Understanding Flutter

Key Points	Notes

1. **How to Create Flutter Project?**
    a. There are two (2) ways to create Flutter project:
        i. Flutter CLI
        ii. Android Studio

    b. Flutter CLI requires Flutter SDK, which you can download and install from https://flutter.dev/docs/get-started/install

    c. Android Studio requires Flutter plugin installation, which you can configure under *Android Studio* > *Configure* > *Plugins* > *Marketplace* > *search for flutter* > *Click Install button*.

    d. To create Flutter and preview project:

Flutter CLI (Command Prompt)

```
> flutter create my_app
> cd my_app
> flutter pub get
> flutter run

* Make sure iOS simulator is up and running
before executing flutter run command (iOS
only).
```

Android Studio

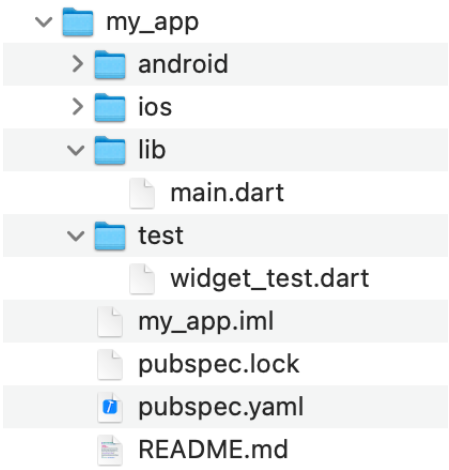
```
1. Open Android Studio
2. Select New Flutter Project
3. Select Device to Run
4. Click Run button (Green Play Button)

* Make sure Device is defined in AVD and
listed in the Android Studio Devices.
```

Summary

Key Points	Notes

2. Flutter Project Structure

a. A created project will contain the following project structure or directory:



Folder/File	Description
android	Contain converted **Kotlin** native source code, which you can continue working with Android platform.
ios	Contain converted **Swift** native source code, which you can continue working with iOS platform.
lib	**Dart** source code for your flutter application.
test	Writing test or instrumented test in Dart.
my_app.iml	IDE file created for Android Studio.
pubspec.lock	Flutter generated files to be used by the application.
pubspec.yaml	
README.md	A markdown file used in version control git and provides information about the application.

Summary

Key Points	Notes

	3. Anatomy of Flutter Application
	a. Basic Flutter Application with text and minimal styling.
	
	(1) `import 'package:flutter/material.dart';`
	
	(2) `void main() {`
	(3) `runApp(`
	(4) `Center(`
	(5) `child: Text(`
	(6) `'Hello, world!',`
	(7) `textDirection: TextDirection.ltr,`
	`),`
	`),`
	`);`
	`}`
	
	b. Code explanation:
	
	(1) Import material design library for styling and UI. You can also import other Dart libraries needed for your Flutter application. You can also use iOS specific UI, which is *cupertino.dart*.
	(2) Application execution/entry point.
	(3) Top level widget method.
	(4) Positioning and alignment widget.
	(5) Text widget.
	(6) Visible text.
	(7) Text widget property, setting direction of the text.

Summary	

Key Points	Notes

c. Basic Flutter Application with scaffolding of Material Design UI.

```
(1) import 'package:flutter/material.dart';

(2) void main() => runApp(My_App());

(3) class My_App extends StatelessWidget {
(4)  @override
(5)  Widget build(BuildContext context) {
(6)    return MaterialApp(
(7)      title: 'Welcome to Flutter',
(8)      home: Scaffold(
(9)        appBar: AppBar(
(10)          title: Text('Welcome to Flutter'),
         ),
(11)         body: Center(
(12)          child: Text('Hello world'),
        ),
      ),
    );
  }
}
```

d. Code Explanation:

(1) Import material design library.
(2) Application execution/entry point.
(3) Create a stateless widget and give a name My_App.
(4) Override annotation, which tell flutter the below method will be overridden.
(5) Override Flutter abstract build method with code implementation.
(6) Return the MaterialApp widget (UI) with the given properties for display.
(7) Set property title to MaterialApp widget.
(8) Set front (home) layout or scaffold of the MaterialApp widget.
(9) Set appBar property
(10) Set title property for appBar.
(11) & (12) Set body widget with text property for appBar.

Summary

Key Points	Notes

	4. The `build` method
	a. Every widget created in Flutter must have a build method and return another widget.
	
	b. Example:
	
	`Widget build(BuildContext context) {`
	`  return MaterialApp(`
	`  …`
	`  );`
	
	5. The `new` and `const` Constructors in Flutter
	a. Many built-in widgets contain `new` and `const` constructors. `const` constructor perform better than `new` constructor. Thus, you are recommended to use `const` as much as you can in your Flutter application development.
	
	b. Flutter make it easier for developer to choose whether to use `new` or `const` constructor by just omitting both in the code. The framework will determine which is the best fit for your constructor and always choose `const` whenever it can.
	
	c. Example:
	
	`Widget build(BuildContext context) {`
	`  return Button(`
	`  child: Text("Submit"),`
	`  );`
	`}`
	
	`// compared to`
	`Widget build(BuildContext context) {`
	`  return new Button(`
	`  child: new Text("Submit"),`
	`  );`
	`}`

Summary

Key Points	Notes

6. Hot Reload

a. Dart uses both ahead-of-time (AOT) and just-in-time (JIT) compilers. This allows developers to compile and run the code as it needs to. In other words, developers can develop and recompile code quickly in the development.

b. Hot reload can be accessed in Android Studio and Visual Studio code by pressing ***Ctrl-S*** for windows and ***Cmd-S*** for macOS.

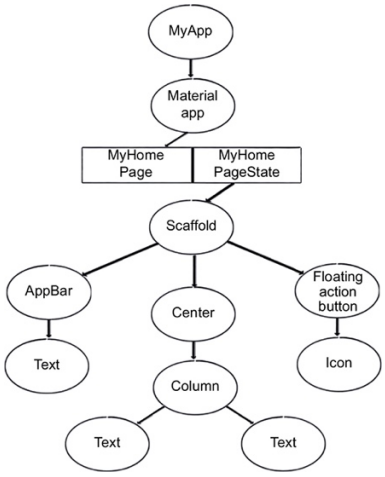
c. For Flutter CLI you can press ***r*** button when you run your app on the simulator (iOS) or emulator (Android).

7. Widget Tree

a. Flutter application is made of many widgets and a widget represent a node in a tree structure, which is similar to **Document Object Model** (DOM) of a web browser.

b. Every time you use `build` method, you will add a new node to the widget tree of your application. The node is connected similar to parent-child relationship.

c. Example:



Widget Tree (Windmill, 2020)

Summary

Key Points	Notes

	d. To form a widget tree is to add another widget in the existing widget or child(ren) in another child(ren).
	
	e. Example:
	
	`return Container(`
	`  child: Padding(`
	`  padding: EdgeInsets.all(8.0),`
	`  child: Text("Padded Text")`
	`  ),`
	`);`
	
	8. Stateless Widgets
	a. A `StatelessWidget` do not contain internal state (data) that changes during the lifetime or lifecycle of the widget. In other words, it cannot update itself.
	
	b. Example:
	
	`class SubmitButton`
	` extends StatelessWidget {`
	`  Widget build(context) {`
	`  return Button(`
	`  child: Text('Submit'),`
	`  );`
	`  }`
	`}`
	
	c. Every stateless widget represent the layout/UI of Flutter application, which can be customized at the property level but does **NOT** carry data with it. However, you can create a reusable stateless widget which the data can be passed from stateful widget or other sources.
	
	d. Stateless widget will be removed from the widget tree permanently when it is destroyed from the widget life cycle (Flutter application is closed).

Summary	

Key Points	Notes

9. Stateful Widgets

a. A `StatefulWidget` contains internal data and would be able to update the data. All stateful widgets have state objects and must be defined using `CreateState` method before a widget can be built.

b. Example:

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    // ..
  }
}
```

c. Every stateful widget does **NOT** contain `build` method but it has an associated state object which does contain `build` method. In other words, stateful widget is similar to stateless widget but it contains state object.

10. setState

a. A `setState` is method to change the state object. The method `createState` is to create a state object. The `build` method is to create a widget and add to the widget tree.

b. Example:

```
void _incrementCounter() {
  setState(() {
    _counter++;
  });
}
```

Summary

Key Points	Notes

11. BuildContext

a. `BuildContext` determine the location of the widget in the tree. Every time build is called a widget reference is created to determine where it is located in the widget tree.

b. Example:

```
Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
            'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style:Theme.of(context).textTheme.headline4,
            ),
          ],
        ),
      ),
    );
  }
```

12. Private Values

a. Any private variables or functions declared in a class can be accessed only within the class. It is represented by underscore.

b. Example:

```
void _incrementCounter() {
  setState(() {
    _counter++;
  });
}
```

Summary

Key Points	Notes

	13. Composition Over Inheritance
	a. In Flutter composition is favour over inheritance for simplicity. Composition is a way to combine simple objects to create a complex objects.
	
	b. In programming composition is where a class has properties which themselves are instances of other classes.
	
	c. Example:
	
	`class Computer{`
	`  final CPU cpu;`
	`  final RAM ram;`
	
	`  Computer(this.cpu, this.ram);`
	`}`
	
	`class CPU{`
	`  final String make;`
	`  final double clockSpeed;`
	`  final String socketType;`
	
	`  CPU(this.make, this.clockSpeed, this.socketType);`
	`}`
	
	`class RAM{`
	`  final String type;`
	`  final int speed;`
	`  final int pins;`
	
	`  RAM(this.type, this.speed, this.pins);`
	`}`
	
	Composition in Flutter (Kinya, 2020)

Summary

Key Points	Notes
	14. References a. Windmill, E. (2020). Flutter in Action (1st Ed.). USA: Manning Publications. b. Flutter Official Documentation. Retrieved on 1 December 2020 from https://flutter.dev/docs c. Kinya, B. (2020). Composition in Flutter. Retrieved on 5 December 2020 from https://kinya.hashnode.dev/composition-in-flutter-ckepqbojt02g7kps1224cbrdg

Summary