**Course: 405-02: Mobile Application Development – 2**

**Unit-4: Introduction of Flutter:**

4.1 Fundamentals of Flutter:

**What is Flutter?**

In general, creating a mobile application is a very complex and challenging task. There are many frameworks available, which provide excellent features to develop mobile applications. For developing mobile apps, Android provides a native framework based on Java and Kotlin language, while iOS provides a framework based on Objective-C/Swift language. Thus, we need two different languages and frameworks to develop applications for both OS. Today, to overcome form this complexity, there are several frameworks have introduced that support both OS along with desktop apps. These types of the framework are known as **cross-platform** development tools.

The cross-platform development framework has the ability to write one code and can deploy on the various platform (Android, iOS, and Desktop). It saves a lot of time and development efforts of developers. There are several tools available for cross-platform development, including web-based tools, such as Ionic from Drifty Co. in 2013, Phonegap from Adobe, Xamarin from Microsoft, and React Native form Facebook. Each of these frameworks has varying degrees of success in the mobile industry. In recent, a new framework has introduced in the cross-platform development family named **Flutter** developed from Google.

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 an **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.

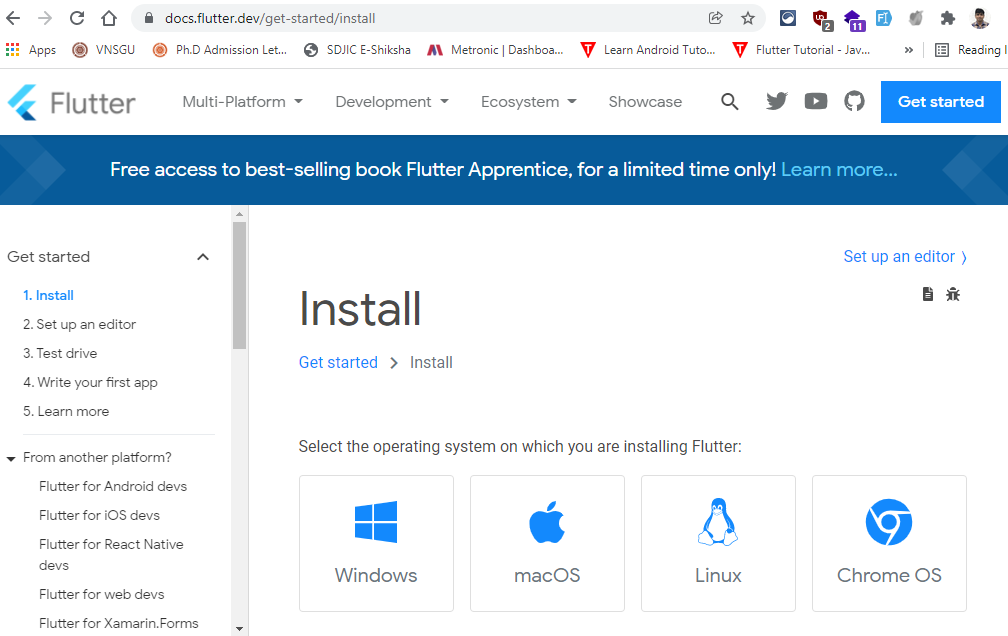
**What makes Flutter unique?**

Flutter is different from other frameworks because it neither uses **WebView** nor the **OEM** widgets that shipped with the device. Instead, it uses its own high-performance rendering engine to draw widgets. It also implements most of its systems such as animation, gesture, and widgets in Dart programing language that allows developers to read, change, replace, or remove things easily. It gives excellent control to the developers over the system.

4.1.1 Installation and Architecture of Flutter

Install the Flutter SDK

**Step 1:** Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website**(https://flutter.dev/)**, click on Get started button, you will get the following screen.



**Get the Flutter SDK**

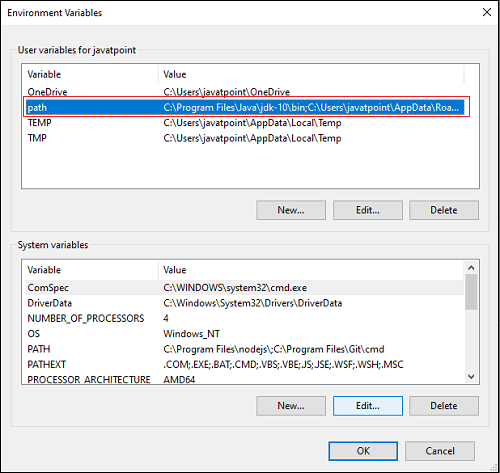
**Step 2:** Download the following installation bundle to get the latest stable release of the Flutter SDK:

*https://storage.googleapis.com/flutter\_infra\_release/releases/stable/windows/flutter\_windows\_2.8.1-stable.zip*

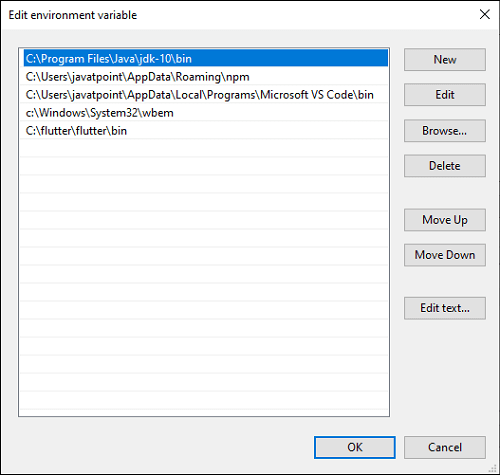
**Step 3:** Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\Users\<your-user-name>\Documents).

**Step 4:** To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

**Step 4.1:** Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.



**Step 4.2:** Now, select path -> click on edit. The following screen appears.

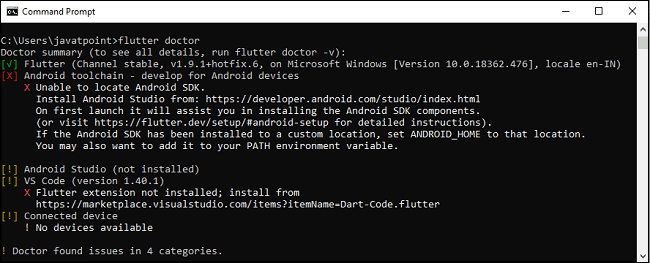


**Step 4.3:** In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.

**Step 5:** Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

$ flutter doctor

**Step 6:** When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

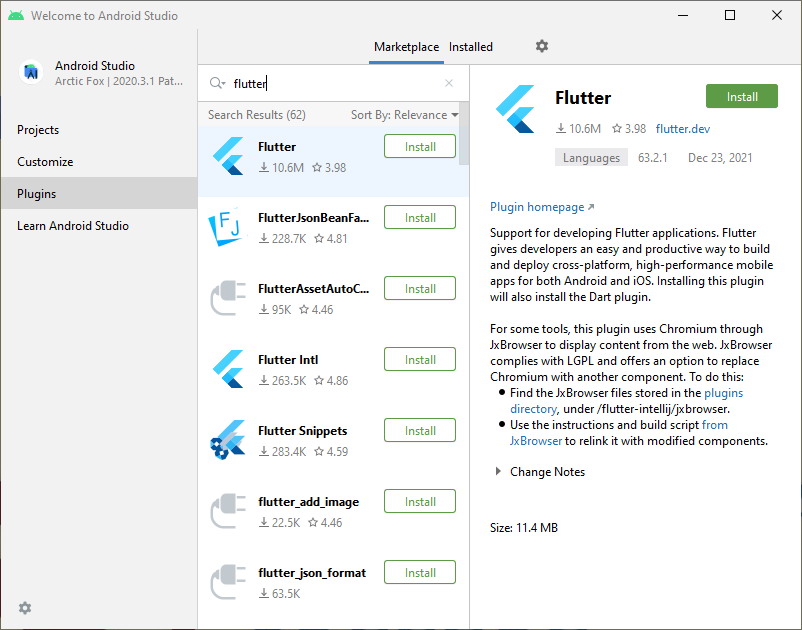


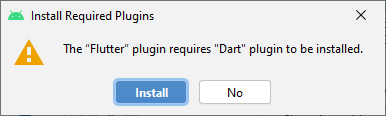
**Install Flutter/Dart Plugins for Android Studio**

Install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

Open the Android Studio and then go to File->Settings->Plugins.

Now search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.





**Set Environment Variables**

Create two System/User Environment Variable and set following values.

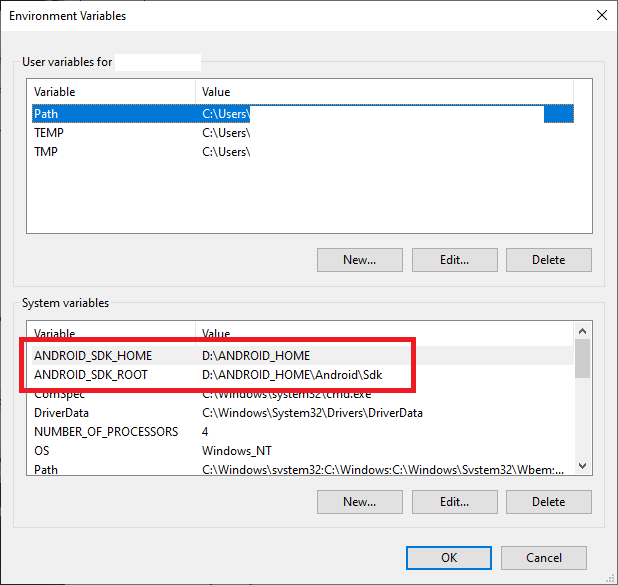
Variable Name: ANDROID\_SDK\_HOME

Value: C:\Users\{YOUR\_USER\_NAME}

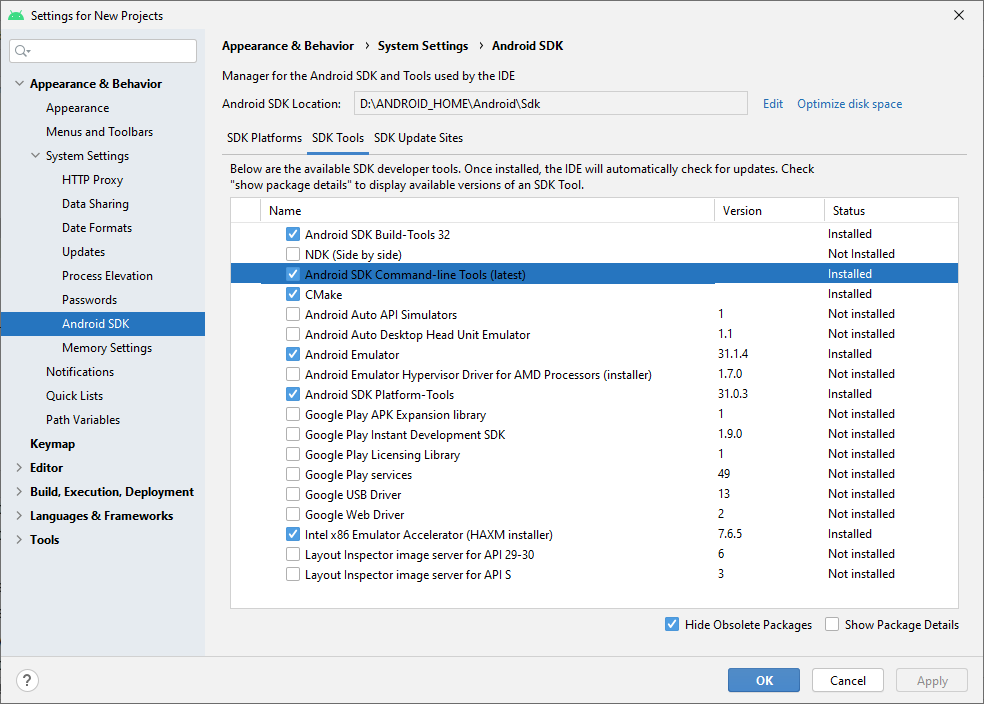
(Or provide any directory where you want to keep all the android libraries and Virtual devices)

Variable Name: ANDROID\_SDK\_ROOT

Value: C:\Users\{YOUR\_USER\_NAME}\AppData\Local\Android\Sdk



**Install Android SDK command line tools**

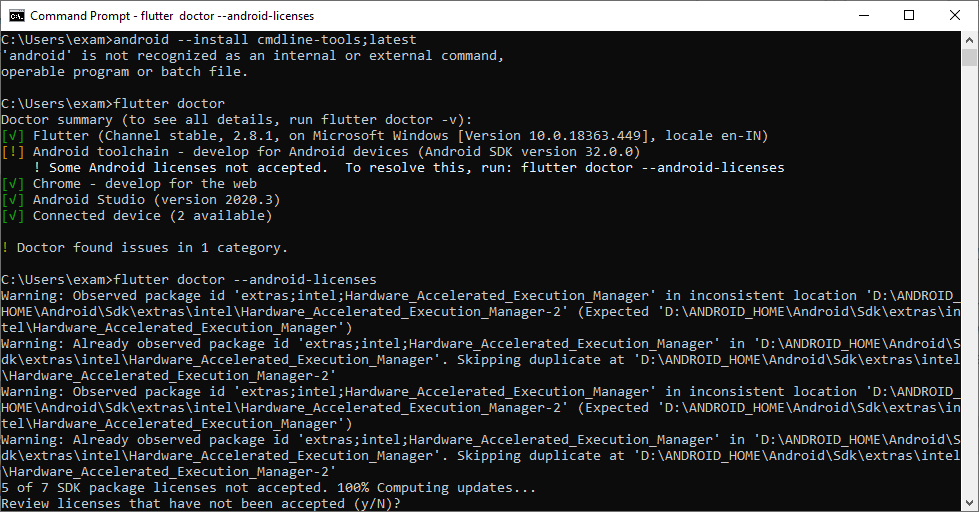


**Accept All Android Licenses:**

We need to accept all the android licenses to run the flutter applications built over Android SDK. To accept all the licenses, we need to run following command in the terminal window.

$ flutter doctor --android-licenses

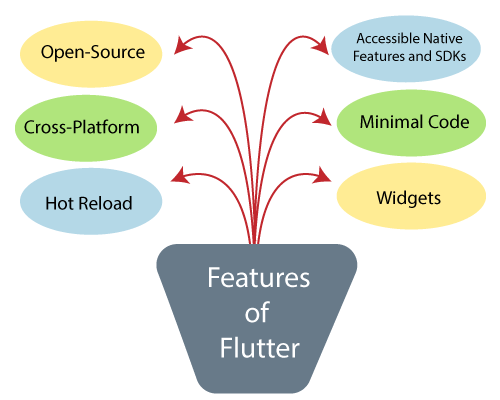
Read the licenses if you want and keep pressing “Y” to accept all the licenses.



Now we are ready to build our first Flutter Application using Android Studio and Android SDK.

4.1.2 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. Here, we are going to discuss its main features for developing the mobile framework.



**Open-Source:** Flutter is a free and open-source framework for developing mobile applications.

**Cross-platform:** This feature allows Flutter to write the code once, maintain, and can run on different platforms. It saves the time, effort, and money of the developers.

**Hot Reload:** Whenever the developer makes changes in the code, then these changes can be seen instantaneously with Hot Reload. It means the changes immediately visible in the app itself. It is a very handy feature, which allows the developer to fix the bugs instantly.

**Accessible Native Features and SDKs:** This feature allows the app development process easy and delightful through Flutter's native code, third-party integration, and platform APIs. Thus, we can easily access the SDKs on both platforms.

**Minimal code:** Flutter app is developed by Dart programming language, which uses JIT and AOT compilation to improve the overall start-up time, functioning and accelerates the performance. JIT enhances the development system and refreshes the UI without putting extra effort into building a new one.

**Widgets:** The Flutter framework offers widgets, which are capable of developing customizable specific designs. Most importantly, Flutter has two sets of widgets: Material Design and Cupertino widgets that help to provide a glitch-free experience on all platforms.

Advantage of Flutter

Flutter fulfills the custom needs and requirements for developing mobile applications. It also offers many advantages, which are listed below.

* It makes the app development process extremely fast because of the hot-reload feature. This feature allows us to change or update the code are reflected as soon as the alterations are made.
* It provides the smoother and seamless scrolling experiences of using the app without much hangs or cuts, which makes running applications faster in comparison to other mobile app development frameworks.
* Flutter reduces the time and efforts of testing. As we know, flutter apps are cross-platform so that testers do not always need to run the same set of tests on different platforms for the same app.
* It has an excellent user interface because it uses a design-centric widget, high-development tools, advanced APIs, and many more features.
* It is similar to a reactive framework where the developers do not need to update the UI content manually.
* It is suitable for MVP (Minimum Viable Product) apps because of its speedy development process and cross-platform nature.

Disadvantages of Flutter

We have seen earlier that the Flutter has many advantages, but it also contains some disadvantages, which are given below.

* The Flutter is a comparatively new language that needs continuous integration support through the maintenance of scripts.
* It provides very limited access to SDK libraries. It means a developer does not have a lot of functionalities to create a mobile application. Such types of functionalities need to be developed by the Flutter developer themselves.
* The Flutter apps do not support the browser. It only supports Android and iOS platforms.
* It uses Dart programming for coding, so a developer needs to learn new technologies. However, it is easy to learn for developers.

**History of Flutter**

Flutter is a free and open-source UI software development kit introduced by Google. It is used to build applications for Android, iOS, Windows, and the web. The first version of Flutter was announced in the year **2015** at the **Dart Developer Summit**. It was initially known as codename **"Sky"** and can run on the Android OS. After the announcement of Flutter, the first Flutter Alpha version (v-0.06) was released in May **2017**.

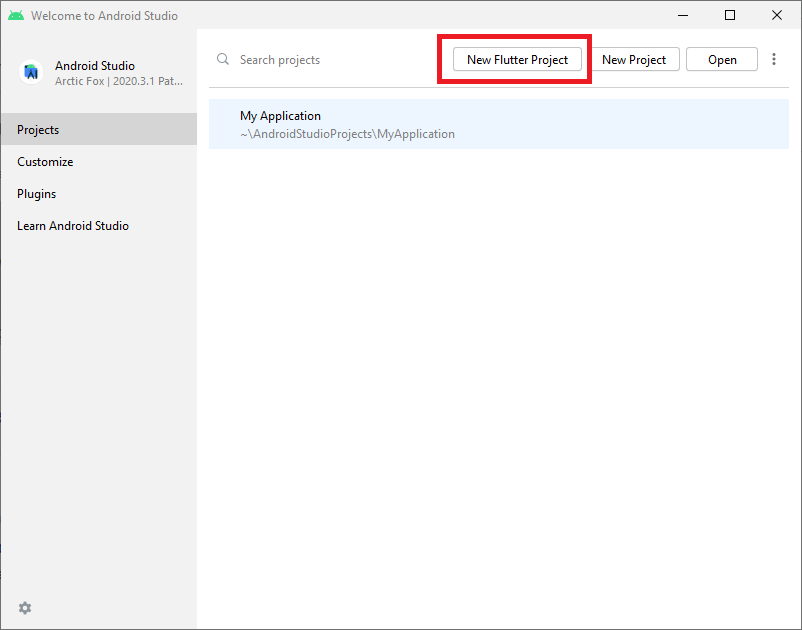
Later, during the keynote of Google Developer days in Shanghai, Google launched the second preview of Flutter in **September 2018** that was the last big release before Flutter 1.0 version. On **December 4, 2018**, the first stable version of the Flutter framework was released at the Flutter Live event, denoting Flutter 1.0. The current stable release of the framework is Flutter v1.9.1+hotfix.6 on October 24, 2019.

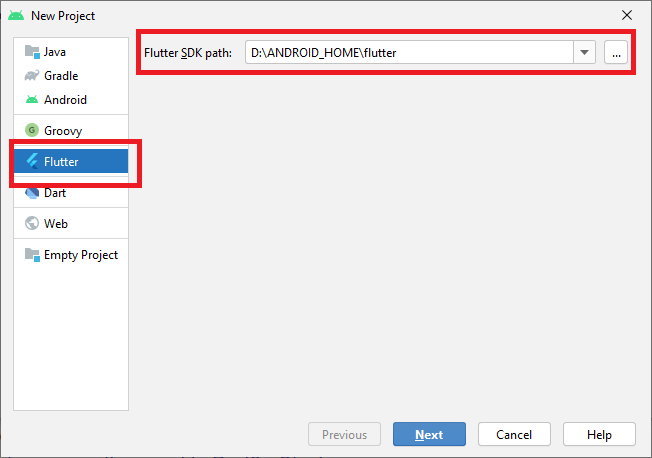
**4.1.3 Creating basic flutter project using Android Studio**

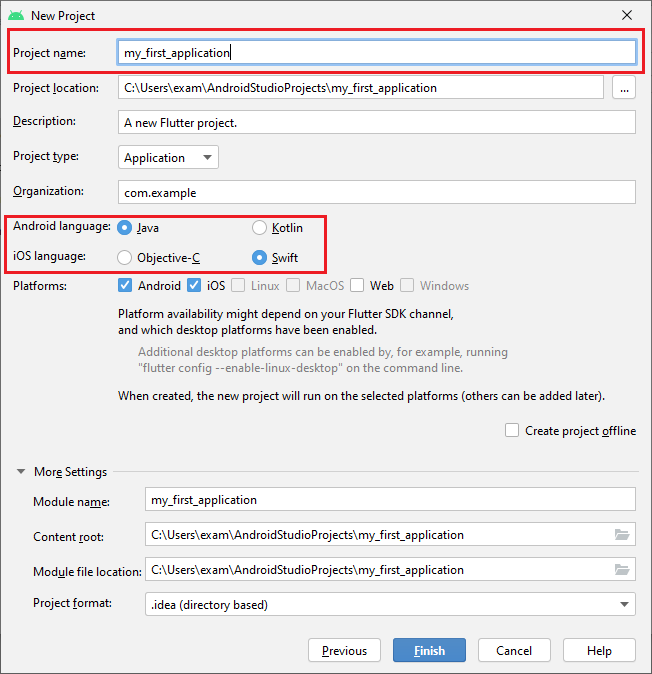
In this section, we are going to learn how to create a simple application in Android Studio to understand the basics of the Flutter application. To create Flutter application, do the following steps:

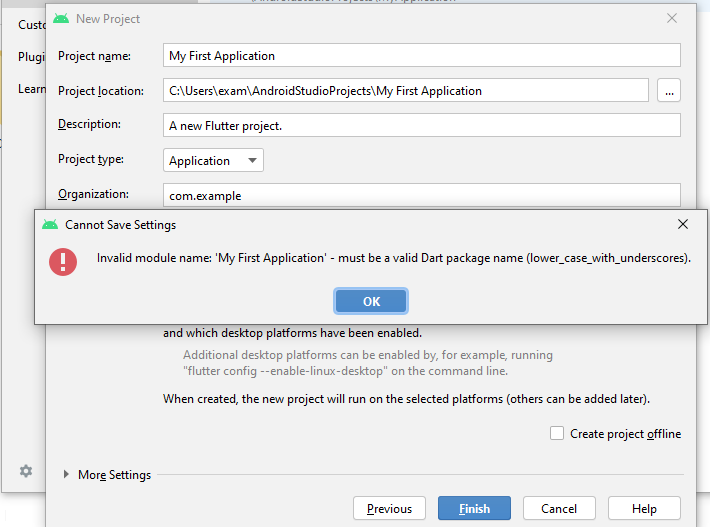
**Step 1:** Open the Android Studio.

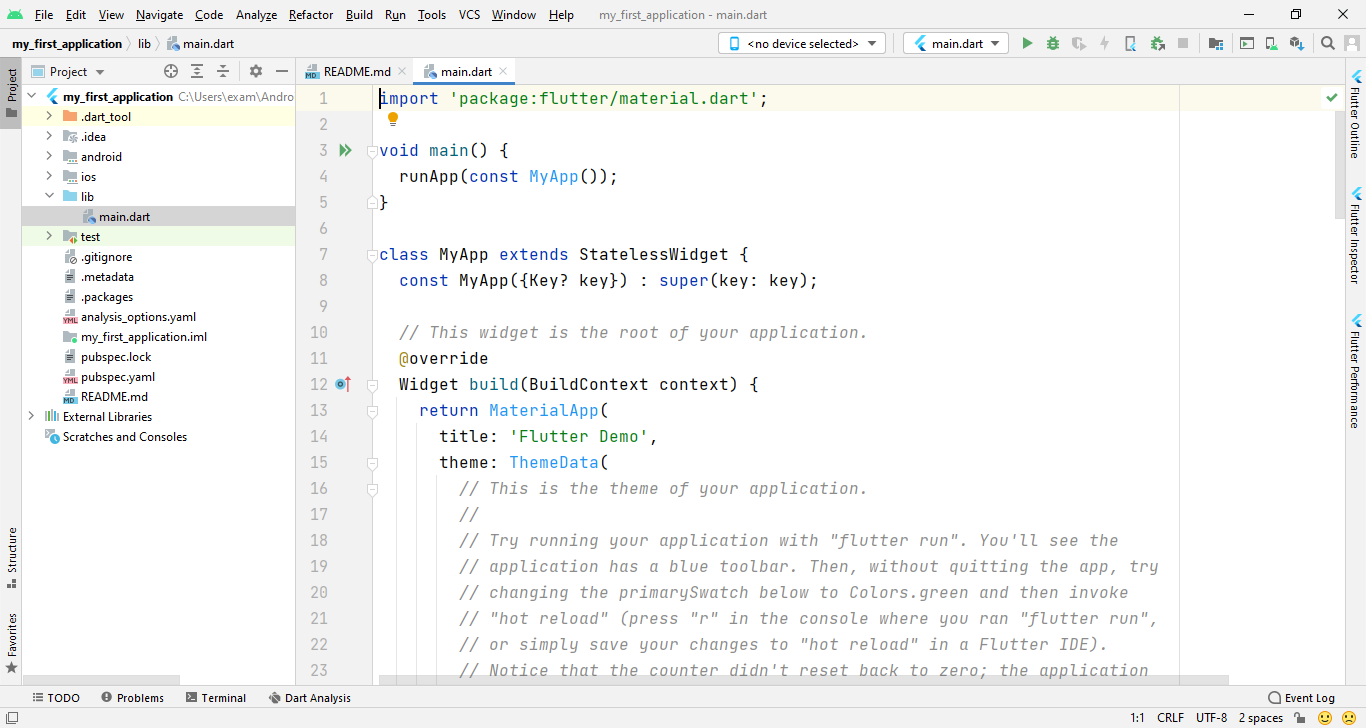
**Step 2:** Create the Flutter project. To create a project, go to File-> New->New Flutter Project. The following screen helps to understand it more clearly.

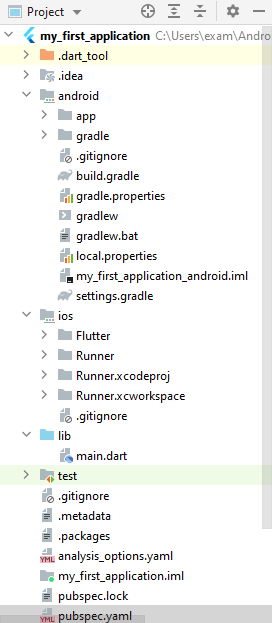












**.idea:** This folder is at the very top of the project structure, which holds the configuration for Android Studio. It doesn't matter because we are not going to work with Android Studio so that the content of this folder can be ignored.

**.android:** This folder holds a complete Android project and used when you build the Flutter application for Android. When the Flutter code is compiled into the native code, it will get injected into this Android project, so that the result is a native Android application. **For Example:** When you are using the Android emulator, this Android project is used to build the Android app, which further deployed to the Android Virtual Device.

**.ios:** This folder holds a complete Mac project and used when you build the Flutter application for iOS. It is similar to the android folder that is used when developing an app for Android. When the Flutter code is compiled into the native code, it will get injected into this iOS project, so that the result is a native iOS application. Building a Flutter application for iOS is only possible when you are working on macOS.

**.lib:** It is an essential folder, which stands for the library. It is a folder where we will do our 99 percent of project work. Inside the lib folder, we will find the Dart files which contain the code of our Flutter application. By default, this folder contains the file **main.dart**, which is the entry file of the Flutter application.

**.test:** This folder contains a Dart code, which is written for the Flutter application to perform the automated test when building the app. It won't be too important for us here.

We can also have some default files in the Flutter application. In 99.99 percent of cases, we don't touch these files manually. These files are:

**.gitignore:** It is a text file containing a list of files, file extensions, and folders that tells Git which files should be ignored in a project. Git is a version-control file for tracking changes in source code during software development Git.

**.metadata:** It is an auto-generated file by the flutter tools, which is used to track the properties of the Flutter project. This file performs the internal tasks, so you do not need to edit the content manually at any time.

**.packages:** It is an auto-generated file by the Flutter SDK, which is used to contain a list of dependencies for your Flutter project.

**flutter\_demoapp.iml:** It is always named according to the Flutter project's name that contains additional settings of the project. This file performs the internal tasks, which is managed by the Flutter SDK, so you do not need to edit the content manually at any time.

**pubspec.yaml:** It is the project's configuration file that will use a lot during working with the Flutter project. It allows you how your application works. This file contains:

* Project general settings such as name, description, and version of the project.
* Project dependencies.
* Project assets (e.g., images).

**pubspec.lock:** It is an auto-generated file based on the **.yaml** file. It holds more detail setup about all dependencies.

**README.md:** It is an auto-generated file that holds information about the project. We can edit this file if we want to share information with the developers.

Open the **main.dart** file and replace the code with the following code snippets.

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
5. **class** MyApp **extends** StatelessWidget {
6. // This widget is the root of your application.
7. @override
8. Widget build(BuildContext context) {
9. **return** MaterialApp(
10. title: 'Hello World Flutter Application',
11. theme: ThemeData(
12. // This is the theme of your application.
13. primarySwatch: Colors.blue,
14. ),
15. home: MyHomePage(title: 'Home page'),
16. );
17. }
18. }
19. **class** MyHomePage **extends** StatelessWidget {
20. MyHomePage({Key key, **this**.title}) : **super**(key: key);
21. // This widget is the home page of your application.
22. **final** String title;
24. @override
25. Widget build(BuildContext context) {
26. **return** Scaffold(
27. appBar: AppBar(
28. title: Text(**this**.title),
29. ),
30. body: Center(
31. child: Text('Hello World'),
32. ),
33. );
34. }
35. }

**Step 8:** Let us understand the above code snippet line by line.

* To start Flutter programming, you need first to import the Flutter package. Here, we have imported a **Material package**. This package allows you to create user interface according to the Material design guidelines specified by Android.
* The second line is an entry point of the Flutter applications similar to the main method in other programming languages. It calls the **runApp** function and pass it an object of **MyApp** The primary purpose of this function is to attach the given widget to the screen.
* Line 5 to 18 is a widget used for creating UI in the Flutter framework. Here, the **StatelessWidget** does not maintain any state of the widget. MyApp extends StatelessWidget that overrides its **build** The build method is used for creating a part of the UI of the application. In this block, the build method uses MaterialApp, a widget to create the root level UI of the application and contains three properties - title, theme, and home.
  1. **Title:** It is the title of the Flutter application.
  2. **Theme:** It is the theme of the widget. By default, it set the blue as the overall color of the application.
  3. **Home:** It is the inner UI of the application, which sets another widget (MyHomePage) for the application.
* Line 19 to 35, the **MyHomePage** is similar to MyApp, except it will return the **Scaffold** Scaffold widget is a top-level widget after the MaterialApp widget for creating the user interface. This widget contains two properties **appBar** and **body**. The appBar shows the header of the app, and body property shows the actual content of the application. Here, **AppBar** render the header of the application, **Center** widget is used to center the child widget, and **Text** is the final widget used to show the text content and displays in the center of the screen.

**Step 9:** Now, run the application. To do this, go to Run->Run main.dart, as shown in the below screen.

4.2 Flutter Widget:

4.2.1 Types of flutter widget:

4.2.1.1 Visible and Invisible

4.2.1.2 StatelessWidget, StatefulWidget

4.2.1.3 Single child widget and Multiple child widget

4.2.2 Visible widget (Constructor and Properties): Text, Image, Button, Icon

4.3.3 Invisible widget (Constructor and Properties): column, row, center, padding, scaffold, stack