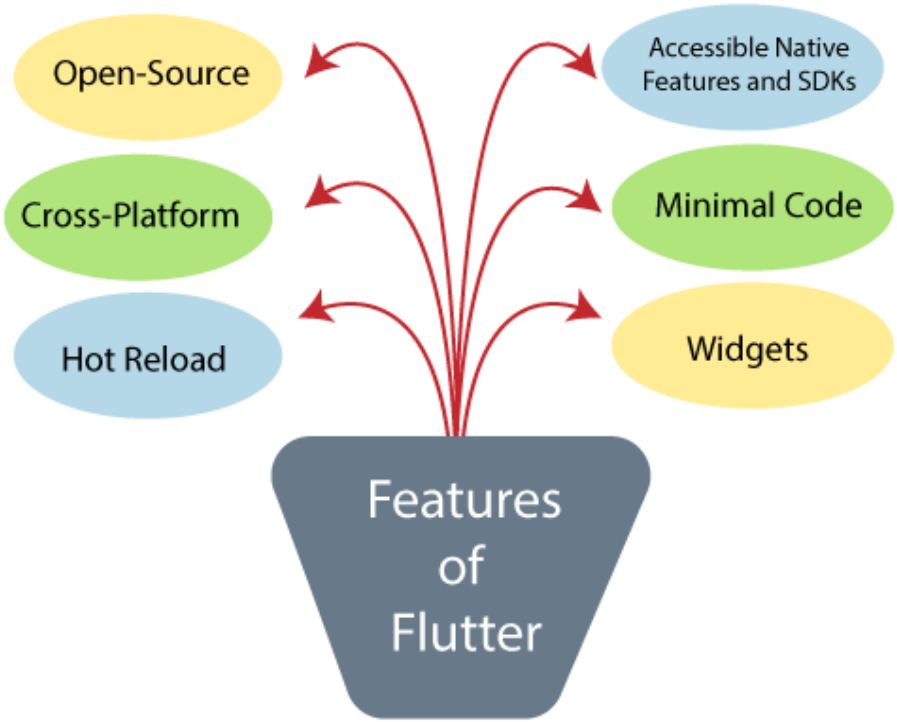
FLUTTER

Flutter is a UI toolkit for creating fast, beautiful, natively compiled applications for mobile, web, and desktop with one programing language and single codebase. It is free and open-source. It was initially developed from **Google** and now manages by European Computer Manufacturers Association (**ECMA)** standard. Flutter apps use Dart programming language for creating an app. The **dart programming** shares several same features as other programming languages, such as Kotlin and Swift, and can be trans-compiled into JavaScript code.

Flutter is mainly optimized for 2D mobile apps that can run on both Android and iOS platforms. We can also use it to build full-featured apps, including camera, storage, geolocation, network, third-party SDKs, and more.

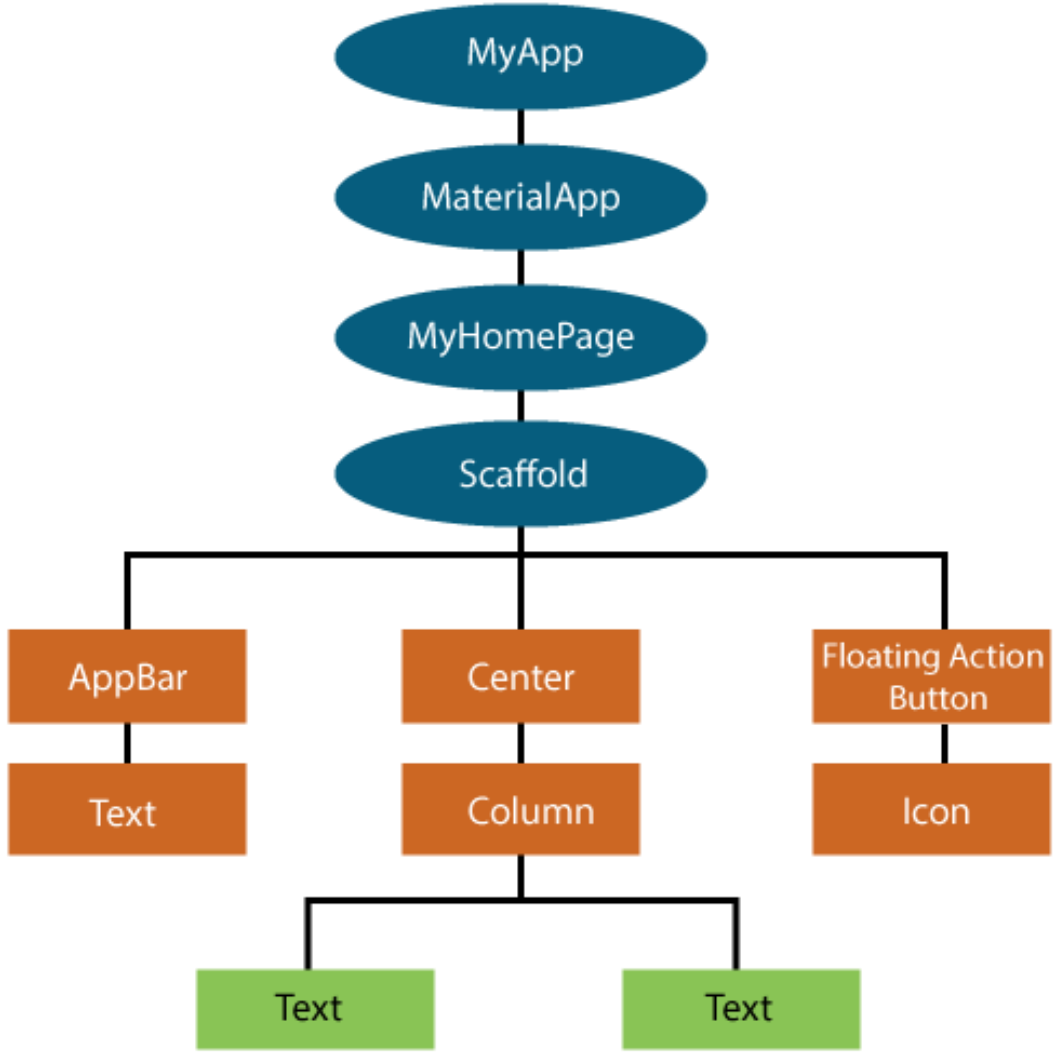
Features of Flutter

Flutter gives easy and simple methods to start building beautiful mobile and desktop apps with a rich set of material design and widgets.



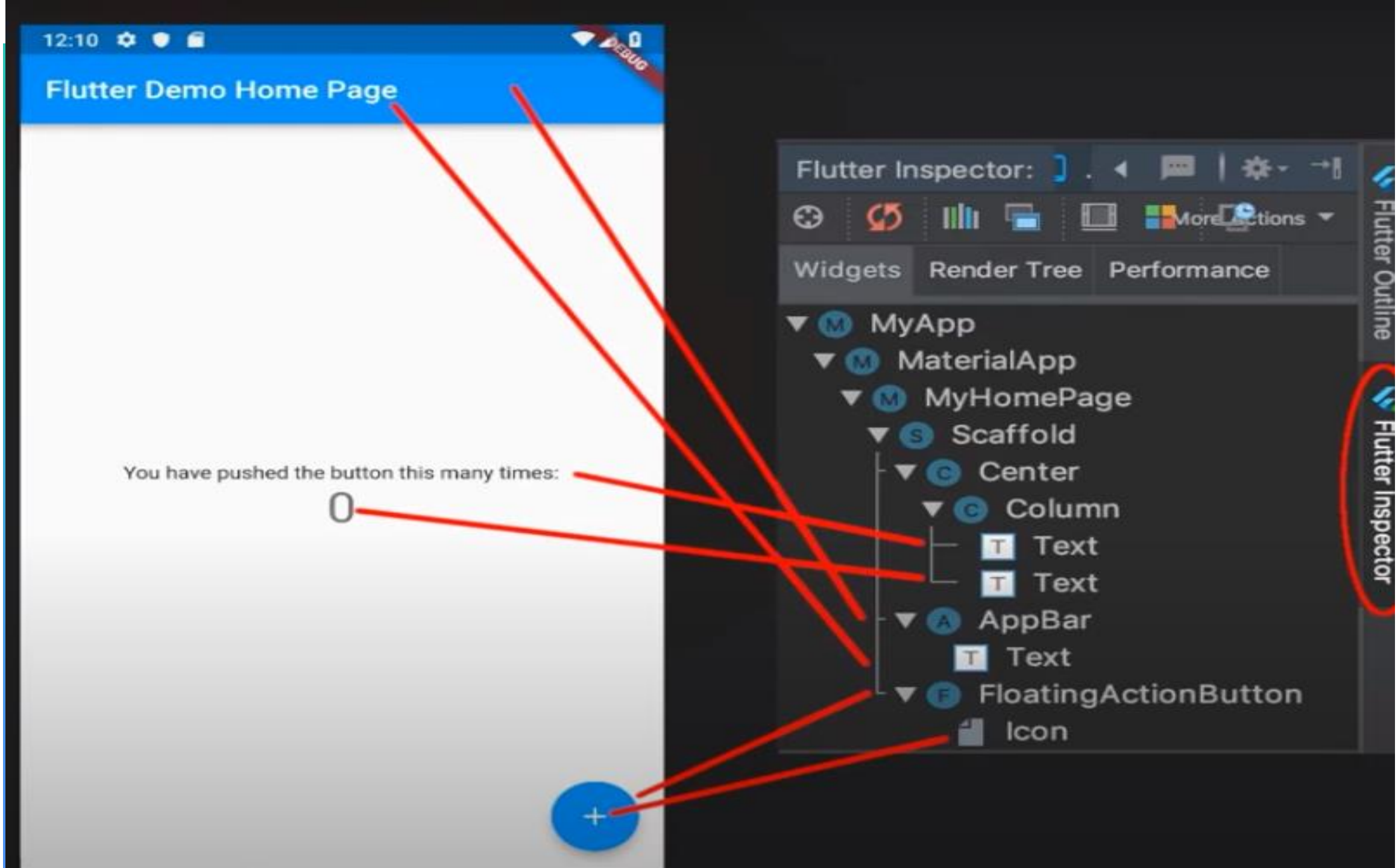
Flutter Widgets

Whenever you are going to code for building anything in Flutter, it will be inside a widget. The central purpose is to build the app out of widgets. It describes how your app view should look like with their current configuration and state. When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.
Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.



We can create the Flutter widget like this:

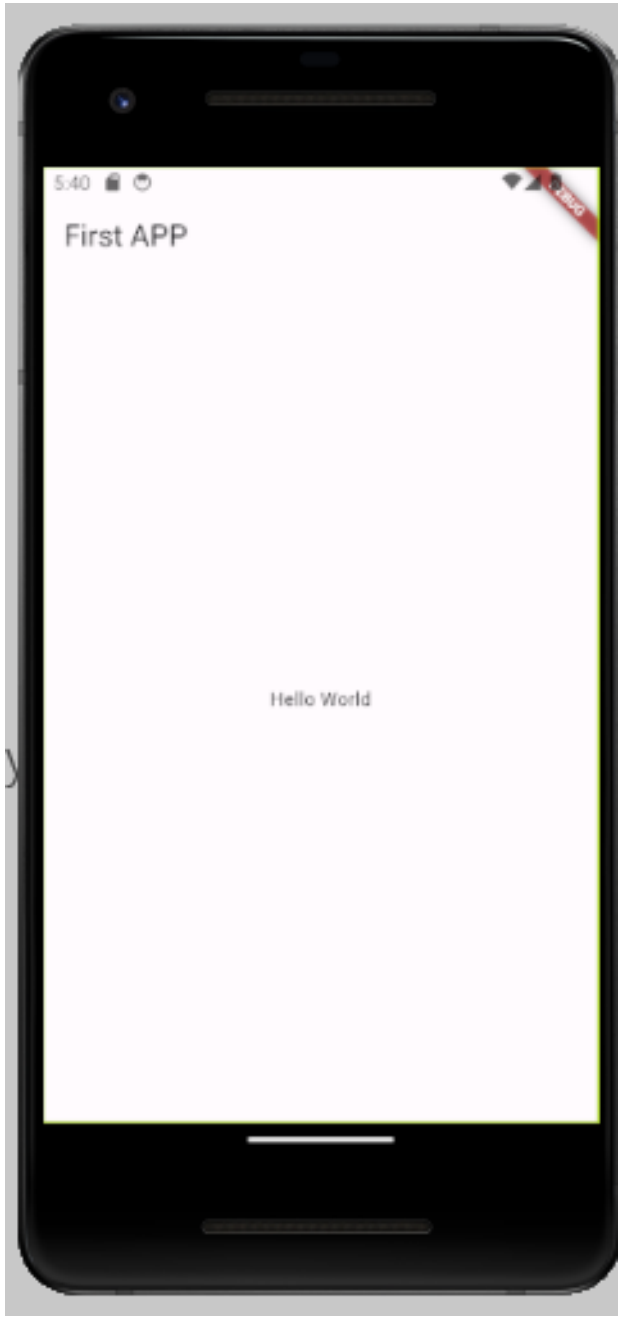
```
Class ImageWidget extends StatelessWidget {
    // Class Stuff
}
```


12:10
DEBUG
Flutter Demo Home Page
You have pushed the button this many times:
0
Flutter Inspector:
More Actions
Widgets
Render Tree
Performance
MyApp
MaterialApp
MyHomePage
Scaffold
Center
Column
Text
Text
AppBar
Text
FloatingActionButton
Icon
Flutter Outline
Flutter Inspector

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

void main() {
  runApp(MyHomePage());
}

class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("First APP"),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```



Types of Widget

Visible widget

The visible widgets are related to the user input and output data. Some of the important types of this widget are:

Text

A Text widget holds some text to display on the screen. We can align the text widget by using **textAlign** property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more. We can use it as like below code snippets.

```
new Text(
Hello, Javatpoint!',
textAlign: TextAlign.center,
style: new TextStyle(fontWeight: FontWeight.bold),
)
```

Button

This widget allows you to perform some action on click. Flutter does not allow you to use the Button widget directly; instead, it uses a type of buttons like a TextButton and a ElevatedButton. We can use it as like below code snippets.

```
// TextButton  Example
new TextButton (
  child: Text("Click here"),
  onPressed: () {
    // Do something here
  },
),

//ElevatedButton Example
new ElevatedButton (
  child: Text("Click here"),
  elevation: 5.0,
  onPressed: () {
    // Do something here
  },
),
```

Image

This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

- **Image:** It is a generic image loader, which is used by **ImageProvider**.
- **asset:** It load image from your project asset folder.
- **file:** It loads images from the system folder.
- **memory:** It load image from memory.
- **network:** It loads images from the network.

To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in **pubspec.yaml** file.

```
assets:
  - assets/
```

```
class MyHomePage extends StatelessWidget {
  // This widget is the home page of your application.
  final String title;

  @override
  Widget build(BuildContext context) {
    return  MaterialApp (
    home: scaffold(
      appBar: AppBar(
        title: Text(this.title),
     ),
      body: Center(
        child: Image.asset('assets/computer.png'),
      ),
     ),
    );
  }
}
```

Icon

This widget acts as a container for storing the Icon in the Flutter. The following code explains it more clearly.

```
Icon( Icons.thumb_up,
 color: Colors.blue,
 size: 100, ),
```

Invisible widget

The invisible widgets are related to the layout and control of widgets. It provides controlling how the widgets actually behave and how they will look onto the screen. Some of the important types of these widgets are:

Column

A column widget is a type of widget that arranges all its children's widgets in a vertical alignment. It provides spacing between the widgets by using
the **mainAxisAlignment** and **crossAxisAlignment** properties. In these properties, the main axis is the vertical axis, and the cross axis is the horizontal axis.

```
new Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    new Text(
      "VegElement",
    ),
    new Text(
      "Non-vegElement"
    ),
  ],
),
```

Row

The row widget is similar to the column widget, but it constructs a widget horizontally rather than vertically. Here, the main axis is the horizontal axis, and the cross axis is the vertical axis.

```
new Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    new Text(
      "VegElement",
    ),
    new Text(
      "Non-vegElement"
    ),
  ],
),
```

Center

This widget is used to center the child widget, which comes inside it. All the previous examples contain inside the center widget.

```
Center(
  child: new column(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      new Text(
        "VegElement",
      ),
      new Text(
        "Non-vegElement"
      ),
    ],
  ),
),
```

Padding

This widget wraps other widgets to give them padding in specified directions. You can also provide padding in all directions. We can understand it from the below example that gives the text widget padding of 6.0 in all directions.

```
Padding(
 padding: const EdgeInsets.all(6.0),
  child: new Text(
    "Element 1",
  ),
),
```

Scaffold

This widget provides a framework that allows you to add common material design elements like AppBar, Floating Action Buttons, Drawers, etc.

Stack

It is an essential widget, which is mainly used for **overlapping** a widget, such as a button on a background gradient.

State Management Widget

In Flutter, there are mainly two types of widget:

- StatelessWidget
- StatefulWidget

StatefulWidget

A StatefulWidget has state information. It contains mainly two classes: the **state object** and the **widget**. It is dynamic because it can change the inner data during the widget lifetime. This widget does not have a **build()** method. It has **createState()** method, which returns a class that extends the Flutters State Class. The examples of the StatefulWidget are Checkbox, Radio, Slider, Form, and TextField.

```
class Car extends StatefulWidget {
  @override
  _CarState createState() => _CarState();
}

class _CarState extends State<Car> {
  @override
  Widget build(BuildContext context) {
    return Container(
      color: const Color(0xFEEFE),
            child: Container(      //child: Container() )
        )
    );
  }
}
```

StatelessWidget

The StatelessWidget does not have any state information. It remains static throughout its lifecycle. The examples of the StatelessWidget are Text, Row, Column, Container, etc.

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

  @override
  Widget build(BuildContext context) {
    return Container(
   color: const Color(0x0xFEEFE));
  }
}
```