Section : IT Systems Development (DSI)
Module : Framework cross-platform workshop
Teaching unit : Mobile Developpment and Web
Level : 3rd Year (Applied license : LMD)

Workshop 2: Using Stateful Widgets

Technical requirements

In order to start with Flutter, we need a few tools:

- A PC with a recent Windows version, or a Mac with a recent version of the macOS or Linux operating system. we can also use a Chrome OS machine, with a few tweaks.
- An Android/iOS setup.
- The Flutter SDK. It's free, light, and open source.
- Physical device/emulator/simulator
- Android Studio/IntelliJ IDEA or Visual Studio Code

To start

1. Create a class that extends StatelessWidget.

2. Override the build() method.

3. Return a widget.

So in the main.dart file, remove the example code and write the code given as follows:

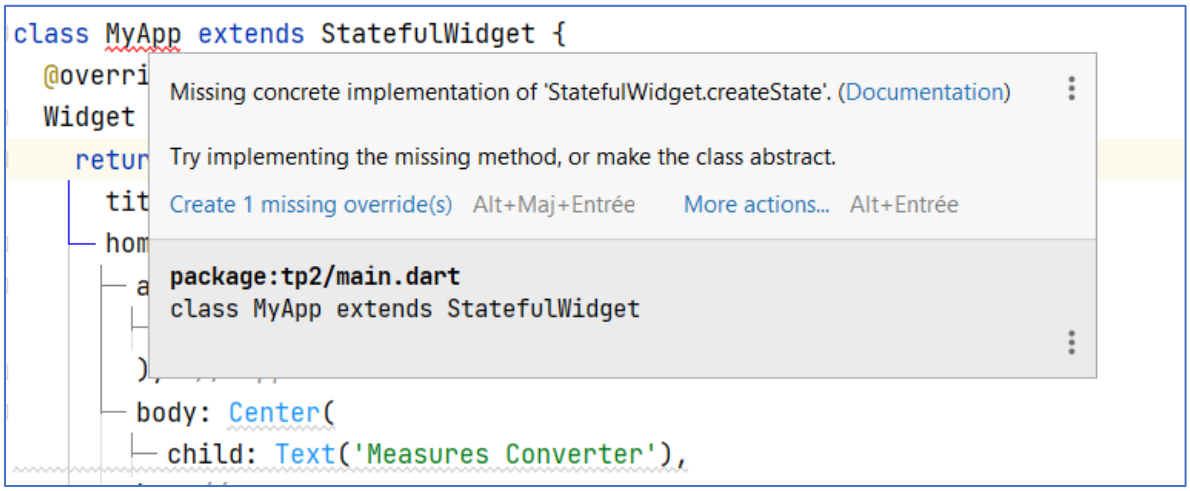
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Measures Converter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Measures Converter'),
        ),
        body: Center(
          child: Text('Measures Converter'),
        ),
      ),
    );
  }
}
```

A Stateless widget is a class that extends a StatelessWidget. Extending a StatelessWidget class requires overriding a build() method.

Using stateful widgets

Transform the MyApp class into a stateful widget,

```
class MyApp extends StatefulWidget {
```



What these errors are trying to tell us is the following:
a. A stateful widget requires a createState() method.
b. In a stateful widget, there is no build() method to override.

So
1- Add the necessary createState() method

```
@override
MyAppState createState() => MyAppState();
```

2- Create a new class called MyAppState, that extends the State, and in particular, the State of MyApp:

```
class MyAppState extends State<MyApp> {}
```

The revised code should look like this:

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatefulWidget {
  @override
  MyAppState createState() => MyAppState();
}
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Measures Converter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Measures Converter'),
        ),
        body: Center(
          child: Text('Measures Converter'),
        ),
```

```
        ),
      );
    }
}
```

To sum it up, from a syntax perspective, the difference between a Stateless widget and a stateful widget is that the former overrides a build() method and returns a widget, whereas a stateful widget overrides a createState() method, which returns a State. The State class overrides a build() method, returning a widget.

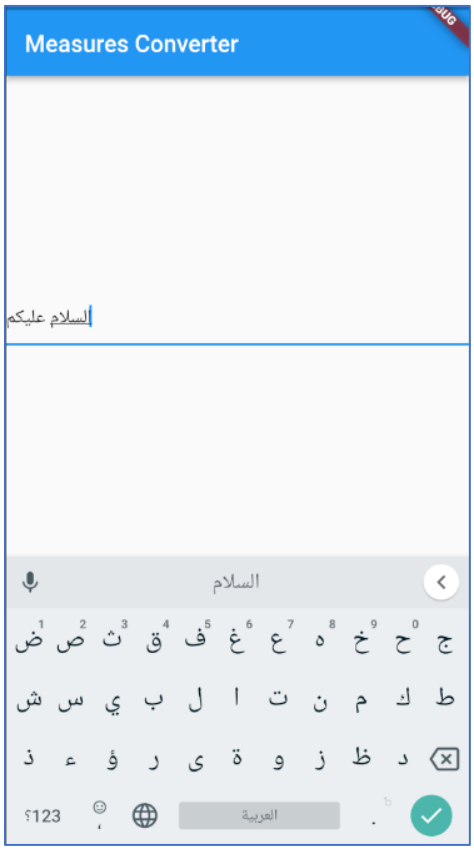
the app layout:

Reading user input from TextField

In the State class, let's add a member called _numberFrom. As shown in the following code, this is a value that will change based on user input:
double _numberFrom;

1- Then, in the body of the build() method, let's delete the text widget, and add TextField instead:

```
body: Center(
  child: TextField(),
),
```



For this application, we'll respond to each change in the content of TextField through the onChanged method, and then we'll update the State.

In order to update the State, you need to call the setState() method

2- let's add a Text widget that will show the content of the TextEdit widget, and then wrap the two widgets into a Column widget:

Before trying the app, let's add another method to the MyAppState class:

```
double _numberFrom = 0;
  @override
  void initState() {
    _numberFrom = 0;
    super.initState();
  }
```

The new code is :

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

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

class MyApp extends StatefulWidget {
  @override
  MyAppState createState() => MyAppState();
}

class MyAppState extends State<MyApp> {
  double _numberFrom = 0;
  @override
  void initState() {
    _numberFrom = 0;
```

```
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Measures Converter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Measures Converter'),
        ),
        body: Center(
          child: Column(
            children: [
              TextField(
                onChanged: (text) {
                  var rv = double.tryParse(text);
                  if (rv != null) {
                    setState(() {
                      _numberFrom = rv;
                    });
                  }
                },
              ),
              Text((_numberFrom == null) ? '' : _numberFrom.toString())
            ],
          ),
        ),
      ),
    );
  }
}
```

Here is a diagram that highlights the steps described previously: with a few variations, you'll notice a similar pattern whenever you use stateful widgets in your apps:

Creating a DropdownButton widget

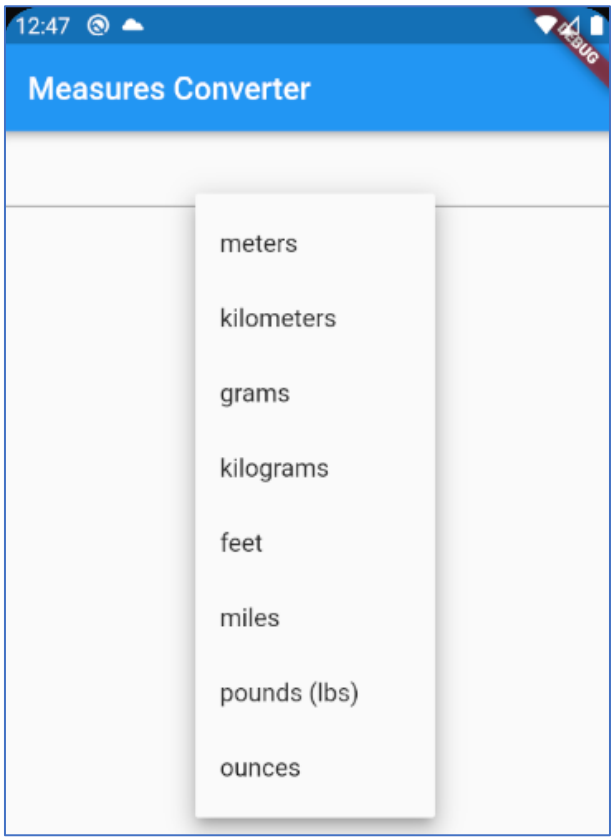
DropdownButton is a widget that lets users select a value from a list of items.

1. Create an instance of DropdownButton, specifying the type of data that will be included in the list.
2. Add an items property that will contain the list of items that will be shown to the user.
3. The items property requires a list of DropdownMenuItem widgets. Therefore, you need to map each value that you want to show into DropdownMenuItem.
4. Respond to the user actions by specifying an event; typically, for DropdownButton, you will call a function in the onChanged property.

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

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

class MyApp extends StatefulWidget {
  @override
  MyAppState createState() => MyAppState();
}

class MyAppState extends State<MyApp> {
  late double _numberFrom;
  @override
  void initState() {
    _numberFrom = 0;
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    var fruits = ['Orange', 'Apple', 'Strawberry', 'Banana'];
    return MaterialApp(
      title: 'Measures Converter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Measures Converter'),
        ),
        body: Center(
          child: Column(
            children: [
              TextField(
                onChanged: (text) {
                  var rv = double.tryParse(text);
                  if (rv != null) {
                    setState(() {
                      _numberFrom = rv;
                    });
                  }
                },
              ),
              DropdownButton<String>(
                  items: fruits.map((String value) {
                    return DropdownMenuItem<String>(
                      value: value,
                      child: Text(value),
                    );
                  }).toList(),
                  onChanged: (value) {}),
              Text((_numberFrom == null) ? '' : _numberFrom.toString())
            ],
          ),
        ),
      ),
    );
  }
}
```

5. Let's create a list of strings that will contain all the measures that we want to deal with. At the beginning of the State class, let's add the following code after fruits list declaration:

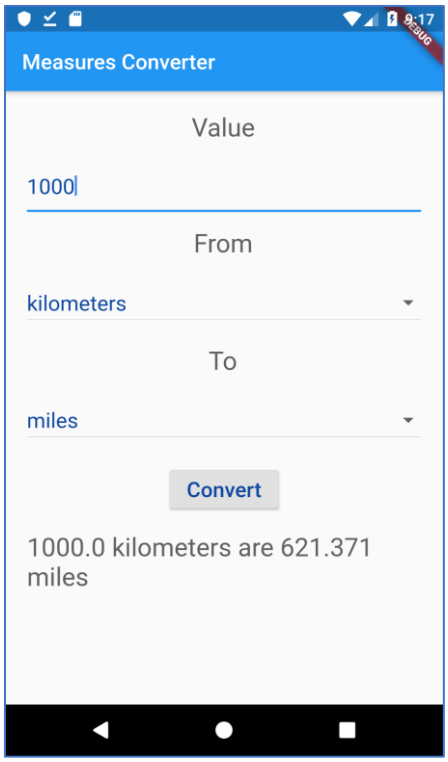
```
final List<String> _measures = [
  'meters',
  'kilometers',
  'grams',
  'kilograms',
  'feet',
  'miles',
  'pounds (lbs)',
  'ounces',
];
```



Complete the application

Use the following UI to complete this application

To do this, you can refer to the 2nd chapter of the book “Flutter Projects”


9:17
DEBUG
Measures Converter
Value
1000
From
kilometers
To
miles
Convert
1000.0 kilometers are 621.371 miles