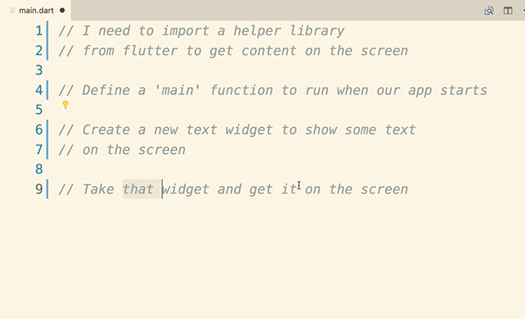
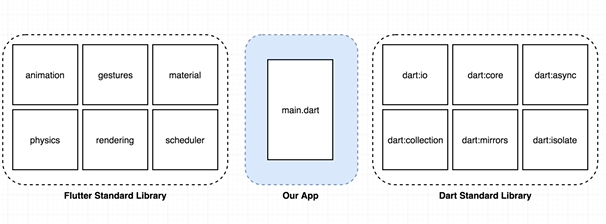


* Flutter application is made up of widgets that represents elements displayed on screen. Widget is a dart class.
* In order to create complex flutter applications, we create reusable widgets that can be used across the application several times.
* Widgets provided by flutter can be access at <https://flutter.io/widgets>

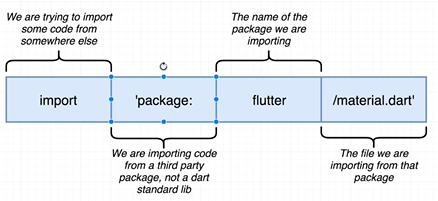
**Four steps to design simple App page**



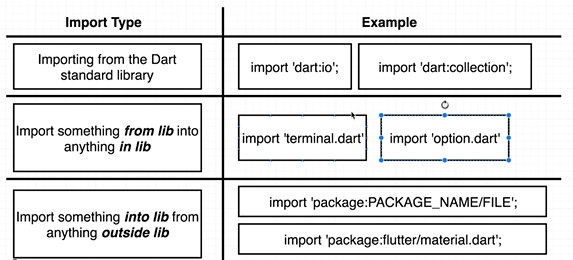
1. **Import Statements**: Flutter apps have access to dart standard library and flutter standard library. In order to use any of these codes, we have to import these library in application file.



In dart, there are three different types of import statements. These are to import third party package, dart standard library code and code written by us.



**Syntax to import third party flutter library**



1. **Main function:** This function is executed when the app runs. All the code to get the content up on the screen goes here.
2. **Create Widget:** Create widget to display on screen using flutter packages.
3. **Show Widget:** Get the created widget to show on screen.

**Code Example:**

void main() {

  //create a new text widget to show some content on screen

  var app = MaterialApp(

    home: Scaffold(

      appBar: AppBar(

        backgroundColor: Colors.green,

        title: Text("Random Images")

      ),

      floatingActionButton : FloatingActionButton(

        backgroundColor: Colors.green,

        child: Icon(Icons.add),

        onPressed: () {

          print("hi there....");

        },

      )

    )

  );

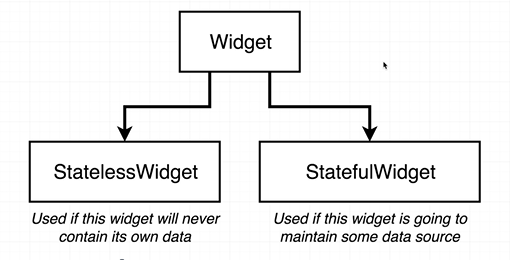
  //take the widget and get it on screen

  runApp(app);

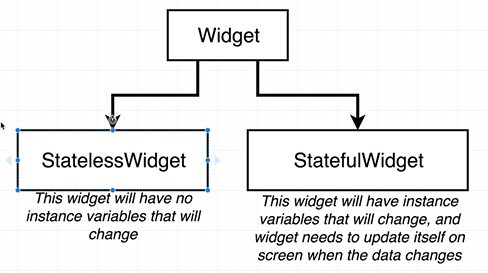
}

**Stateless vs Stateful Widget**

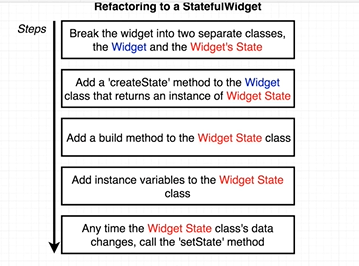
Custom widget needs to extend the base class that can be a stateful or stateless widget. A widget is class in flutter that controls the elements on screen. It contains bunch of functions & different variables that helps to show some content on screen and perform operations. StatelessWidget and Stateful Widget are subclasses of Widget class with the main difference whether the widget contains its own state or data.

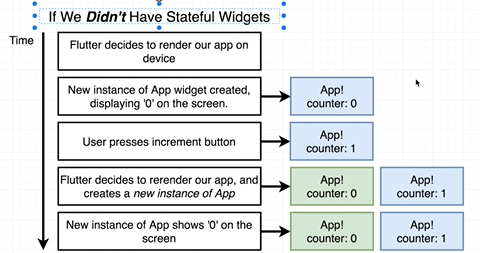
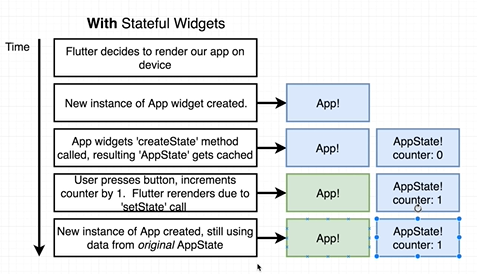


Stateless Widget must implements a build method that returns the widgets that this widget will show on the screen. Build method accepts the context as parameter and return Widget.



To implement stateful widget, we need to have two separate classes, one containing the widget itself and other containing the widget state. Widget class must implement createState method that returns the instance of widget state. Widget state should have instance variables which changes over the time and must implement any change in instance variable inside setState method only for change to reflect and update the screen.





**Named Constructor:** We can add multiple constructor to single class in dart. We can add a named constructor with below syntax:

**ClassName.ConstructorName(param){body}**

**Parse JSON in Dart:** We need to import dart package ‘dart:convert’ which contains json object having functions to json.

Import “dart:convert”;

Void main(){

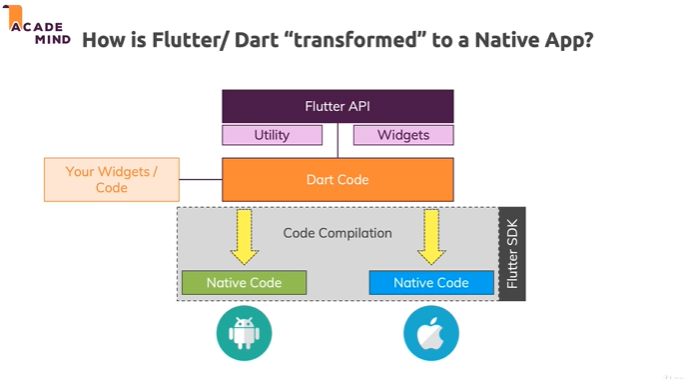
Var rawJson = ‘{“id”:1, “url”:”blabla.com”}’;

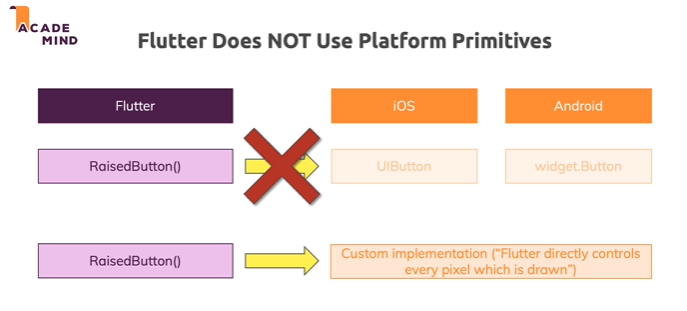
Var parsedJson = json.decode(rawJson);

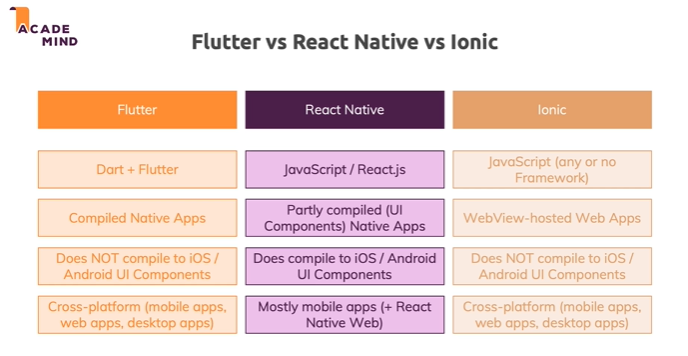
}

**What is flutter?**









**Dart Basics**

**What is Dart?**

Dart is an **object-oriented programming language** developed by Google. Whilst technically not restricted, it’s primarily used for creating **frontend user interfaces** for the web (with AngularDart or Flutter for Web) and mobile apps (Flutter).

It’s under active development, **compiled to native machine code** (when used for building mobile apps), inspired by modern features of other programming languages (mainly Java, JavaScript, C#) and **strongly typed**.

As already mentioned, Dart is a compiled language. That means, that your code isn’t executed like you write it, but instead, a compiler parses + transform it (to machine code).

**What are "Types"?**

Not all programming languages are strongly typed - but Dart is. What does that mean though?

Every value you use in your program (e.g. some user input you're saving) **has a type** - it could be "text" for example. In Dart (and in pretty much all other programming languages), that would not be called "text" though but "**string**".

On the other hand, if you're working with the age of a user, you might be using a**number**without any decimal places - a so-called "**integer**" value. Numbers with decimal places are called "**doubles**" (or "float" - in other programming languages).

These are some basic types - Dart has way more than these basic types though. You will learn way more about these (and all the other important) types throughout the course.

At the moment, these are types you should keep in mind:

**String** => "Hello" *or*'Hello' => Text, has to be wrapped in quotes in your code (you can use double or single quotes, just be consistent once you made your choice).

**int** => 20*or*-54 => Numbers without decimal places

**double** => -3.99*or*85.9421 => Numbers WITH decimal places

**num** => The "parent type" of double and int. You should rarely use it - be more specific about which type of value you need (int or double) if possible!

It's also important to keep in mind that EVERY value in Dart is an object. More on that can be found below (=> *"What does "Object-oriented" mean?"*)

**Variables & Functions**

In your programs, you typically need to store some values. Not necessarily in a database or in a file but in memory. You might need to store some intermediate result, the input of a user before you process it or some information about your Flutter widget (e.g. "Should it currently be displayed on the screen?").

You store such information (= values) in so-called **variables**.

Variables are data containers - they have names and store values of any type.

For example:

1. var myAge = 30;

is a variable that stores an integer (int) of value 30.

The var keyword tells Dart that myAge is a variable. Alternatively to var, you could also use the type name - in addition to informing Dart about the variable, you would then also "inform Dart" about the type of data stored in the variable:

1. int myAge = 30;

However, Dart has a feature called "**type inference**". This means, that Dart is pretty smart about inferring types of values. If you created the variable with var, Dart is still able to **infer** that myAge is of type int because you initialize the variable (i.e. you assign a value right from the start) with an integer value.

Because of that built-in type inference, it's considered a**good practice** to NOT explicitly define the type but instead use var when defining variables. Hence this snippet would be preferred:

1. var myAge = 30;

That changes if you create a variable without an initial value - then, you should inform Dart about which type of data you plan on saving in there:

1. int myAge;
2. myAge = 30;

Besides variables, another core feature of ANY programming language are "functions".

Functions allow you to "outsource" code into "re-usable code snippets". Here's an example:

1. var price1 = 9.99;
2. var price2 = 10.99;
3. var total = price1 + price2;
5. var numOfRounds1 = 10;
6. var numOfRounds2 = 45;
7. var totalRounds = numOfRounds1 + numOfRounds2;

In this example, we have the same logic of adding two numbers in two different places of our code. Instead of repeating ourselves, it would be great to put that logic into a function that we can call whenever as often as we want. Here's the changed snippet, using a function:

1. num addNumbers(num n1, num n2) { // use num as a type because it should work with int and double
2. return n1 + n2;
3. }
5. var price1 = 9.99;
6. var price2 = 10.99;
7. var total = addNumbers(price1, price2);
9. var numOfRounds1 = 10;
10. var numOfRounds2 = 45;
11. var totalRounds = addNumbers(numOfRounds1, numOfRounds2);

Of course, on first sight, this code now got longer and more complex. But the advantage is, that if you ever need to change your addition logic, you do it in **one place** => Inside of the function. You don't have to edit multiple places in code just to change the + into a - for example (if you wanted to do that).

Functions can take arguments (the data between the ( )) which are basically variables, scoped to the function. "**Scoping**" means that you can use the variables only inside of the block statement (= function body, the code between the { }) where you defined them.

In addition, functions can also return values - like the result of the addition in this example.

**What does "Object-oriented" mean?**

Dart is an object-oriented programming language - that means that **every value in Dart is an object**. Even a simple number.

What are “objects” though?

**Objects are data structures** - you find them in a lot of programming languages. In Dart, every value is an object, even primitive values like text (= String) or numbers (= Integers and Doubles). But you also have more complex built-in objects (e.g. Lists of data) and you can build your own objects.

You often build your own objects if you want to express more complex relations between data or if you want to encapsulate certain functionality in “one building block”.

Objects are created with the help of “**Classes**” because every object needs a **blueprint** (=> the class) based on which you can then create (“instantiate”) it.

Here’s an example class definition:

1. class Person {
2. var name = 'Max';
3. var age = 30;
5. void greet() {
6. print('Hi, I am ' + name + ' and I am ' + age.toString() + ' years old!';
7. }
8. }

In this example, we define a Person class which has two class-level variables (also called “instance fields” or “**properties**”) and one class-level function (also called **“method"**).

As you can see, we also use types in classes - for both properties (variables) and methods (functions).

You can also see, that inside of the greet method, we can access the class properties name and age without issues (age.toString() is used to covert the integer value to a string whilst outputting it in a longer string).

The class only serves as a blueprint though! On its own, it does not give you an object! Instead, you can now create objects based on this class:

1. class Person {
2. var name = 'Max';
3. var age = 30;
5. void greet() {
6. print('Hi, I am ' + name + ' and I am ' + age.toString() + ' years old!';
7. }
8. }
10. void main() {
11. var myself = Person();
12. print(myself.name); // use the . to access class properties & methods
13. }

As a side note: The main function is a special function in Dart - **it’s the function which Dart will execute first**, when your app starts.  
Inside of main, we then create a new object based on Person by using Person(). This process is called “instantiating the class”, hence we create “an instance of Person”.

The **type of**myself would then be Person because**classes always also act as types**!