ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

CET215: Mobile Application Development

Lecture 5: Stateful Widget, Actions, dividers
[Counter APP]

© Spring 2025– **Dr. Ahmed Elrefaiy**
Ahmed.elrefai@sut.edu.eg

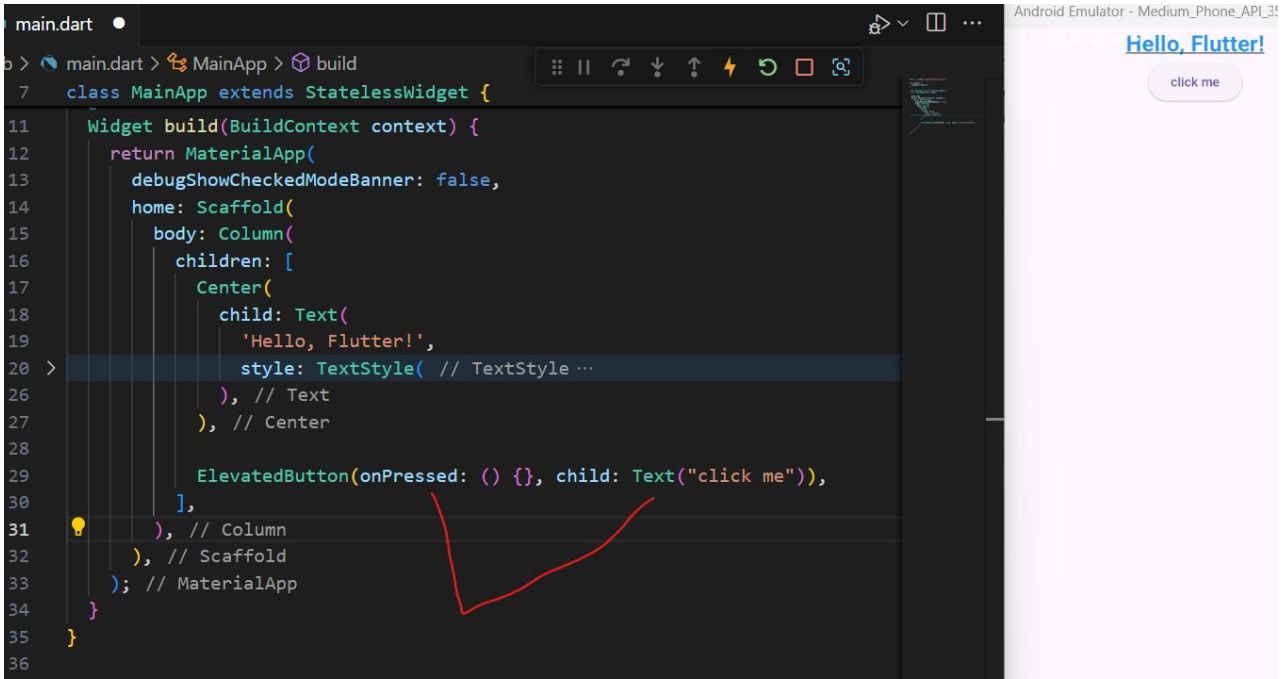
Text Styling in Flutter

- Flutter uses the **Text widget** combined with a **TextStyle** to style text content.
- **fontStyle** – Can set italic style (e.g. FontStyle.italic).


```
Text(
  'Hello, Flutter!',
  style: TextStyle(
    fontSize: 24,
    fontWeight: FontWeight.bold,
    color: Colors.blue,
    decoration: TextDecoration.underline,
  ),
)
```

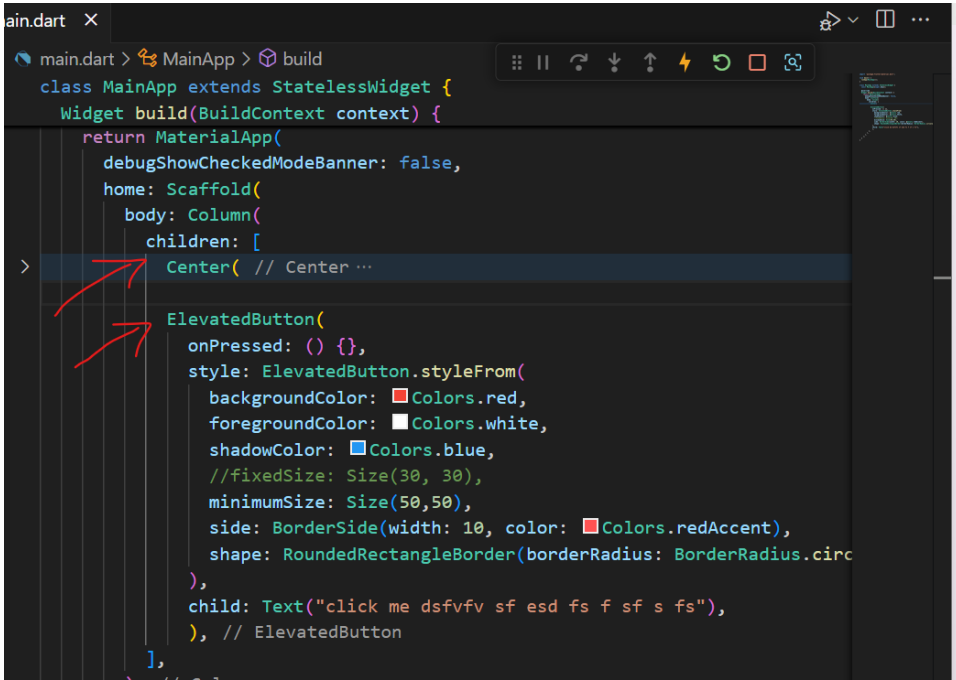
ElevatedButton Widget

- The **ElevatedButton** widget provides a Material-styled button with elevation. Its basic structure includes an **onPressed** callback and a **child** (usually a Text label):



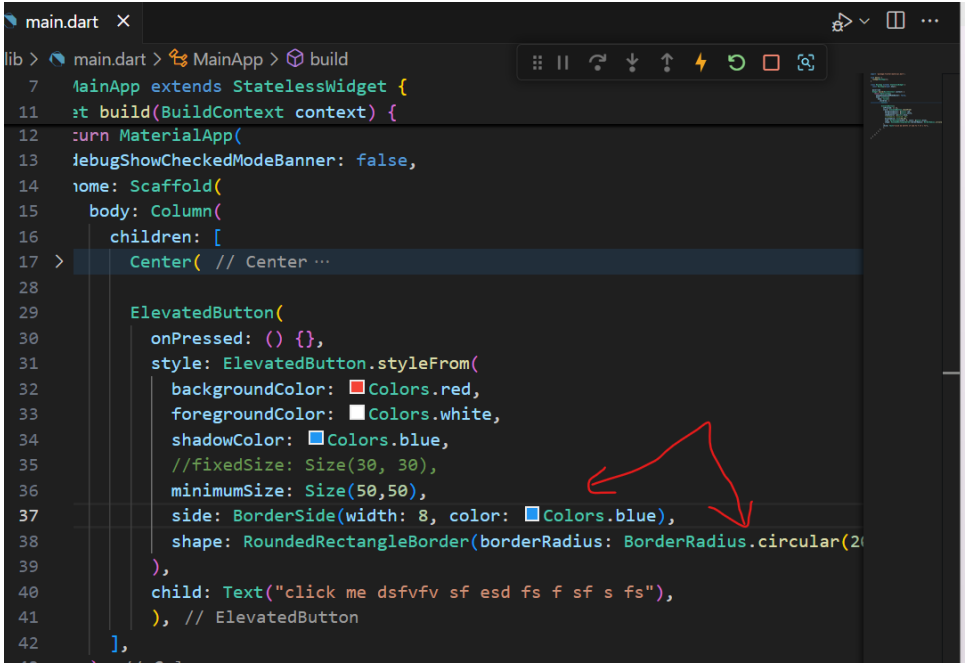
ElevatedButton Widget (Style)

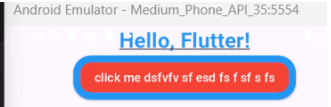
- To customize the appearance, use the **style** property with ElevatedButton.styleFrom
- Background Color → backgroundColor
- Fixed size button → fixedSize: Size(width, height)
  - to make the button a specific size that doesn't change
- Minimum Size → minimumSize: Size(minWidth, minHeight)
  - The button can grow larger if its child content is larger, but will not shrink below these dimensions.



ElevatedButton Widget (Style)

- Shape → You can adjust corner radius or borders
- RoundedRectangleBorder(borderRadius: BorderRadius.circular(20))





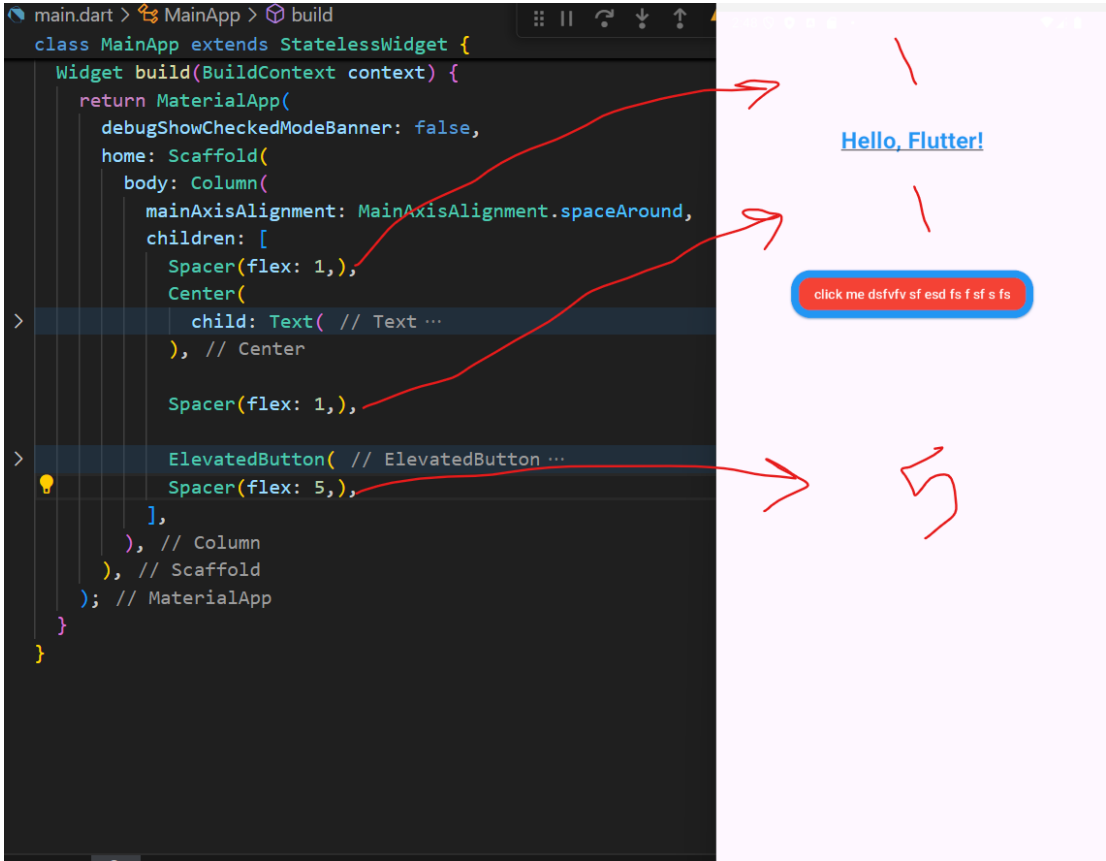
control spacing between Elements

- Using Axis Alignment as first option
- Using other Widgets like:
  - Spacer
  - SizedBox

```
lass MainApp extends StatelessWidget {
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    home: Scaffold(
      body: Column(
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: [
          Center(
            child: Text( // Text ···
          ), // Center

          ElevatedButton( // ElevatedButton ···
        ],
      ), // Column
    ), // Scaffold
  ); // MaterialApp
}
```

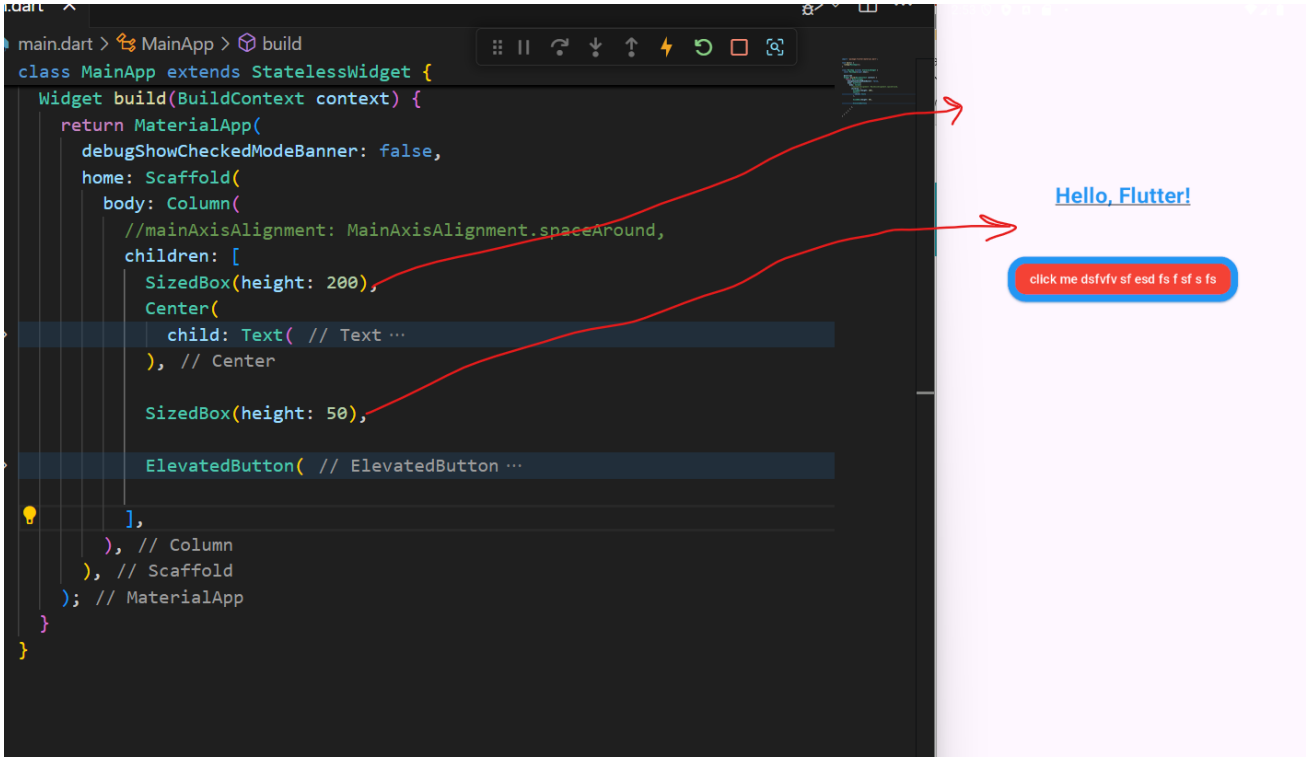
Spacer

- Spacer Widget
- A `Spacer()` is essentially an empty, expandable widget that fills available space in a flex container (Row/Column).
- It uses a (flex factor) to distribute space proportionally.
- By default, `Spacer()` has a flex of 1



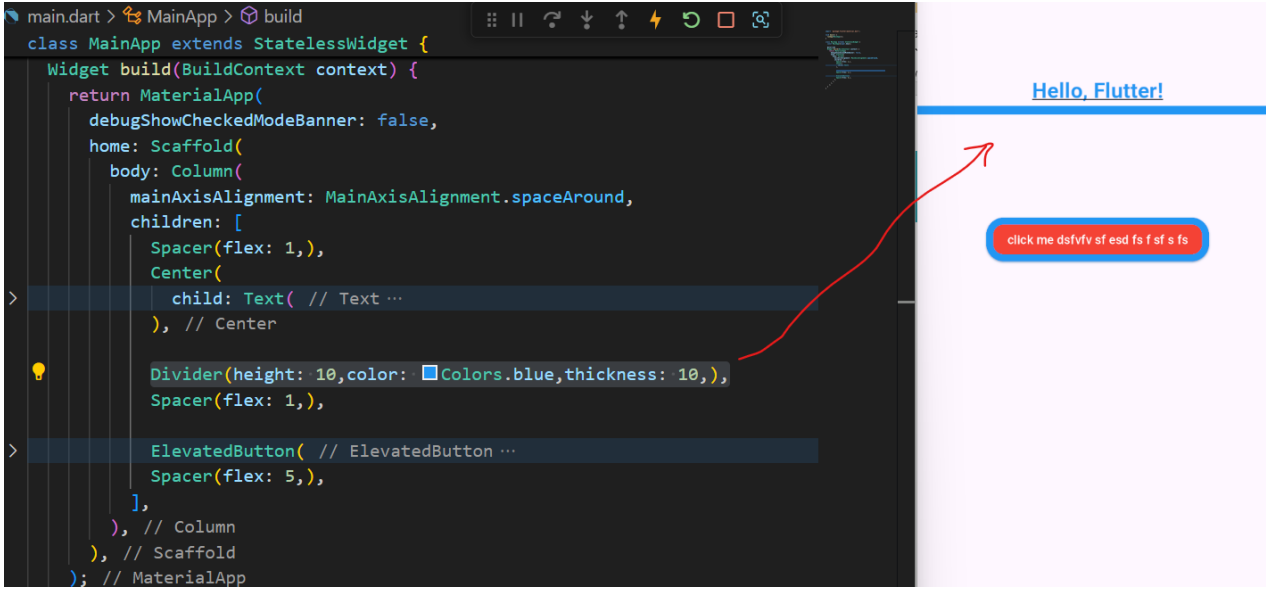
SizedBox

- provides a fixed-size box.
- It can be used as an invisible spacer with a specific width or height,
- or to give a widget a constrained box to live in.



Dividers in Columns and Rows

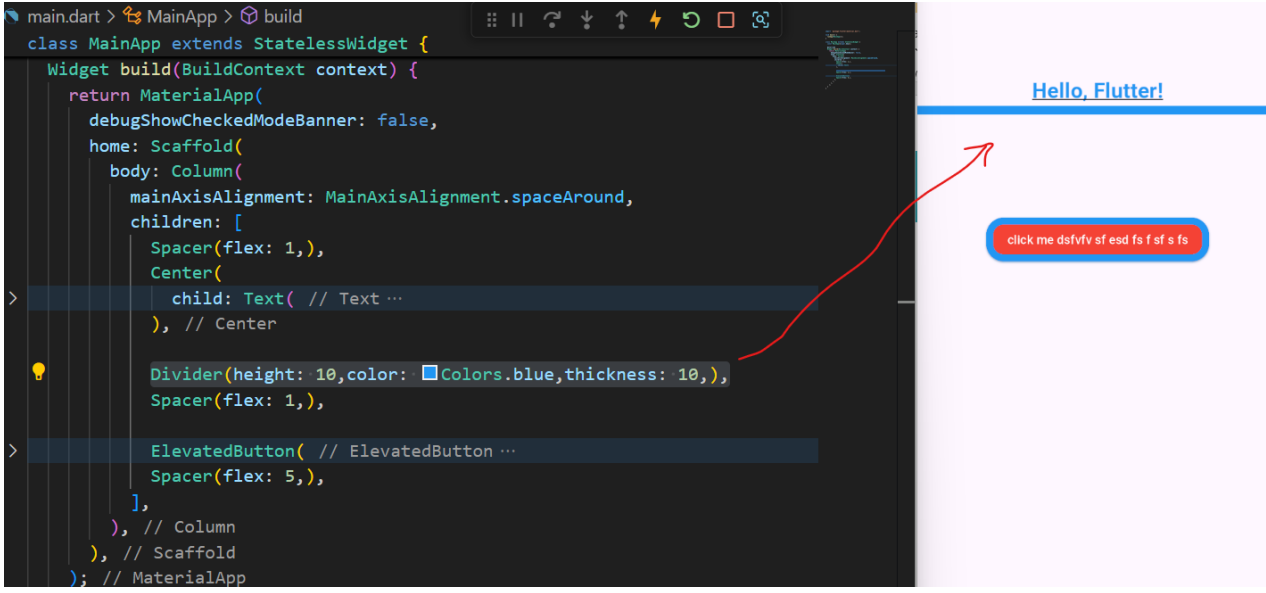
- Flutter provides Two types:
  - Divider (horizontal line) [used with column]
  - VerticalDivider (vertical line) [used with row]
- These widgets used to visually separate content.



Dividers in Columns and Rows

- **Divider (Horizontal) - Key properties:**

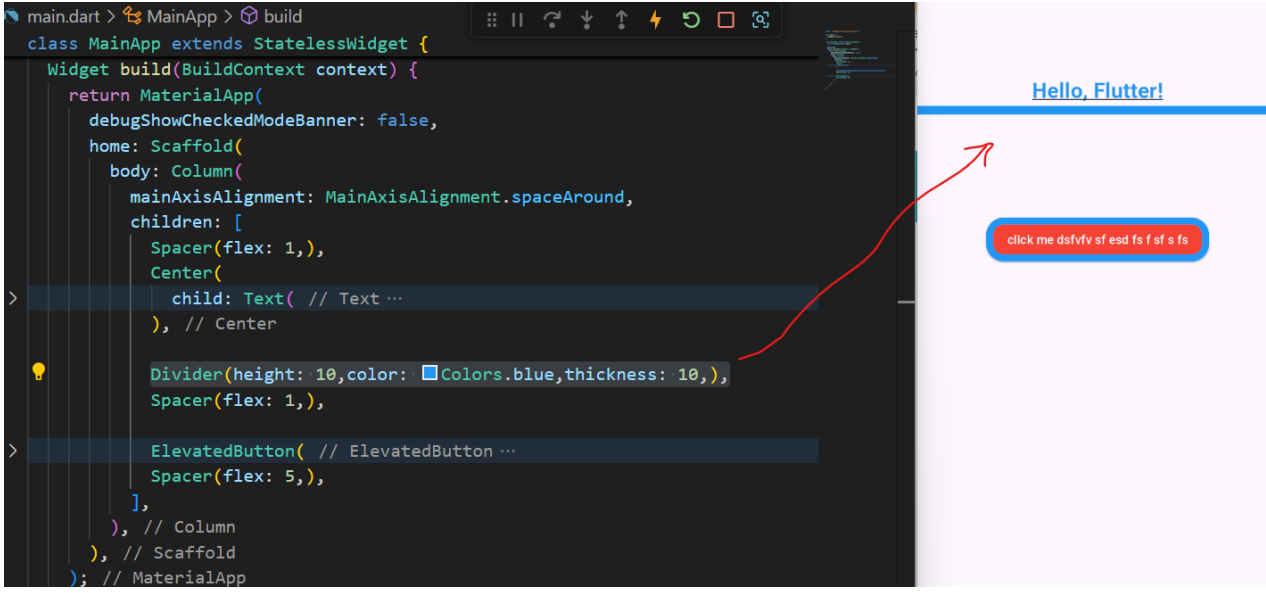
- **color:** the line color (e.g. Colors.grey).
- **thickness:** the line thickness (e.g. thickness: 2 for a 2-pixel thick line).
- **height:** total height of the line



Dividers in Columns and Rows

- **VerticalDivider - Key properties:**

- **color:** the line color (e.g. Colors.grey).
- **thickness:** the line thickness (e.g. thickness: 2 for a 2-pixel thick line).
- **width:** total width of the line



StatefulWidget & State Management

- **Widget types in flutter:**
  - Stateless
  - Stateful
- **Stateless:** (static) - it can’t change once built.

- Not all widgets are static; some need to change dynamically:
- **StatefulWidget** in Flutter are those that can maintain mutable state
- meaning the widget can rebuild with new information over time
- Ex. Counter APP

StatefulWidget & State Management

- **Implemented as two classes:**
- The **StatefulWidget** class itself (which is usually just a lightweight container for configuration)
- and a corresponding **State class** (where the mutable state lives).
- The framework calls **createState()** on the widget to instantiate the state object
- This **build** method is called whenever the state changes to render the updated UI.
- When you need to update the state, you must call **setState(() { ... })** inside the State class. This tells Flutter that the state has changed and triggers a rebuild of the widget

StatefulWidget & State Management

- **Why use StatefulWidget?**
- If a widget needs to update its UI based on user interaction or other events
- Ex.
  - like toggling a favorite icon,
  - updating a counter,
  - fetching data, etc.,

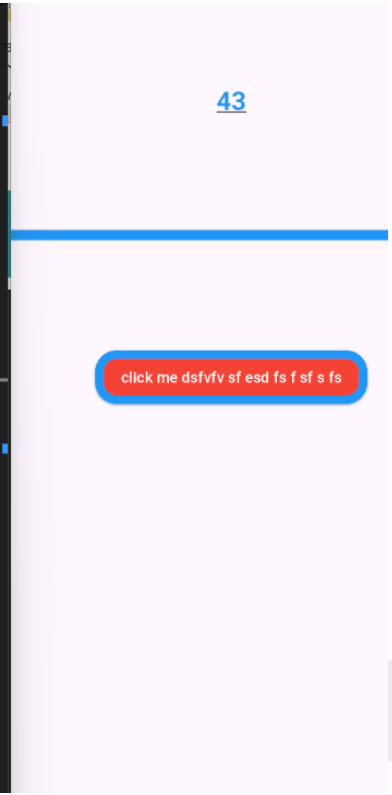
Counter APP

```dart
class MainApp extends StatefulWidget {
  @override
  State<MainApp> createState() => _MainAppState();
}

class _MainAppState extends State<MainApp> {
  int x = 0;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        body: Column(
          mainAxisAlignment: MainAxisAlignment.spaceAround,
          children: [
            Spacer(flex: 1,),
            Center(
              child: Text(
                '$x',
                style: TextStyle(
                  fontSize: 24,
                  fontWeight: FontWeight.bold,
                  color: Colors.blue,
                  decoration: TextDecoration.underline,
```

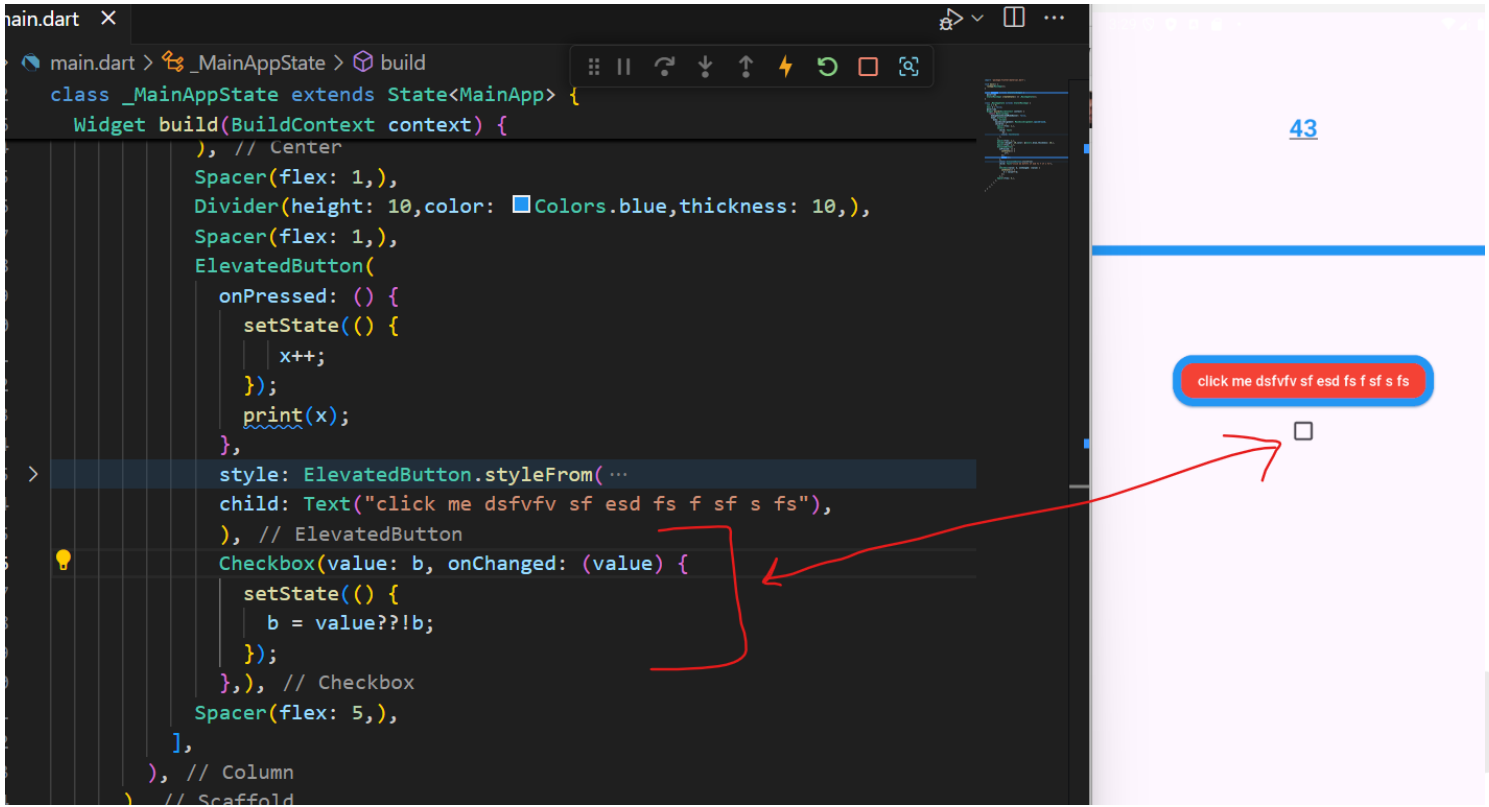
Counter APP

main.dart > _MainAppState > build

```
class _MainAppState extends State<MainApp> {
  Widget build(BuildContext context) {
        mainAxisAlignment: MainAxisAlignment.spaceAround,
        children: [
          Spacer(flex: 1,),
          Center(
            child: Text(
              '$x',
              style: TextStyle( // TextStyle ···
            ), // Text
          ), // Center
          Spacer(flex: 1,),
          Divider(height: 10,color: Colors.blue,thickness: 10,),
          Spacer(flex: 1,),
          ElevatedButton(
            onPressed: () {
              setState(() {
                x++;
              });
              print(x);
            },
            style: ElevatedButton.styleFrom( ···
            child: Text("click me dsfvfv sf esd fs f sf s fs"),
          ), // ElevatedButton
```



Toggle Checkbox



Comparison

Comparison between StatelessWidget and StatefulWidget

Aspect	StatelessWidget	StatefulWidget
Use Case	Static content (no change after rendering).	Dynamic content that updates based on user interaction or other events.
State Management	No state management.	Manages internal state, UI updates with `setState`.
Implementation	Single class that extends `StatelessWidget`.	Two classes: StatefulWidget and State class.
Example Widgets	Text, Icon, Image, simple layouts.	Interactive widgets like counters, forms, or toggles.
Lifecycle Methods	No lifecycle methods needed.	Provides lifecycle methods like `initState`, `dispose`, and `didUpdateWidget`.
Performance	Lightweight and efficient.	Slightly heavier due to state tracking and UI rebuilding.
Syntax Complexity	Simple structure.	Requires splitting widget into two classes (`Widget` & `State`).