Flutter Basic Widgets

Container () Widget:

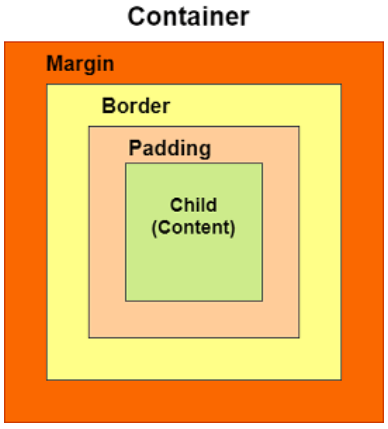
The container in Flutter is a **parent widget that can contain multiple child widgets** and manage them efficiently through width, height, padding, background color, etc. It is a widget that combines common painting, positioning, and sizing of the child widgets. It is also a class to store one or more widgets and position them on the screen according to our needs. Generally, it is similar to a box for storing contents. It allows many attributes to the user for decorating its child widgets, such as using **margin**, which separates the container with other contents.

A container widget is same as ** ** tag in html. If this widget does not contain any child widget, it will fill the whole area on the screen automatically. Otherwise, it will wrap the child widget according to the specified height & width. It is to **note that** this widget cannot render directly without any parent widget. We can use Scaffold widget, Center widget, Padding widget, Row widget, or Column widget as its parent widget.

Why we need a container widget in Flutter?

If we have a widget that needs some background styling may be a color, shape, or size constraints, we may try to **wrap it in a container widget**. This widget helps us to compose, decorate, and position its child widgets. If we wrap our widgets in a container, then without using any parameters, we would not notice any difference in its appearance. But if we add any properties such as color, margin, padding, etc. in a container, we can style our widgets on the screen according to our needs.

A basic container has a margin, border, and padding properties surrounding its child widget, as shown in the below image:



Constructors of the container class

The following are the syntax of container class constructor:

```
Container({Key key,
        AlignmentGeometry alignment,
        EdgeInsetsGeometry padding,
        Color color,
        double width,
        double height,
        Decoration decoration,
        Decoration foregroundDecoration,
        BoxConstraints constraints,
        Widget child,
        Clip clipBehavior: Clip.none
});
```

Properties of Container widget

Let us learn some of the essential properties of the container widget in detail.

1. child: This property is used to store the child widget of the container. Suppose we have taken a Text widget as its child widget that can be shown in the below example:

```
Container(
    child: Text("Hello! I am in the container widget", style: TextStyle(fontSize: 25)),
)
```

2. color: This property is used to set the **background color of the text**. It also changes the background color of the entire container. See the below example:

```
Container(
    color: Colors.green,
    child: Text("Hello! I am in the container widget", style: TextStyle(fontSize: 25)),
)
```

3. height and width: This property is used to set the container's height and width according to our needs. By default, the container always takes the space based on its child widget. See the below code:

```
1. Container(
2.     width: 200.0,
3.     height: 100.0,
4.     color: Colors.green,
5.     child: Text("Hello! I am in the container widget", style: TextStyle(fontSi
   ze: 25)),
6. )
```

4. margin: This property is used to surround the e**mpty space around the container**. We can observe this by seeing white space around the container. Suppose we have used the **EdgeInsets.all(25)** that set the equal margin in all four directions, as shown in the below example:

```
1. Container(
2.     width: 200.0,
3.     height: 100.0,
4.     color: Colors.green,
5.     margin: EdgeInsets.all(20),
6.     child: Text("Hello! I am in the container widget", style: TextStyle(fontSi
   ze: 25)),
7. )
```

5. padding: This property is used to **set the distance** between the border of the container (all four directions) and its child widget. We can observe this by seeing the space between the container and the child widget. Here, we have used an EdgeInsets.all(35) that set the space between text and all four container directions:

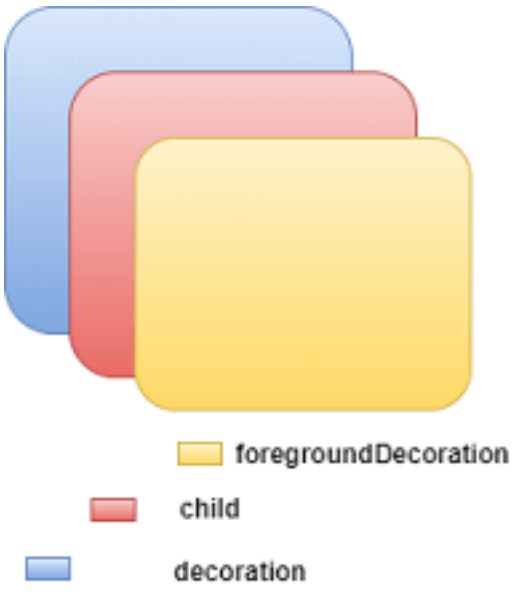
```
1. Container(
2.     width: 200.0,
3.     height: 100.0,
4.     color: Colors.green,
5.     padding: EdgeInsets.all(35),
6.     margin: EdgeInsets.all(20),
```

```
7.     child: Text("Hello! I am in the container widget", style: TextStyle(fontSi
       ze: 25)),
8.  )
```

6. alignment: This property is used to **set the position** of the child within the container. Flutter allows the user to align its element in various ways such as center, bottom, bottom center, topLeft, centerRight, left, right, and many more. In the below example, we are going to align its child into the bottom right position.

```
1.  Container(
2.     width: 200.0,
3.     height: 100.0,
4.     color: Colors.green,
5.     padding: EdgeInsets.all(35),
6.     margin: EdgeInsets.all(20),
7.     alignment: Alignment.bottomRight,
8.     child: Text("Hello! I am in the container widget", style: TextStyle(fontSi
       ze: 25)),
9.  )
```

7. decoration: This property allows the developer to **add decoration on the widget**. It decorates or paint the widget behind the child. If we want to decorate or paint in front of a child, we need to use the **forgroundDecoration** parameter. The below image explains the difference between them where the foregroundDecoration covers the child and decoration paint behind the child.



The decoration property supported many parameters, such as color, gradient, background image, border, shadow, etc. It is to make sure that **we can either use the color property in a container or decoration, but not in both**. See the below code where we have added a border and shadow property to decorate the box:

```
1.  import 'package:flutter/material.dart';
2.
3.  void main() => runApp(MyApp());
4.
5.  /// This Widget is the main application widget.
6.  class MyApp extends StatelessWidget {
7.
8.    @override
9.    Widget build(BuildContext context) {
10.    return MaterialApp(
11.      home: Scaffold(
12.        appBar: AppBar(
13.          title: Text("Flutter Container Example"),
14.        ),
15.        body: Container(
16.          padding: EdgeInsets.all(35),
17.          margin: EdgeInsets.all(20),
18.          decoration: BoxDecoration(
19.            border: Border.all(color: Colors.black, width: 4),
20.            borderRadius: BorderRadius.circular(8),
21.            boxShadow: [
22.              new BoxShadow(color: Colors.green, offset: new Offset(6.0, 6.0),
    ),
23.            ],
24.          ),
25.          child: Text("Hello! I am in the container widget decoration box!!",
26.              style: TextStyle(fontSize: 30)),
27.        ),
28.      ),
29.    );
30.  }
```

31.}

We will see the output as below screenshot:

8. transform: The transform property allows developers to **rotate the container**. It can rotate the container in any direction, i.e., change the container coordinate in the parent widget. In the below example, we will rotate the container in the **z-axis**.

1. Container(

```
    width: 200.0,
    height: 100.0,
    color: Colors.green,
    padding: EdgeInsets.all(35),
    margin: EdgeInsets.all(20),
    alignment: Alignment.bottomRight,
    transform: Matrix4.rotationZ(0.1),
    child: Text("Hello! I am in the container widget", style: TextStyle(fontSize: 25)),
)
```

9. constraints: This property is used when we want to **add additional constraints to the child**. It contains various constructors, such as tight, loose, expand, etc. Let's see how to use these constructors in our app:

tight: If we use size property in this, it will give fixed value to the child.

```
Container(
    color: Colors.green,
    constraints: BoxConstraints.tight(Size size)
      : minWidth = size.width, maxWidth = size.width,
        minHeight = size.height, maxHeight = size.height;
    child: Text("Hello! I am in the container widget", style: TextStyle(fontSize: 25)),
)
```

expand: Here, we can choose the height, width, or both values to the child.

```
Container(
    color: Colors.green,
    constraints: BoxConstraints.expand(height: 60.0),
    child: Text("Hello! I am in the container widget", style: TextStyle(fontSize: 25)),
)
```

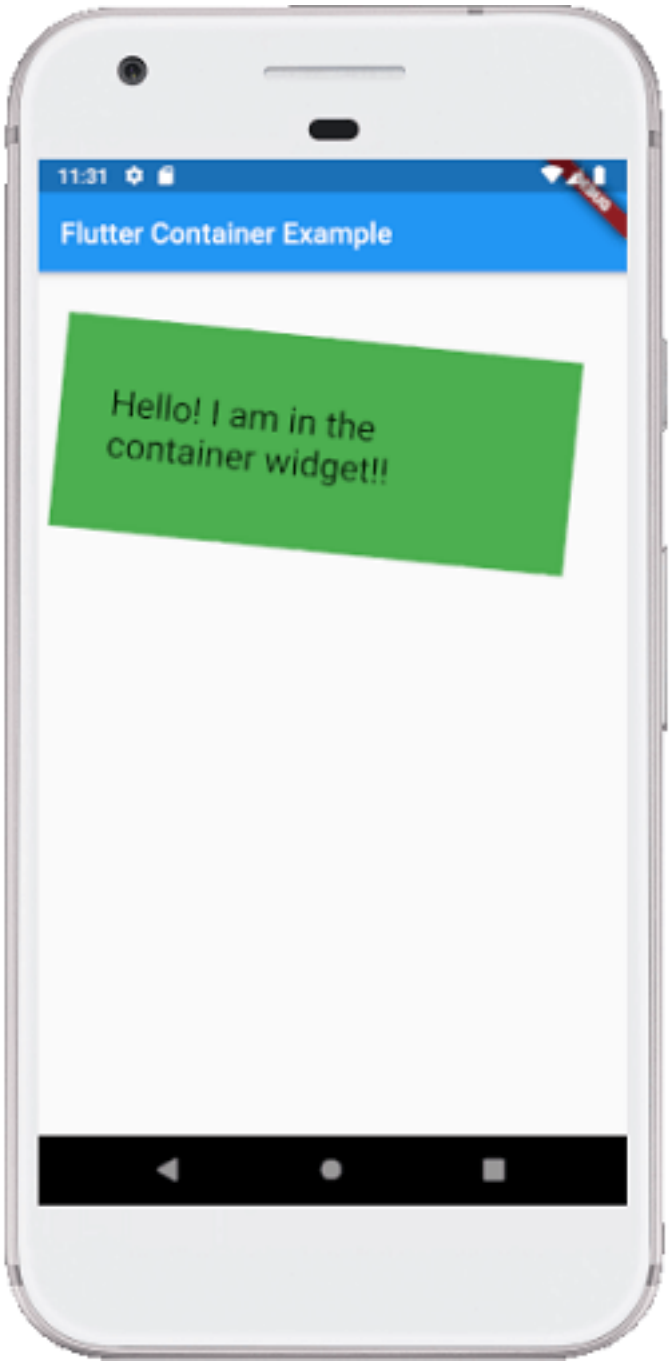
Let us understand it with an example where we will try to cover most of the container properties. Open the **main.dart** file and replace it with the below code:

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

void main() => runApp(MyApp());

/// This Widget is the main application widget.
class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
   return MaterialApp(
    home: MyContainerWidget(),
   );
  }
}

class MyContainerWidget extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   home: Scaffold(
    appBar: AppBar(
     title: Text("Flutter Container Example"),
    ),
    body: Container(
     width: double.infinity,
     height: 150.0,
     color: Colors.green,
     margin: EdgeInsets.all(25),
     padding: EdgeInsets.all(35),
     alignment: Alignment.center,
     transform: Matrix4.rotationZ(0.1),
     child: Text("Hello! I am in the container widget!!",
        style: TextStyle(fontSize: 25)),
    ),
   ),
  );
 }
```

38.}

Output

When we run this app, it will give the following screenshot:

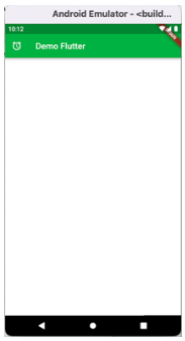


Extra- Work: If this widget does not contain any child widget, it will fill the whole area on the screen automatically.

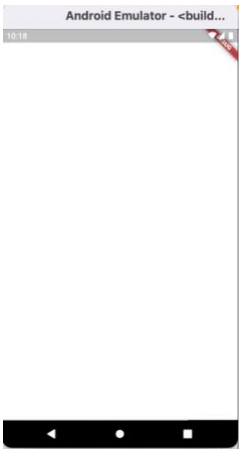
main.dart

lib > main.dart > ...

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

Run | Debug | Profile
void main() {

  runApp(const MyHomePage());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        backgroundColor: Colors.teal,
        appBar: AppBar(
          title: const Text('Demo Flutter'),
          backgroundColor: Colors.green,
          leading: const Icon(Icons.access_alarm),
        ), // AppBar
        body: Container(
          color: Colors.white,
        ), // Container
      ), // Scaffold
    ); // MaterialApp
  }
}
```



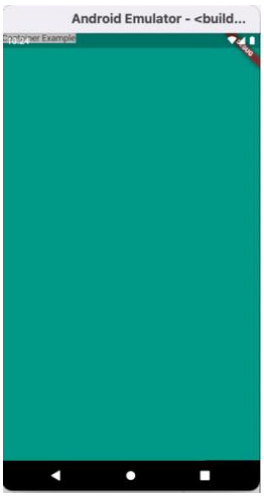
Task- What will happen if you give the don’t use the app bar widget.



Task- what will happen if you provide a child property of container.

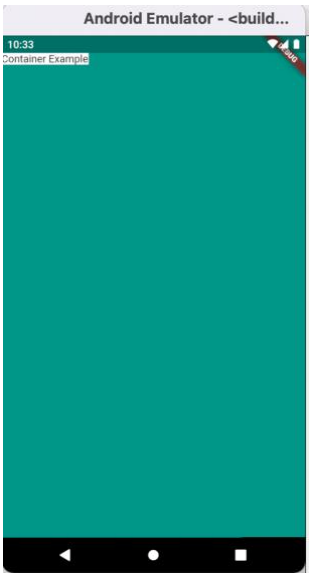
- *Result: Containers with children, size themselves to their children*.

```
10  @override
11  Widget build(BuildContext context) {
12    return MaterialApp(
13      home: Scaffold(
14        backgroundColor: Colors.teal,
15        body: Container(
16          color: Colors.white,
17          child:const Text('Container Example'),
18        ), // Container
19      ), // Scaffold
20    ); // MaterialApp
21  }
```



As we see the output our text has been gone on the top bar near the notch of the phone or border of the phone. So how to save it not to go there. For this, flutter have provided with the widget called SafeArea which helps us to put our widget inside the top bar where time or other things are shown.

```
15          body: SafeArea(
16            child: Container(
17              color: Colors.white,
18              child:const Text('Container Example'),
19            ), // Container
20          ), // SafeArea
```



We can increase the size of container by using the Property of Container – width and height. (Do yourself).

More about **margin and padding:**

```
10    @override
11    Widget build(BuildContext context) {
12      return MaterialApp(
13        home: Scaffold(
14          backgroundColor: Colors.teal,
15          body: SafeArea(
16            child: Container(
17              width: 100,
18              height: 100,
19              margin:const EdgeInsets.all(50),
20              padding: const EdgeInsets.all(30),
21              color: Colors.white,
22              child:const Text('Container Example'),
23            ), // Container
24          ), // SafeArea
25        ), // Scaffold
26      ); // MaterialApp
27    }
28  }
```

Task- Add a container inside another container with two different colors for each container. Use the property of container height, width, margin, padding etc.