1. Widgets

Everything is a Widget in Flutter.

- Key Question:
  - How can Flutter manage components in a GUI for applications in a cross-platform way?

Short answers:

- ⚡ Declarative, modern, scalable pattern (similar to React)
- UI = Stateless + Stateful parts
  - `StatelessWidget` ↔️ Functional Component
  - `StatefulWidget` ↔️ Hooks / Class Component
- 🌀 **State change = redraw**

- Sub Questions: What is State in Flutter?
  - How do I make `Stateless and stateful` Flutter apps?
  - What are `MaterialDesign` and `Scaffolding`?

The Start

- In this section, we introduce Flutter.
    - We read the simplest Flutter program.
    - We understand Dart and its relationship to Java/JavaScript.

simplest.dart

- The simplest Dart program contains only a couple of lines of code.
- The simplest1.dart and simplest2.dart programs show the power of Flutter that can support most platforms with the same source.``

simplest1.dart: The Simplest Flutter Apps

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(home: Text('Hello, ASE 456 Students')));
}
```

- We need only five lines of code!
- Use `dartpad.dev` to run this code.

simplest2.dart: Let's make it more readable

```
import 'package:flutter/material.dart';
void main() {
  runApp(
    MaterialApp(
      home: Text('Hello'), // <-keep trailing commas
    ), // <- Dart Convention
  );
}
```

- In Flutter, we keep trailing commas for auto formatting.

The Material Design by Google

- The `flutter/material.dart` instructs Flutter to use the Material design (Google style).
- We can choose Cupertino (iOS) or Fluent (Microsoft) for other OS GUI styles.

Declarative Syntax

```
// Instruct Flutter to make a Text widget.
// Flutter decides where to put the widget.
MaterialApp(
  home: Text('Hello, ASE 456 Students'),
),
```

- Flutter uses declarative syntax.
- Modern Android and iOS follow this style of GUI programming.

Short Introduction to Dart/Flutter

- We are going to discuss Dart in detail in later topics.
- In this sub-section, we discuss Dart only enough to understand the example code.

Dart vs. Java and JavaScript

```
# dart
import "..."
void main() {...}
# Java
import ...
public static void main() {...}
```

- Flutter is written in Dart.
- Dart is similar to Java, but simple.

```
//java
import java.util.ArrayList;

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

- In Java, we use `import` to use libraries.
- Java uses `.` to specify package and directory information; Flutter uses `:` and `/`.

Dart vs. JavaScript

- Dart inventors are JavaScript experts who designed the JavaScript async system.
- Dart is similar to JavaScript/TypeScript.

```
// JavaScript
import { MaterialApp } from '@package/flutter/material';
// Flutter
import 'package:flutter/material.dart';
```

Dart as OOP & FP Language

- Dart is an OOP language
  - OOP Polymorphism and Inheritance are widely used.
- Dart is an FP language
  - Higher-level functions and Lambda expressions are widely used.

Dictionary as Argument

```
MaterialApp(home: Text('Hello, ASE 456 Students'),),
```

- Dart can use a dictionary for arguments.
- This example uses a key (home) and value (Text) `home: Text(...)` to give an argument to the MaterialApp() function.

main()

```
void main() {
  ...
  runApp(...)
}
```

- `void main()` is the starting point of a program.
- It is the same as Java/C/C++.

runApp()

```
runApp(
  MaterialApp(...);
);
```

- In this code, the runApp() function starts the Material design GUI app.
- In Flutter, we don't use the `new` operator.

Using arrow function (Lambda expression) =>

```
void main() => runApp(MaterialApp(home:MyApp()));

void main() {
  return runApp(MaterialApp(home:MyApp()));
}
```

- The two code examples are the same.
- When we have only one statement in a method, we can use `=>`.

Stateless

- In this section, we discuss the Stateless widget.
- The Stateless widget cannot do a lot of things per se, but it serves as the central widget of the Flutter application.

Stateless1.dart

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

void main() => runApp(home:MyApp());
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text('Hello, ASE 456 Students');
  }
}
```

Stateless Widget

- Stateless means `no changes` in the Widget.
- Anything that is not changed in a GUI app can be regarded as Stateless.
- Stateless is efficient because we have to draw the GUI only once.
- To make a stateless widget, a class

No changes -> Stateless

- This is a "picture" of a window.
- This picture window has nothing to update, so it's stateless.

The build() Method

```
class MyApp extends StatelessWidget {
   ...
   Widget build(BuildContext context) {...}
}
```

- In Flutter, the build() method contains the GUI components.
- Flutter will call the `build` method anytime to draw components on the screen.

Drawing Stateless Widget

```
Widget build(BuildContext context) {
  return Text('Hello, ASE 456 Students');
}
```

- In this example, we have only one component.
- The Text widget displays a string: no update in GUI needed.

Stateful

- In this section, we discuss the Stateful widget.
- The Stateful widget can be regarded as a placeholder widget, and it needs another widget to store GUI variables: the State widget.

stateful1.dart: State<T> and Stateful Widgets

```dart
// State<T> Widget
class _MyAppState extends State<MyApp> {
  @override
  Widget build() {...}
}
// Stateful Widget
class MyApp extends StatefulWidget {
  @override
  State<MyApp> createState() => _MyAppState();
}
```

Stateful: the Idea

- The real window has states: open or closed.
- When we have a state in a GUI, it means the GUI should be updated.

StatefulWidget for States

- To describe states in Flutter, we use the Stateful Widget.
- The Stateful widget pairs with the State<T> Widget.
    - The T in State<T> is the Stateful widget type.
- Any GUI variables are stored in the State<T> widget.

State<T> Widget - build()

```
class _MyAppState extends State<MyApp> {
  Widget build(BuildContext context) {
    return Text('Hello, ASE 456 Students');
  }
```

- The State<T> Widget has the `build()` method.
- This build method is called anytime it needs to redraw itself.

Stateful Widget - createState()

```
class MyApp extends StatefulWidget {
  @override
  State<MyApp> createState() => _MyAppState();
}
```

- We need to make a Stateful widget to connect to the State widget through the `createState()` method.
- The createState method returns the State<MyApp>.

stateful2.dart: GUI Variables

- When we need GUI variables, we keep them in the State Widgets.

```
class _MyState extends State<MyStateful> {
  String _str = "Hello"; // Variables used in GUI (GUI variables)

  @override
  Widget build(BuildContext context) {
    return Text(_str);
  }
}
```

setState() to trigger the build()

```
// Notify Flutter Widgets to call build()
// with the changed GUI variables
setState(() {_counter++;});
```

- To update the GUI display, we invoke `setState()` to trigger the `build()` function with the changed GUI variables.

MaterialApp (stateful3.dart)

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp( // <- We build MaterialApp Stateful Widget
      home: MyStateful(),
    );
  }
}
```

- In this example, we use a Stateful Widget (MyStageful) as an argument to the MaterialApp class.

Decorator Design Pattern

- This is also called the `decorator design pattern`.
- We call this technique `Dependency Injection` as we specify what object is decorated by the MaterialApp.

Typical Stateful Flutter Program

```
MaterialApp(home: MyStateful(),) // Google Style
Cuppertino(home: MyStateful(),) // iOS Style
```

- In this `decorator` design pattern, we decorate the MyStateful widget with the MaterialApp or Cuppertino style.

The two Placeholders

```
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {... build(MaterialApp());}
class MyStateful extends StatefulWidget {... getState();}
class _MyState extends State<MyStateful> { ... build();}
```

- The Stateless widget is a placeholder for the Stateful widget.
- The Stateful widget is a placeholder for the State class.

The Case of No Placeholders

```
void main() => runApp(MaterialApp(home:MyApp()));

class MyApp extends StatefulWidget {
  const MyApp();
  @override
  State<MyApp> createState() => _MyAppState();
}
```

- For simple apps, we can use a Stateful Widget without the Stateless Widget placeholder.

Summary

```dart
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  Widget build(BuildContext context) {
      return MaterialApp(home: MyStateful(),);
  }
}
```

- Stateless MyApp as a placeholder.
- It has the build function to redraw from the `setState()` function.

```
class MyStateful extends StatefulWidget {
  State<MyStateful> createState() => _MyState();
}
class _MyState extends State<MyStateful> {
  Widget build(BuildContext context) { GUI }
}
```

- Stateful and State<T> Widget
- T is the type of Stateful Widget: MyStateful in this example
- The `createState()` returns the State<T> object.