الجمهورية العربية السورية
جامعة دمشق
كلية الهندسة المعلوماتية

Programming Languages

Lecture Six

Flutter – Installation & Basics

Prepared By

Eng. Rawan Koroni

Eng. Rimon Koroni

1- <u>Introduction</u>

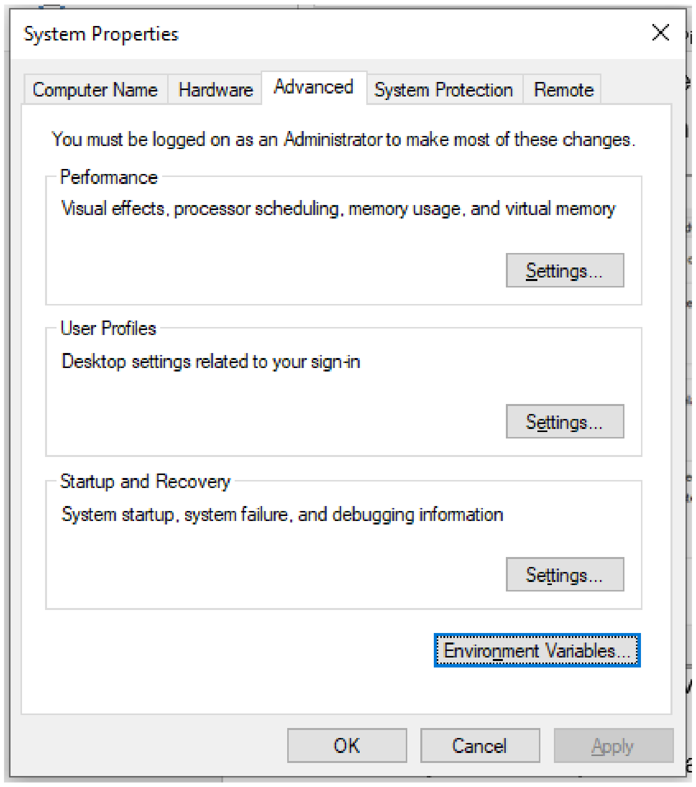
- In general, developing a mobile application is a complex and challenging task. There are many frameworks available to develop a mobile application. Android provides a native framework based on Java language and iOS provides a native framework based on Objective-C / Swift language.
- However, to develop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

2- Flutter Features

- 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.

3- Installation in Windows

- Go to URL: https://flutter.dev/docs/get-started/install/windows and download the latest Flutter SDK
- Unzip the zip archive in a folder, say C:\src\flutter\
- Update the system path to include flutter bin directory.
  - From system properties select advanced tab:


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

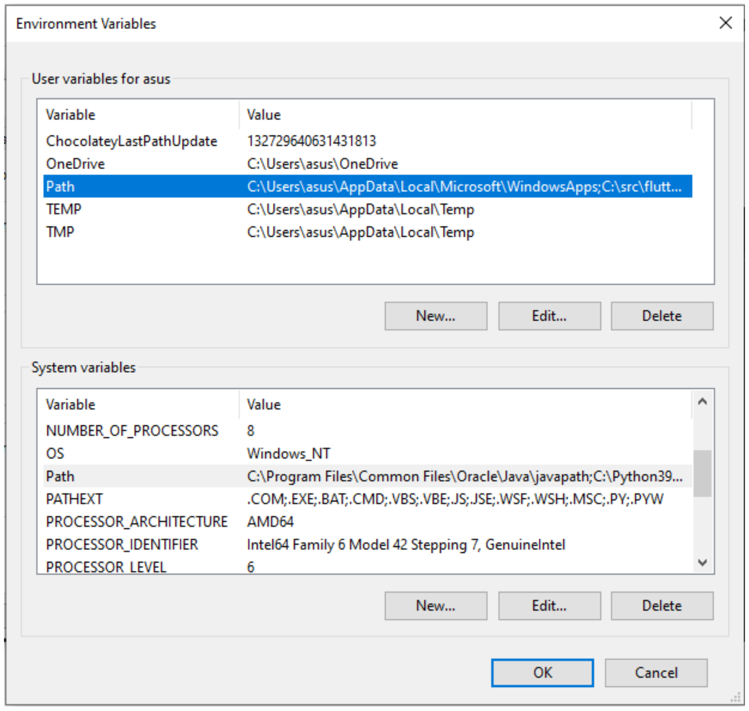
User Profiles
Desktop settings related to your sign-in
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK | Cancel | Apply

- Press on Environment Variables



Environment Variables

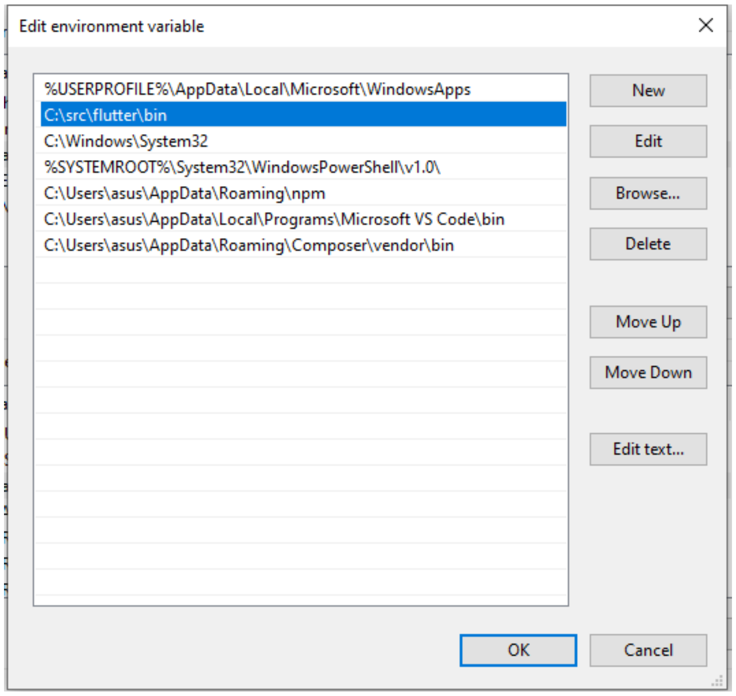
User variables for asus

Variable	Value
ChocolateyLastPathUpdate	132729640631431813
OneDrive	C:\Users\asus\OneDrive
Path	C:\Users\asus\AppData\Local\Microsoft\WindowsApps;C:\src\flutt...
TEMP	C:\Users\asus\AppData\Local\Temp
TMP	C:\Users\asus\AppData\Local\Temp

System variables

Variable	Value
NUMBER_OF_PROCESSORS	8
OS	Windows_NT
Path	C:\Program Files\Common Files\Oracle\Java\javapath;C:\Python39...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 42 Stepping 7, GenuineIntel
PROCESSOR_LEVEL	6

- Add C:\src\flutter\bin to Path



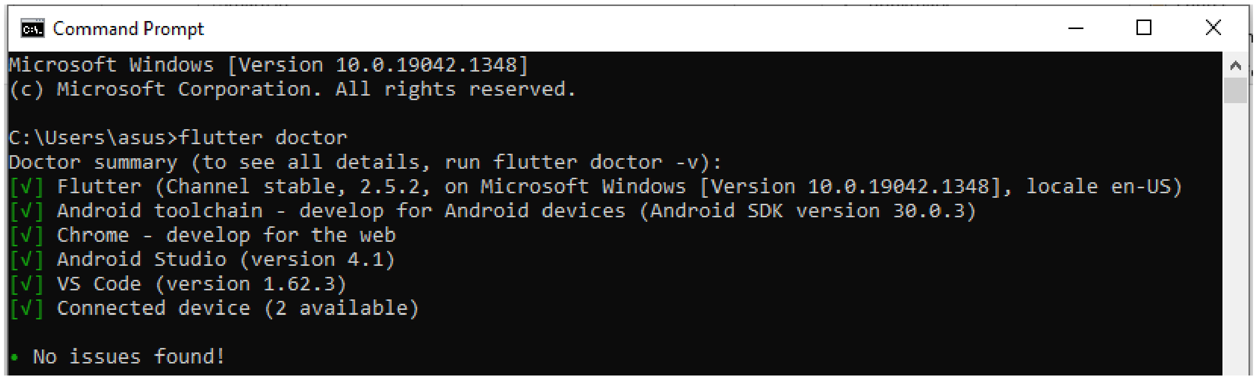
Edit environment variable

```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\src\flutter\bin
C:\Windows\System32
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Users\asus\AppData\Roaming\npm
C:\Users\asus\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\asus\AppData\Roaming\Composer\vendor\bin
```

- To check if flutter installed successfully type the following in the command prompt:

  flutter doctor

- Running the above command will analyze the system and show its report as shown below:



```
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\asus>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.5.2, on Microsoft Windows [Version 10.0.19042.1348], locale en-US)
[√] Android toolchain - develop for Android devices (Android SDK version 30.0.3)
[√] Chrome - develop for the web
[√] Android Studio (version 4.1)
[√] VS Code (version 1.62.3)
[√] Connected device (2 available)

• No issues found!
```

The report says that all development tools are available

- Install the latest Android Studio, if reported by flutter doctor.
  You can download it from https://developer.android.com/studio
- Start an android emulator or connect a real android device to the system.
- Install Flutter and Dart plugin for Android Studio. It provides startup template to create new Flutter application, an option to run and debug Flutter application in the Android studio itself …etc.
  - Open Android Studio.
  - Click File → Settings → Plugins.
  - Select the Flutter plugin and click Install.
  - Click Yes when prompted to install the Dart plugin.
  - Restart Android studio.

4- Flutter Using VS Code

You can also build apps with Flutter using any text editor combined with flutter command-line tools. However, we recommend using VS code because is a lightweight editor and easy to use

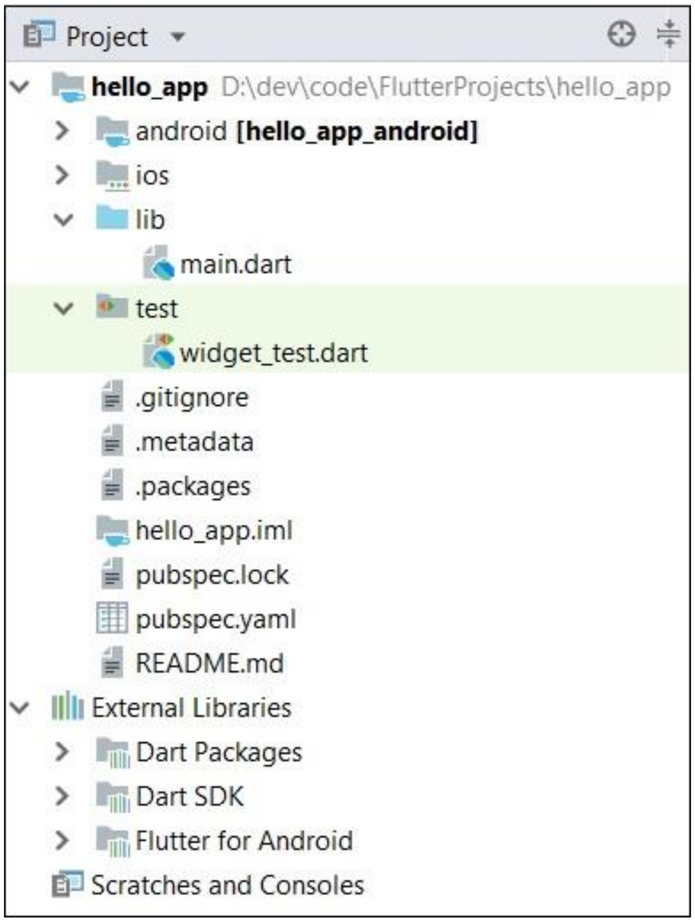
- Download VS Code from https://code.visualstudio.com/
- Install the Flutter and Dart plugins
  - Start VS Code.
  - Invoke View > Command Palette….
  - Type "install", and select Extensions: Install Extensions.
  - Type "flutter" in the extensions search field, select Flutter in the list, and click install. This also installs the required Dart plugin.
- Validate your setup with the Flutter Doctor
  - Invoke View > Command Palette….
  - Type "doctor", and select the Flutter: Run Flutter Doctor.
  - Review the output in the OUTPUT pane for any issues. Make sure to select Flutter from the dropdown in the different Output Options

5- Example

Let's create a simple Flutter application to understand the basics of creating a flutter application in the Android Studio.

- Open Android Studio
- Create Flutter Project. For this, click File → New → New Flutter Project
- Select Flutter Application. For this, select Flutter Application and click Next

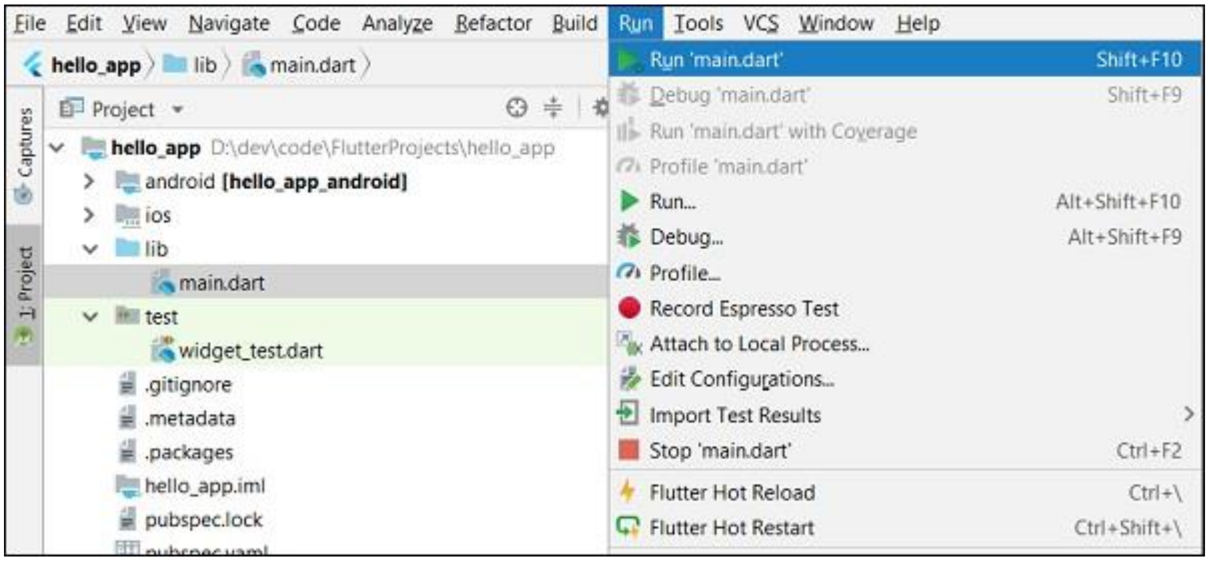
- Select flutter template “application”
- Configure the application as below and click Next
  - Project name: hello_app
  - Flutter SDK Path: <path_to_flutter_sdk>
  - Project Location: <path_to_project_folder>
  - Description: Flutter based hello world application
- Set the company domain as **com.example.hello_app** and click **Finish**
- Android Studio creates a fully working flutter application with minimal functionality.
- The structure of the application and its purpose is as follows:



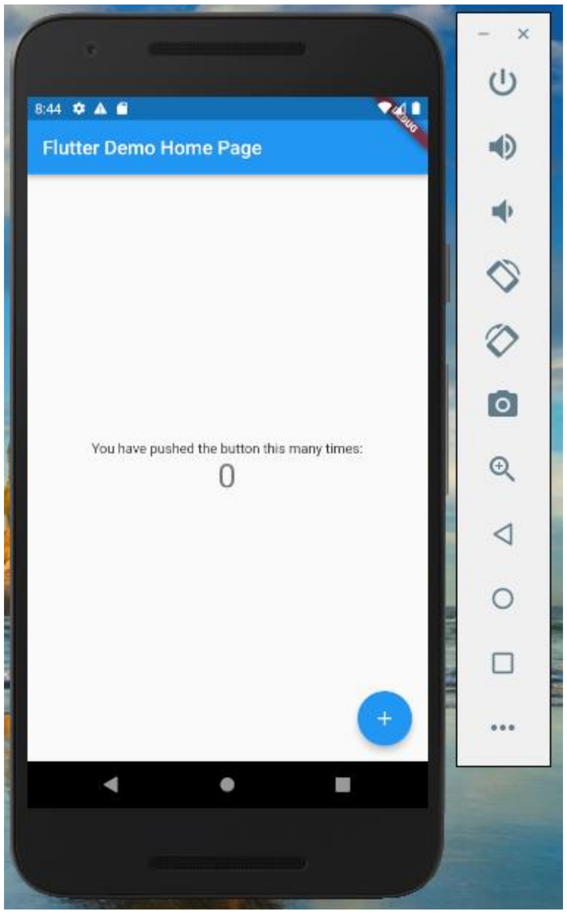
```
Project
hello_app D:\dev\code\FlutterProjects\hello_app
  android [hello_app_android]
  ios
  lib
    main.dart
  test
    widget_test.dart
  .gitignore
  .metadata
  .packages
  hello_app.iml
  pubspec.lock
  pubspec.yaml
  README.md
External Libraries
  Dart Packages
  Dart SDK
  Flutter for Android
Scratches and Consoles
```

- **android** – Auto generated source code to create android application
- **ios** – Auto generated source code to create ios application
- **lib** – Main folder containing Dart code written using flutter framework
- **ib/main.dart** – Entry point of the Flutter application
- **test** – Folder containing Dart code to test the flutter application
- **test/widget_test.dart** – Sample code
- **.gitignore** – Git version control file
- **.metadata** – auto generated by the flutter tools
- **.packages** – auto generated to track the flutter packages
- **.iml** – project file used by Android studio
- **pubspec.yaml** – Used by Pub, Flutter package manager
- **pubspec.lock** – Auto generated by the Flutter package manager, Pub
- **README.md** – Project description file written in Markdown format

➢ Now, run the application using, Run → Run main.dart


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
hello_app lib main.dart
Project
hello_app D:\dev\code\FlutterProjects\hello_app
android [hello_app_android]
ios
lib
main.dart
test
widget_test.dart
.gitignore
.metadata
.packages
hello_app.iml
pubspec.lock
pubspec.yaml
Captures
1: Project
Run 'main.dart' Shift+F10
Debug 'main.dart' Shift+F9
Run 'main.dart' with Coverage
Profile 'main.dart'
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Profile...
Record Espresso Test
Attach to Local Process...
Edit Configurations...
Import Test Results
Stop 'main.dart' Ctrl+F2
Flutter Hot Reload Ctrl+\
Flutter Hot Restart Ctrl+Shift+\

➢ Finally, the output of the application is as follows:



6- Introduction to Dart Programming

➢ Dart is an open-source general-purpose programming language. It is originally developed by Google. Dart is an object-oriented language with C-style syntax. It supports programming concepts like interfaces, classes, unlike other programming languages Dart doesn't support arrays. Dart collections can be used to replicate data structures such as arrays, generics, and optional typing.

- The following code shows a simple Dart program

```
void main() {
   print("Dart language is easy to learn");
}
```

- Variable is named storage location and Data types simply refers to the type and size of data associated with variables and functions. Dart uses var keyword to declare the variable. The syntax of var is defined below:

```
var name = 'Dart';
```

- The final and const keyword are used to declare constants. They are defined as below:

```
void main() {
   final a = 12;
   const pi = 3.14;
   print(a);
   print(pi);
}
```

- Dart language supports the following data types:
  - **Numbers** – It is used to represent numeric literals – Integer and Double.
  - **Strings** − It represents a sequence of characters. String values are specified in either single or double quotes.
  - **Booleans** − Dart uses the bool keyword to represent Boolean values – true and false.
  - **Lists** − It is used to represent a collection of objects. A simple List can be defined as below

```
void main() {
   var list = [1,2,3,4,5];
   print(list);
}
```

The list shown above produces [1,2,3,4,5] list

- **Map** - can be defined as shown here:

```
void main() {
   var mapping = {'id': 1,'name':'Dart'};
   print(mapping);
}
```

- **Dynamic** – If the variable type is not defined, then its default type is dynamic. The following example illustrates the dynamic type variable:

```
void main() {
   dynamic name = "Dart";
   print(name);
}
```

➢ **Decision making:** A decision making block evaluates a condition before the instructions are executed. Dart supports If, If..else and switch statements

➢ **Loops:** are used to repeat a block of code until a specific condition is met. Dart supports for, for..in , while and do..while loops:

```
void main() {
   for( var i = 1 ; i <= 10; i++ ) {
      if(i%2==0) {
         print(i);
      }
   }
}
```

- **Functions**: A function is a group of statements that together performs a specific task. Let us look into a simple function in Dart as shown here

```
void main() {
   add(3,4);
}
void add(int a,int b) {
   int c;
   c = a+b;
   print(c);
}
```

- **Object oriented programming**: Dart is an object-oriented language. It supports object-oriented programming features like classes, interfaces, etc.

  A class is a blueprint for creating objects. A class definition includes the following:

  - Fields
  - Getters and setters
  - Constructors
  - Functions

  Now, let us create a simple class using the above definitions

```
class Employee {
   String name;

   //getter method
   String get emp_name {
      return name;
   }
   //setter method
   void set emp_name(String name) {
      this.name = name;
   }
   //function definition
   void result() {
      print(name);
   }
}
void main() {
   //object creation
   Employee emp = new Employee();
   emp.name = "employee1";
   emp.result(); //function call
}
```