Practical No. 11

Title: Introduction to Flutter

Aim: Create an application to demonstrate Flutter

Introduction

In general, developing a mobile application is a complex and challenging task. There are many frameworks available to develop a mobile application. Android provides a native framework based on Java language and iOS provides a native framework based on Objective-C / Swift language.

However, to develop an application supporting both the OSs, we need to code in two different languages using two different frameworks. To help overcome this complexity, there exists mobile frameworks supporting both OS. These frameworks range from simple HTML based hybrid mobile application framework (which uses HTML for User Interface and JavaScript for application logic) to complex language specific framework (which do the heavy lifting of converting code to native code). Irrespective of their simplicity or complexity, these frameworks always have many disadvantages, one of the main drawback being their slow performance.

In this scenario, Flutter – a simple and high performance framework based on Dart language, provides high performance by rendering the UI directly in the operating system's canvas rather than through native framework.

Flutter also offers many ready to use widgets (UI) to create a modern application. These widgets are optimized for mobile environment and designing the application using widgets is as simple as designing HTML.

To be specific, Flutter application is itself a widget. Flutter widgets also supports animations and gestures. The application logic is based on reactive programming. Widget may optionally have a state. By changing the state of the widget, Flutter will automatically (reactive programming) compare the widget's state (old and new) and render the widget with only the necessary changes instead of re-rendering the whole widget.

We shall discuss the complete architecture in the coming chapters.

Features of Flutter

Flutter framework offers the following features to developers –

- Modern and reactive framework.
- Uses Dart programming language and it is very easy to learn.
- Fast development.
- Beautiful and fluid user interfaces.
- Huge widget catalog.
- Runs same UI for multiple platforms.
- High performance application.

Advantages of Flutter

Flutter comes with beautiful and customizable widgets for high performance and outstanding mobile application. It fulfills all the custom needs and requirements. Besides these, Flutter offers many more advantages as mentioned below –

- Dart has a large repository of software packages which lets you to extend the capabilities of your application.
- Developers need to write just a single code base for both applications (both Android and iOS platforms). *Flutter* may to be extended to other platform as well in the future.
- Flutter needs lesser testing. Because of its single code base, it is sufficient if we write automated tests once for both the platforms.
- Flutter's simplicity makes it a good candidate for fast development. Its customization capability and extendibility makes it even more powerful.
- With Flutter, developers has full control over the widgets and its layout.
- Flutter offers great developer tools, with amazing hot reload.

Disadvantages of Flutter

Despite its many advantages, flutter has the following drawbacks in it –

- Since it is coded in Dart language, a developer needs to learn new language (though it is easy to learn).
- Modern framework tries to separate logic and UI as much as possible but, in Flutter, user interface and logic is intermixed. We can overcome this using smart coding and using high level module to separate user interface and logic.
- Flutter is yet another framework to create mobile application. Developers are having a hard time in choosing the right development tools in hugely populated segment.

Exercise - Create android application to demonstrate Flutter

Implementation:
Program:

Output:

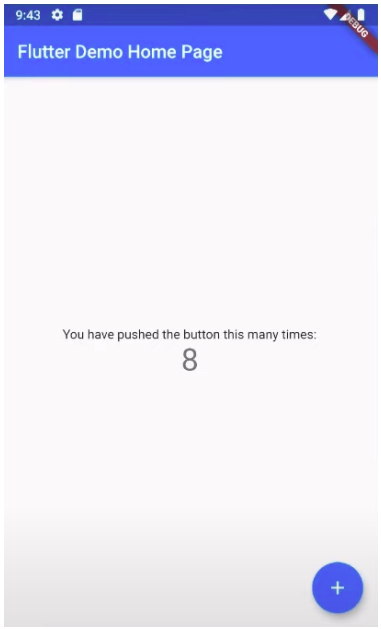
main.dart

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

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

class MyApp extends StatelessWidget {
 const MyApp({super.key});

 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       // This is the theme of your application.
       //
       // Try running your application with "flutter run". You'll see the
       // application has a blue toolbar. Then, without quitting the app,
try
       // changing the primarySwatch below to Colors.green and then invoke
       // "hot reload" (press "r" in the console where you ran "flutter
run",
       // or simply save your changes to "hot reload" in a Flutter IDE).
       // Notice that the counter didn't reset back to zero; the
application
       // is not restarted.
       primarySwatch: Colors.blue,
     ),
     home: const MyHomePage(title: 'Flutter Demo Home Page'),
   );
 }
}

class MyHomePage extends StatefulWidget {
 const MyHomePage({super.key, required this.title});

 // This widget is the home page of your application. It is stateful,
meaning
 // that it has a State object (defined below) that contains fields that
affect
 // how it looks.

 // This class is the configuration for the state. It holds the values (in
this
```

```
 // case the title) provided by the parent (in this case the App widget)
and
 // used by the build method of the State. Fields in a Widget subclass are
 // always marked "final".

 final String title;

 @override
 State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
   setState(() {
     // This call to setState tells the Flutter framework that something
has
     // changed in this State, which causes it to rerun the build method
below
     // so that the display can reflect the updated values. If we changed
     // _counter without calling setState(), then the build method would
not be
     // called again, and so nothing would appear to happen.
     _counter++;
   });
 }

 @override
 Widget build(BuildContext context) {
   // This method is rerun every time setState is called, for instance as
done
   // by the _incrementCounter method above.
   //
   // The Flutter framework has been optimized to make rerunning build
methods
   // fast, so that you can just rebuild anything that needs updating
rather
   // than having to individually change instances of widgets.
   return Scaffold(
     appBar: AppBar(
       // Here we take the value from the MyHomePage object that was
created by
       // the App.build method, and use it to set our appbar title.
       title: Text(widget.title),
     ),
     body: Center(
```

```
       // Center is a layout widget. It takes a single child and positions
it
       // in the middle of the parent.
       child: Column(
         // Column is also a layout widget. It takes a list of children
and
         // arranges them vertically. By default, it sizes itself to fit
its
         // children horizontally, and tries to be as tall as its parent.
         //
         // Invoke "debug painting" (press "p" in the console, choose the
         // "Toggle Debug Paint" action from the Flutter Inspector in
Android
         // Studio, or the "Toggle Debug Paint" command in Visual Studio
Code)
         // to see the wireframe for each widget.
         //
         // Column has various properties to control how it sizes itself
and
         // how it positions its children. Here we use mainAxisAlignment
to
         // center the children vertically; the main axis here is the
vertical
         // axis because Columns are vertical (the cross axis would be
         // horizontal).
         mainAxisAlignment: MainAxisAlignment.center,
         children: <Widget>[
           const Text(
             'You have pushed the button this many times:',
           ),
           Text(
             '$_counter',
             style: Theme.of(context).textTheme.headline4,
           ),
         ],
       ),
     ),
     floatingActionButton: FloatingActionButton(
       onPressed: _incrementCounter,
       tooltip: 'Increment',
       child: const Icon(Icons.add),
     ), // This trailing comma makes auto-formatting nicer for build
methods.
   );
 }
}
```

Output:


9:43
Flutter Demo Home Page
DEBUG
You have pushed the button this many times:
8
+