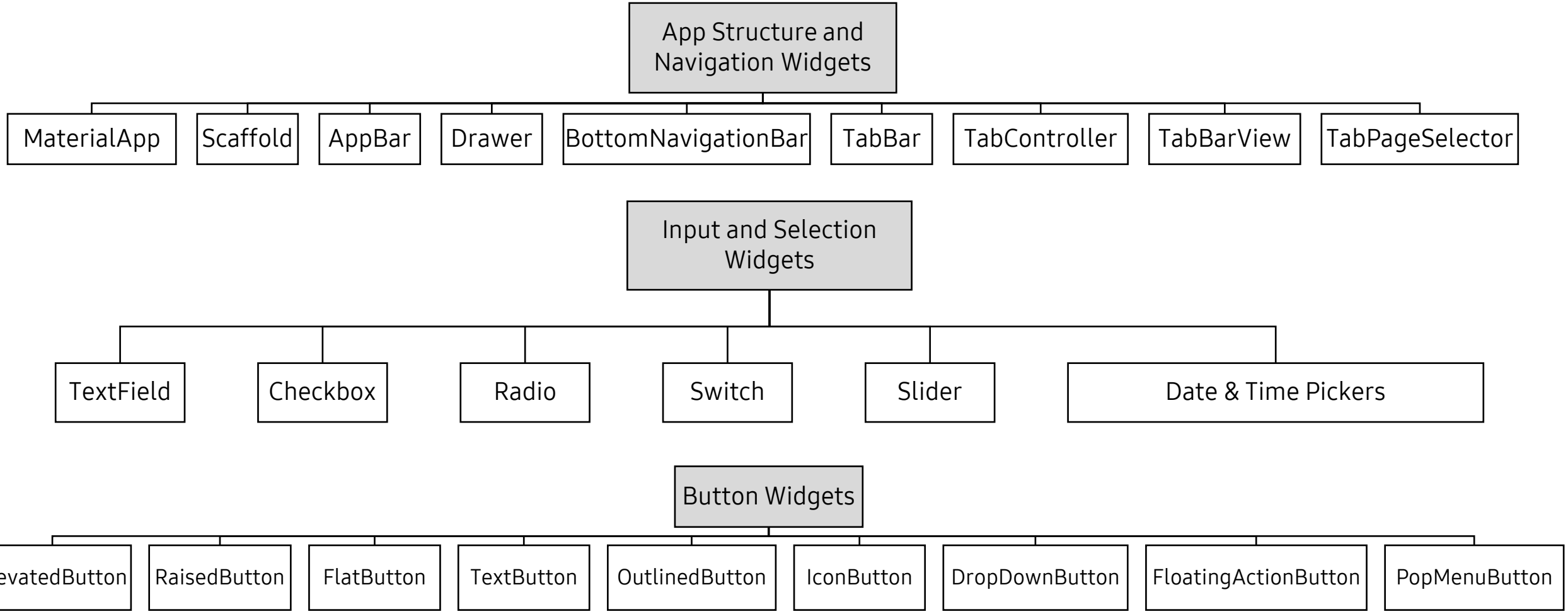


- In other words, widgets are configurations (instructions) for different parts of the UI.
- Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description.
- Placing the widgets together creates the widget tree.

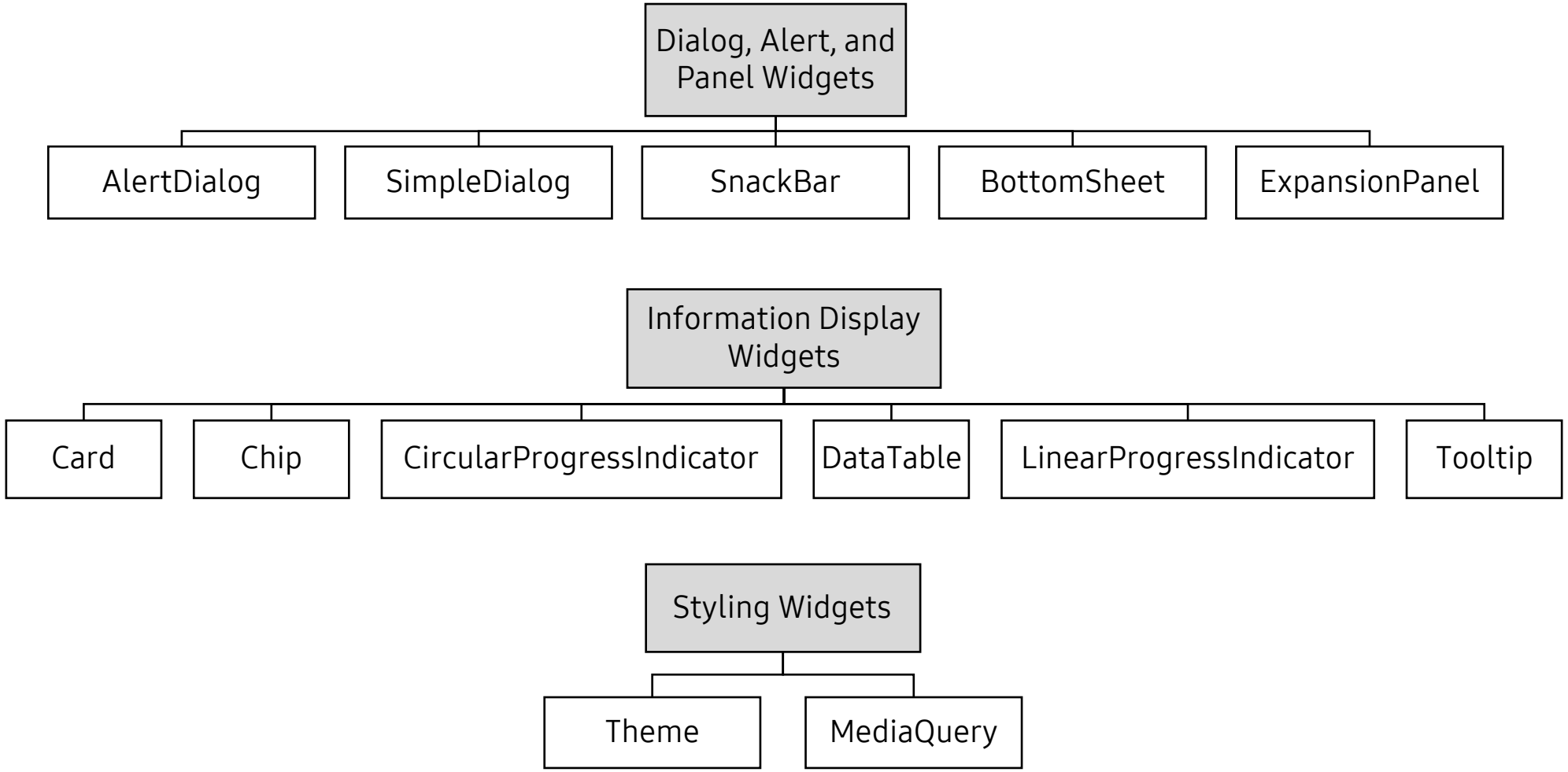
Common Widgets



Common Widgets



Common Widgets



Types of Widgets

- Flutter uses two commonly used widgets:

StatelessWidget

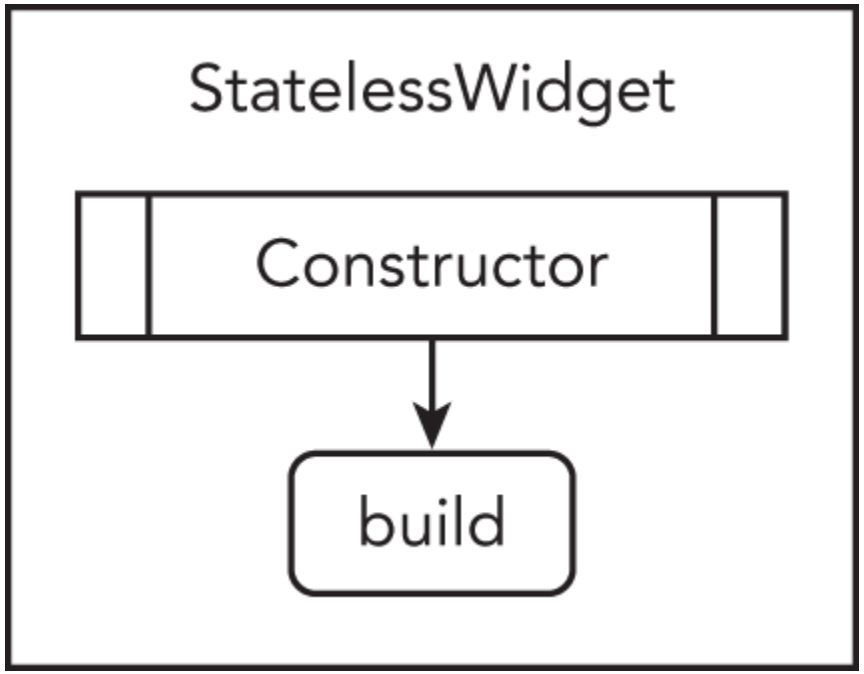
Widgets whose state cannot be altered once they are built

- The stateless widget is declared with one class, the StatelessWidget class.
- Stateless widget are useful when the part of the user interface you are describing does not depend on anything other than the configuration information in the object itself and the BuildContext in which the widget is inflated.
- These widgets are immutable once they are built i.e. any amount of change in the variables, icons, buttons, or retrieving data can not change the state of the app.

StatelessWidget Life Cycle

- Basic structure of a stateless widget.
  - Inherits `StatelessWidget` class.
  - Overrides and implements `build` method and return a widget.

```
class MyApp extends StatelessWidget {


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



StatelessWidget Life Cycle

- The build method of a stateless widget is typically only called in three situations:
  - First time the widget is inserted in the tree.
  - When the widget's parent changes its configuration.
  - When an InheritedWidget it depends on changes.

StatelfullWidget

Widgets whose state can be altered once they are built

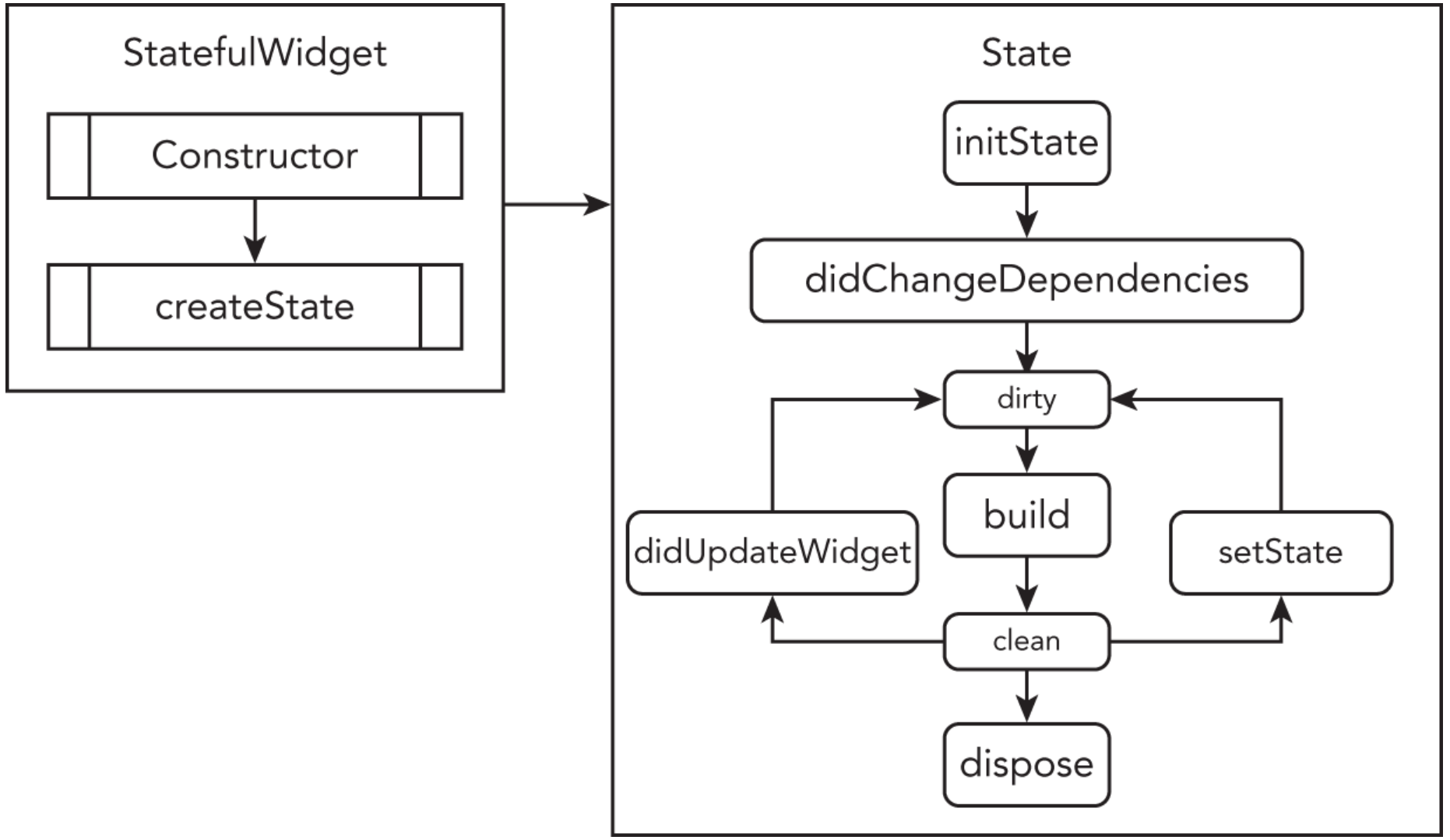
- The stateful widget is declared with two classes, the StatefulWidget class and the State class.
- These states are mutable and can be changed multiple times in their lifetime.
- Stateful widgets are useful when the part of the user interface you are describing can change dynamically.

StatelfullWidget

- Basic structure of a statelfull widget.
  - Inherits `StatefullWidget` class.
  - Overrides `createState` method and returns a state.
  - Inherits the `State` class, embraces the state variable and returns the widget with `build` method.

```
class MyApp extends StatefulWidget {

  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

StatefullWidget Life Cycle

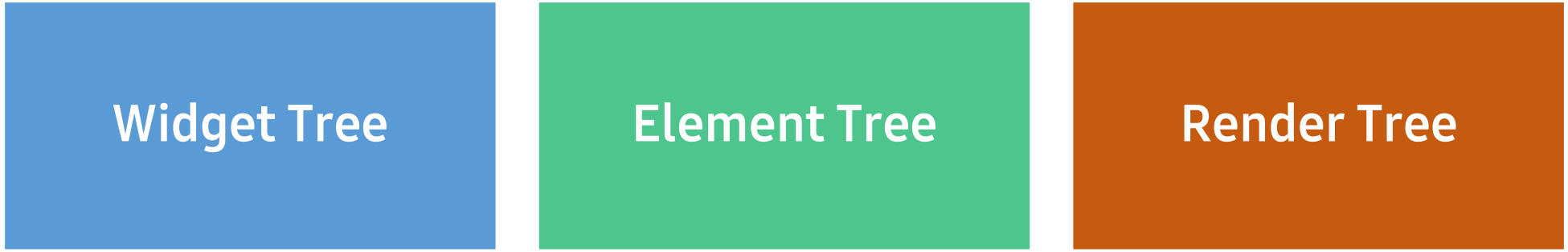


StatefullWidget Life Cycle

Life Cycle Method	Description
`initState()`	Called once when this object is inserted into the tree.
`dispose()`	Called when this object is removed from the tree permanently.
`didChangeDependencies()`	Called when the State object changes.
`didUpdateWidget(Contacts oldWidget)`	Called when the widget configuration changes.
`deactivate()`	Called when this object is removed from the tree.
`build(BuildContext context)`	Can be called multiple times to build the UI, and the BuildContext handles the location of this widget in the widget tree.
`setState()`	Notifies the framework that the state of this object has changed to schedule calling the build for the State object.

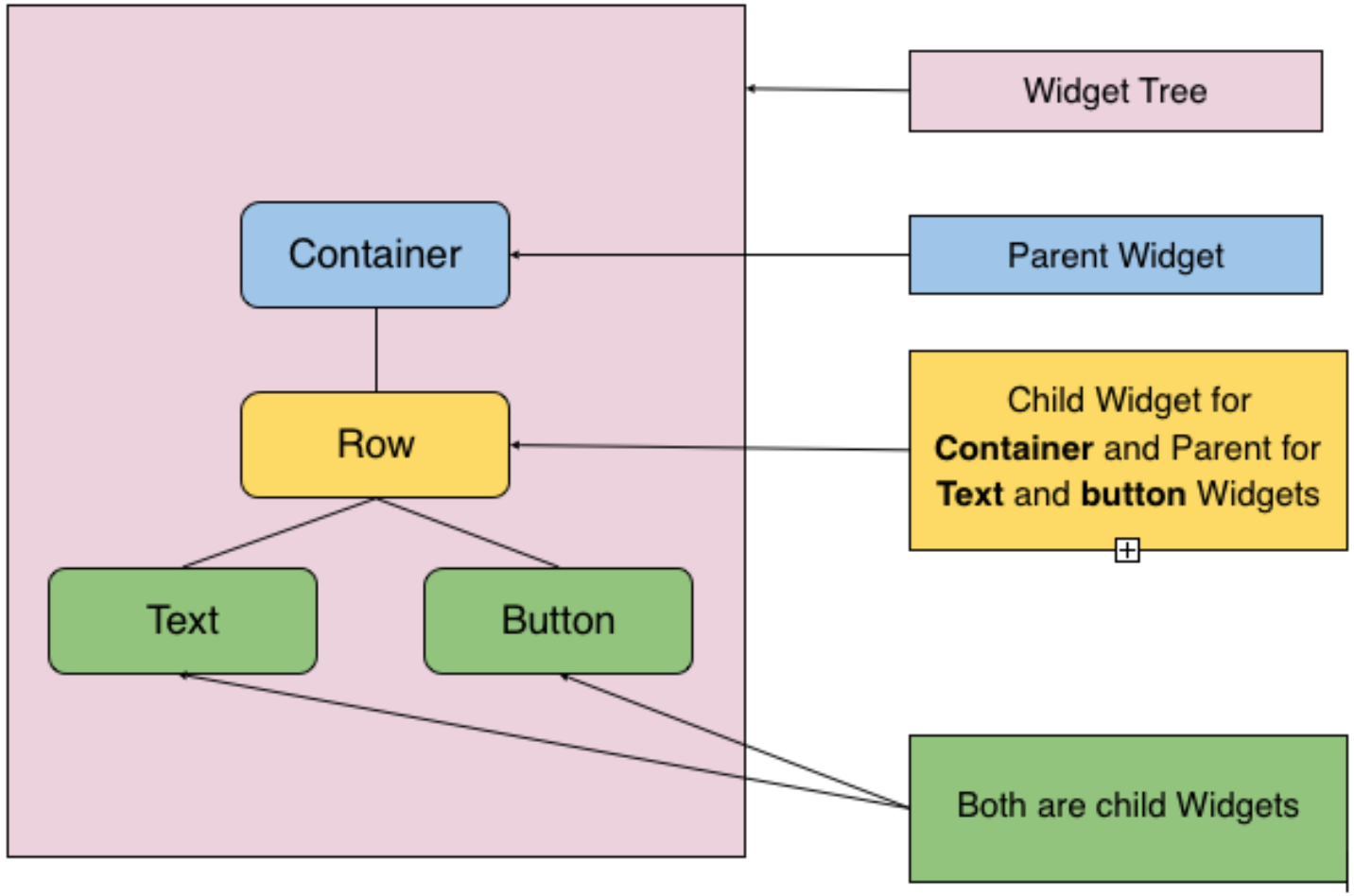
Widget, Element & Render Tree

The widget tree is created when you compose (nest) widgets; this is known as composition. Three trees are created:

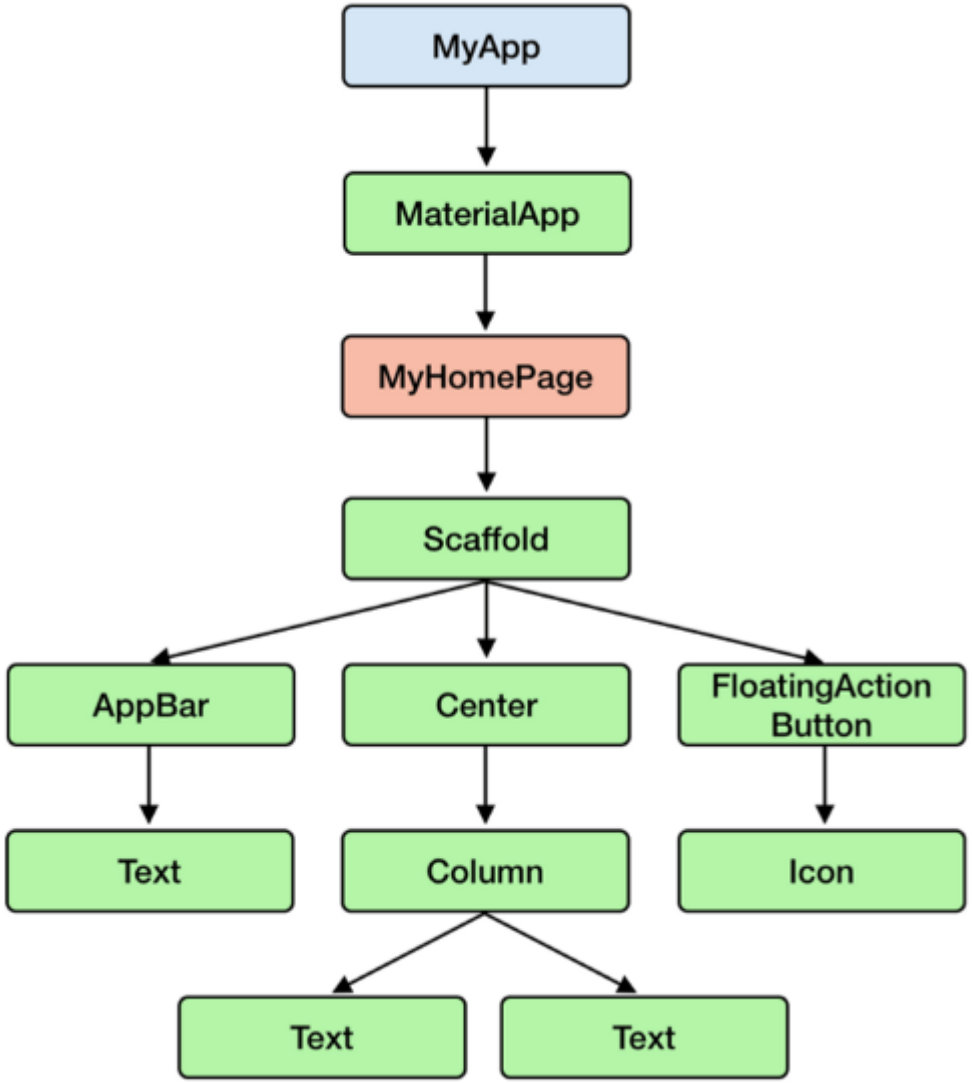


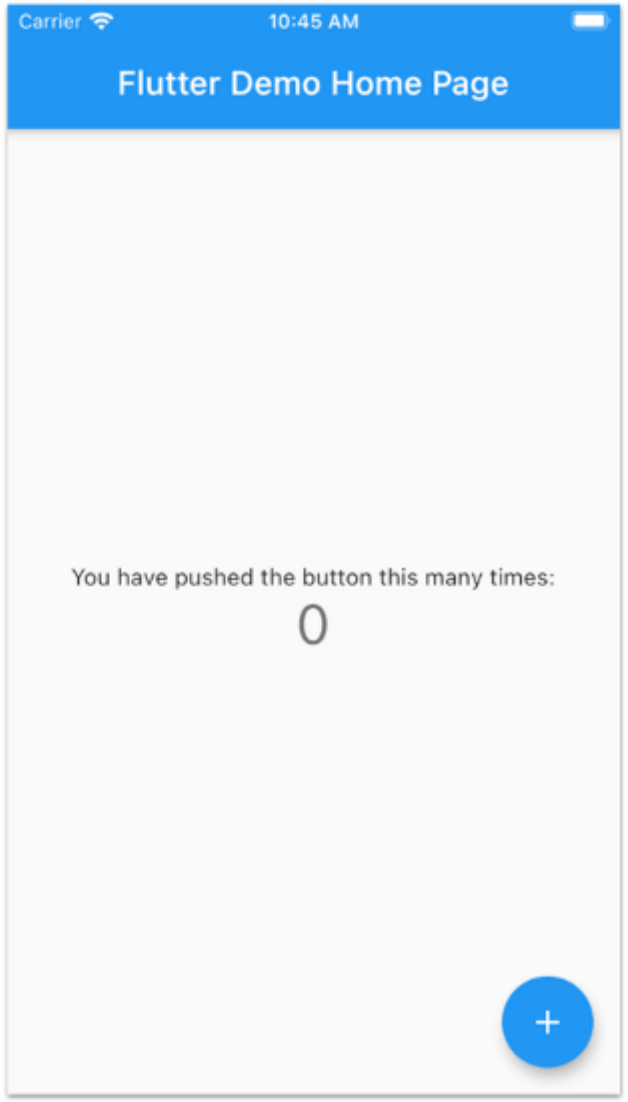
- **Widget Tree:** nested set of widgets to compose the user interface.
- **Element Tree:** represents each element mounted (rendered) on the screen.
- **Render Tree:** a low-level layout and painting system that inherits from the RenderObject.

Widget, Element & Render Tree

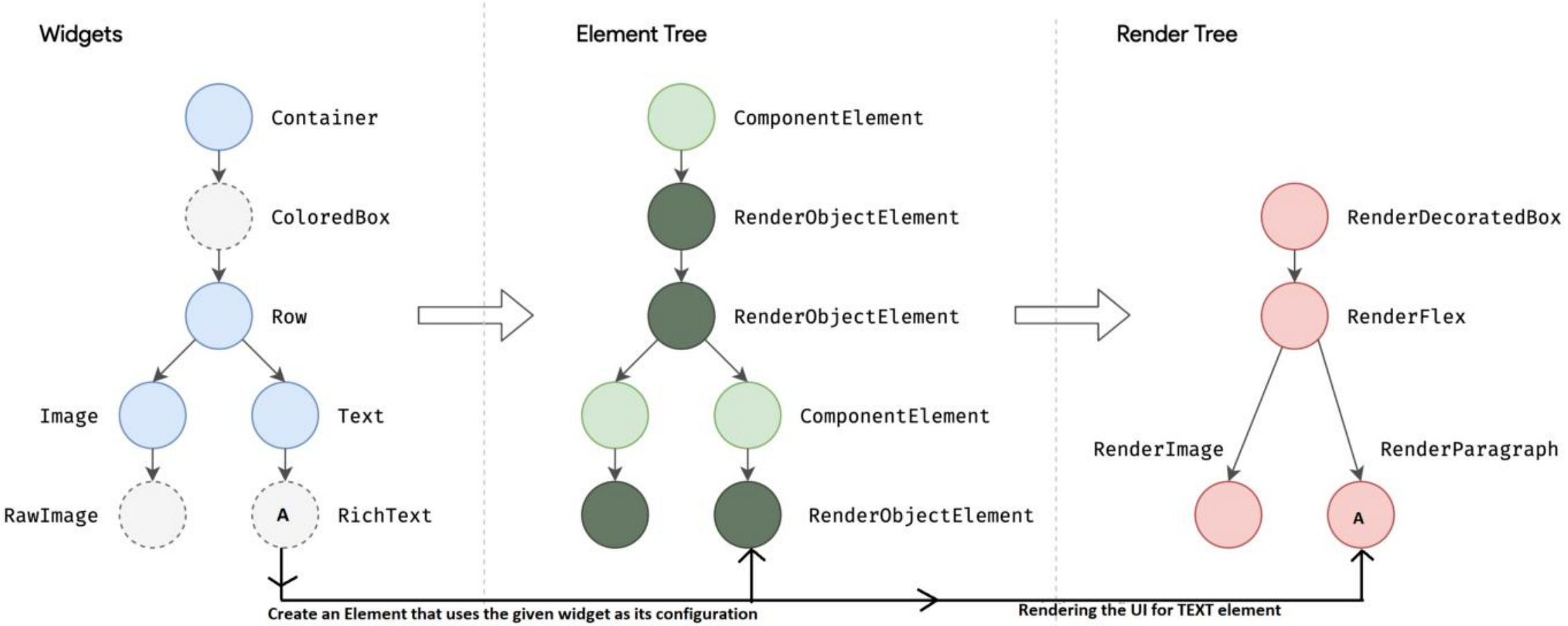


Widget, Element & Render Tree





Widget, Element & Render Tree



Widget, Element & Render Tree

- Widget is simply a configuration. We tell the framework what we want to see in different portions of the screen using widgets.
- Widgets are immutable, so we can’t update them. Once declared in the widget tree, they can’t be changed.
- Element is the information hub configured by a widget.
- There is a corresponding element in the element tree for every widget in the widget tree.
- When we want to know about the size, parent, children, and location of a widget in the widget tree, we get the answer from its element.
- Furthermore, the State object of a StatefulWidget is permanently associated with the element of the StatefulWidget.

Widget, Element & Render Tree

- Once we provide the initial widget tree, the framework creates the corresponding element and render trees from top to down.
- The nodes in the element and render trees are updated (if possible) in one top-down pass.
- Widget tree is maintained by app developers: widgets only hold the configuration data.
- Element tree is the logical structure of the UI: it is (re)constructed by the widget tree and knows about the location of the widget in the tree.

Widget, Element & Render Tree

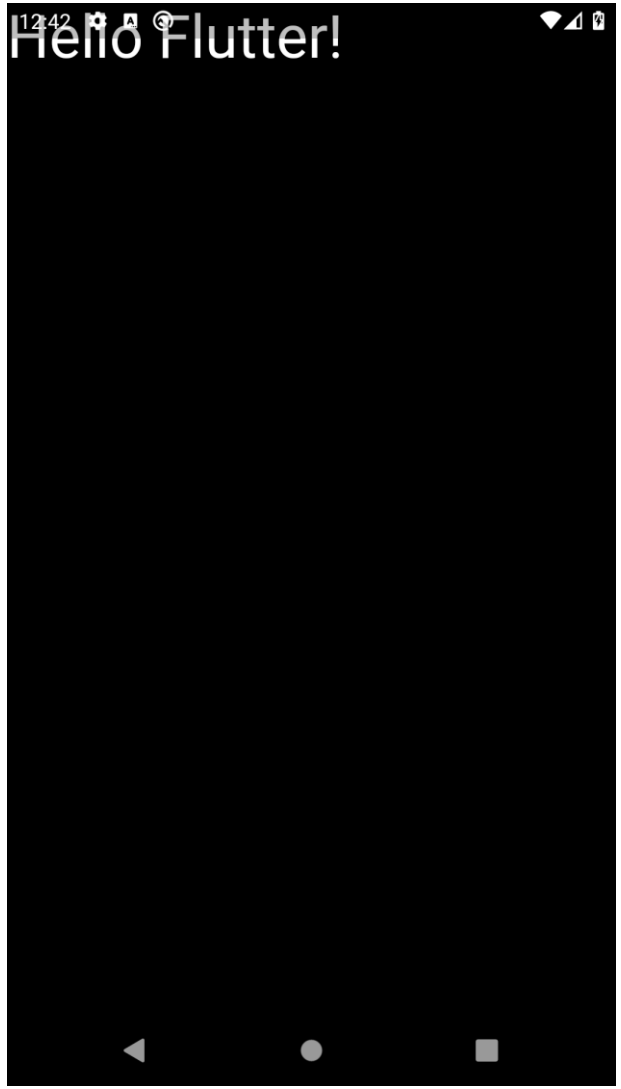
- Render tree is the data structure that is managed by the element tree: the geometry is computed during the layout phase and used in the painting and hit testing.

Text Widgets

Example 01: Text Demo

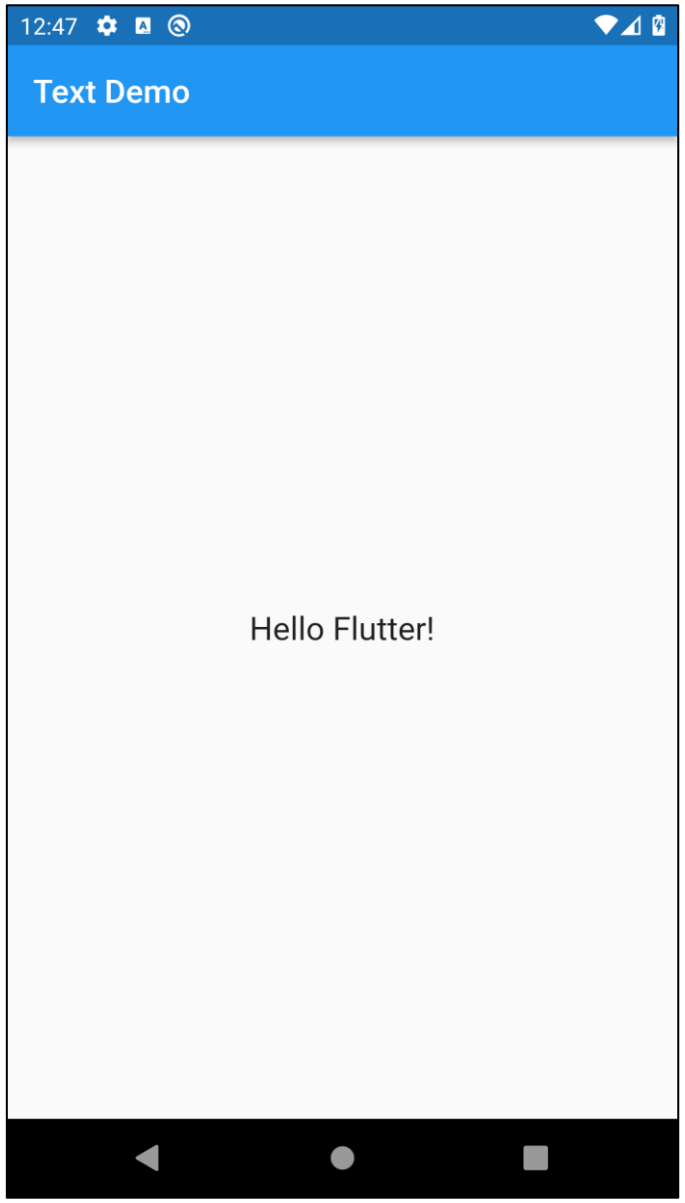
- Text widget represents a static text portion within the UI.

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

void main() {
  runApp(const Text(
    'Hello Flutter!',
    textDirection: TextDirection.ltr,
    style: TextStyle(fontSize: 40.0),
  ));
}
```



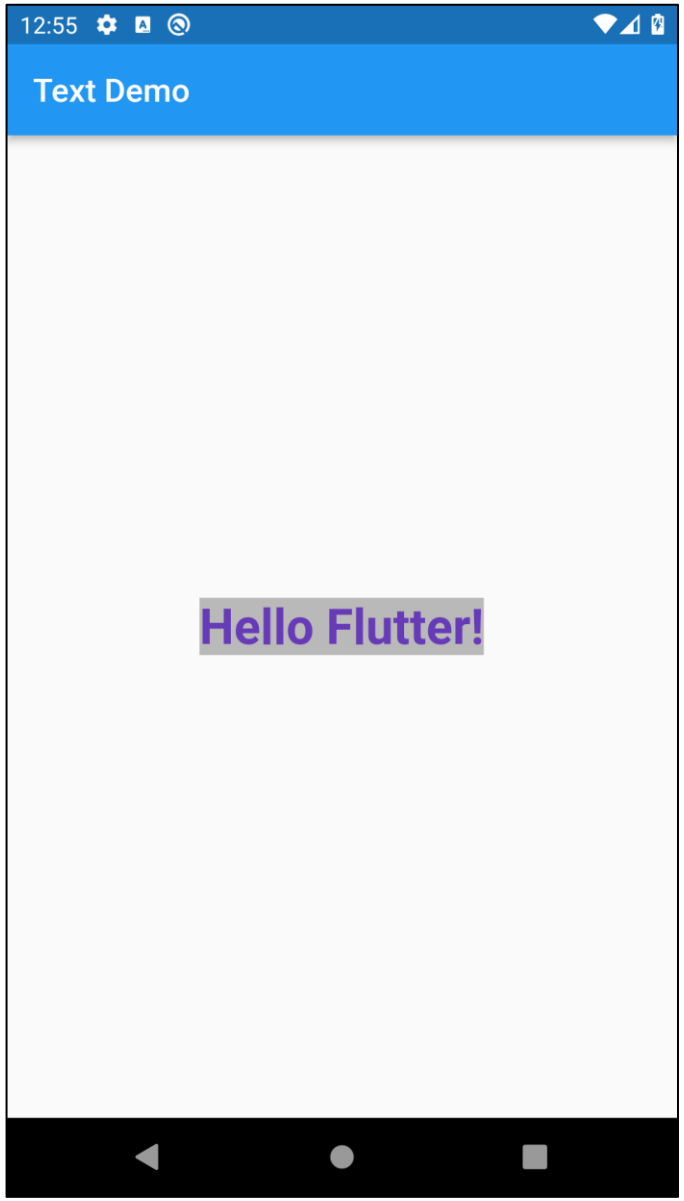
Example 02: Text Demo (Material App)

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

void main() {
  runApp(MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Text Demo',
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Text Demo'),
      ),
      body: const Center(
        child: Text(
          'Hello Flutter!',
          style: TextStyle(fontSize: 20.0),
        ),
      ),
    ),
  ));
}
```



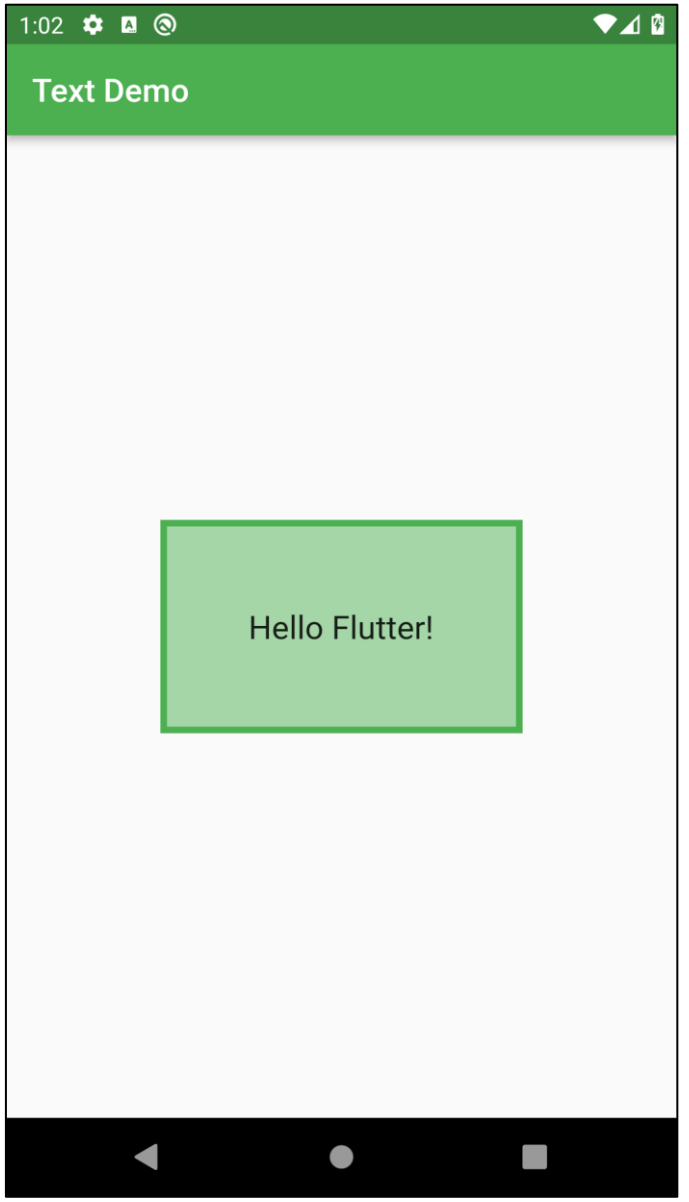
Example 03: Text Demo (Material App)

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

void main() {
  runApp(MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Text Demo',
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Text Demo'),
      ),
      body: const Center(
        child: Text(
          'Hello Flutter!',
          style: TextStyle(
            color: Colors.deepPurple,
            fontSize: 30.0,
            fontWeight: FontWeight.bold,
            backgroundColor: Colors.black26,
          ),
        ),
      ),
    ),
  ));
}
```



Example 04: Text Demo (Container)

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

void main() {
  runApp(MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Text Demo',
    theme: ThemeData(primarySwatch: Colors.green),
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Text Demo'),
      ),
      body: Center(
        child: Container(
          padding: const EdgeInsets.all(50.0),
          decoration: BoxDecoration(
              color: Colors.green[200],
              border: Border.all(
                  color: Colors.green, style: BorderStyle.solid, width: 4.0)),
          child: const Text(
            'Hello Flutter!',
            style: TextStyle(fontSize: 20.0),
          ),),),),
  ));
}
```



Example 05: Text Demo (TextStyle)

```dart
Text(
  'Hello Flutter!',
  style: TextStyle(
    fontSize: 40.0,
    color: Colors.deepOrange,
    fontWeight: FontWeight.bold,
    decoration: TextDecoration.combine([
      TextDecoration.underline,
      TextDecoration.overline,
    ]),
    decorationThickness: 2.0,
    decorationColor: Colors.green,
    decorationStyle: TextDecorationStyle.wavy,
    letterSpacing: 10.0,
    wordSpacing: 20.0,
  ),
)
```

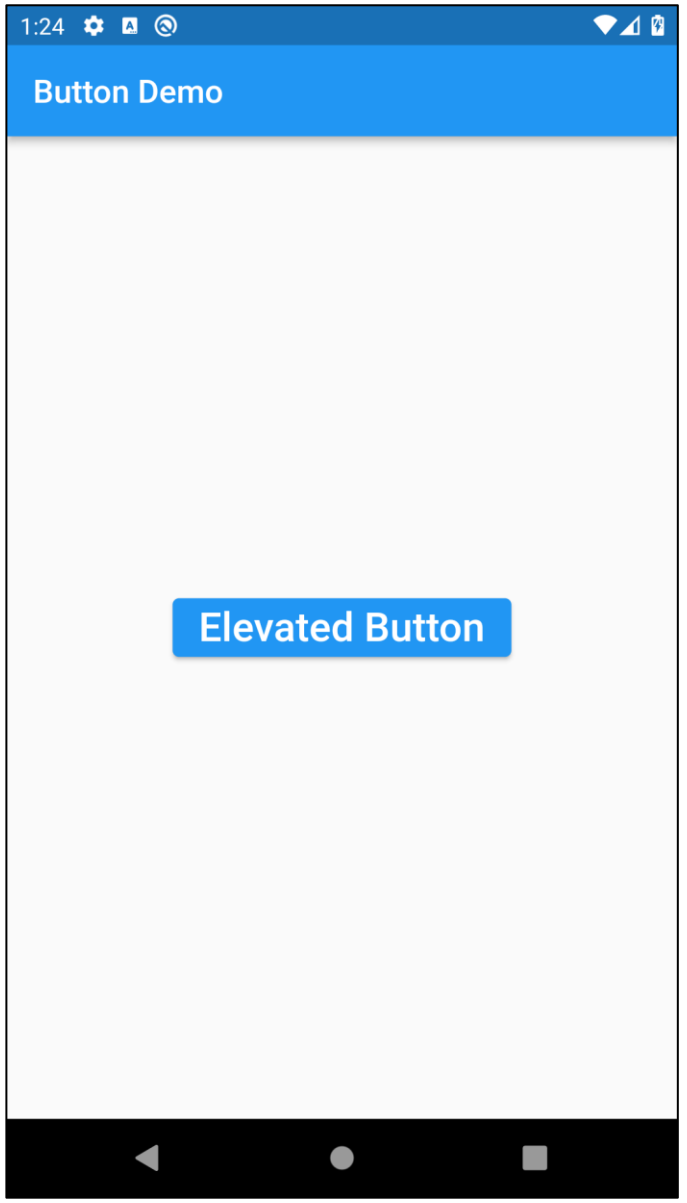
Example 06: Text Demo (Foreground, BoxShadow)

```
Text(
  'Hello Flutter!',
  style: TextStyle(
    fontSize: 40.0,
    fontWeight: FontWeight.bold,
    letterSpacing: 10.0,
    foreground: Paint()
      ..color = Colors.deepOrange
      ..strokeWidth = 2.0
      ..style = PaintingStyle.stroke,
    shadows: const [
      BoxShadow(
          color: Colors.black38,
          offset: Offset(15.0, 15.0),
          blurRadius: 15.0,
          blurStyle: BlurStyle.normal)
    ],
  ),
)
```

Button Widgets

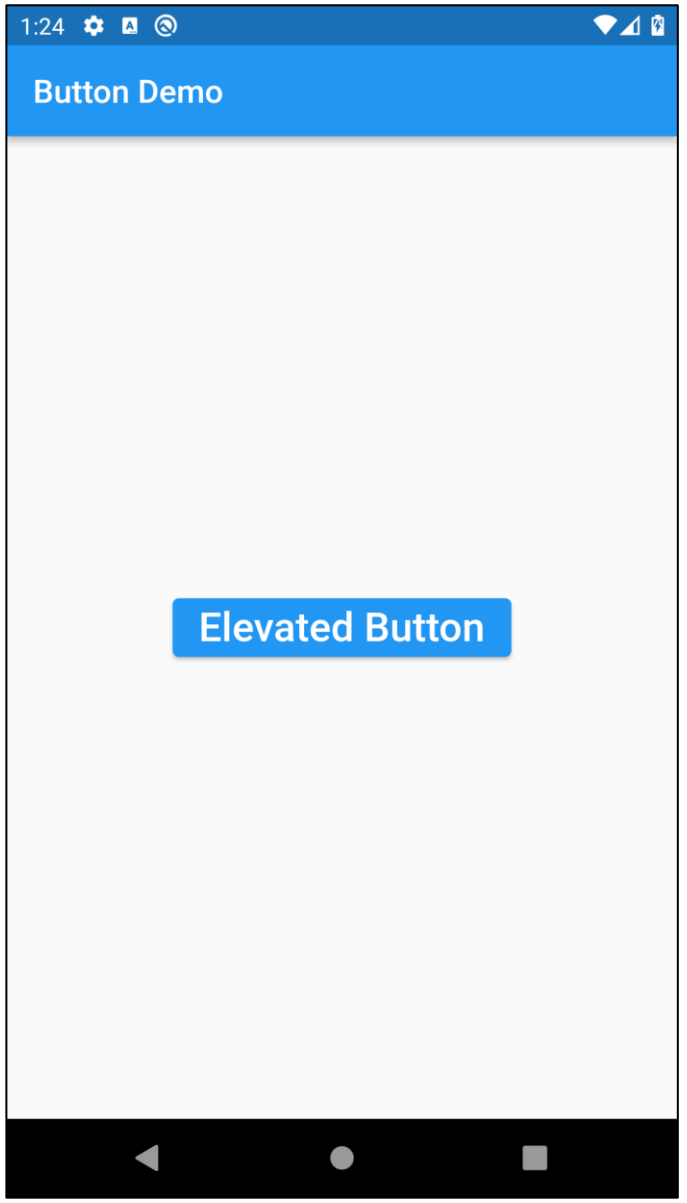
Example 07: Elevated Button Demo

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

void main() {
  runApp(MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Button Demo',
    theme: ThemeData(primarySwatch: Colors.deepOrange),
    home: MyApp(),
  ));
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
```



Example 07: Elevated Button Demo

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Button Demo'),
    ),
    body: Center(
      child: ElevatedButton(
        onPressed: () {
          print('Button Clicked...');
        },
        child:
            const Text('Elevated Button’,
            style: TextStyle(fontSize: 25.0)),
      ),
    ),
  );
 }
}
```

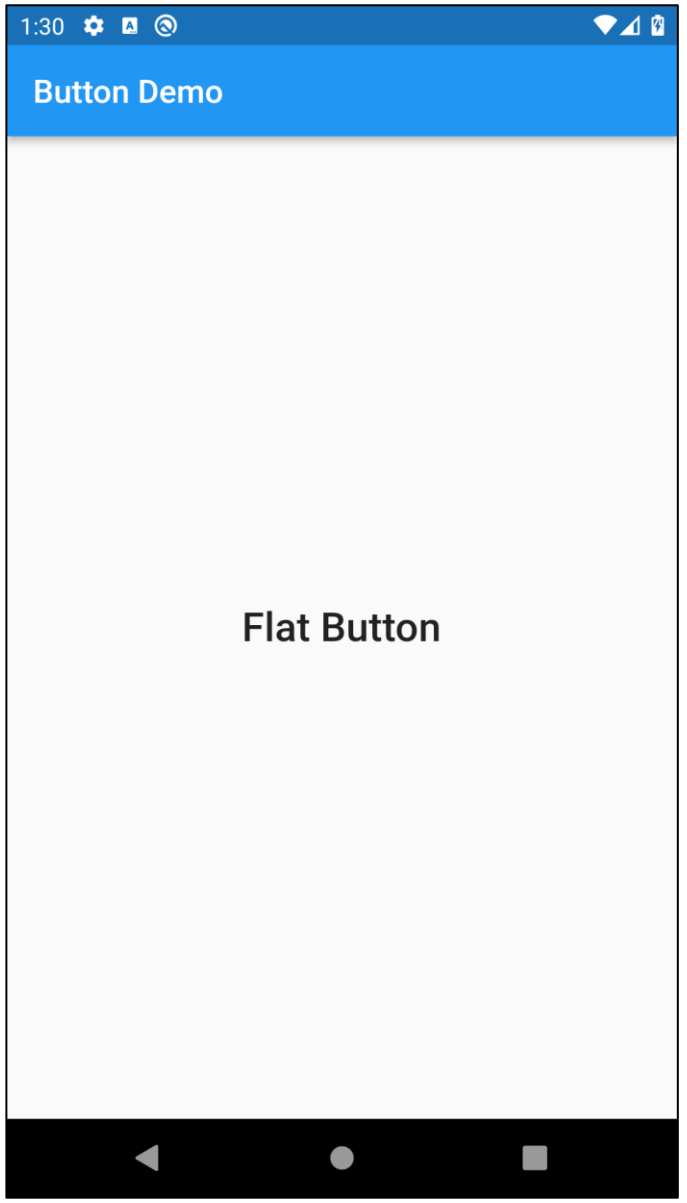


Example 08: Raised Button Demo

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Button Demo'),
    ),
    body: Center(
      child: RaisedButton(
        onPressed: () {
          print('Button Clicked...');
        },
        child:
            const Text('Raised Button’,
            style: TextStyle(fontSize: 25.0)),
      ),
    ),
  );
 }
}
```

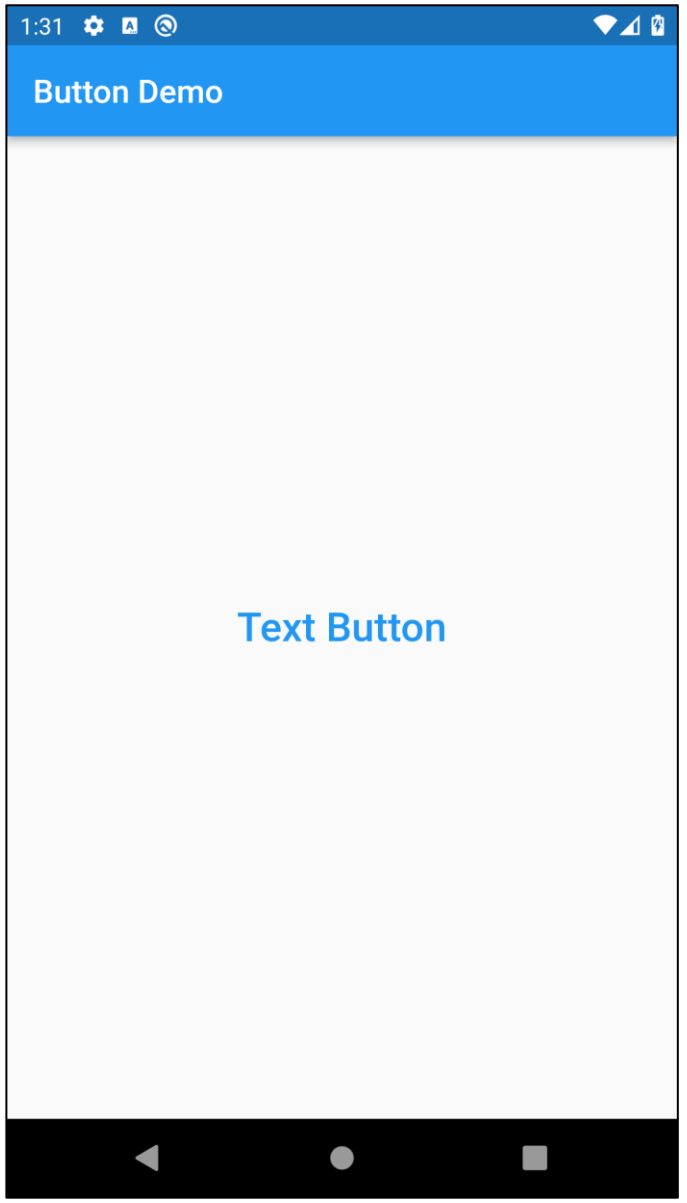
Example 09: Flat Button Demo

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Button Demo'),
    ),
    body: Center(
      child: FlatButton(
        onPressed: () {
          print('Button Clicked...');
        },
        child:
            const Text('Flat Button',
            style: TextStyle(fontSize: 25.0)),
      ),
    ),
  );
 }
}
```



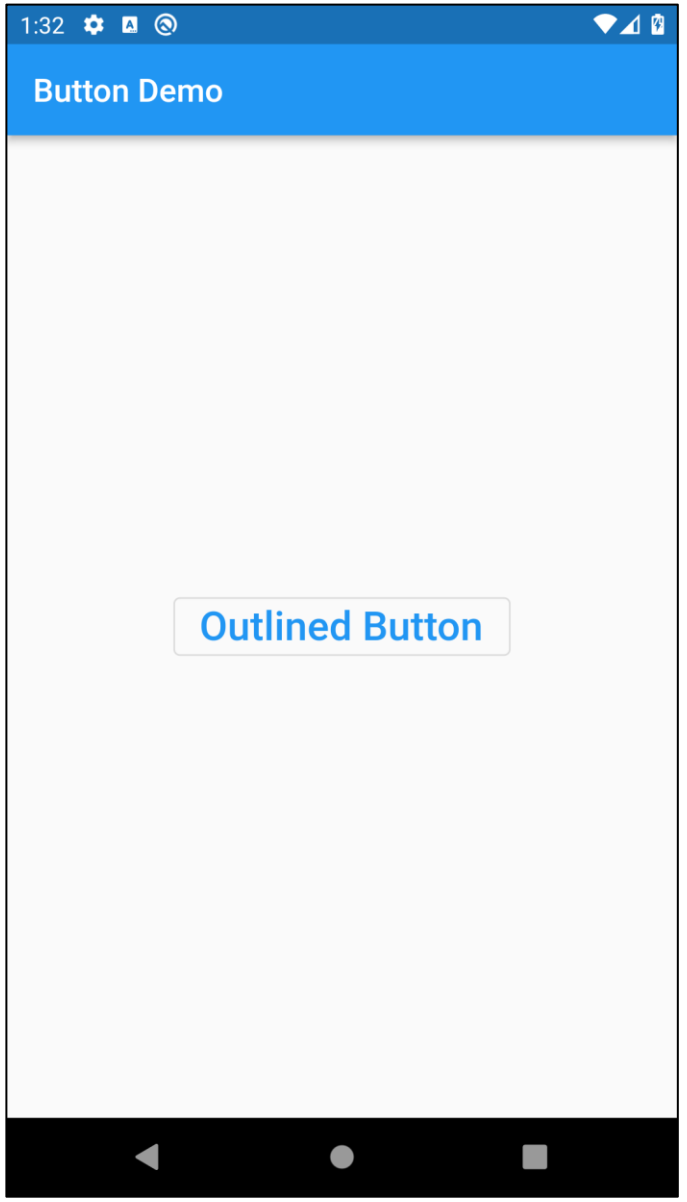
Example 10: Text Button Demo

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Button Demo'),
    ),
    body: Center(
      child: TextButton(
        onPressed: () {
          print('Button Clicked...');
        },
        child:
            const Text('Text Button',
            style: TextStyle(fontSize: 25.0)),
      ),
    ),
  );
 }
}
```



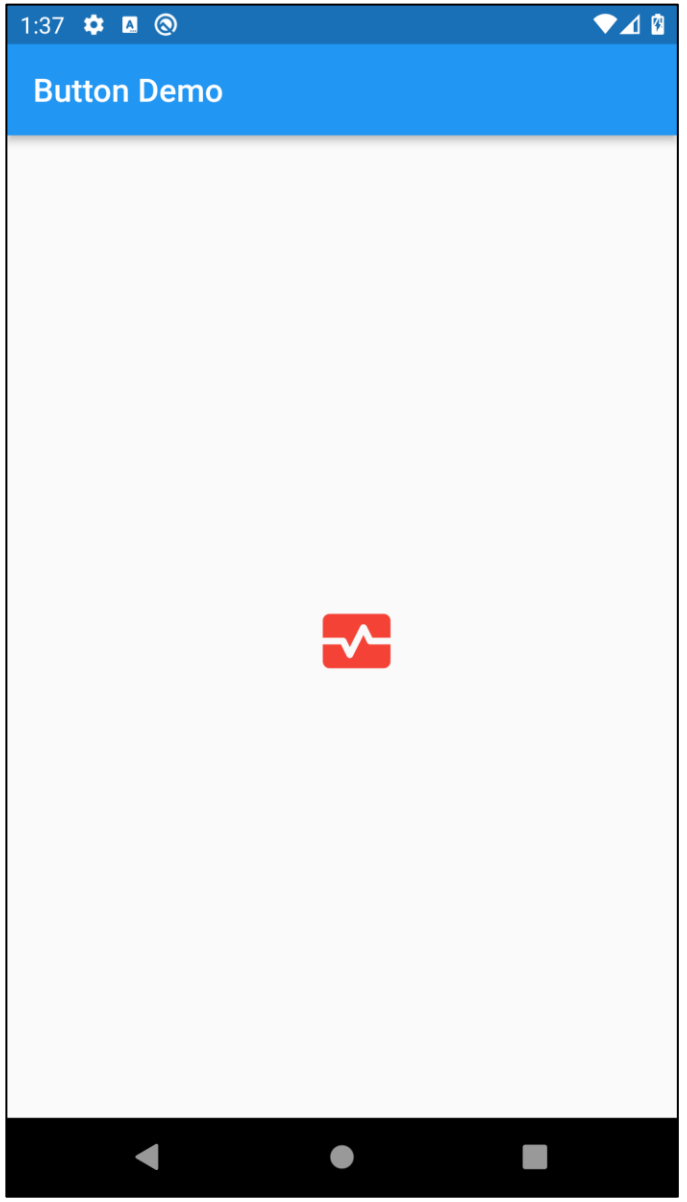
Example 11: Outlined Button Demo

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Button Demo'),
    ),
    body: Center(
      child: OutlinedButton(
        onPressed: () {
          print('Button Clicked...');
        },
        child:
            const Text('Outlined Button',
            style: TextStyle(fontSize: 25.0)),
      ),
    ),
  );
 }
}
```



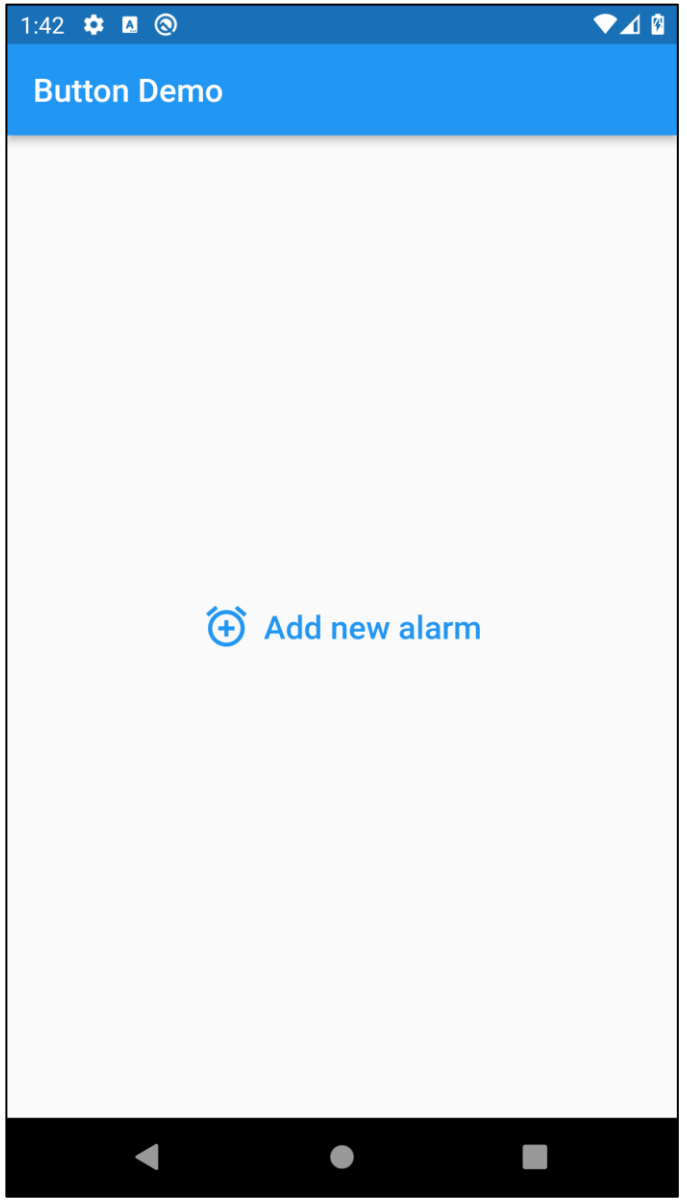
Example 12: Icon Button Demo

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Button Demo'),
    ),
    body: Center(
      child: IconButton(
        onPressed: () {
          print('Button Clicked...');
        },
        icon: const Icon(
          Icons.monitor_heart,
          color: Colors.red,
          size: 50.0,
        ),),),
  );
  }
}
```



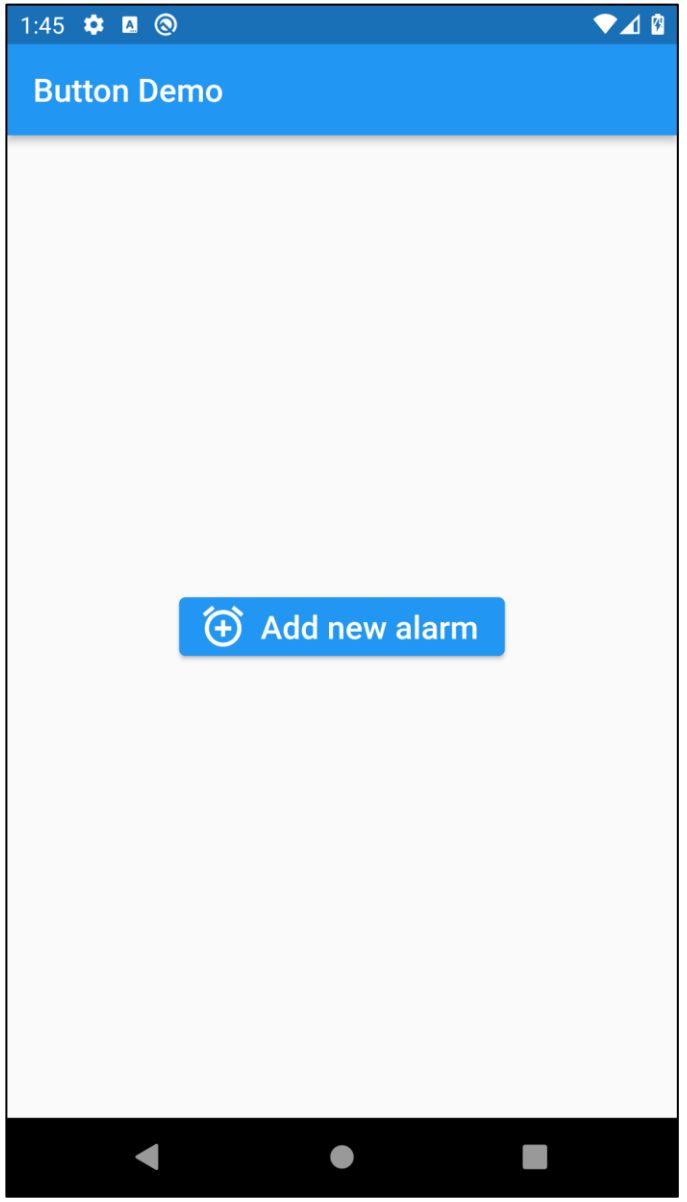
Example 13: Text Button with Icon Demo

```
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Button Demo'),
      ),
      body: Center(
        child: TextButton.icon(
          onPressed: () {
            print('Button Clicked...');
          },
          label: const Text(
            'Add new alarm',
            style: TextStyle(fontSize: 20.0),
          ),
          icon: const Icon(
            Icons.alarm_add,
            color: Colors.blue,
            size: 30.0,
          ),),),);
  }
```



Example 14: Elevated Button with Icon Demo

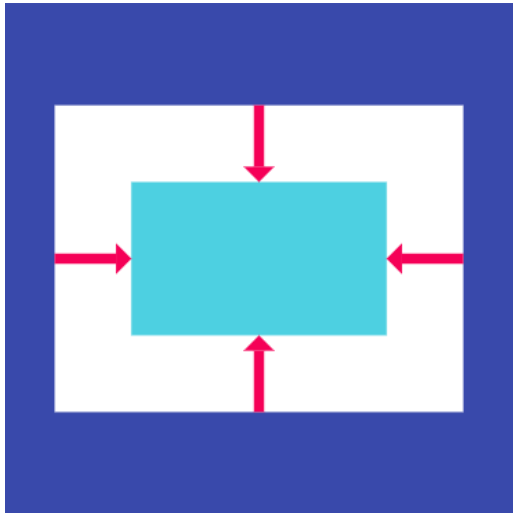
```
@override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Button Demo'),
      ),
      body: Center(
        child: ElevatedButton.icon(
          onPressed: () {
            print('Button Clicked...');
          },
          label: const Text(
            'Add new alarm',
            style: TextStyle(fontSize: 20.0),
          ),
          icon: const Icon(
            Icons.alarm_add,
            color: Colors.white,
            size: 30.0,
          ),),),);
  }
```



Center Widget

Center Widget

A widget that centers its child within itself



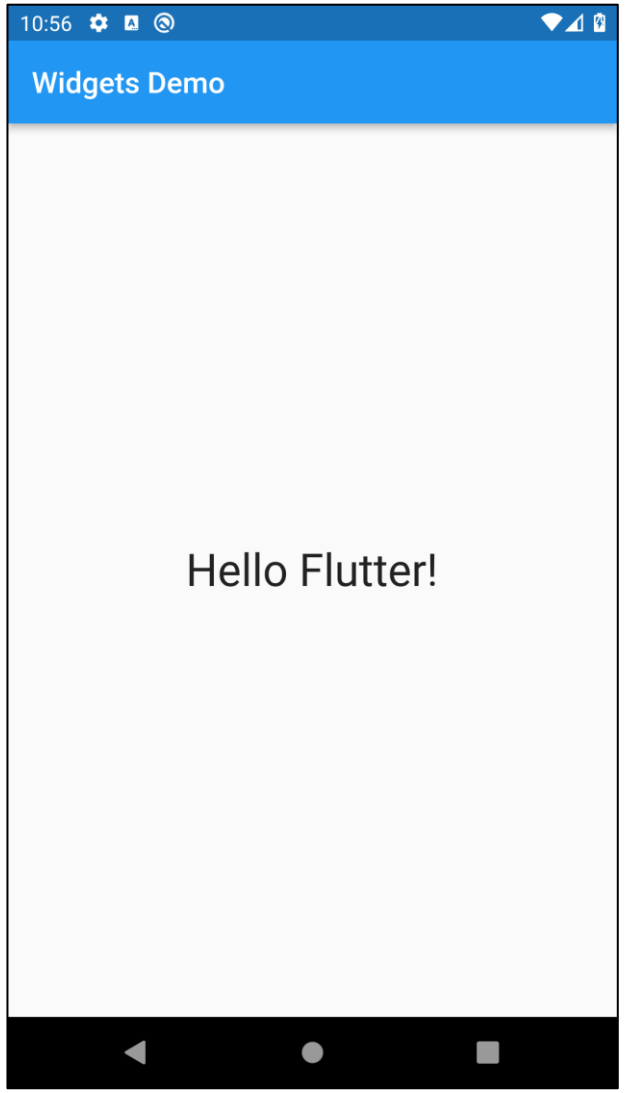
- This widget will be as big as possible if its dimensions are constrained and widthFactor and heightFactor are null.
- If a dimension is unconstrained and the corresponding size factor is null then the widget will match its child's size in that dimension.

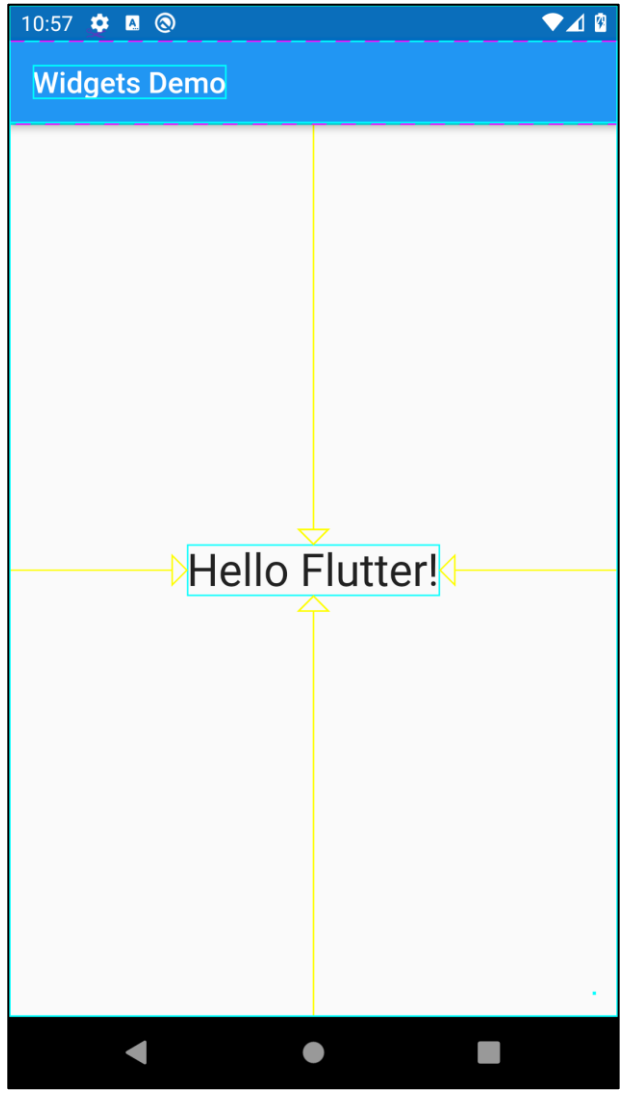
Example 15: Center Widget Demo

```
Center(
    child: Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
),
```

To paint debug information, enable the following property in main function

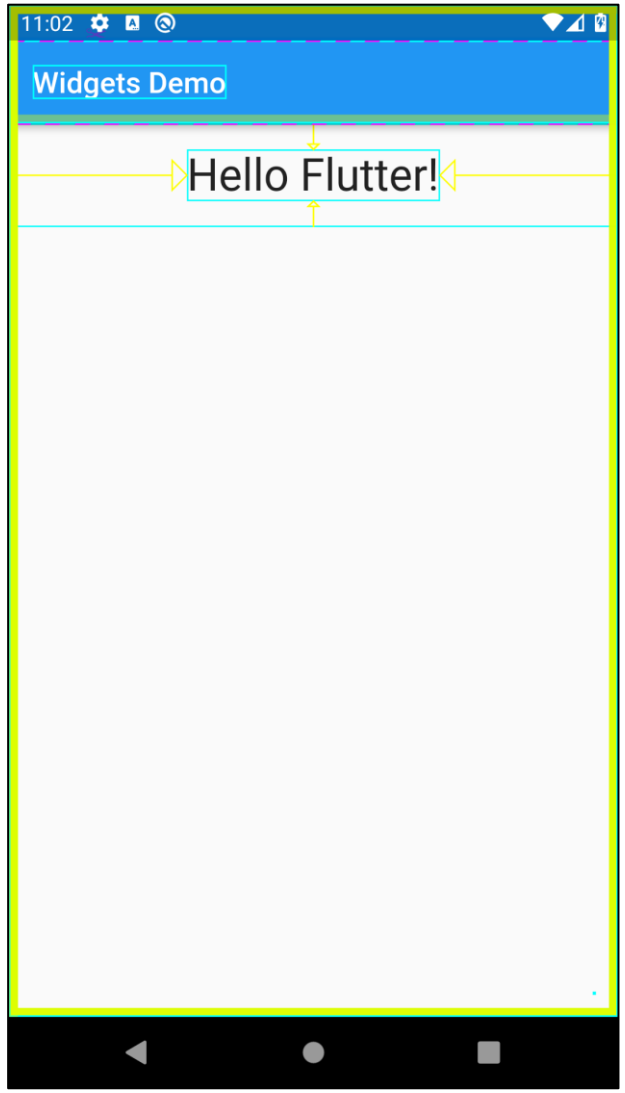
```
void main() {
  debugPaintSizeEnabled = true;
  .
  .
  .
}
```





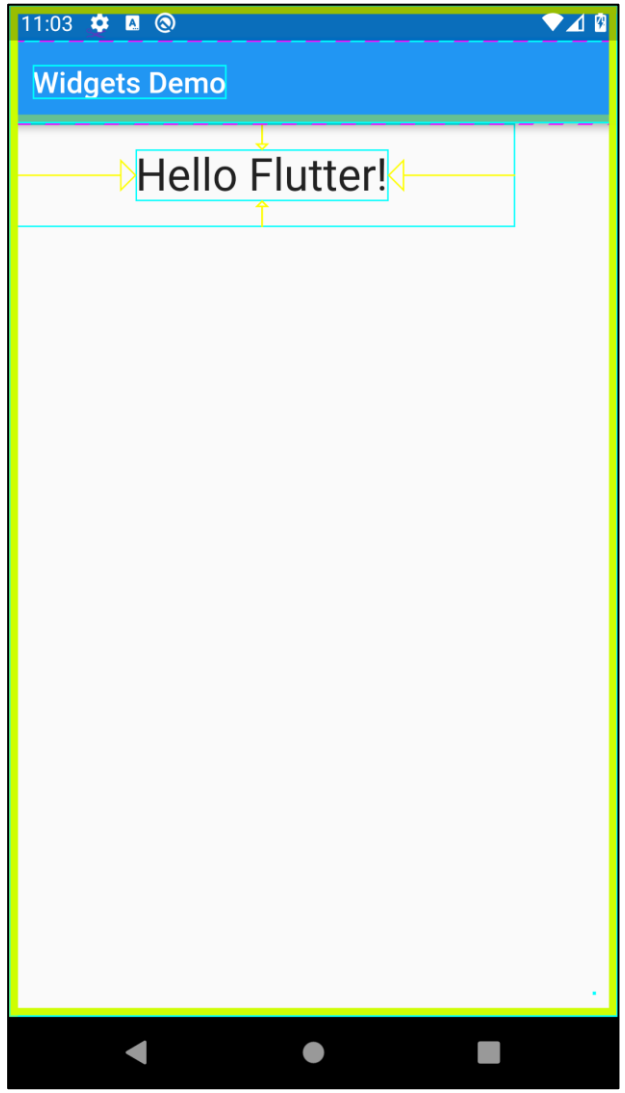
Example 16: Center Widget Demo

```
Center(
    heightFactor: 2.0,
    child: Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
),
```



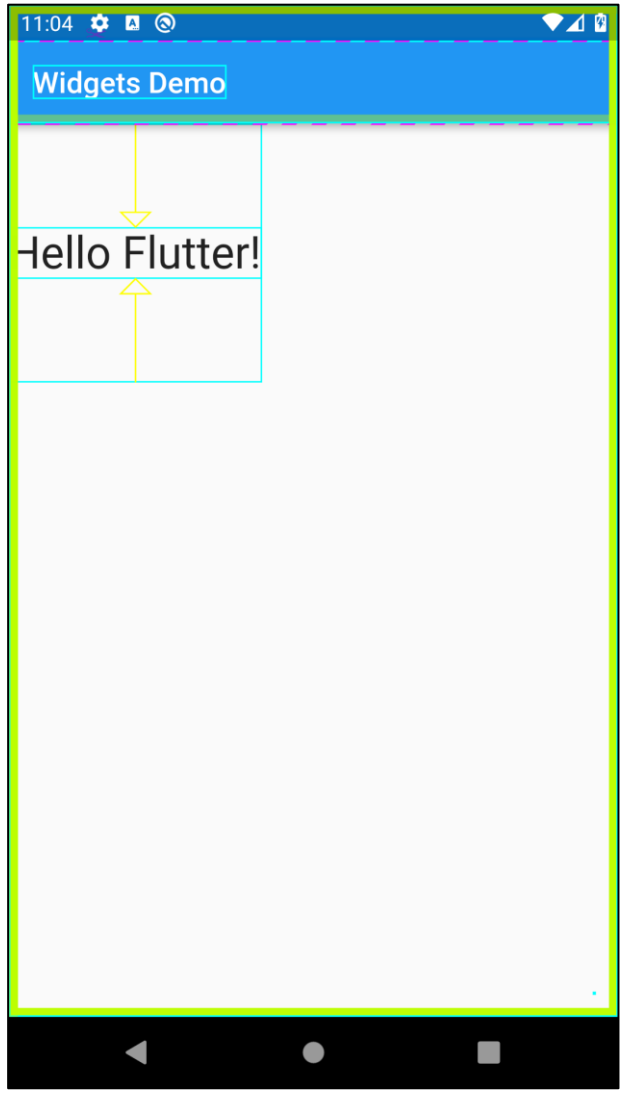
Example 17: Center Widget Demo

```
Center(
    heightFactor: 2.0,
    widthFactor: 2.0,
    child: Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
),
```



Example 18: Center Widget Demo

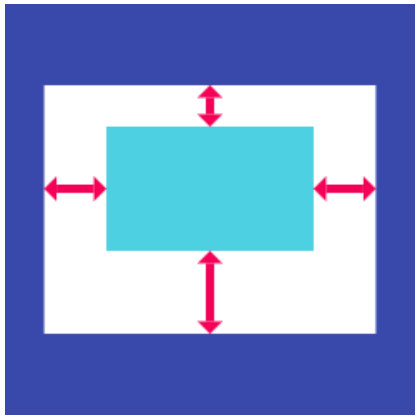
```
Center(
    heightFactor: 5.0,
    widthFactor: 1.0,
    child: Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
),
```



Padding Widget

Padding Widget

A widget that insets its child by the given padding

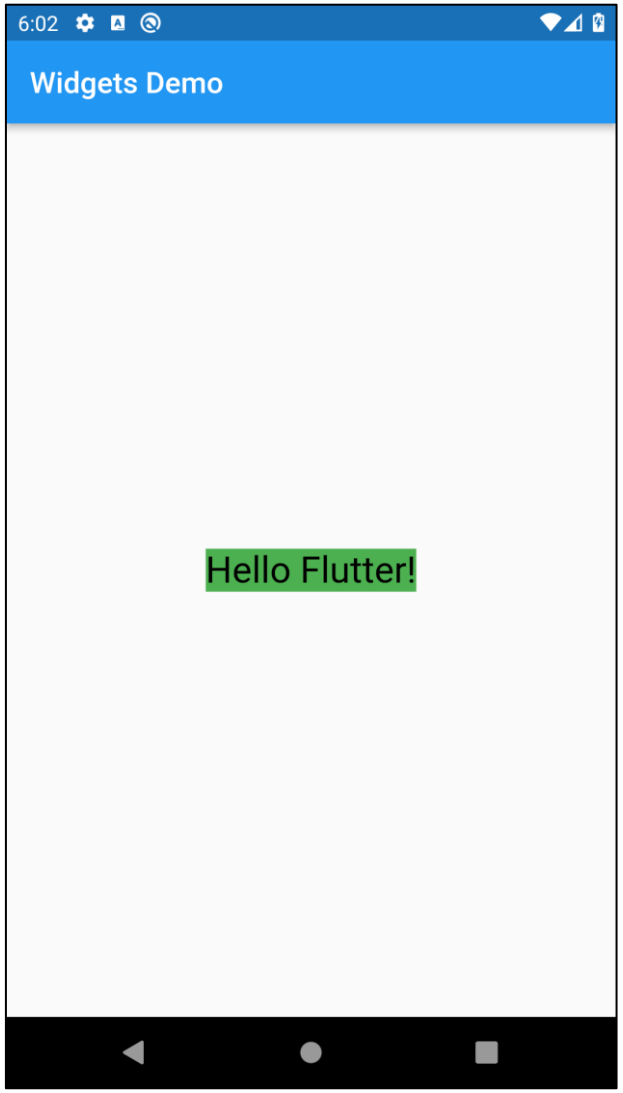


- Why use a Padding widget rather than a Container with a Container.padding property?
- There isn't really any difference between the two. If you supply a Container.padding argument, Container simply builds a Padding widget for you.
- Container doesn't implement its properties directly. Instead, Container combines a number of simpler widgets together into a convenient package.

Example 19: Padding Widget Demo

```
Container(
    color: Colors.green,
    child: const Text(
      'Hello Flutter!',
      style: TextStyle(
        fontSize: 25.0,
        color: Colors.black,
      ),
    ),
)
```

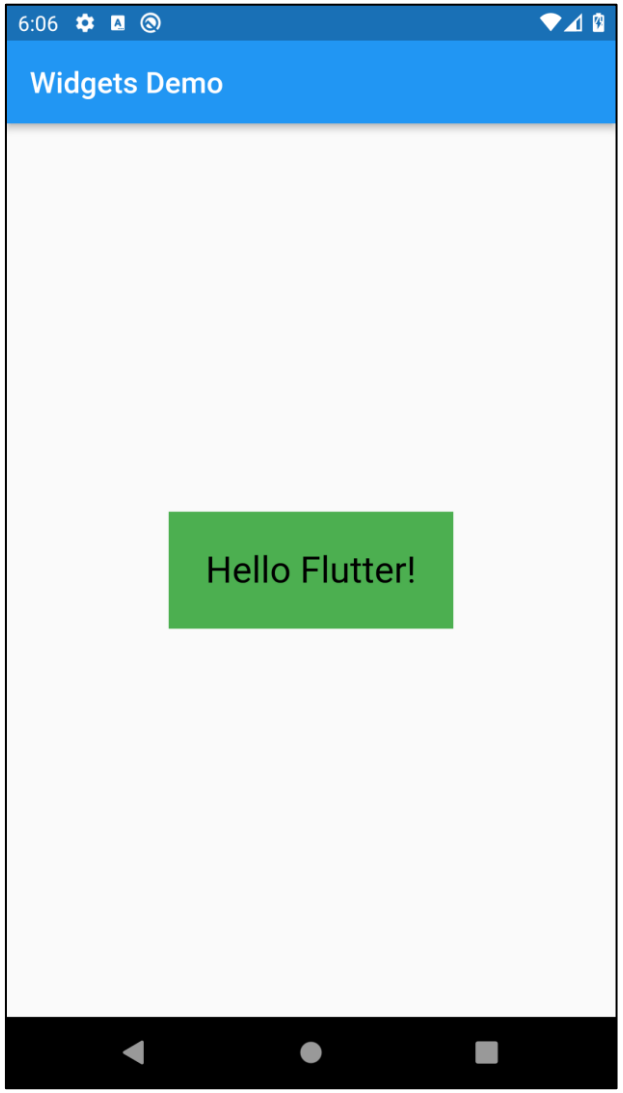
Without padding assigned



Example 20: Padding Widget Demo (EdgeInsets.all)

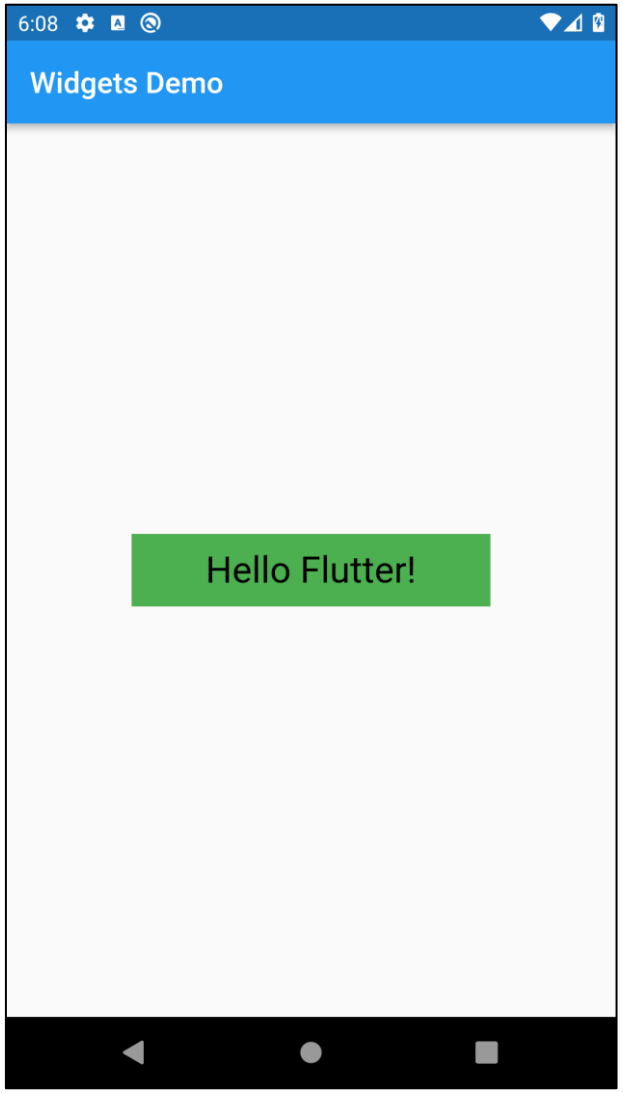
```
Container(
  color: Colors.green,
  child: const Padding(
    padding: EdgeInsets.all(25.0),
    child: Text(
      'Hello Flutter!',
      style: TextStyle(
        fontSize: 25.0,
        color: Colors.black,
      ),
    ),
  ),
)
```

With padding assigned



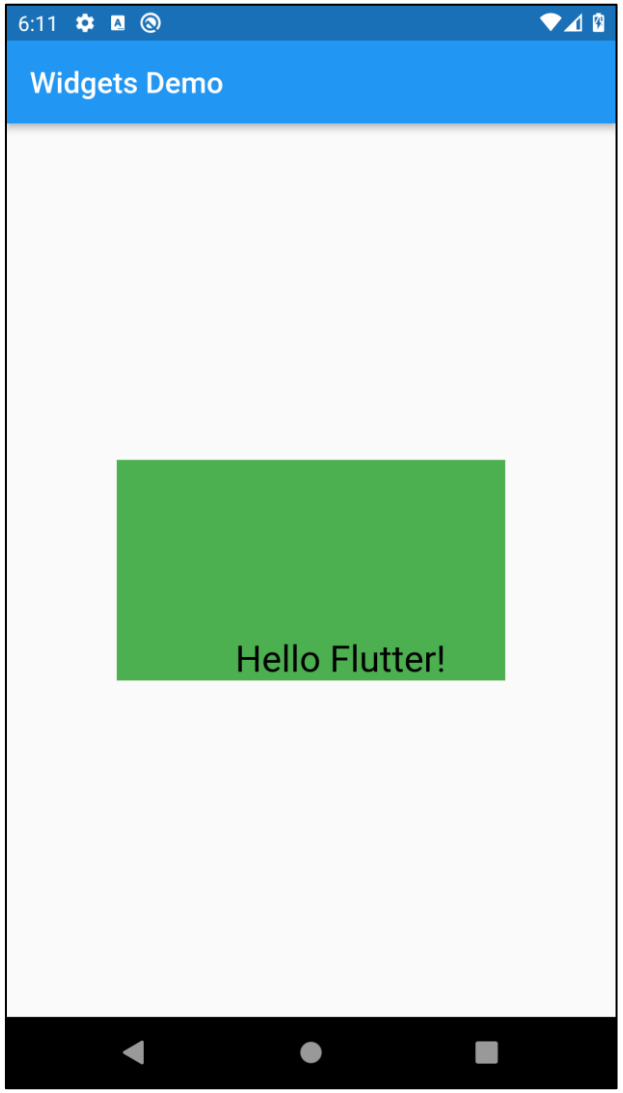
Example 21: Padding Widget Demo (EdgeInsets.symmetric)

```
Container(
  color: Colors.green,
  child: const Padding(
    padding: EdgeInsets.symmetric(
      horizontal: 50.0,
      vertical: 10.0),
    child: Text(
      'Hello Flutter!',
      style: TextStyle(
        fontSize: 25.0,
        color: Colors.black,
      ),
    ),),)
```



Example 22: Padding Widget Demo (EdgeInsets.only)

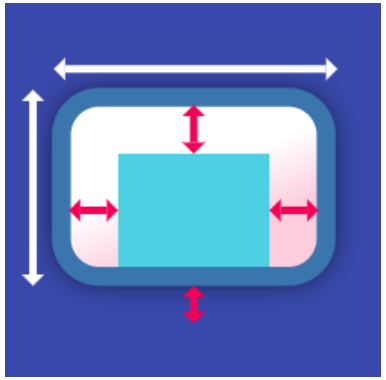
```
Container(
  color: Colors.green,
  child: const Padding(
    padding: EdgeInsets.only(left: 80.0, top: 120.0, right: 40.0),
    child: Text(
      'Hello Flutter!',
      style: TextStyle(
        fontSize: 25.0,
        color: Colors.black,
      ),
    ),
  ),
)
```



Container Widget

Container Widget

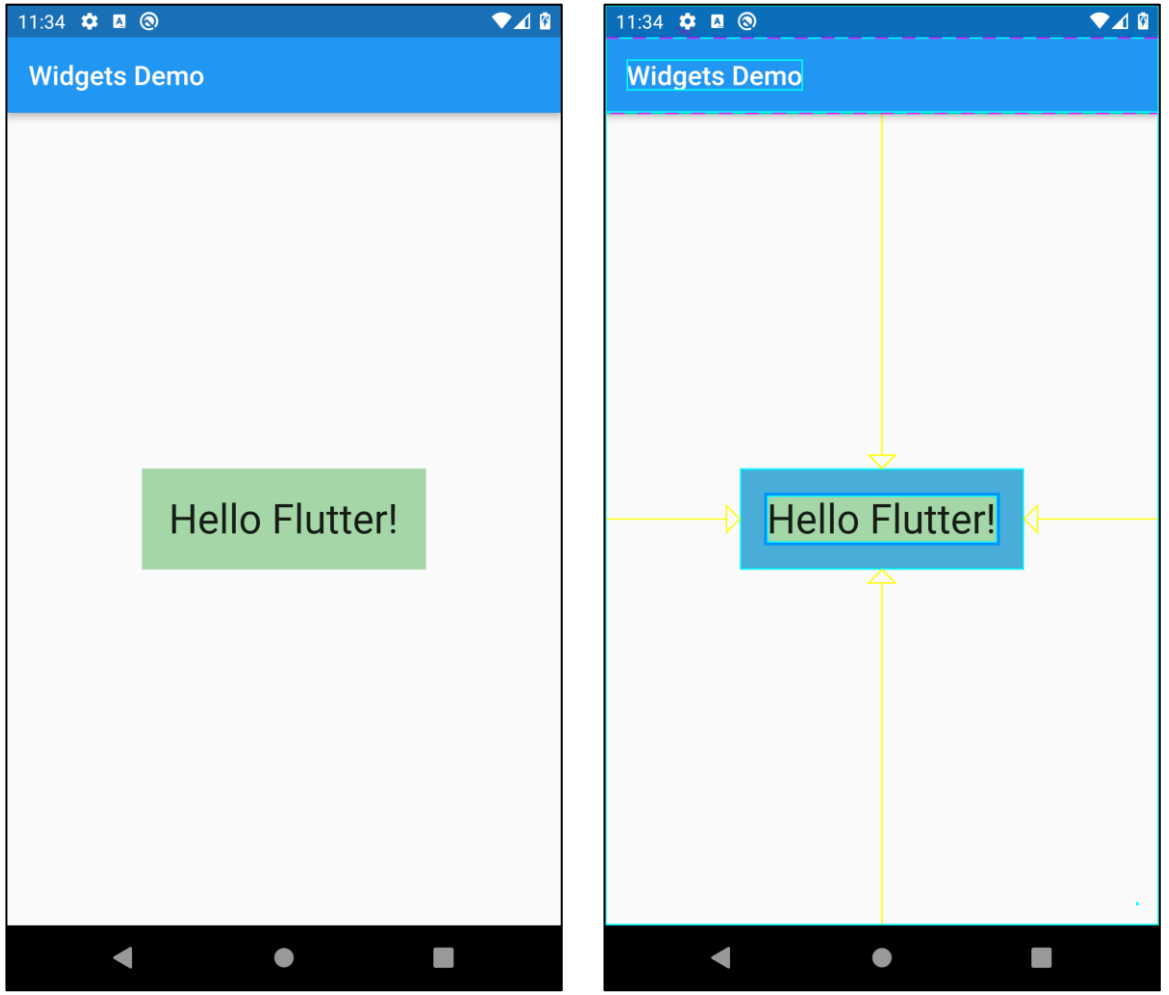
A convenience widget that combines common painting, positioning, and sizing widgets.



- A container first surrounds the child with padding, then applies additional constraints to the padded extent.
- The child is then surrounded by margin space.
- During painting, the container first applies the transform, then paints the decoration to fill the padded extent.
- It then paints the child, and finally paints the foregroundDecoration.

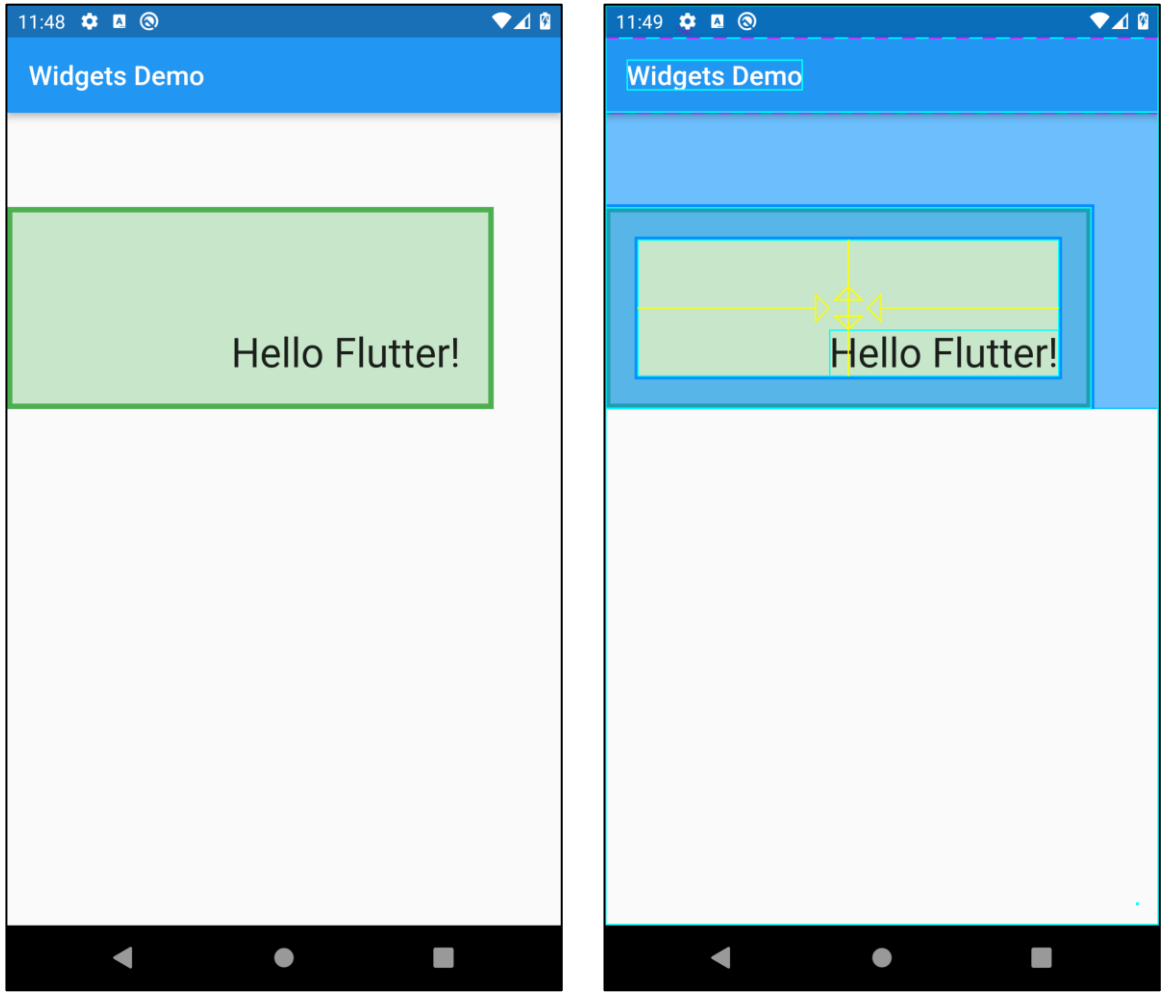
Example 23: Container Widget Demo

```
Container(
      color: Colors.green[200],
      padding: const EdgeInsets.all(20.0),
      child: const Text(
        'Hello Flutter!',
        style: TextStyle(fontSize: 30.0),
      ),
)
```



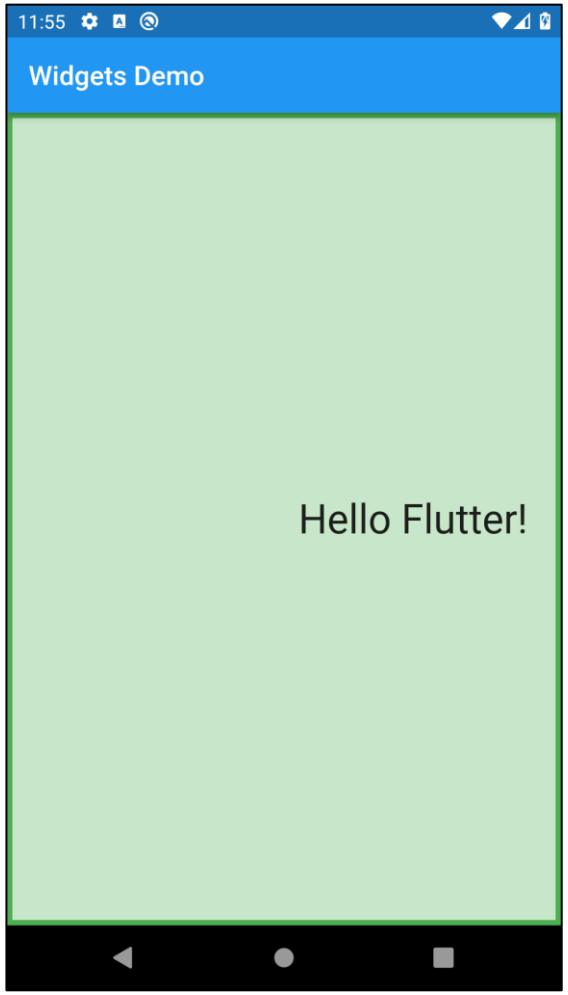
Example 24: Container Widget Demo (BoxDecoration)

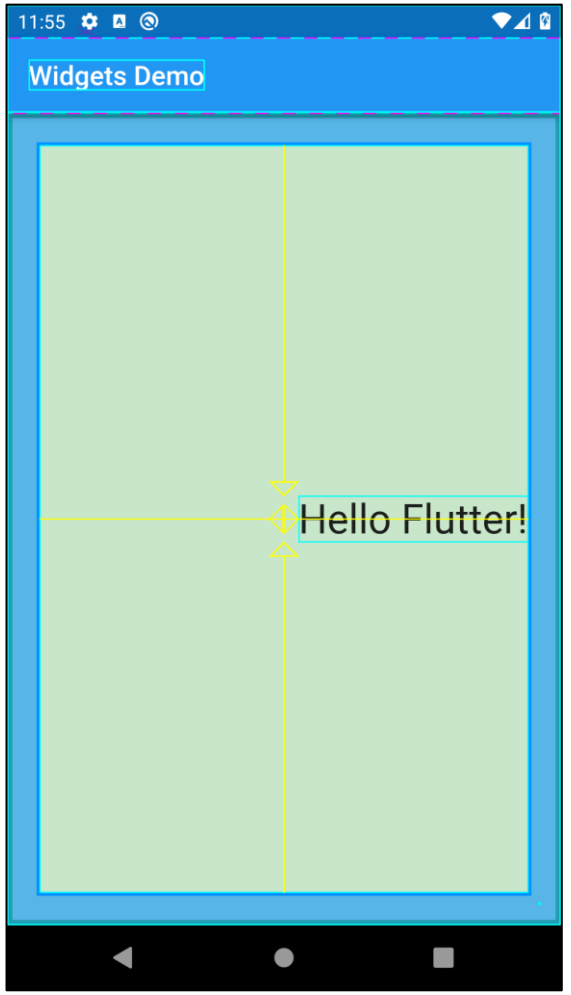
```
Container(
    decoration: BoxDecoration(
      color: Colors.green[100],
      border: Border.all(
        color: Colors.green,
        style: BorderStyle.solid,
        width: 4.0,
      ),
    ),
    alignment: Alignment.bottomRight,
    margin: const EdgeInsets.only(
        right: 50.0, top: 70.0),
    padding: const EdgeInsets.all(20.0),
    width: 500.0,
    height: 150.0,
    child: const Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
)
```



Example 25: Container Widget Demo (BoxConstraints)

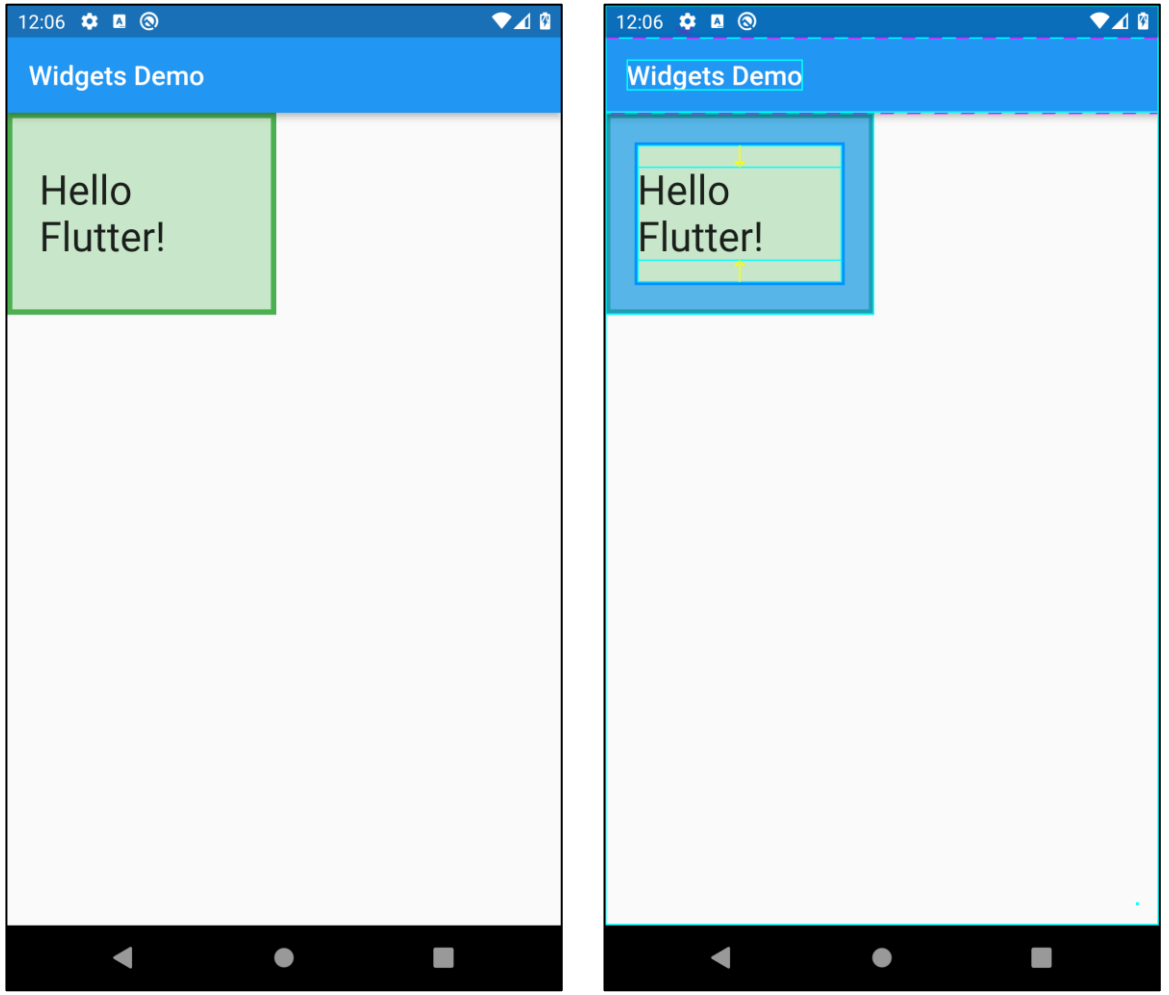
```
Container(
    decoration: BoxDecoration(
      color: Colors.green[100],
      border: Border.all(
        color: Colors.green,
        style: BorderStyle.solid,
        width: 4.0,
      ),
    ),
    constraints: const BoxConstraints.expand(),
    alignment: Alignment.centerRight,
    padding: const EdgeInsets.all(20.0),
    width: 500.0,
    height: 150.0,
    child: const Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
)
```





Example 26: Container Widget Demo (BoxConstraints)

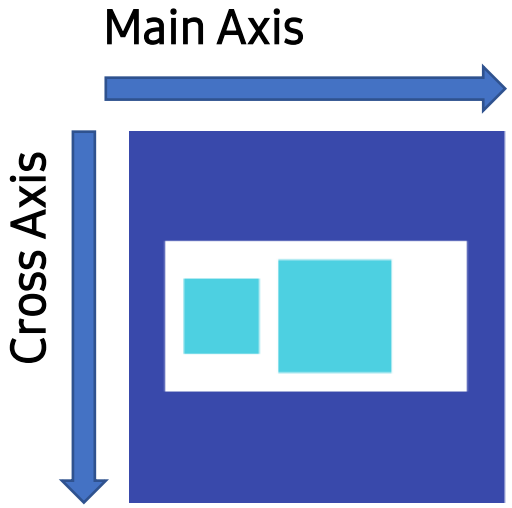
```
Container(
    decoration: BoxDecoration(
      color: Colors.green[100],
      border: Border.all(
        color: Colors.green,
        style: BorderStyle.solid,
        width: 4.0,
      ),
    ),
    constraints: const BoxConstraints(
      minWidth: 100.0,
      maxWidth: 200.0,
    ),
    alignment: Alignment.centerRight,
    padding: const EdgeInsets.all(20.0),
    width: 500.0,
    height: 150.0,
    child: const Text(
      'Hello Flutter!',
      style: TextStyle(fontSize: 30.0),
    ),
)
```



Row Widget

Row Widget

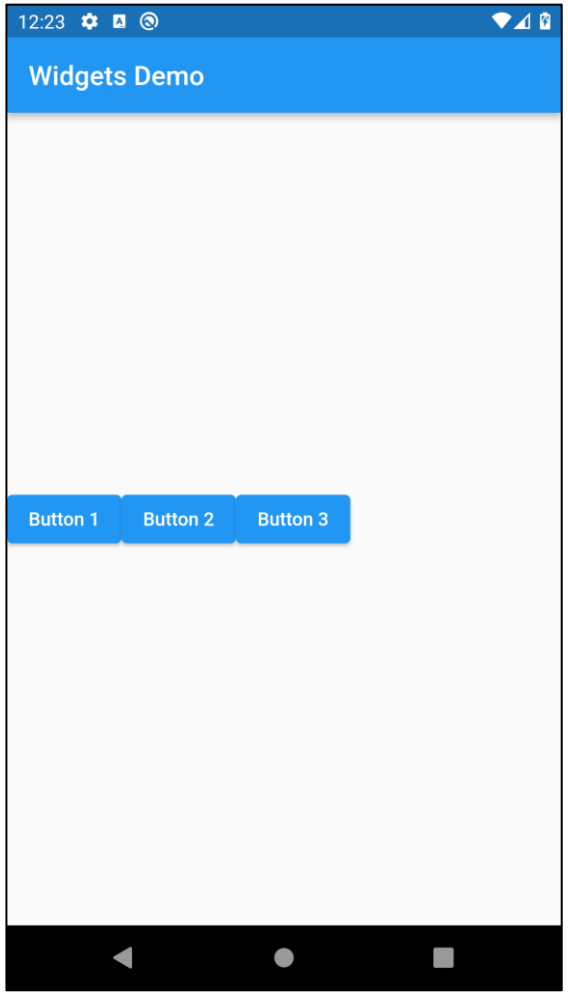
A widget that displays its children in a horizontal array

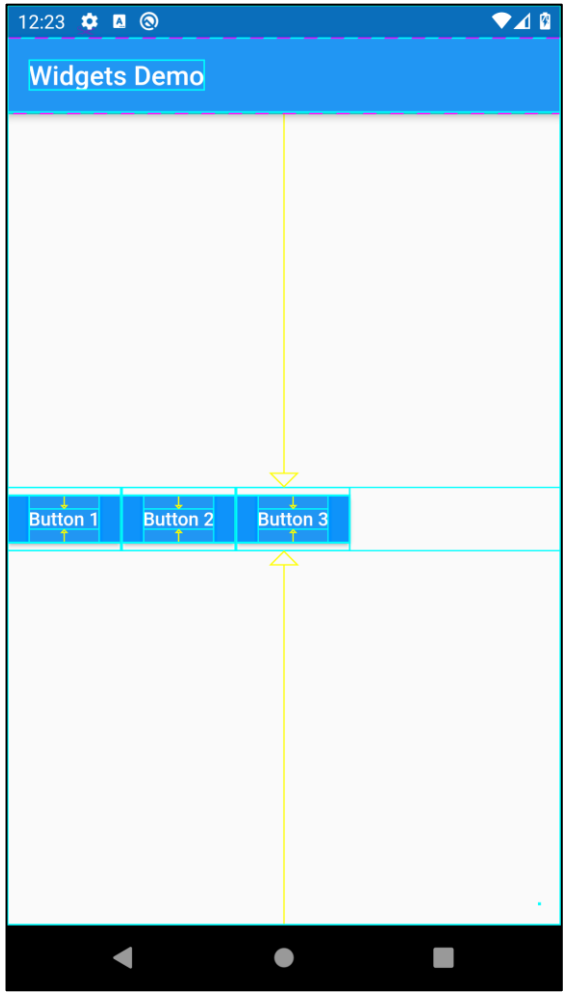


- To cause a child to expand to fill the available horizontal space, wrap the child in an Expanded widget.

Example 27: Row Widget Demo

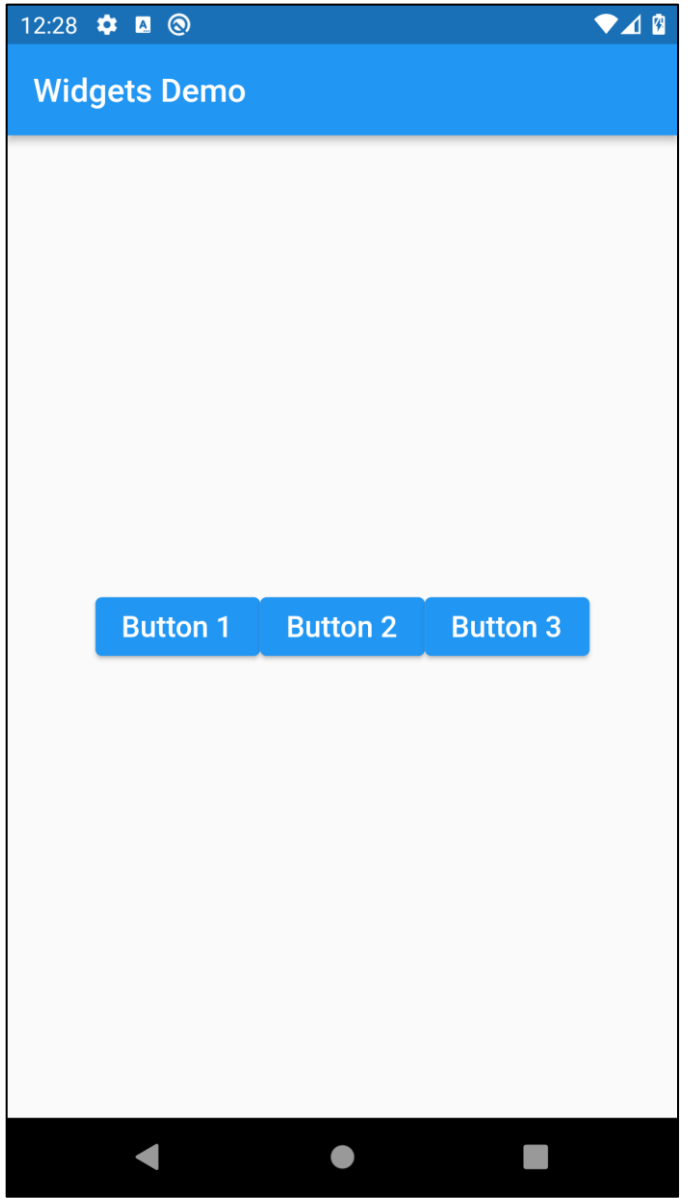
```
Row(
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text('Button 1'),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text('Button 2'),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text('Button 3'),
        ),
      ],
)
```





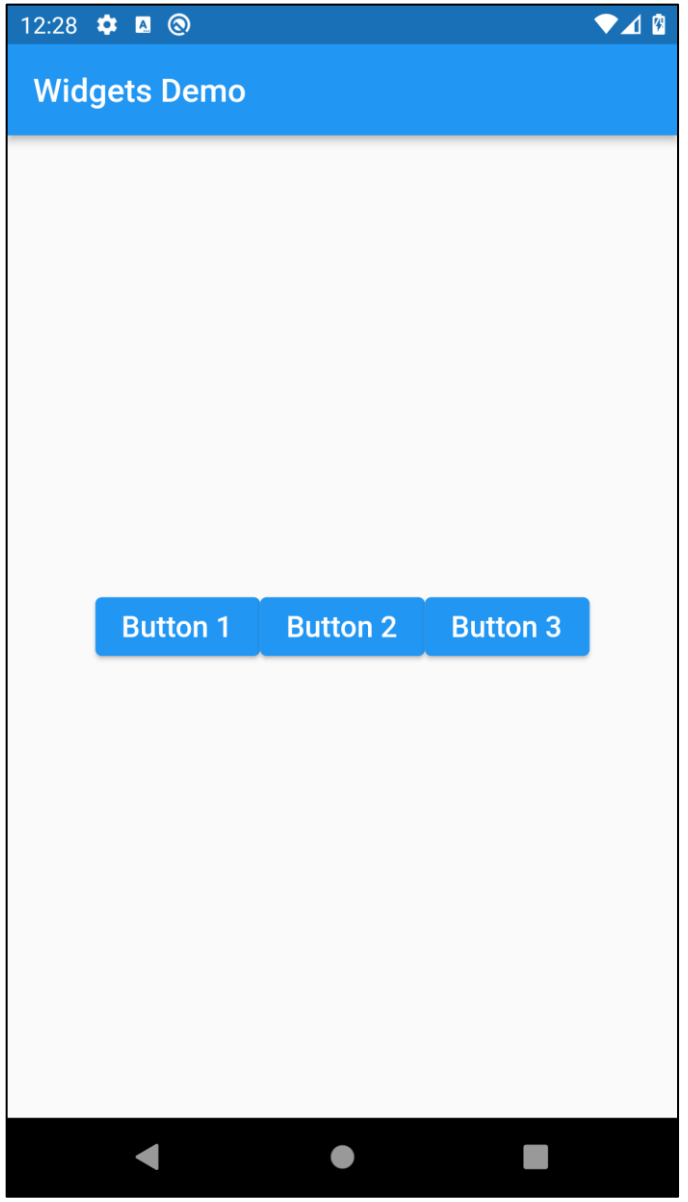
Example 28: Row Widget Demo (MainAxisAlignment)

```
Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 2',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 3',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
      ],
)
```



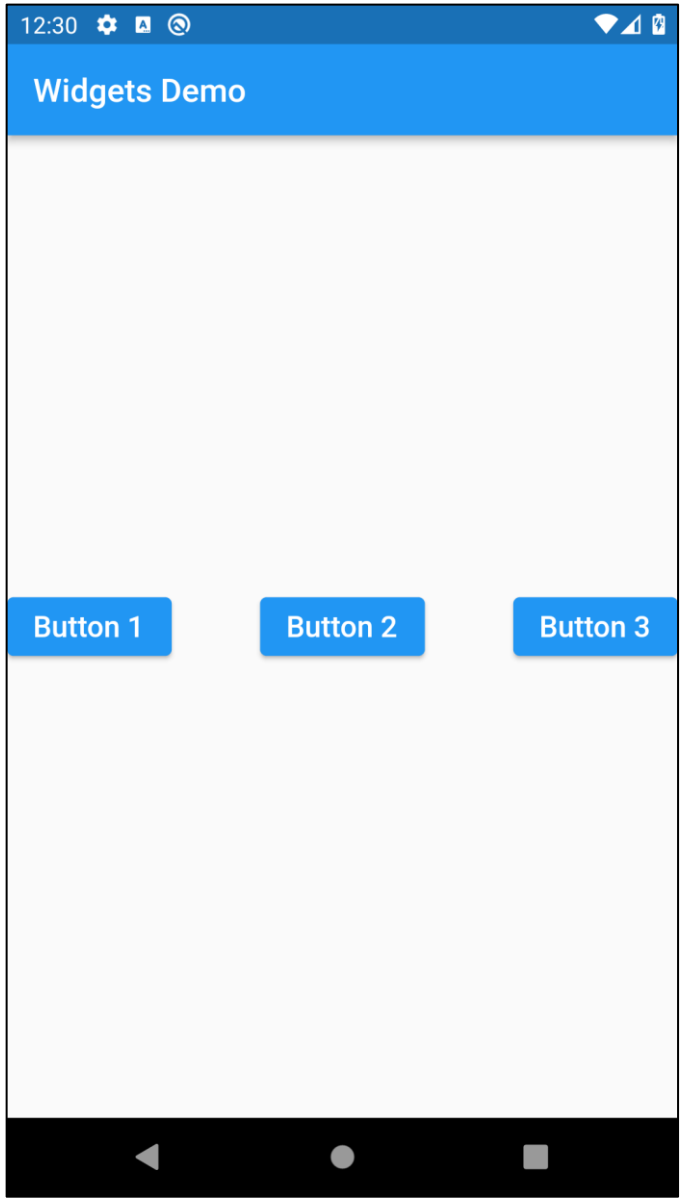
Example 29: Row Widget Demo (center)

```
Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 2',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 3',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
      ],
)
```



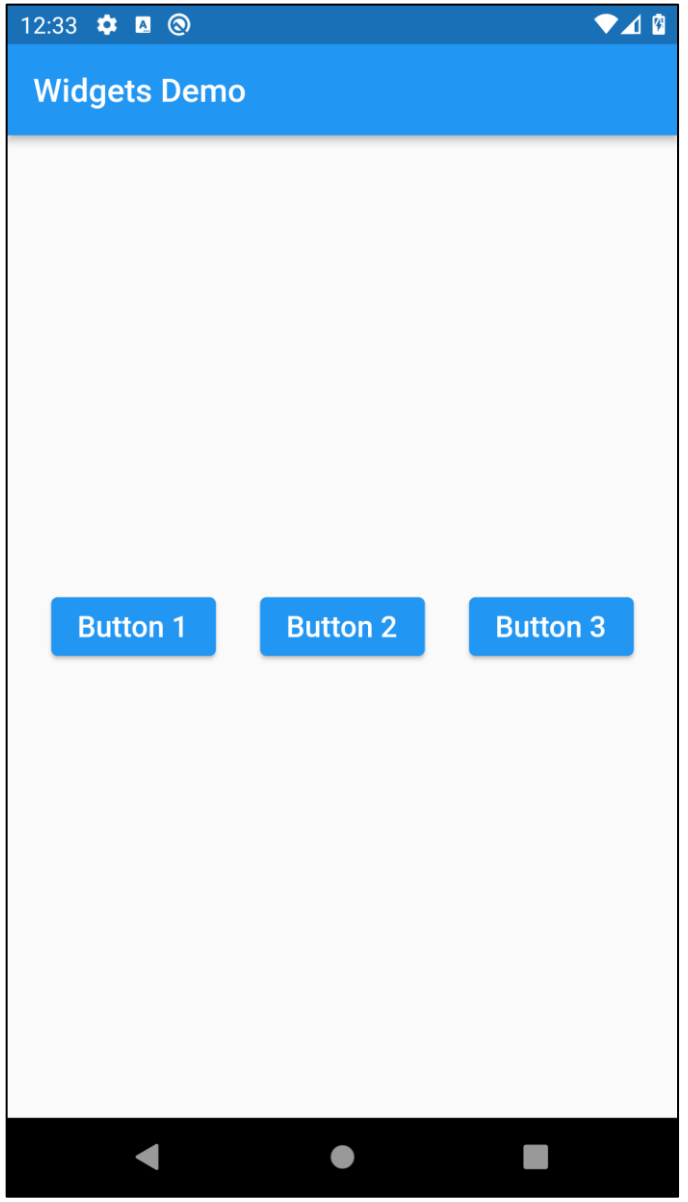
Example 30: Row Widget Demo (spaceBetween)

```
Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 2',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 3',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
      ],
)
```



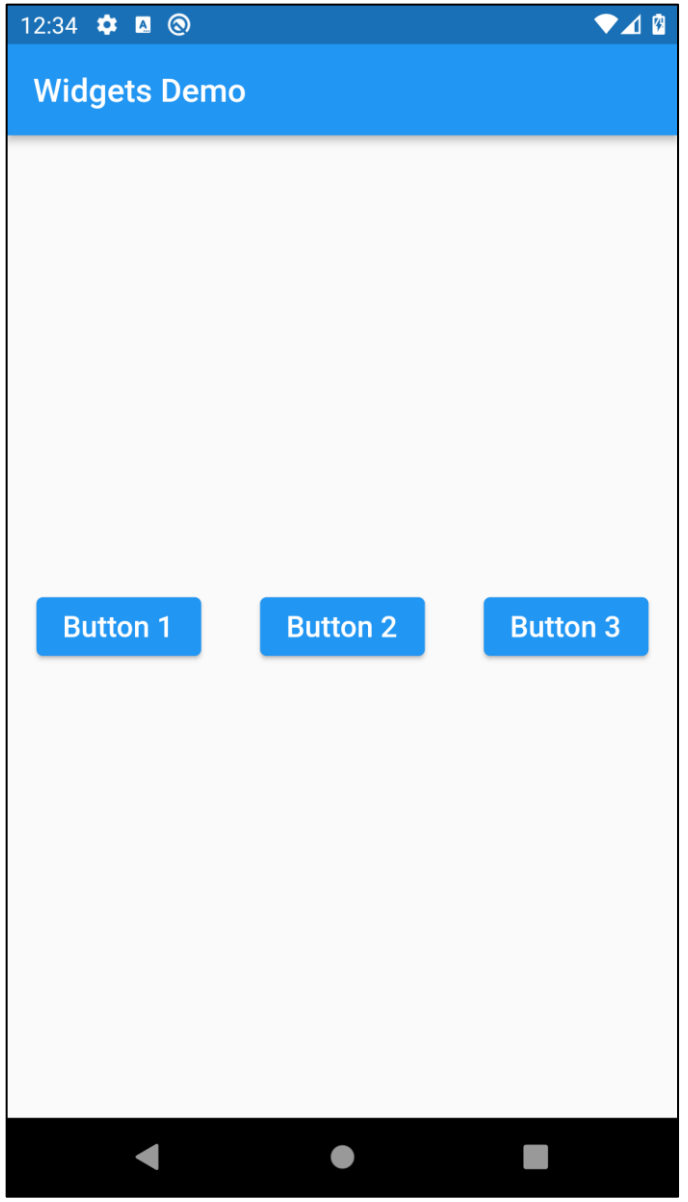
Example 31: Row Widget Demo (spaceEvenly)

```
Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 2',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 3',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
      ],
)
```



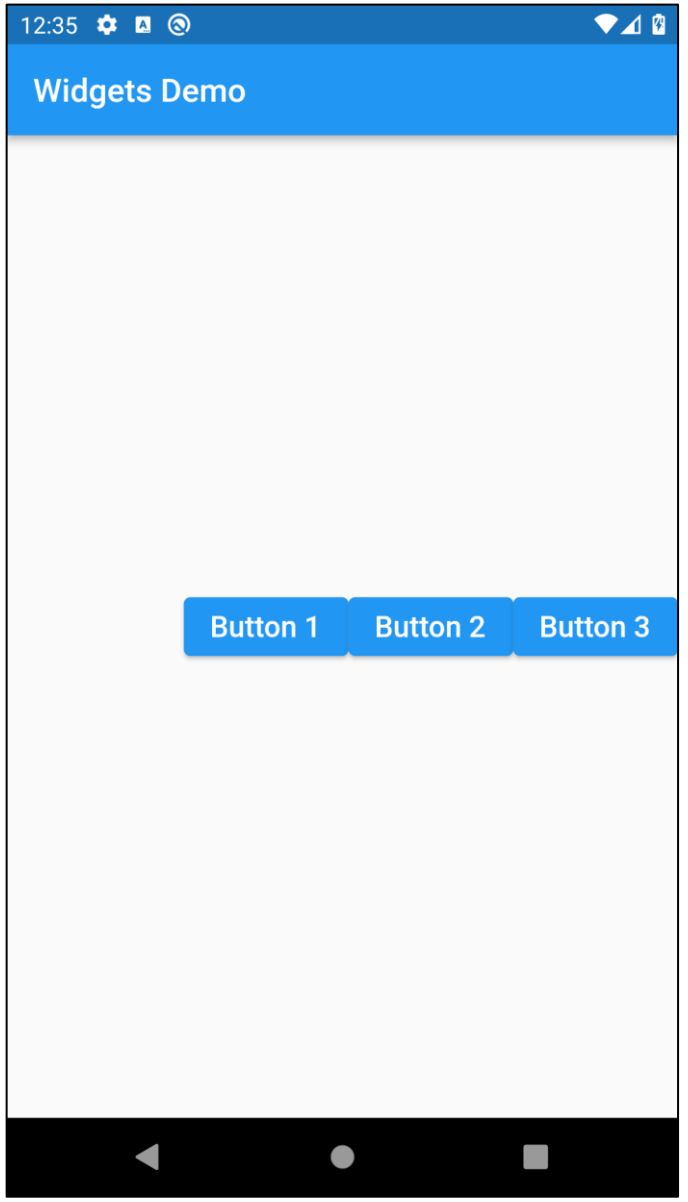
Example 32: Row Widget Demo (SpaceArround)

```
Row(
      mainAxisAlignment: MainAxisAlignment.spaceArround,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 2',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 3',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
      ],
)
```



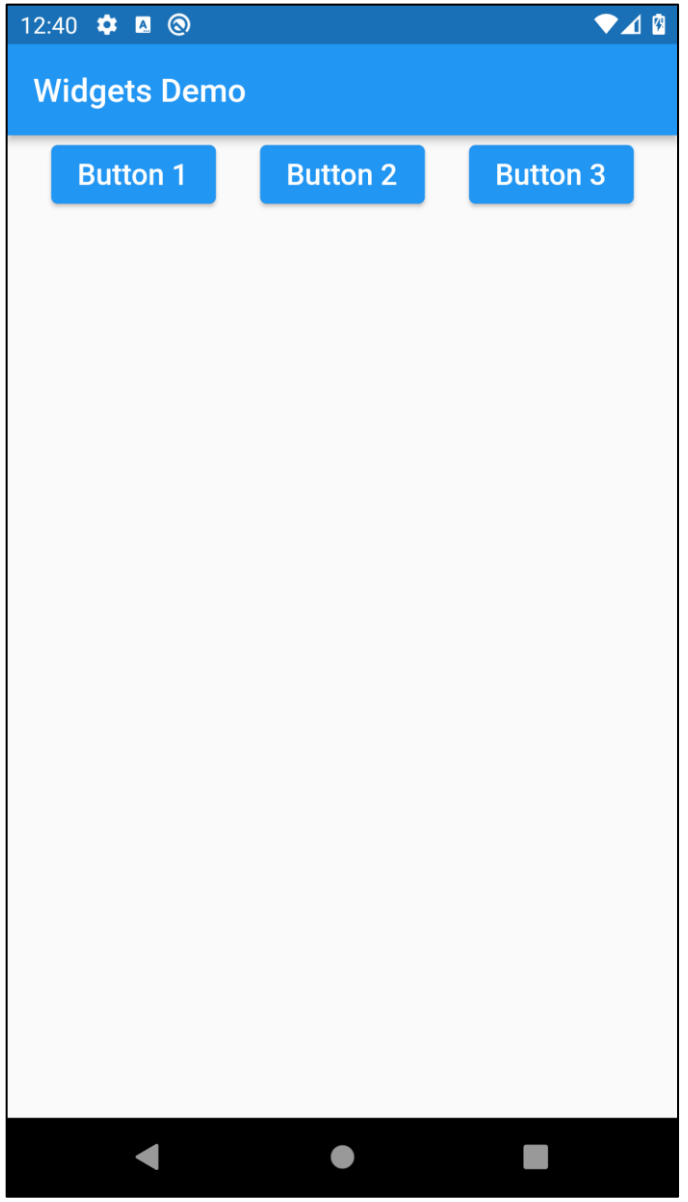
Example 33: Row Widget Demo (end)

```
Row(
      mainAxisAlignment: MainAxisAlignment.end,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 2',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 3',
            style: TextStyle(fontSize: 18.0),
          ),
        ),
      ],
)
```



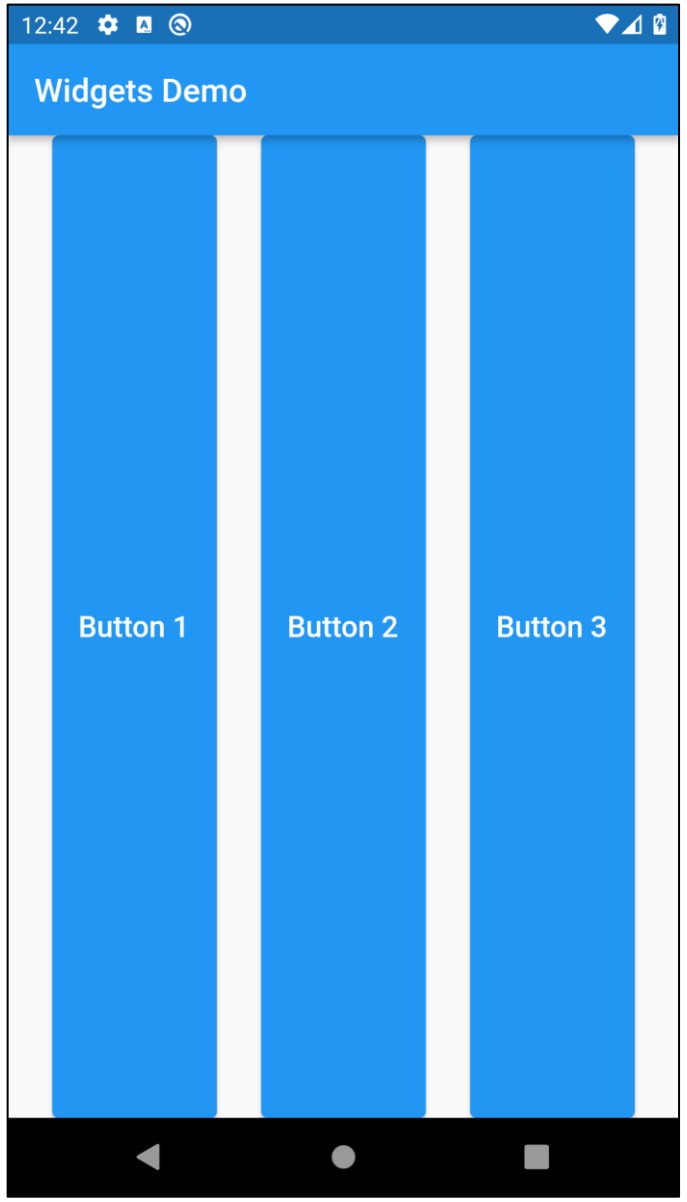
Example 34: Row Widget Demo (spaceEvenly)

```
Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      ElevatedButton(
        onPressed: () {},
        child: const Text(
          'Button 1',
          style: TextStyle(fontSize: 18.0),
        ),
      ),

.
.
.
.
```



Example 35: Row Widget Demo (CrossAxisAlignment)

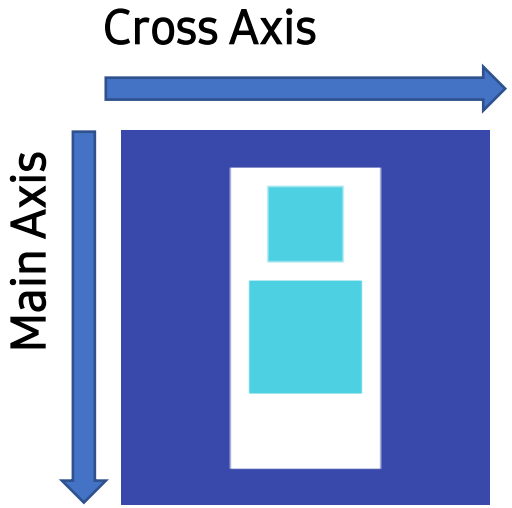
```
Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: <Widget>[
      ElevatedButton(
        onPressed: () {},
        child: const Text(
          'Button 1',
          style: TextStyle(fontSize: 18.0),
        ),
      ),

.
.
.
.
```



Column Widget

Column Widget

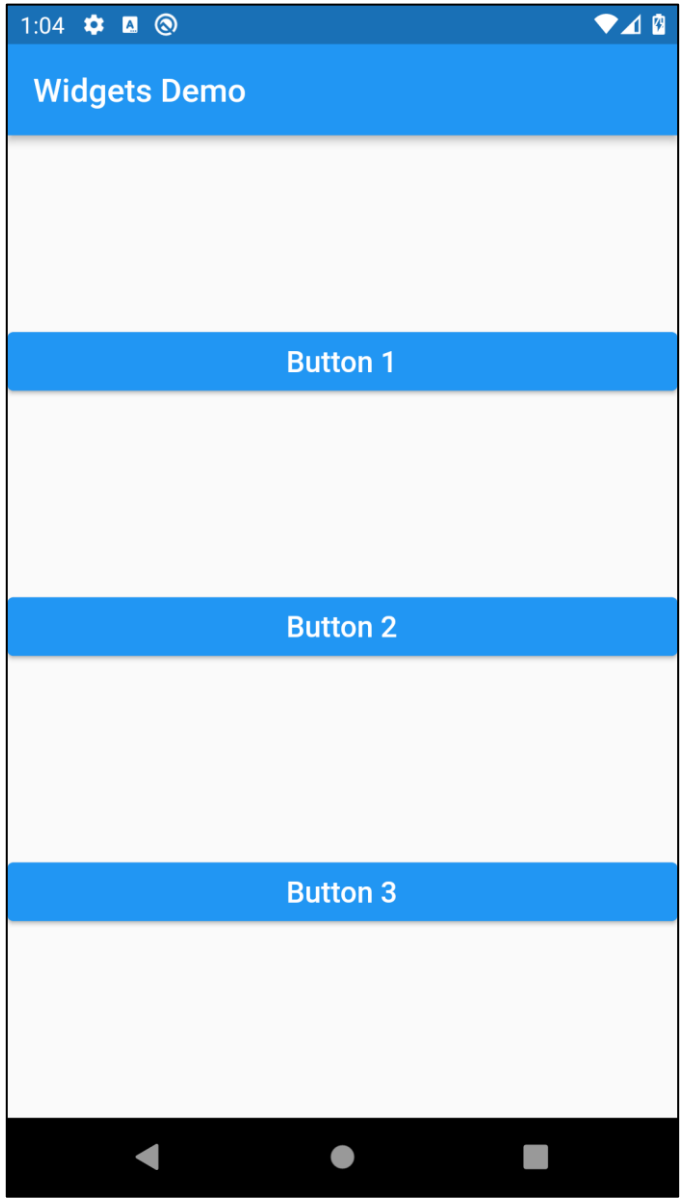
A widget that displays its children in a vertical array



- To cause a child to expand to fill the available vertical space, wrap the child in an Expanded widget.
- Similar to that of Row widget except the main and cross axes are interchanged.

Example 36: Column Widget Demo

```
Column(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    mainAxisSize: MainAxisSize.max,
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: <Widget>[
      ElevatedButton(
        onPressed: () {},
        child: const Text(
          'Button 1',
          style: TextStyle(fontSize: 18.0),
        ),
      ),
      ElevatedButton(
        onPressed: () {},
        child: const Text(
          'Button 2',
          style: TextStyle(fontSize: 18.0),
        ),
      ),
      ElevatedButton(
        onPressed: () {},
        child: const Text(
          'Button 3',
          style: TextStyle(fontSize: 18.0),
        ), ), ],
)
```



Wrap Widget

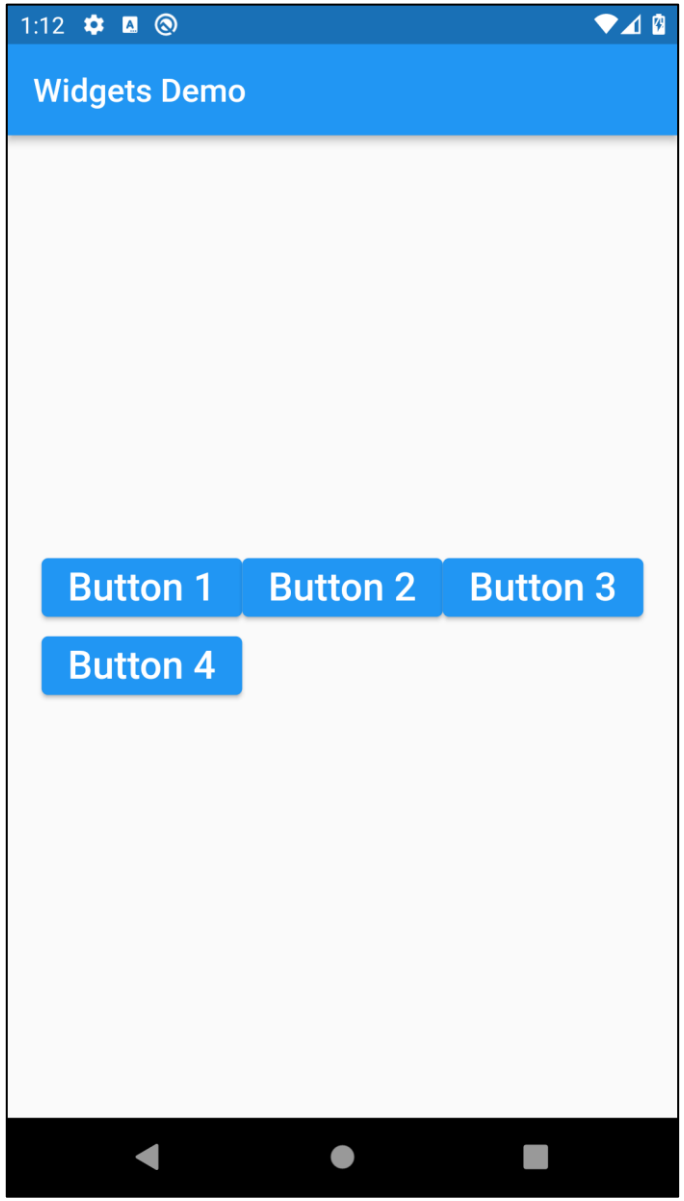
Wrap Widget

A widget that displays its children in multiple horizontal or vertical runs

- A Wrap lays out each child and attempts to place the child adjacent to the previous child in the main axis, given by direction, leaving spacing space in between.
- If there is not enough space to fit the child, Wrap creates a new run adjacent to the existing children in the cross axis.

Example 37: Wrap Widget Demo

```
Wrap(
      direction: Axis.horizontal,
      children: <Widget>[
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 1',
            style: TextStyle(fontSize: 24.0),
          ),
        ),
        .
        .
        .
        ElevatedButton(
          onPressed: () {},
          child: const Text(
            'Button 4',
            style: TextStyle(fontSize: 24.0),
          ),
        ),
      ],
  )
```



Expanded Widget

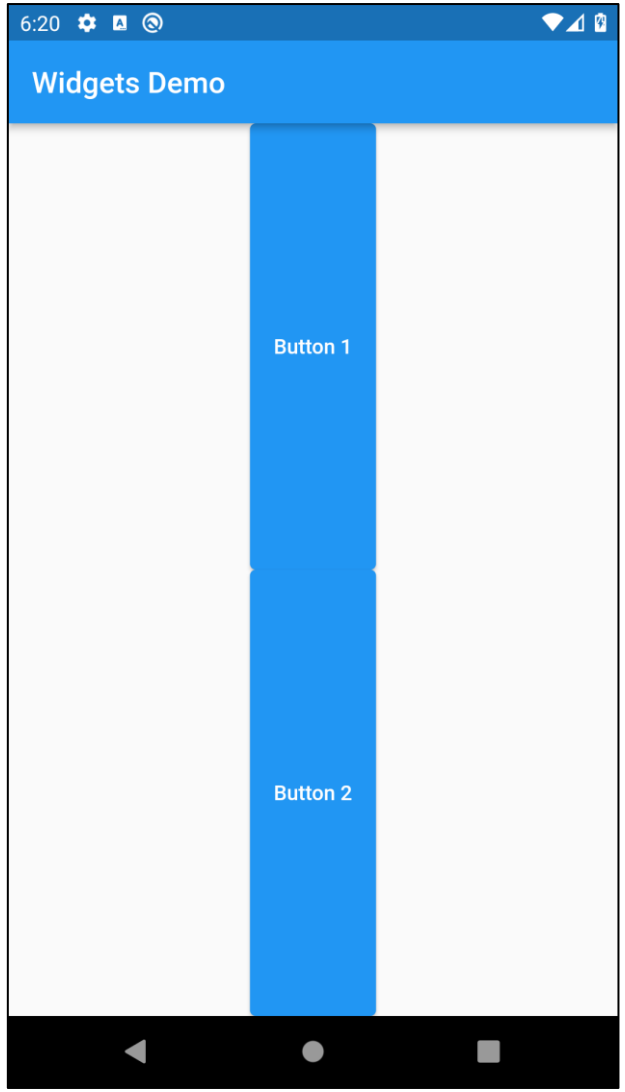
Expanded Widget

A widget that expands a child of a Row, Column, or Flex so that the child fills the available space

- Using an Expanded widget makes a child of a Row, Column, or Flex expand to fill the available space along the main axis (e.g., horizontally for a Row or vertically for a Column).
- If multiple children are expanded, the available space is divided among them according to the flex factor.

Example 38: Expanded Widget Demo

```
Column(
  children: <Widget>[
    Expanded(
      child: ElevatedButton(
        onPressed: () {},
        child: const Text('Button 1'),
      ),
    ),
    Expanded(
      child: ElevatedButton(
        onPressed: () {},
        child: const Text('Button 2'),
      ),
    ),
  ],
)
```



Stack & Positioned Widget

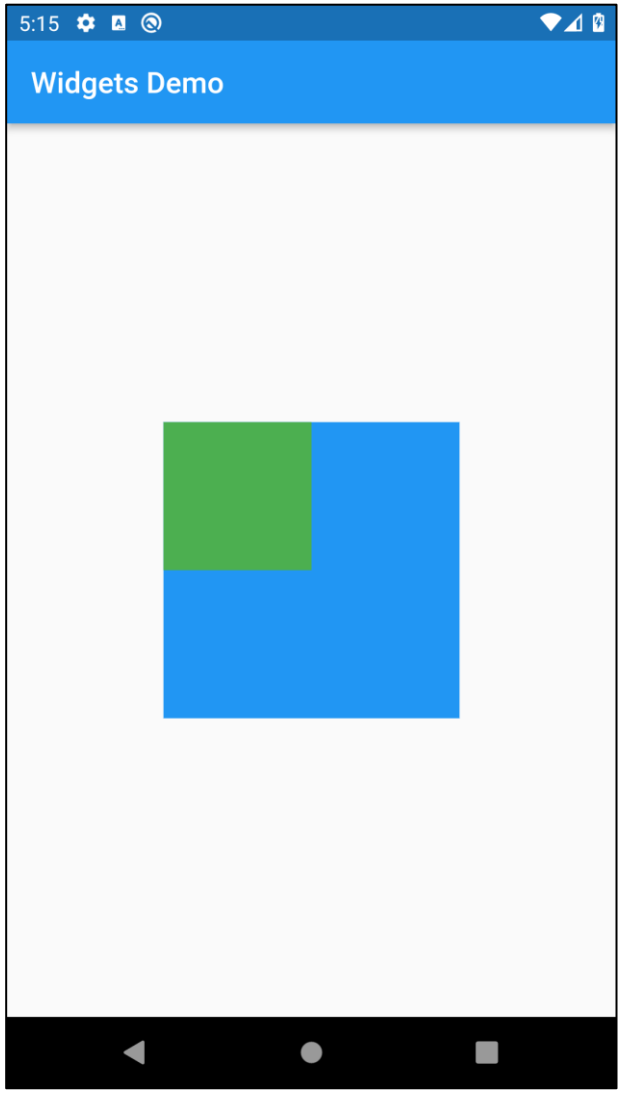
Stack & Positioned Widget

A widget that positions its children relative to the edges of its box

- The stack lays out widgets in their paint order, the widget created later stacks at the top.
- Each child of a Stack widget is either positioned or non-positioned.
- Positioned children are those wrapped in a Positioned widget that has at least one non-null property.
- The stack sizes itself to contain all the non-positioned children.
- The positioned children are then placed relative to the stack according to their top, right, bottom, and left properties.

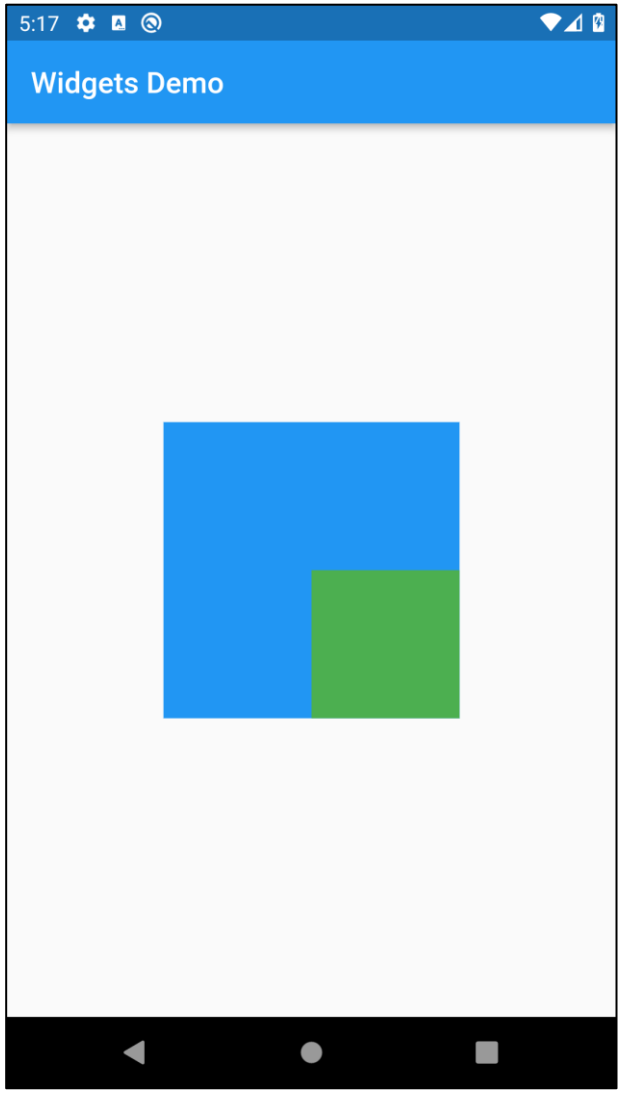
Example 39: Stack & Positioned Widget Demo

```
Stack(
    children: <Widget>[
      Container(
        width: 200,
        height: 200,
        color: Colors.blue,
      ),
      Container(
        width: 100,
        height: 100,
        color: Colors.green,
      ),
    ],
)
```



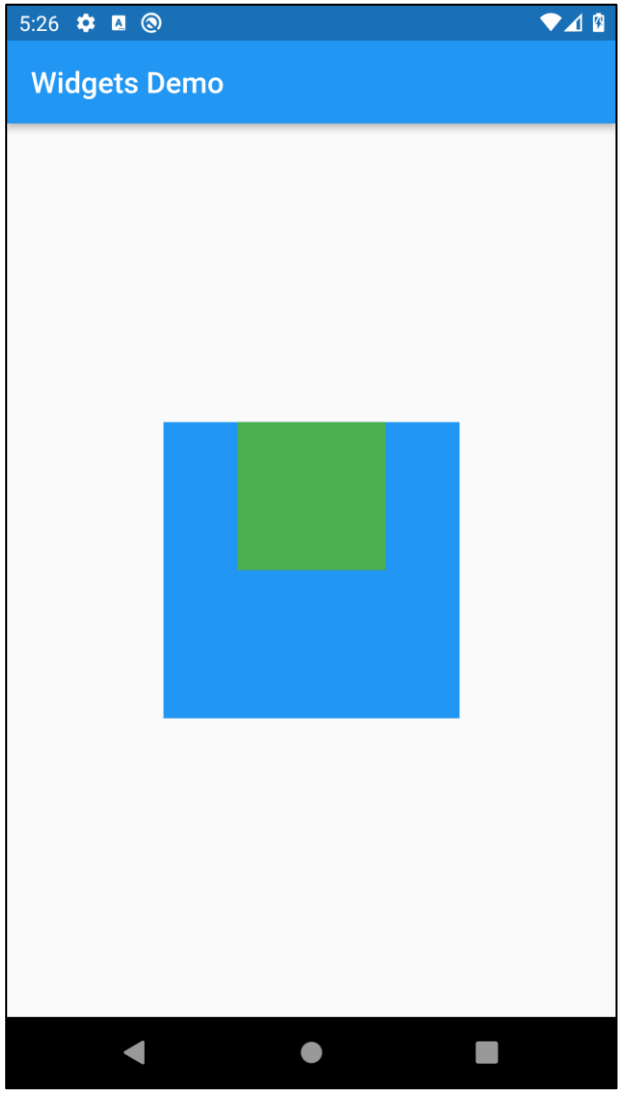
Example 40: Stack & Positioned Widget Demo (right, bottom)

```
Stack(
    children: <Widget>[
      Container(
        width: 200,
        height: 200,
        color: Colors.blue,
      ),
      Positioned(
        right: 0,
        bottom: 0,
        child: Container(
          width: 100,
          height: 100,
          color: Colors.green,
        ),
      ),
    ],
)
```



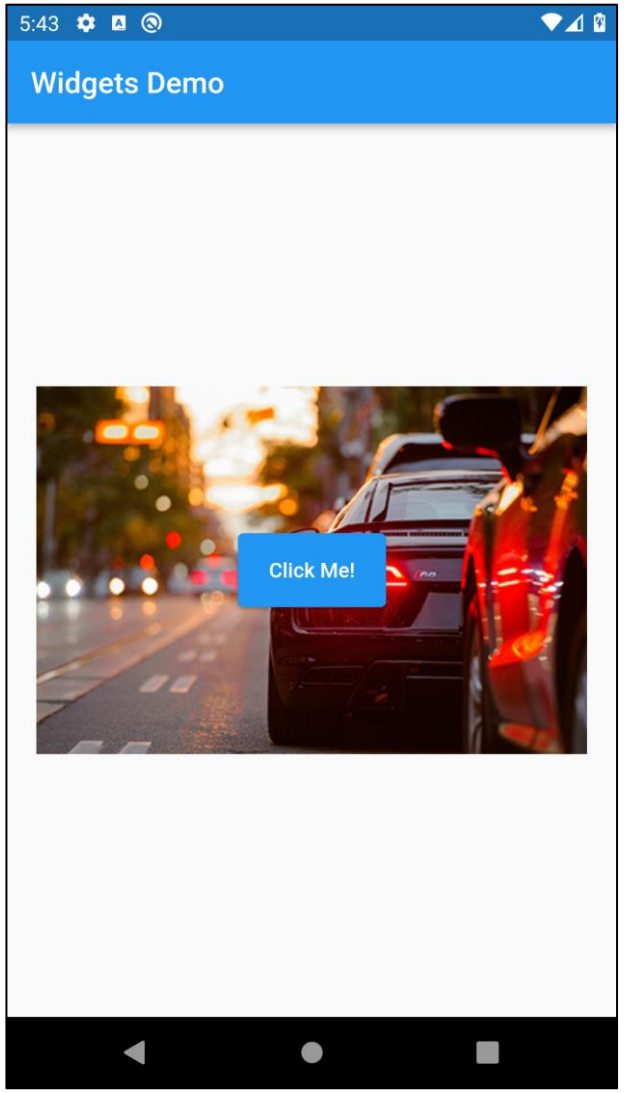
Example 41: Stack & Positioned Widget Demo (Alignment)

```
Stack(
  alignment: Alignment.topCenter,
  children: <Widget>[
    Container(
      width: 200,
      height: 200,
      color: Colors.blue,
    ),
    Container(
      width: 100,
      height: 100,
      color: Colors.green,
    ),
  ],
)
```



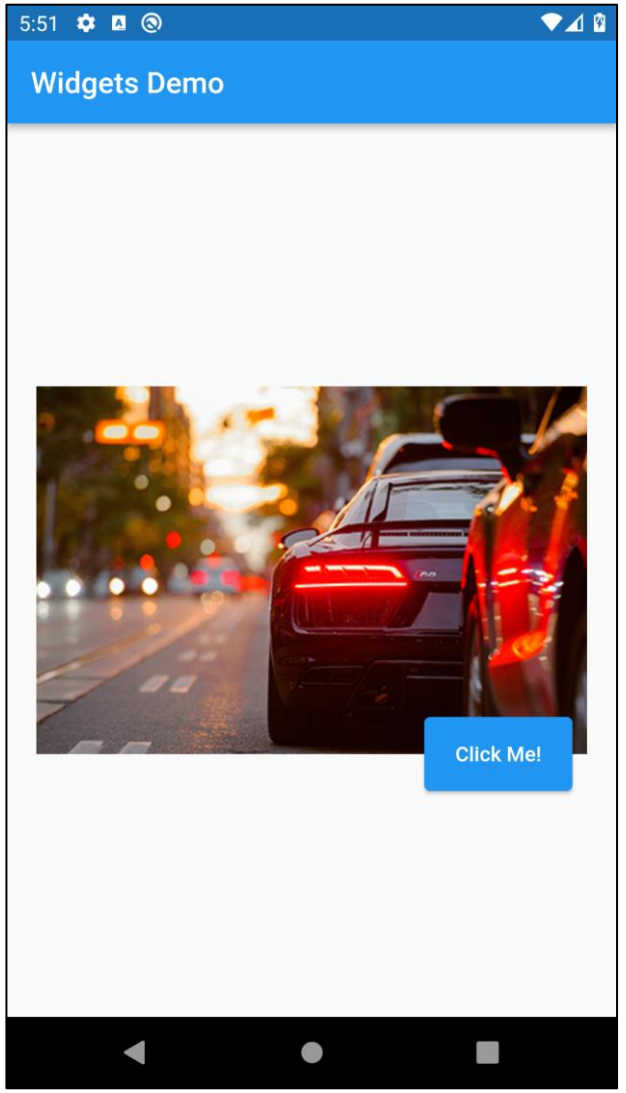
Example 42: Stack & Positioned Widget Demo (Image)

```
Stack(
  alignment: Alignment.center,
  children: <Widget>[
    Container(
      child: const Image(
        image: AssetImage('assets/images/background.jpg'),
      ),
    ),
    Container(
      width: 100,
      height: 50,
      child: ElevatedButton(
        onPressed: () {},
        child: const Text('Click Me!'),
      ),
    ),
  ],
),
```



Example 43: Stack & Positioned Widget Demo (Clip)

```
child: Stack(
  alignment: Alignment.center,
  clipBehavior: Clip.none,
  children: <Widget>[
    Container(
      child: const Image(
        image: AssetImage('assets/images/background.jpg'),
      ),
    ),
    Positioned(
      right: 10,
      bottom: -25,
      child: Container(
        width: 100,
        height: 50,
        child: ElevatedButton(
          onPressed: () {},
          child: const Text('Click Me!'),
        ),
      ),
    ),
  ],
)
```



MediaQuery Widget

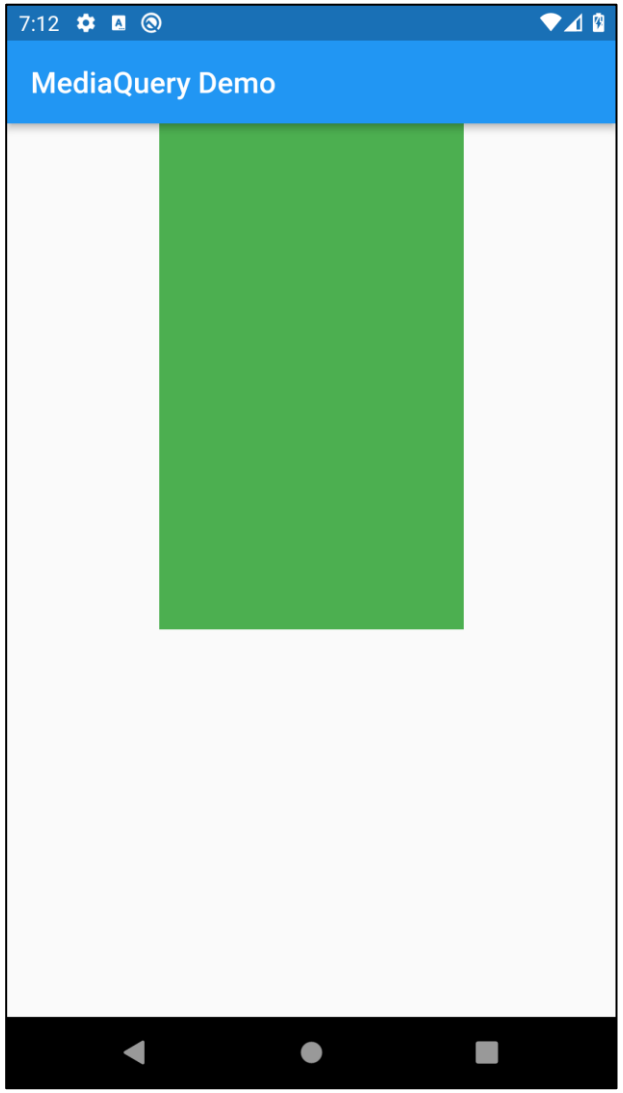
MediaQuery Widget

MediaQuery provides a higher-level view of the current app's screen size and can also give more detailed information about the device and its layout preferences

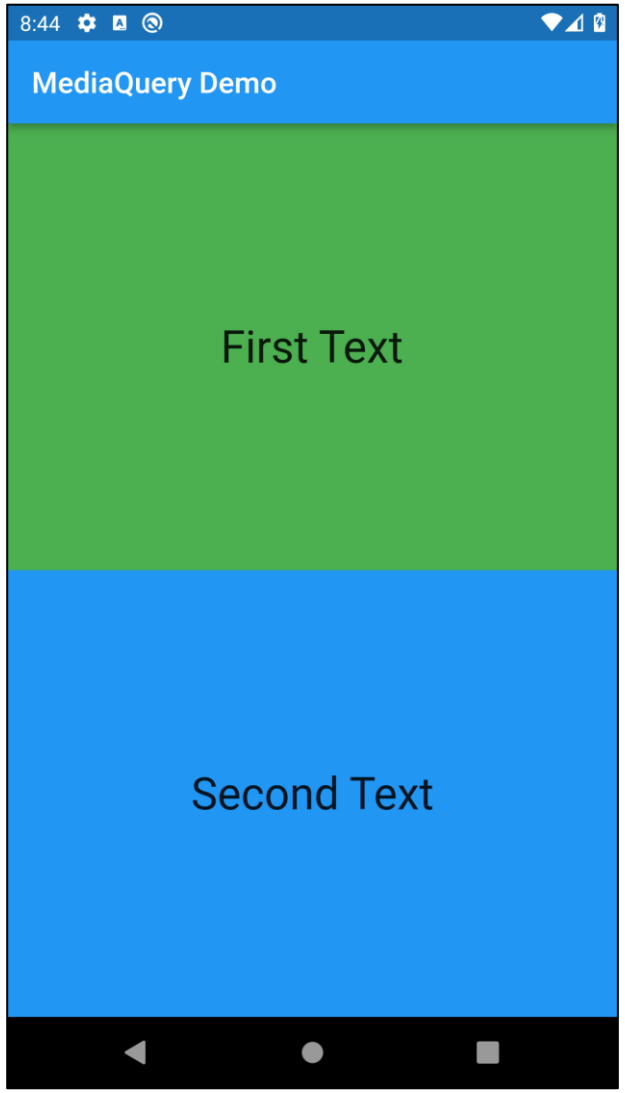
- During the process of developing an app for both phones and tablets, it is standard practice to have different UI layouts for different screen sizes for a better user experience.
- If the user has a preference set for different font sizes or wants to curtail animations.
- This is where MediaQuery comes into action, you can get information about the current device size, as well as user preferences, and design your layout accordingly.
- It can simply be accessed by calling `MediaQuery.of` in the build method.

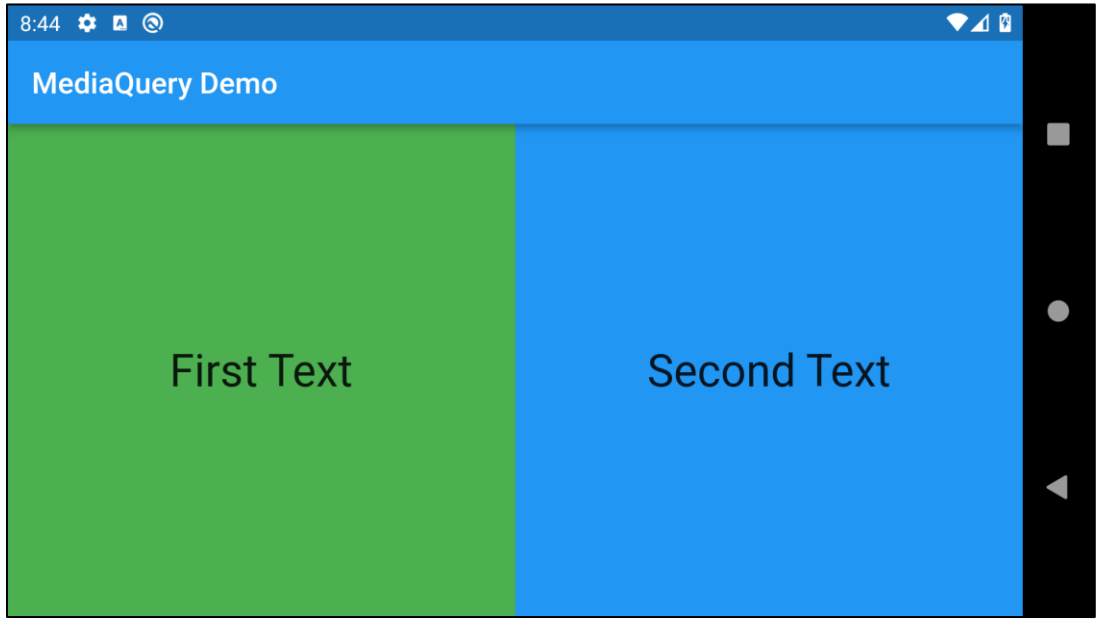
Example 44: MediaQuery Widget Demo (Size)

```
@override
Widget build(BuildContext context) {
  Size size = MediaQuery.of(context).size;

  return Scaffold(
    appBar: AppBar(
      title: const Text('MediaQuery Demo'),
    ),
    body: Center(
      child: Column(children: <Widget>[
        Container(
          width: size.width / 2.0,
          height: size.height / 2.0,
          color: Colors.green,
        ),
      ]),
    ),
  );
}
```



Example 45: MediaQuery Widget Demo (Orientation)





Example 45: MediaQuery Widget Demo (Orientation)

```
@override
  Widget build(BuildContext context) {
    Orientation orientation = MediaQuery.of(context).orientation;

    return Scaffold(
      appBar: AppBar(title: const Text('MediaQuery Demo')),
      body: Center(
        child: Builder(builder: (context) {
          if (orientation.index == Orientation.portrait.index) {
            return Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: <Widget>[
                Expanded(
                  child: Container(
                    alignment: Alignment.center,
                    color: Colors.green,
                    child: const Text(
                      'First Text',
                      style: TextStyle(fontSize: 30.0),
                    ),
                  ),
                ),
```

Example 45: MediaQuery Widget Demo (Orientation)

```
        Expanded(
          child: Container(
            alignment: Alignment.center,
            color: Colors.blue,
            child: const Text(
              'Second Text',
              style: TextStyle(fontSize: 30.0),
            ),
          ),
        ),
      ],
    );
  } else {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: <Widget>[
        Expanded(
          child: Container(
            alignment: Alignment.center,
            color: Colors.green,
            child: const Text(
              'First Text',
```

Example 45: MediaQuery Widget Demo (Orientation)

```
                    style: TextStyle(fontSize: 30.0),
                  ),
                ),
              ),
              Expanded(
                child: Container(
                  alignment: Alignment.center,
                  color: Colors.blue,
                  child: const Text(
                    'Second Text',
                    style: TextStyle(fontSize: 30.0),
                  ),
                ),
              ),
            ],
          );
        }
      }),
    ),
  );
}
```

SafeArea Widget

SafeArea Widget

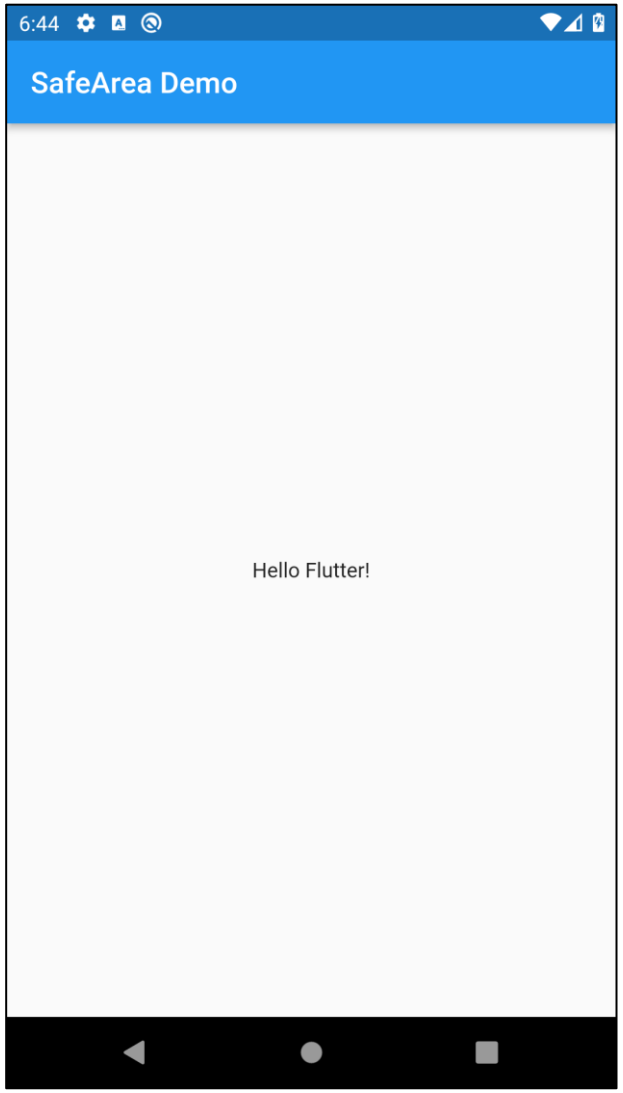
A widget that insets its child by sufficient padding to avoid intrusions by the operating system

- For example, this will indent the child by enough to avoid the status bar at the top of the screen.
- It will also indent the child by the amount necessary to avoid The Notch on the iPhone X, or other similar creative physical features of the display.
- When a minimum padding is specified, the greater of the minimum padding or the safe area padding will be applied.

Example 46: SafeArea Widget Demo

```
Scaffold(
  appBar: AppBar(
    title: const Text('SafeArea Demo'),
  ),
  body: const Center(
    child: Text('Hello Flutter!'),
  ),
)
```

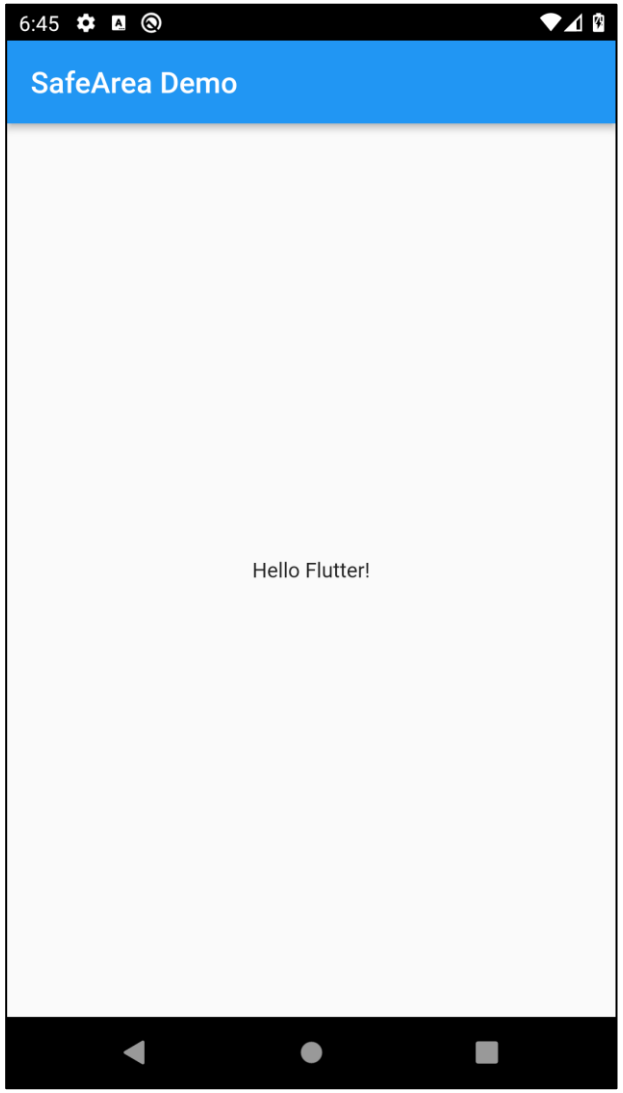
Without SafeArea assigned



Example 46: SafeArea Widget Demo

```
SafeArea(
  child: Scaffold(
    appBar: AppBar(
      title: const Text('SafeArea Demo'),
    ),
    body: const Center(
      child: Text('Hello Flutter!'),
    ),
  ),
)
```

With SafeArea assigned



Assets and Images

Assets and Images

An asset is a file that is bundled and deployed with your app, and is accessible at runtime

- Flutter apps can include both code and assets (sometimes called resources).
- Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP).

Specifying Assets

- Flutter uses the pubspec.yaml file, located at the root of your project, to identify assets required by an app.

To add individual asset files:

```
flutter:
  assets:
    - assets/my_icon.png
    - assets/background.png
```

To include all assets under a directory

```
flutter:
  assets:
    - directory/
    - directory/subdirectory/
```

Note: To add files located in subdirectories, create an entry per directory.

Loading Images

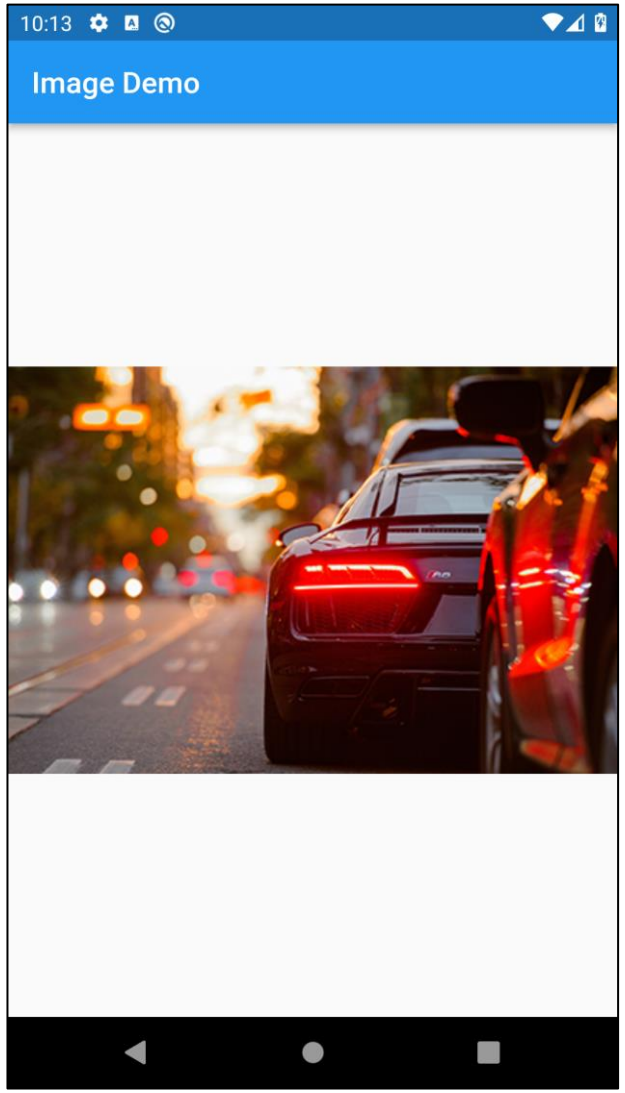
- To load an image, use the AssetImage class in a widget's build() method.
- For example, your app can load the background.png image from the asset:

```
return const Image(image: AssetImage('graphics/background.png'));
```

Example 47: AssetImage Demo

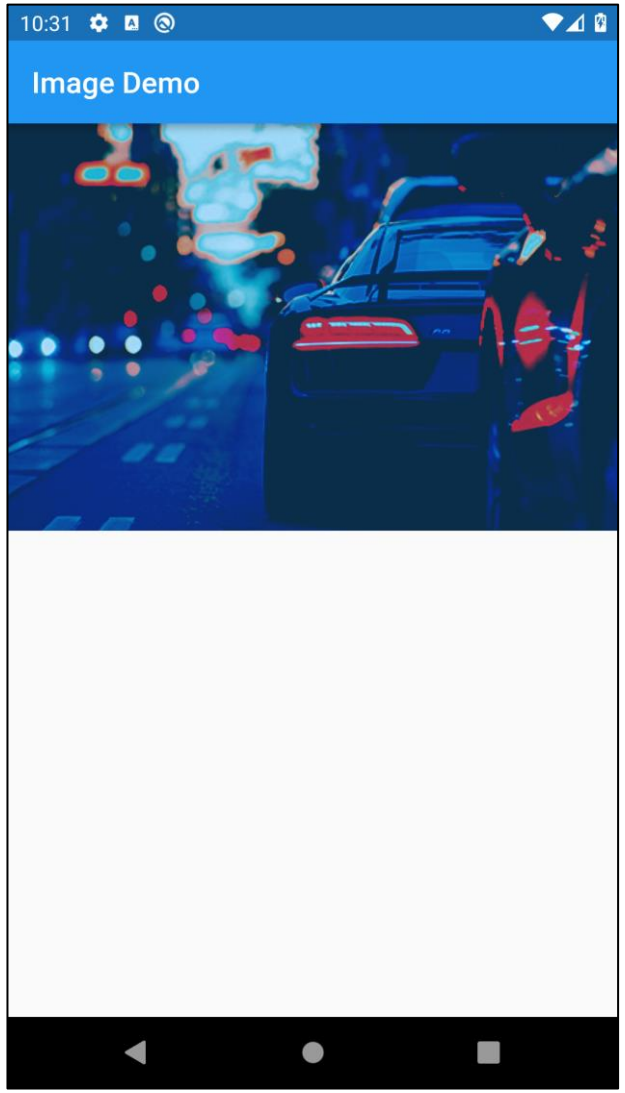
- Copy the Image background.jpg into assets/images folder.
- Ass entry in pubspec.yaml file.

```
Image(
  image: AssetImage('assets/images/background.jpg'),
)
```



Example 48: Image.asset Demo

```
Scaffold(
  appBar: AppBar(title: const Text('Image Demo')),
  body: Center(
    child: Container(
      alignment: Alignment.topCenter,
      child: Image.asset('assets/images/background.jpg',
          color: Colors.blue,
          colorBlendMode: BlendMode.colorBurn,
          scale: 1.0,
          opacity: const AlwaysStoppedAnimation<double>(0.7)),
    ),
  ),
)
```



Example 49: Image.network Demo

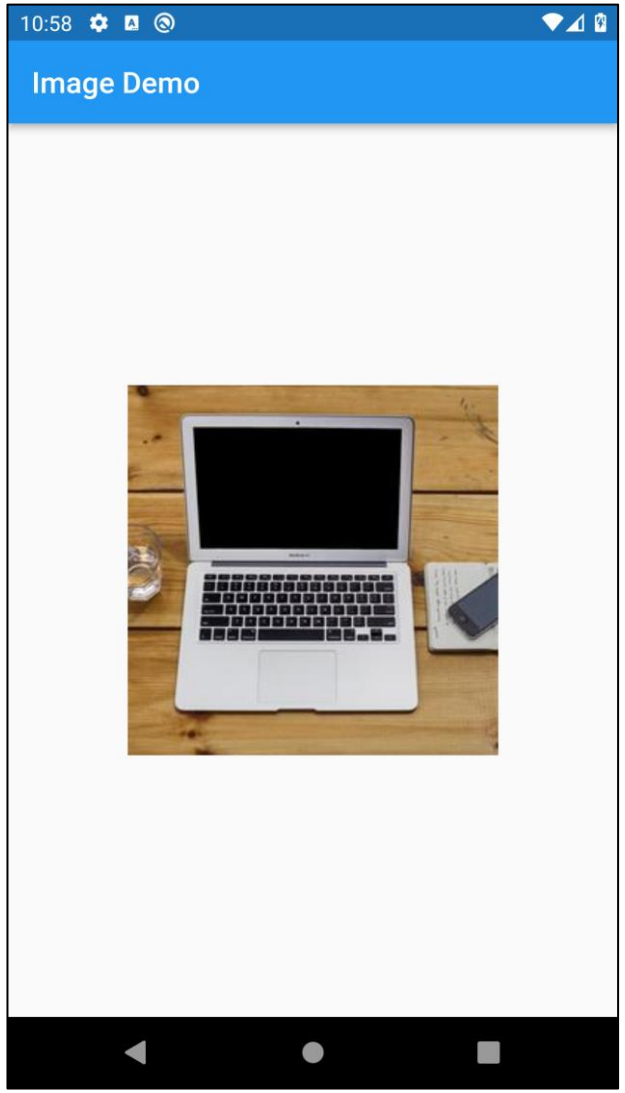
Add internet access permission

- Navigate to android-> app-> src-> main-> AndroidManifest.xml.
- Add these lines outside of application scope.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Example 49: Image.network Demo

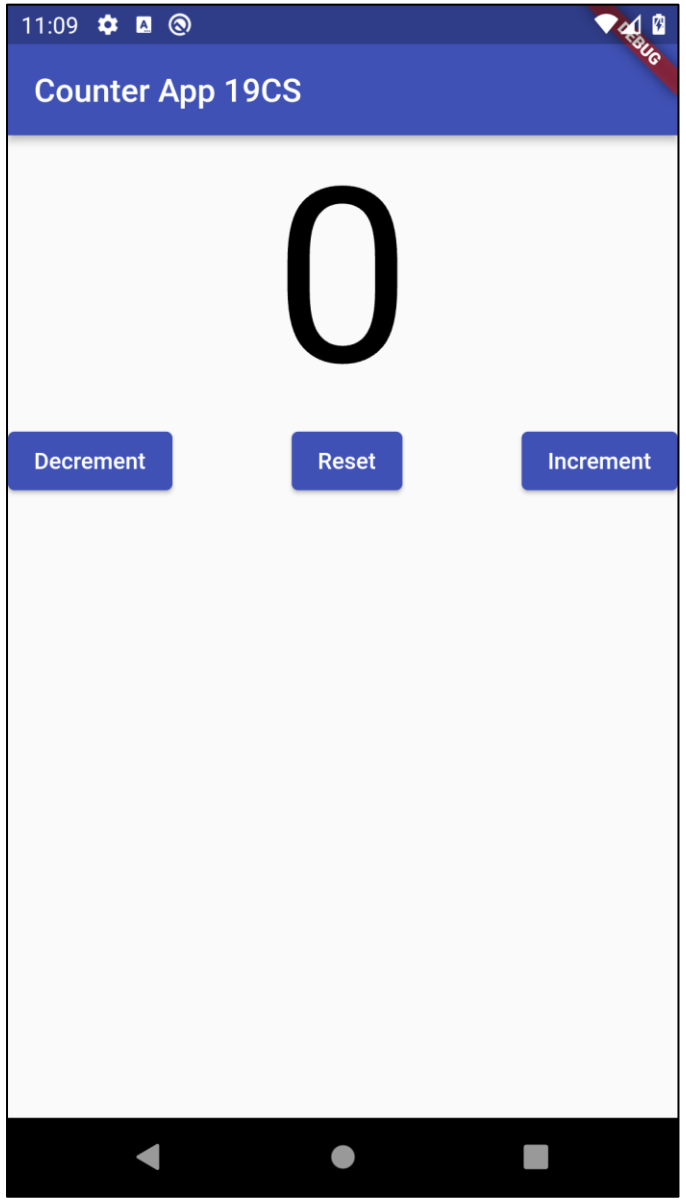
```
Scaffold(
  appBar: AppBar(title: const Text('Image Demo')),
  body: Center(
    child: Container(
      child: Image.network('https://picsum.photos/250?image=9'),
    ),
  ),
)
```

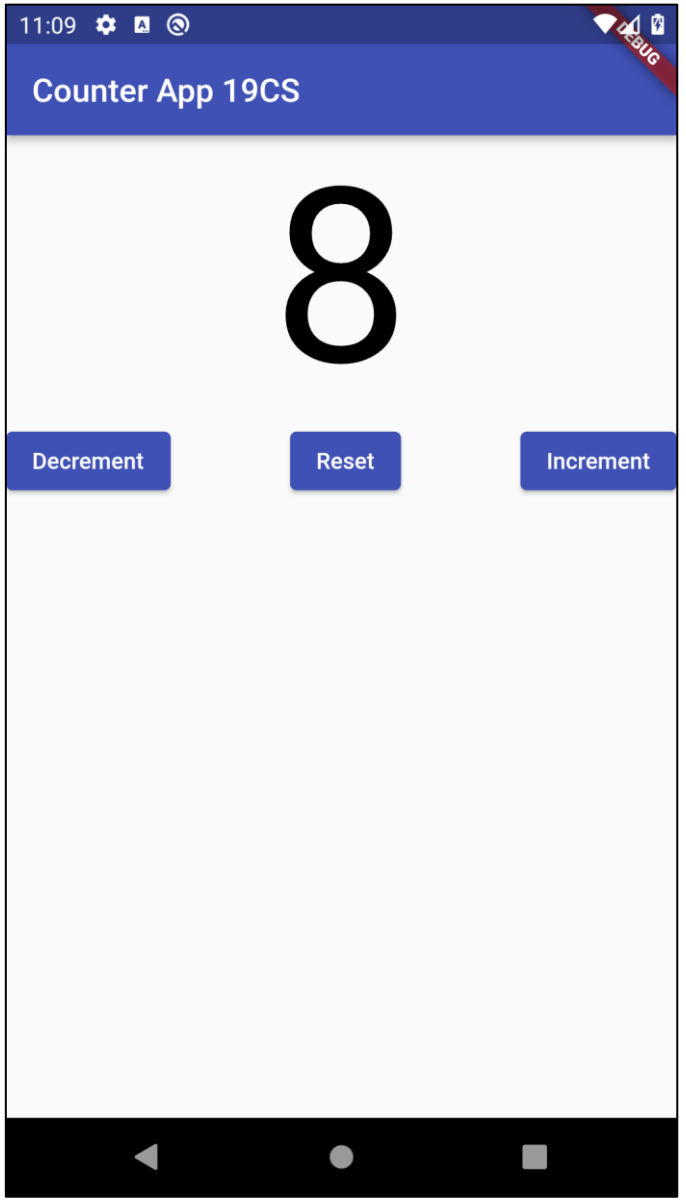


App 01: Counter App

- Develop an app in flutter with one screen.
- The app maintains an integer counter variable which is incremented, decremented and resettled with button actions.

main (counter app-basic).dart





App 01: Counter App

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

void main() {
  runApp(const CounterApp());
}

class CounterApp extends StatelessWidget {
  const CounterApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Counter',
      theme: ThemeData(
        primarySwatch: Colors.indigo,
      ),
      home: const CounterAppHomePage(title: 'Counter App 19CS'),
    );
  }
}
```

App 01: Counter App

```
class CounterAppHomePage extends StatefulWidget {
  const CounterAppHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<CounterAppHomePage> createState() => _CounterAppHomePageState();
}
```

App 01: Counter App

```
class _CounterAppHomePageState extends State<CounterAppHomePage> {
  int _counter = 0;

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

  void _decrementCounter() {
    setState(() {
      if (_counter > 0) {
        _counter--;
      }
    });
  }

  void _resetCounter() {
    setState(() {
      _counter = 0;
    });
  }
```

App 01: Counter App

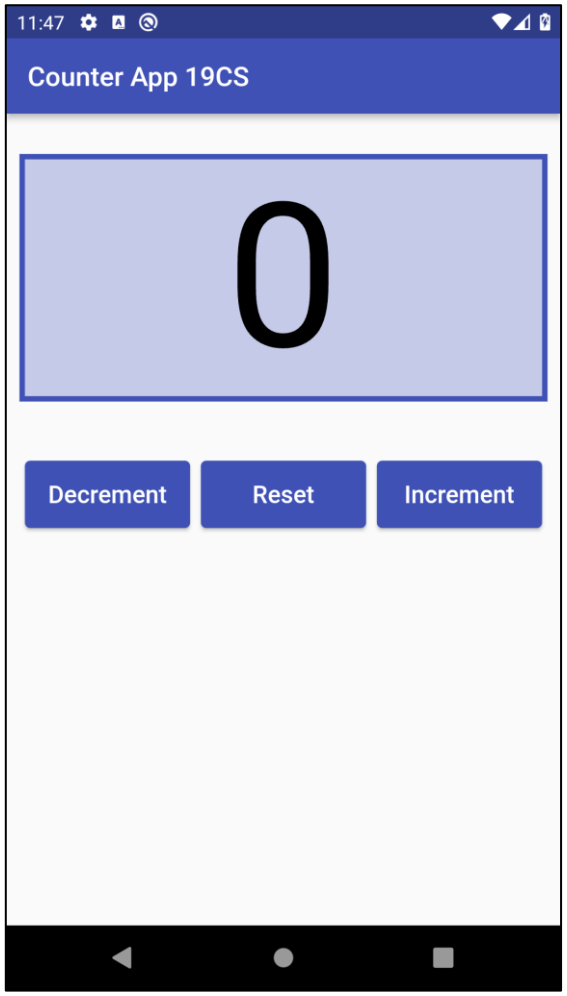
```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: <Widget>[
          Text(
            '$_counter',
            style: const TextStyle(
              color: Colors.black,
              fontSize: 150.0,
            ),
          ),
```

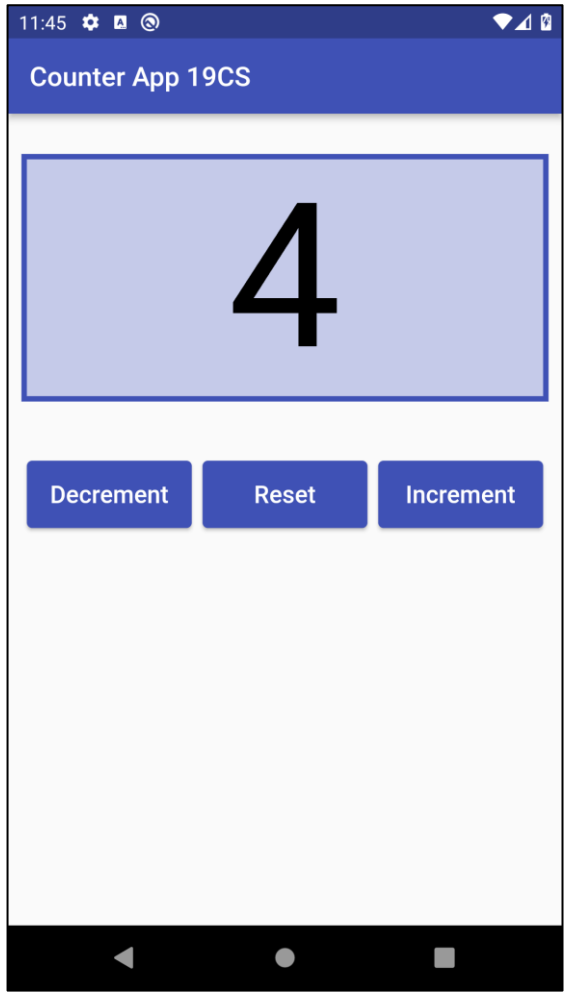
App 01: Counter App

```
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                ElevatedButton(
                  onPressed: _decrementCounter,
                  child: const Text('Decrement'),
                ),
                ElevatedButton(
                  onPressed: _resetCounter,
                  child: const Text('Reset'),
                ),
                ElevatedButton(
                  onPressed: _incrementCounter,
                  child: const Text('Increment'),
                ),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

App 02: Counter App (Styled)

The modified version of the same app applied with different style and layout widgets.





main (counter app-modified).dart