# Module 7

Introduction to Dart and Flutter

# Flutter Installation

# Reference link

- Installation
  - https://docs.flutter.dev/get-started/install/windows
  - https://www.youtube.com/watch?v=fDnqXmLSqtg
  - https://www.youtube.com/watch?v=kEoyVAF25do

# What is Flutter?

- Flutter is a cross-platform SDK from Google.
- **Flutter** was initially launched by Google in 2017, is emerging as one of the fastest application development platforms.
- It's feature-rich and productive user interface frameworks is allowing developers to build cross-platform applications seamlessly.
- This open-source and free software development kit (SDK) allows designing native iOS and Android applications using a particular codebase.
- Flutter's constructive toolkit provides all features required for cross-platform application development.
- Flutter is a portable user interface toolkit complete with tools and widgets.
- It provides developers with a platform to build and deploy natively compiled, visually attractive applications for various platforms with ease.
- With Flutter and Dart developers can build IOS and Android apps with just one codebase and only one programming language.

# What is Dart?

- **Dart** is a platform-independent, open-source, and object-oriented programming language that comprises a range of useful features for a software developer.
- It is a **client side programming language** that renders an extensive range of app development utilities, such as a collection of design features, dynamic typing, interface, classes, and optional typing.
- Dart is developed for both server and browser.
- Dart is an open-source framework which means it is free to use and available on every browser.
- It is a Google-developed framework, comes with a BSD license, and is approved by the ECMA standard.
- It is a garbage-collected and class-based language with the C-style syntax.
- Dart has its package manager called Pub, and this is a major point that distinguishes Dart from other languages.
- Developers can use Pub to create Flutter and Dart applications.
- Dart is **Object-oriented language** and is quite similar to that of **Java Programming**. Dart is extensively use to create single-page websites and web-applications. Best **example** of dart application is **Gmail**.

# Flutter vs Dart Comparison

| | Flutter | Dart |
|---|---|---|
| **Description** | Open-Source UI SDK | Client side programming language for web and mobile apps |
| **Category** | Framework | Programming language |
| **Programming Language** | Dart | Dart |
| **Initial Release Date** | 2017 | 2013 |
| **Developer** | Google | Goole |
| **Open Source** | Yes | Yes |
| **Free to Use** | Yes | Yes |
| **License** | BSD 3-Clause "New" or "Revised" License | BSD 3-Clause "New" or "Revised" License |
| **Advantages** | Same UI across multiple platforms<br>Native performance<br>Own Rendering Engine | Easy to learn<br>High performance<br>Stability |
| **Popular apps** | Philips<br>MGM Resorts<br>ByteDance | Flutter |


## AlertDialog

Alerts are urgent interruptions requiring acknowledgement that inform the user about a situation. The AlertDialog widget implements this component.

Documentation


## CupertinoAlertDialog

An iOS-style alert dialog.

Documentation

# Structure of the Dart language

- Every language specification defines its own syntax. A Dart program is composed of −
  - Variables and Operators
  - Classes
  - Functions
  - Expressions and Programming Constructs
  - Decision Making and Looping Constructs
  - Comments
  - Libraries and Packages
  - Typedefs
  - Data structures represented as Collections / Generics

# Simple hello world example

```
main() {
   print("Hello World!");
}
```

# Identifiers in Dart

- Identifiers are names given to elements in a program like variables, functions etc.
- The rules for identifiers are –
  - Identifiers can include both, characters and digits. However, the identifier cannot begin with a digit.
  - Identifiers cannot include special symbols except for underscore (_) or a dollar sign ($).
  - Identifiers cannot be keywords.
  - They must be unique.
  - Identifiers are case-sensitive.
  - Identifiers cannot contain spaces.

| Valid identifiers | Invalid identifiers |
| --- | --- |
| firstName | Var |
| first_name | first name |
| num1 | first-name |
| $result | 1number |

# Keywords in Dart

| | | | | |
|---|---|---|---|---|
| abstract 1 | continue | false | new | this |
| as 1 | default | final | null | throw |
| assert | deferred 1 | finally | operator 1 | true |
| async 2 | do | for | part 1 | try |
| async* 2 | dynamic 1 | get 1 | rethrow | typedef 1 |
| await 2 | else | if | return | var |
| break | enum | implements 1 | set 1 | void |
| case | export 1 | import 1 | static 1 | while |
| catch | external 1 | in | super | with |
| class | extends | is | switch | yield 2 |
| const | factory 1 | library 1 | sync* 2 | yield* 2 |

# Comments in Dart

```
// this is single line comment

/* This is a
   Multi-line comment
*/
```

# Variables

- Variables in dart can be declared in two ways
  - Using var keyword

```
var name = 'Smith';
```

  - by using the type of the variable

```
String name = 'Smith';
int num = 10;
```

# The dynamic keyword

- Variables declared without a static type are implicitly declared as dynamic.
- Variables can be also declared using the dynamic keyword in place of the var keyword.

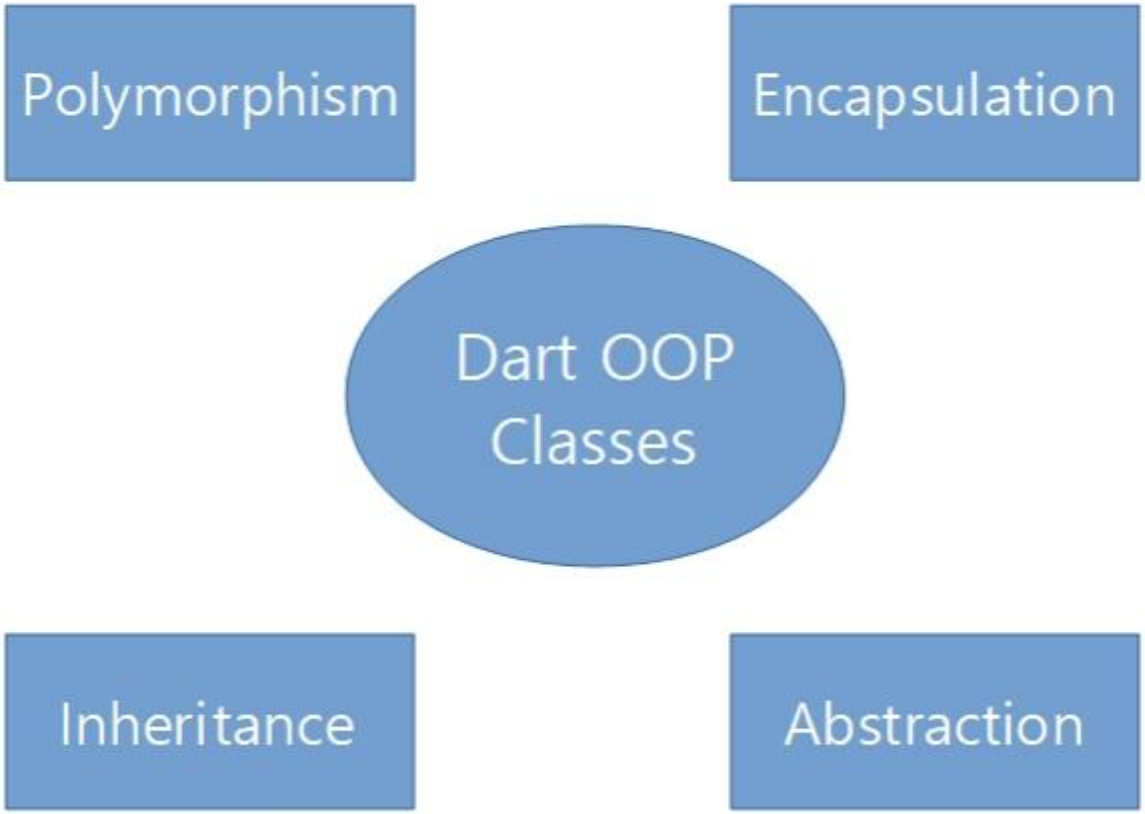
```
void main() {
   dynamic x = "tom";
   print(x);
}
```

- Dynamic is a special type and it can assume any type at runtime; therefore, any value can be cast to dynamic too.
- *The Dart analyzer can infer types for fields, methods, local variables, and most generic type arguments. When the analyzer doesn't have enough information to infer a specific type, it uses the dynamic type.*

# Introduction to OOP in Dart

- In Dart, everything is an object, including the built-in types.
- Upon defining a new class, even when you don't extend anything, it will be a *descendant of an object*.
- Dart implicitly does this for you.
- Dart is called a **true object-oriented** language.
- Even functions are objects.

# Dart OOP features



# Dart – Classes And Objects

- Dart is an object-oriented programming language, so it supports the concept of class, object … etc.
- In Dart, we can define classes and objects of our own.
- We use the **class** keyword to do so.

**Declaring class in Dart –**

**Syntax:**

```
class class_name {

   // Body of class
}
```

# Example

```
class TestClass {
   void disp() {
      print("Hello World");
   }
}
void main() {
   TestClass c = new TestClass();
   c.disp();
}
```

# packages in Dart Programming

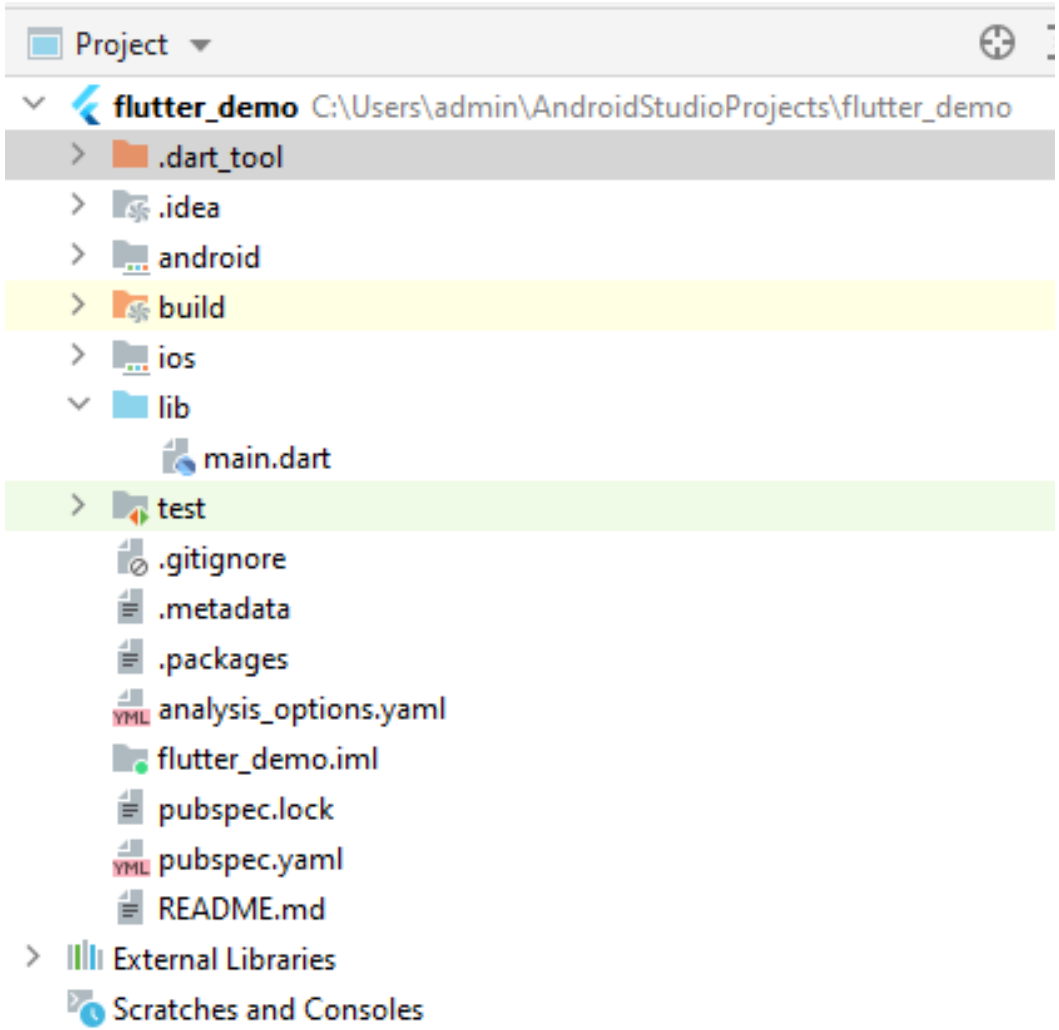
- A package is a mechanism to encapsulate a group of programming units.
- Applications might at times need integration of some third-party libraries or plugins.
- Every language has a mechanism for managing external packages like Maven or Gradle for Java, Nuget for .NET, npm for Node.js, etc.
- The package manager for Dart is **pub**.
- Pub helps to install packages in the repository.
- The repository of packages hosted can be found at https://pub.dartlang.org/.

# pubspec.yaml file

- The **package metadata** is defined in a file, **pubspec.yaml**.
- YAML is the acronym for **Yet Another Markup Language**.
- The **pub** tool can be used to download all various libraries that an application requires.
- Every Dart application has a **pubspec.yaml** file which contains the application dependencies to other libraries and metadata of applications like application name, author, version, and description.

```
name: 'vector_victor'
version: 0.0.1
description: An absolute bare-bones web app.
...
dependencies: browser: '>=0.10.0 <0.11.0'
```

# Structure of the application



# Various components of the structure of the application

- **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

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

void main() {
  runApp(
    const Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

# First flutter application

**main.dart**

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

void main() {
  runApp(MyApp());
}
//void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome'),
        ),
        body: const Center(
          child: DecoratedBox(decoration:
BoxDecoration(color: Colors.blueGrey),
          child: Padding(padding: const EdgeInsets.all(
10.0),
          child: Text('First flutter application')
          ),
          ),
        ),
      ),
    );
  }
}
```

- This example creates a Material app.
- Material is a visual design language that is standard on mobile and the web.
- Flutter offers a rich set of Material widgets.
- It's a good idea to have a uses-material-design: true entry in the flutter section of your pubspec.yaml file.
- This will allow you to use more features of Material, such as their set of predefined Icons.
- The main() method uses arrow (=>) notation.
- Use arrow notation for one-line functions or methods.
- The app extends StatelessWidget, which makes the app itself a widget.
- In Flutter, almost everything is a widget, including alignment, padding, and layout.
- The Scaffold widget, from the Material library, provides a default app bar, and a body property that holds the widget tree for the home screen.
- A widget's main job is to provide a build() method that describes how to display the widget in terms of other, lower level widgets.
- The body for this example consists of a Center widget containing a Text child widget.
- The Center widget aligns its widget subtree to the center of the screen.

- MaterialApp is **the starting point of your app**, it tells Flutter that you are going to use Material components and follow material design in your app.
- Scaffold is used under MaterialApp , it gives you many basic functionalities, like AppBar , BottomNavigationBar , Drawer , FloatingActionButton etc.
- The Scaffold is designed to be the single top-level container for a MaterialApp although it is not necessary to nest a Scaffold.