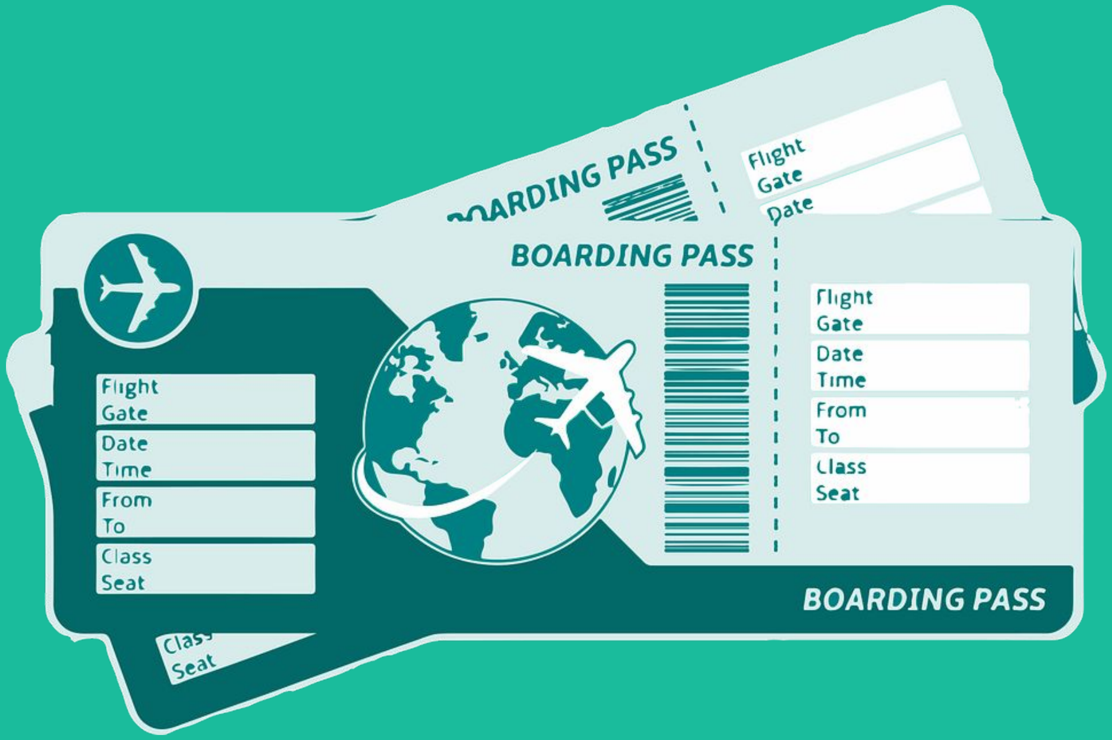
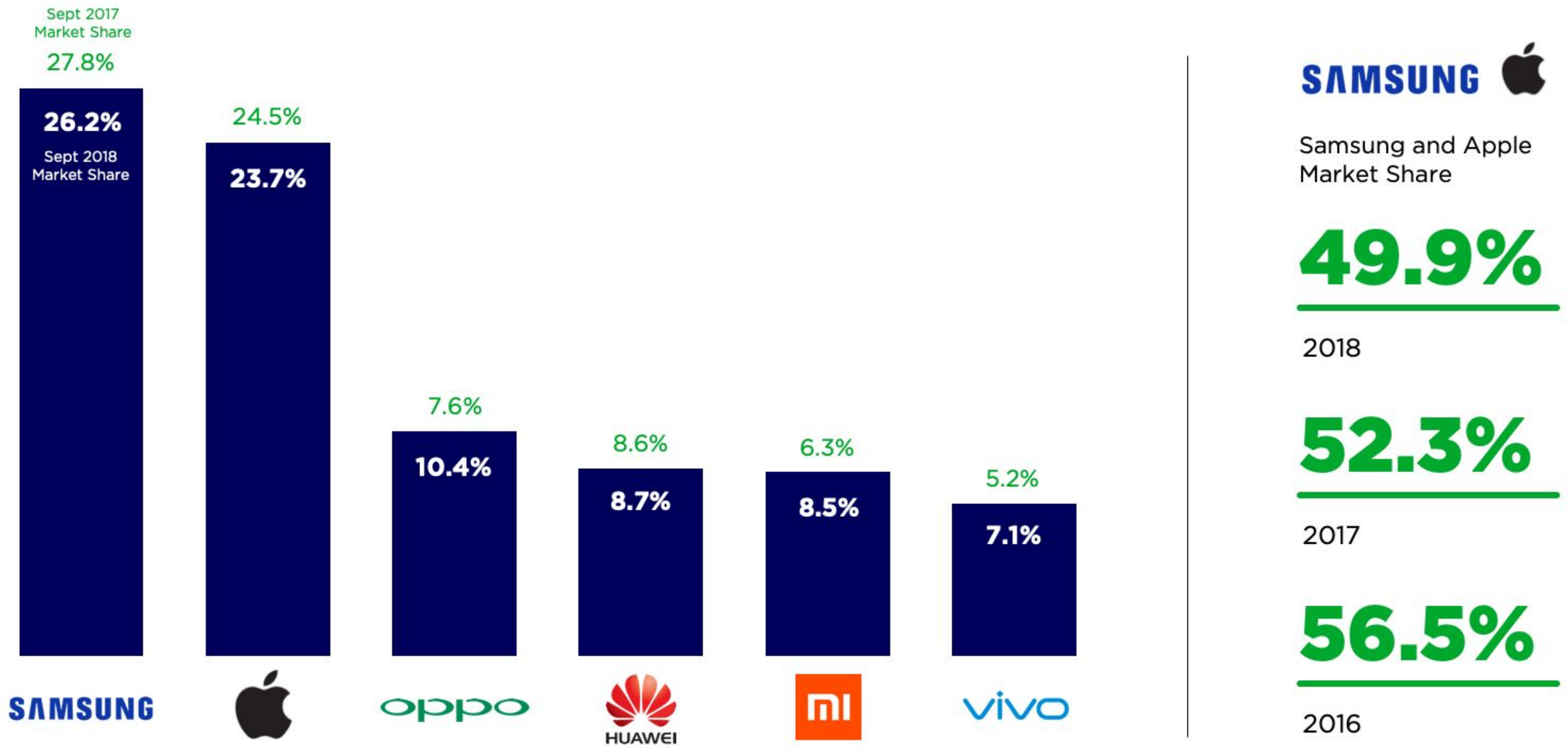


Chapter 11. Mobile Applications

Bilkent University | CS443 | 2021, Spring | Dr. Orçun Dayıbaş

Introduction

- **What is it?**

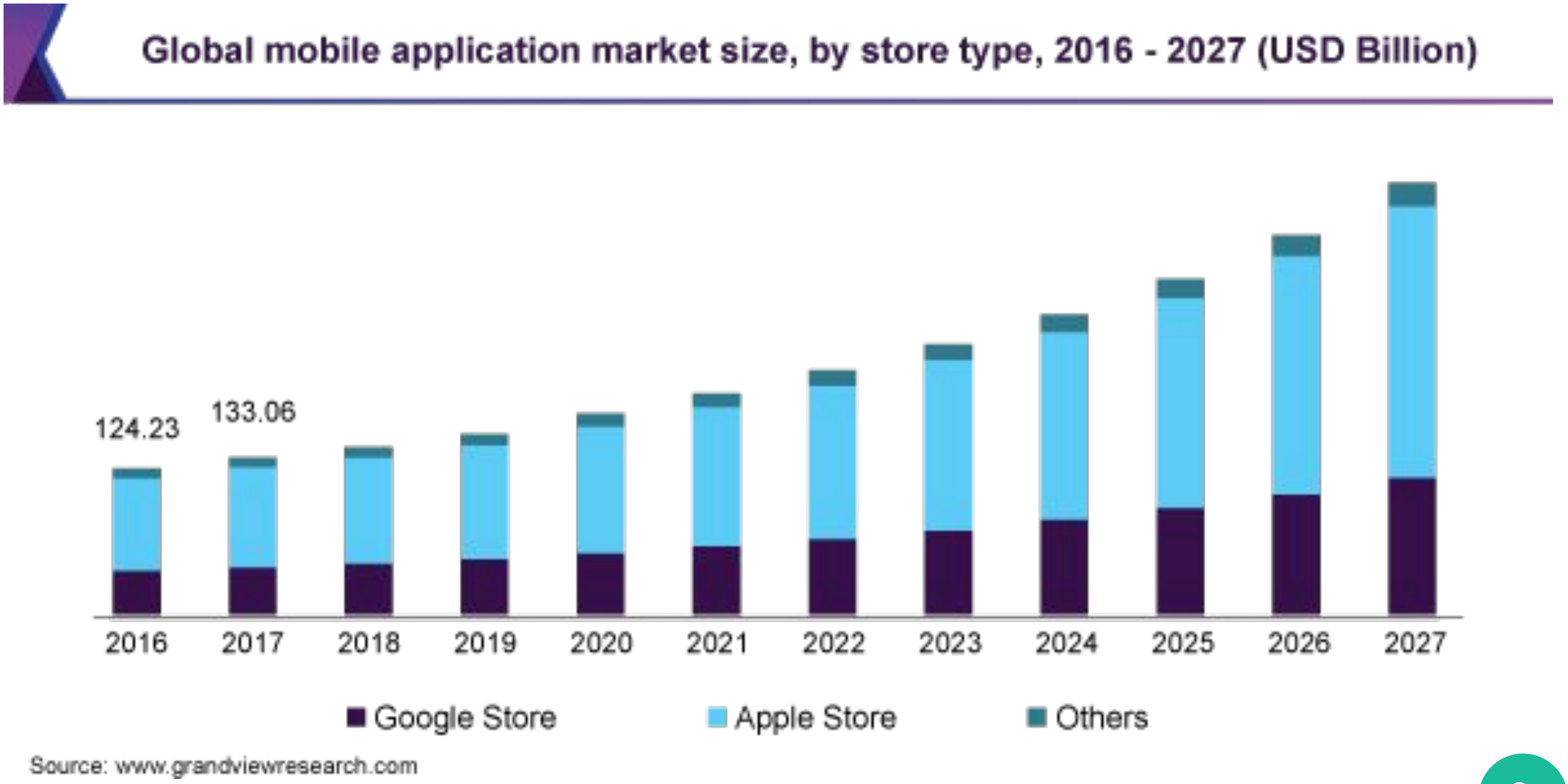


Based on the top 100 countries in terms of smartphone users, representing more than 98% of the world's smartphone population. Based on monthly active use in September 2018.

newzoo.com/global-mobile-report

Introduction

- **What is it?**

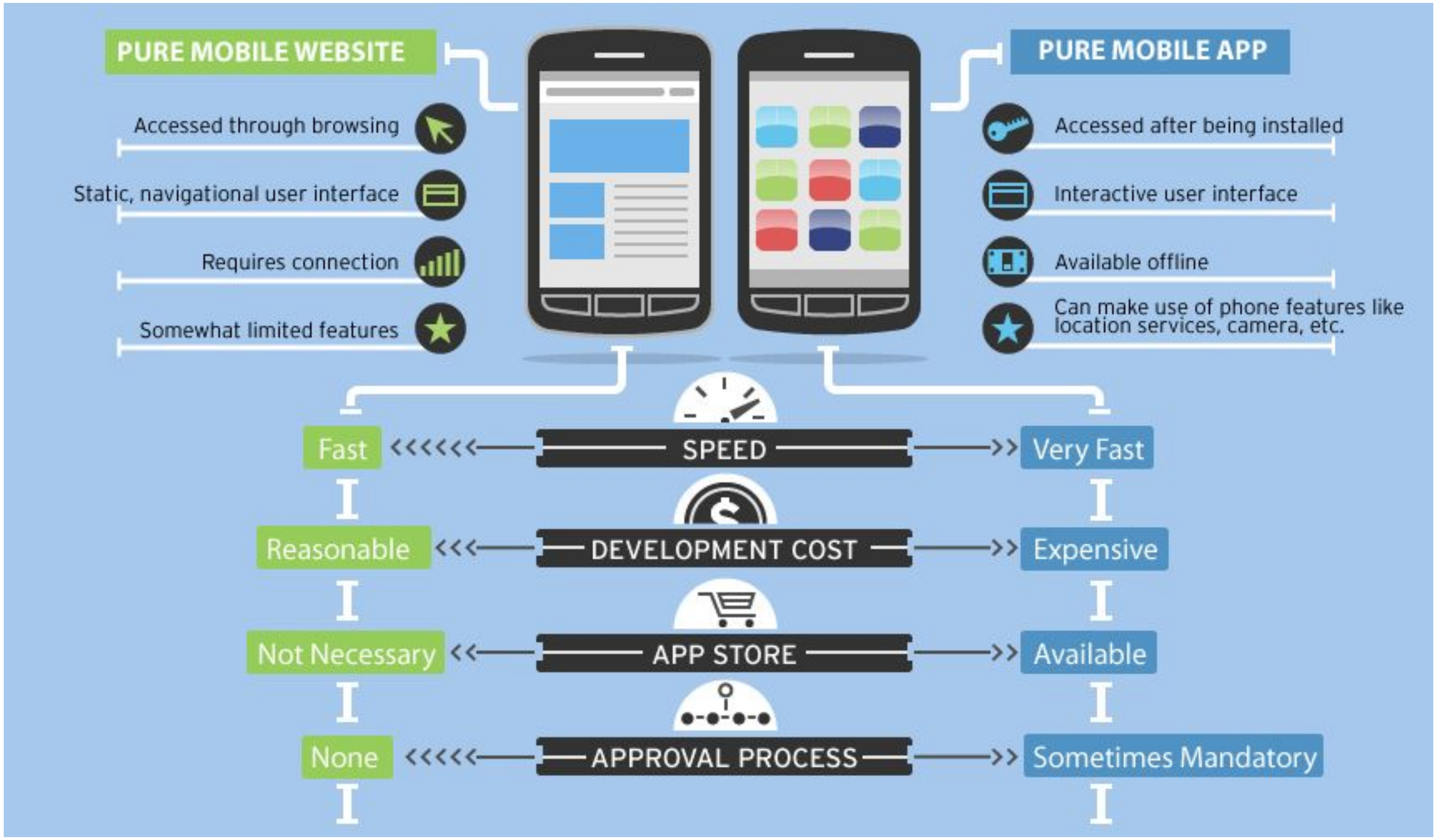


Introduction

- **Why is it harder to develop?**
    - **Fragmented**: there are tons of different devices to be supported
    - **Constrained**: Almost every resource is very constrained, we have lots of trade-offs
    - **Ubiquitous**: Need to work well in all the different contexts a user might be in

Introduction

- **Mobile App vs. Mobile website**


PURE MOBILE WEBSITE
Accessed through browsing
Static, navigational user interface
Requires connection
Somewhat limited features
PURE MOBILE APP
Accessed after being installed
Interactive user interface
Available offline
Can make use of phone features like location services, camera, etc.
Fast
SPEED
Very Fast
Reasonable
DEVELOPMENT COST
Expensive
Not Necessary
APP STORE
Available
None
APPROVAL PROCESS
Sometimes Mandatory

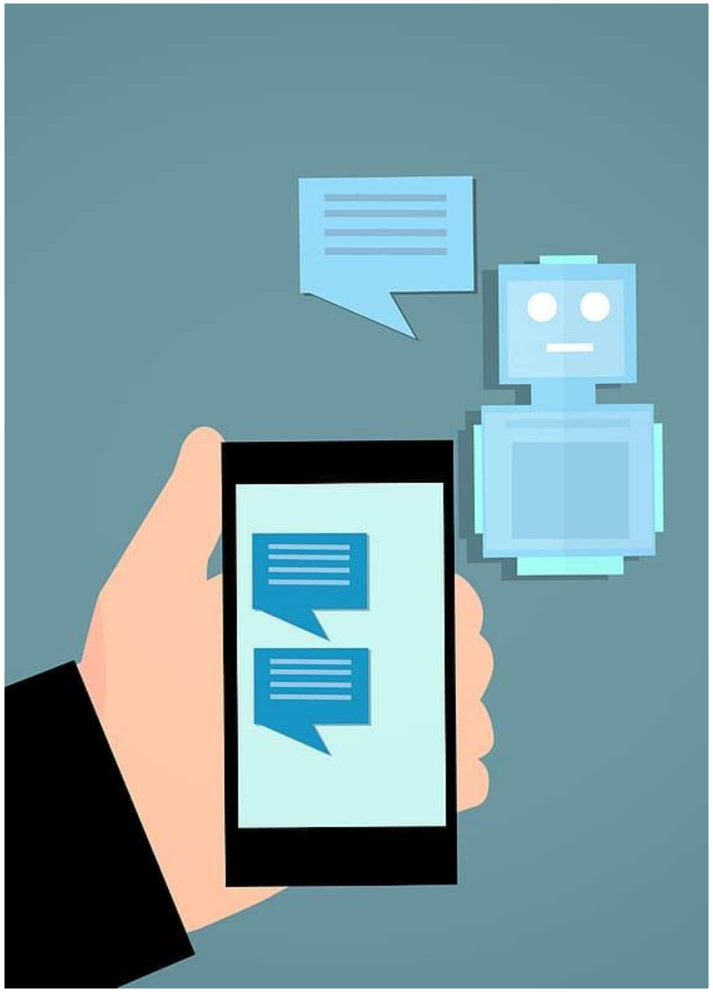
https://www.businessinsider.com/a-mobile-website-might-be-what-your-business-needs-2013-3

Introduction

- **Single platform vs. multiple platform**
  - Single platform: native applications
    - Pros: performance, responsiveness, continuous support and wide variety of features in SDKs
    - Cons: two different code bases (cost)
  - Multiple platform: hybrid or native applications
    - Hybrid Applcations: Under the hood web applications working in a native sandbox application (Ex: Cordova, Ionic, etc.)
      - Pros: Easy to roll-out a feature
      - Cons: bad UX (in general), bad performance
    - Cross-platform Applications:
      - Pros: faster than hybrid, better UX than hybrid
      - Cons: another layer to access components (slower)

Dart & Flutter

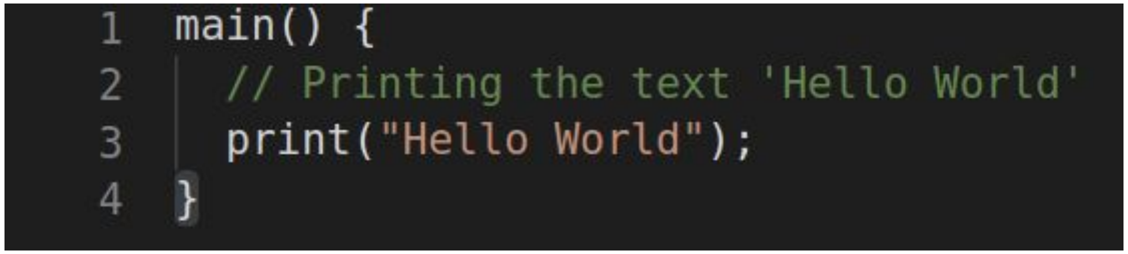
- **Why Dart & Flutter?**
  - Fast
  - Optimized for UI
    - Asynchronous by design
    - Single-threaded
  - Productive Development
    - Flutter has a hot reload feature
    - Flutter provides static analysis
  - Fast on all Platforms
    - Dart has an AOT (Ahead of Time) compiler
    - Flutter is written in Dart → Easy to extend



https://www.educative.io/courses/learn-dart-first-step-to-flutter

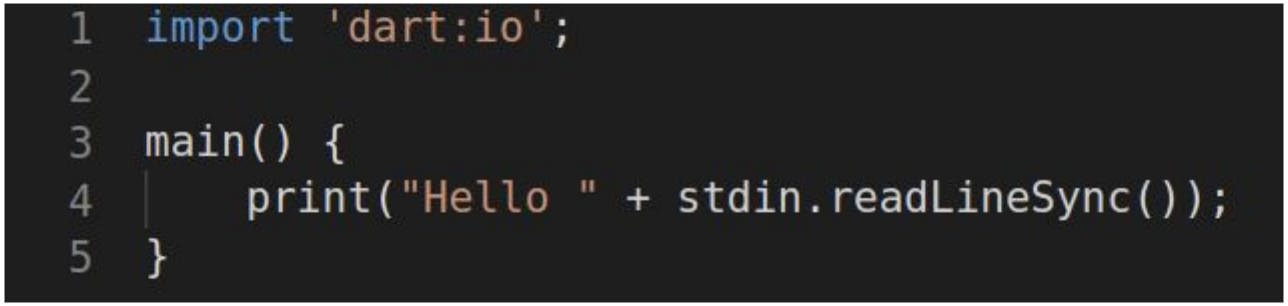
Dart

- **Hello world**
  - Dart uses imperative programming style
  - “void main()” is the main entry point



```dart
main() {
  // Printing the text 'Hello World'
  print("Hello World");
}
```

- **Libraries**
  - Ex: dart:io provides I/O operations

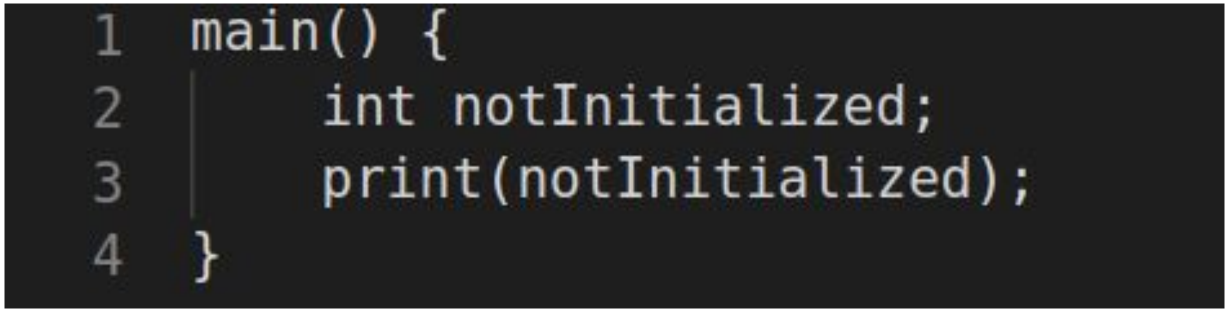


```dart
import 'dart:io';

main() {
    print("Hello " + stdin.readLineSync());
}
```

https://www.educative.io/courses/learn-dart-first-step-to-flutter

Dart

- **Object-oriented & Statically typed**
  - Very similar to Java
  - Language specification: https://dart.dev/guides/language/spec
  - Built-in data types: Numbers, Strings, Booleans, Lists, Sets, Maps, Runes, Symbols
  - Value & reference types

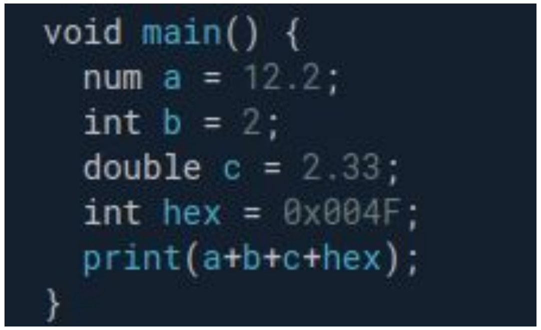


```
main() {
    int notInitialized;
    print(notInitialized);
}
```

https://dartpad.dartlang.org/

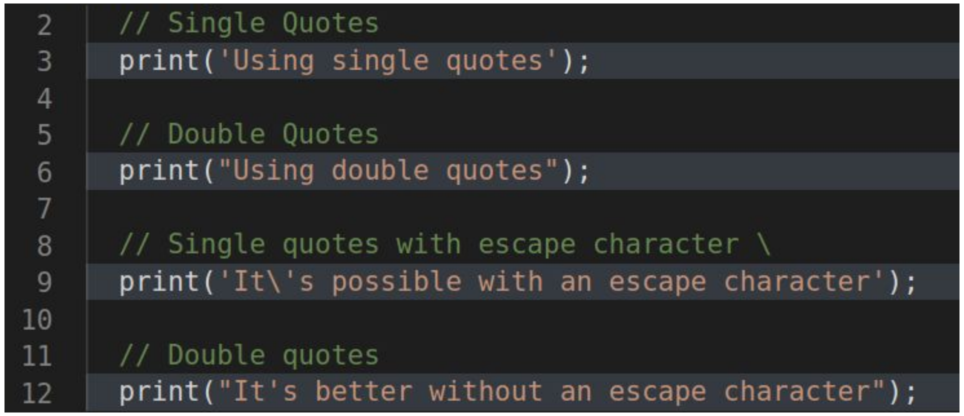
Dart

- **Numbers**
  - num, integer and double

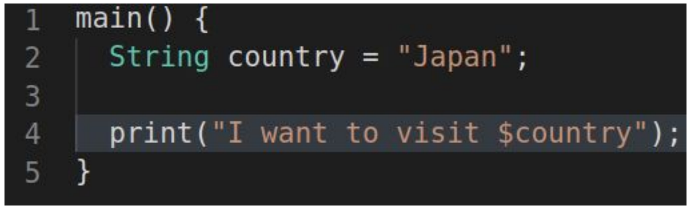


```dart
void main() {
  num a = 12.2;
  int b = 2;
  double c = 2.33;
  int hex = 0x004F;
  print(a+b+c+hex);
}
```

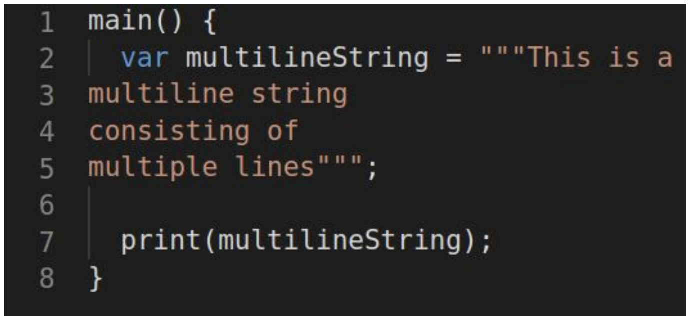
- **Strings**



```dart
// Single Quotes
print('Using single quotes');

// Double Quotes
print("Using double quotes");

// Single quotes with escape character \
print('It\'s possible with an escape character');

// Double quotes
print("It's better without an escape character");
```



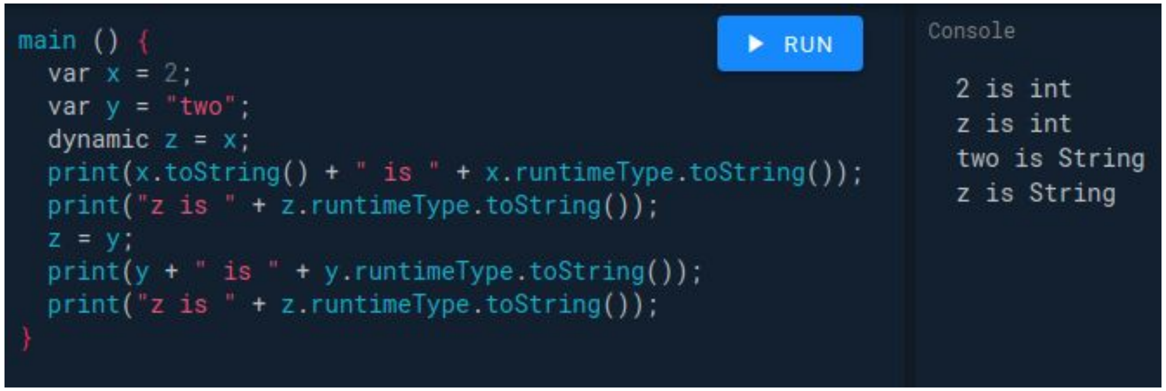
```dart
main() {
  String country = "Japan";

  print("I want to visit $country");
}
```



```dart
main() {
  var multilineString = """This is a
multiline string
consisting of
multiple lines""";

  print(multilineString);
}
```

Dart

- **Type inference**
  - var → type is fixed by the first assignment
  - dynamic → a variable can hold objects of many types

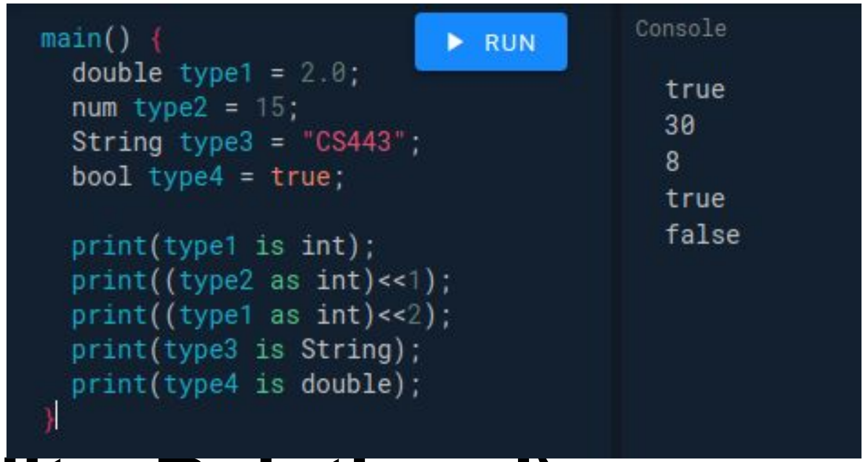


RUN

```
main () {
  var x = 2;
  var y = "two";
  dynamic z = x;
  print(x.toString() + " is " + x.runtimeType.toString());
  print("z is " + z.runtimeType.toString());
  z = y;
  print(y + " is " + y.runtimeType.toString());
  print("z is " + z.runtimeType.toString());
}
```

Console

```
2 is int
z is int
two is String
z is String
```



RUN

```
main() {
  double type1 = 2.0;
  num type2 = 15;
  String type3 = "CS443";
  bool type4 = true;

  print(type1 is int);
  print((type2 as int)<<1);
  print((type1 as int)<<2);
  print(type3 is String);
  print(type4 is double);
}
```

Console

```
true
30
8
true
false
```

- **Operators (Arithmetic, Equality, Relational)**
  - Nothing special (very similar to Java)

Dart

- **Other operators**

Operator	Name	Meaning
( )	Function application	Represents a function call
[ ]	List access	Refers to the value at the specified index in the list
.	Member access	Refers to a property of an expression; example: foo.bar selects property bar from expression foo
?.	Conditional member access	Like ., but the leftmost operand can be null; example: foo?.bar selects property bar from expression foo unless foo is null (in which case the value of foo?.bar is null)

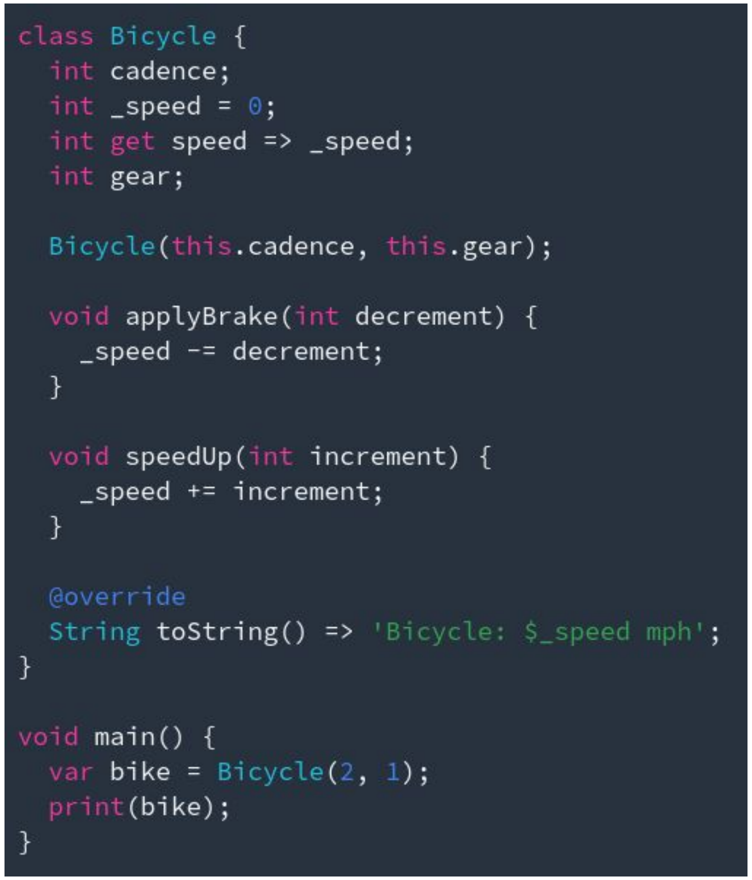
- **Cascade notation**
  - to make a sequence of operations on the same object

```
querySelector('#confirm') // Get an object.
  ..text = 'Confirm' // Use its members.
  ..classes.add('important')
  ..onClick.listen((e) => window.alert('Confirmed!'));
```

```
var button = querySelector('#confirm');
button.text = 'Confirm';
button.classes.add('important');
button.onClick.listen((e) => window.alert('Confirmed!'));
```

Dart

- **Control flow statements**
  - if, else, for, while, do-while, etc.
  - Nothing special (very similar to Java)
- **Collections**
  - List, Set, Map, etc.
  - Nothing special (very similar to Java)
- **Class definition**
  - Neither main() nor Bicycle is declared as public, because all identifiers are public by default.
  - Dart doesn't have keywords for public, private, or protected.
  - “new” is optional (to create an instance)



```dart
class Bicycle {
  int cadence;
  int _speed = 0;
  int get speed => _speed;
  int gear;

  Bicycle(this.cadence, this.gear);

  void applyBrake(int decrement) {
    _speed -= decrement;
  }

  void speedUp(int increment) {
    _speed += increment;
  }

  @override
  String toString() => 'Bicycle: $_speed mph';
}

void main() {
  var bike = Bicycle(2, 1);
  print(bike);
}
```

[Java equivalent of this class]

Dart

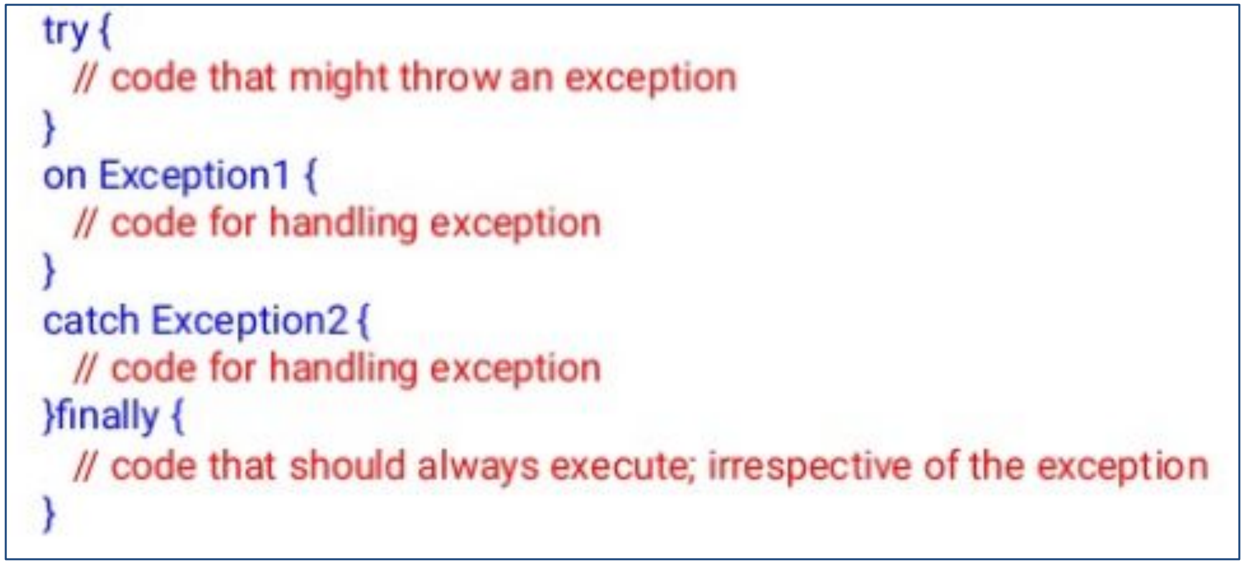
- **Class definition (link)**
    - Uninitialized variables (even numbers) have the value null
    - Privacy for any identifier prefixed with an underscore character.
    - By default, Dart provides implicit getters and setters for all public instance variables.
    - Constructor without a body is a valid definition

```
Bicycle(this.cadence, this.speed, this.gear);
```

```
Bicycle(int cadence, int speed, int gear) {
  this.cadence = cadence;
  this.speed = speed;
  this.gear = gear;
}
```

Dart

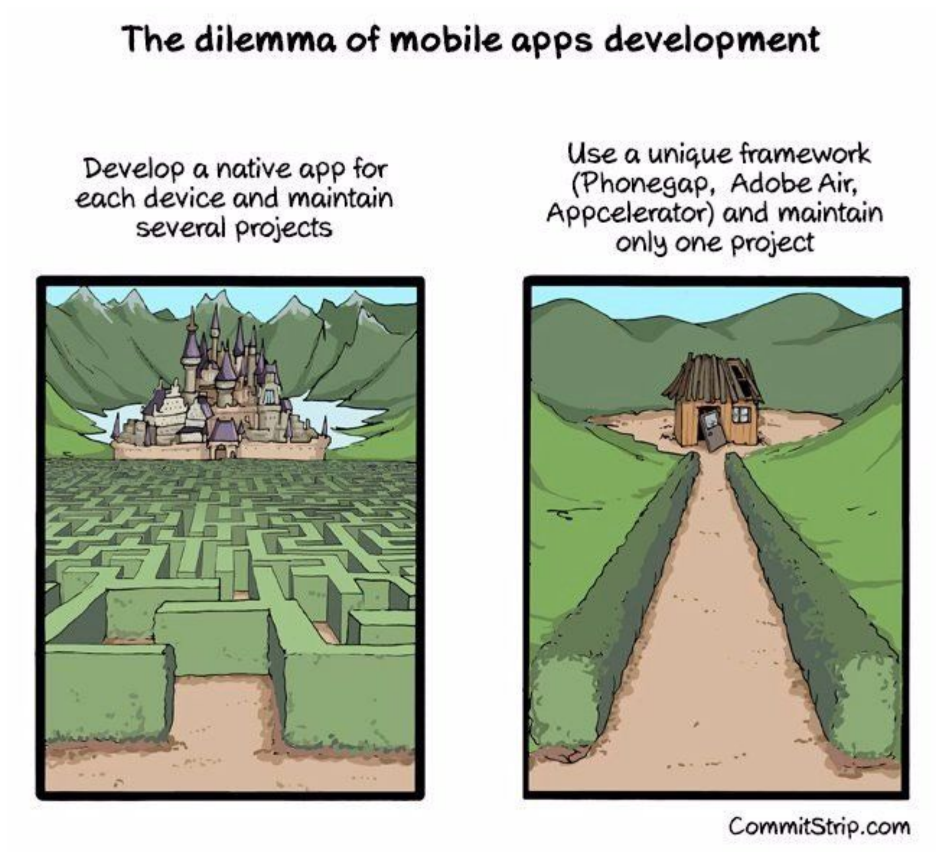
- **Function definition**
    - You can define top-level functions
    - "=>" can be used as a syntactic sugar ([example]())
- **Exception Handling**
    - syntax



```
try {
   // code that might throw an exception
}
on Exception1 {
   // code for handling exception
}
catch Exception2 {
   // code for handling exception
}finally {
   // code that should always execute; irrespective of the exception
}
```

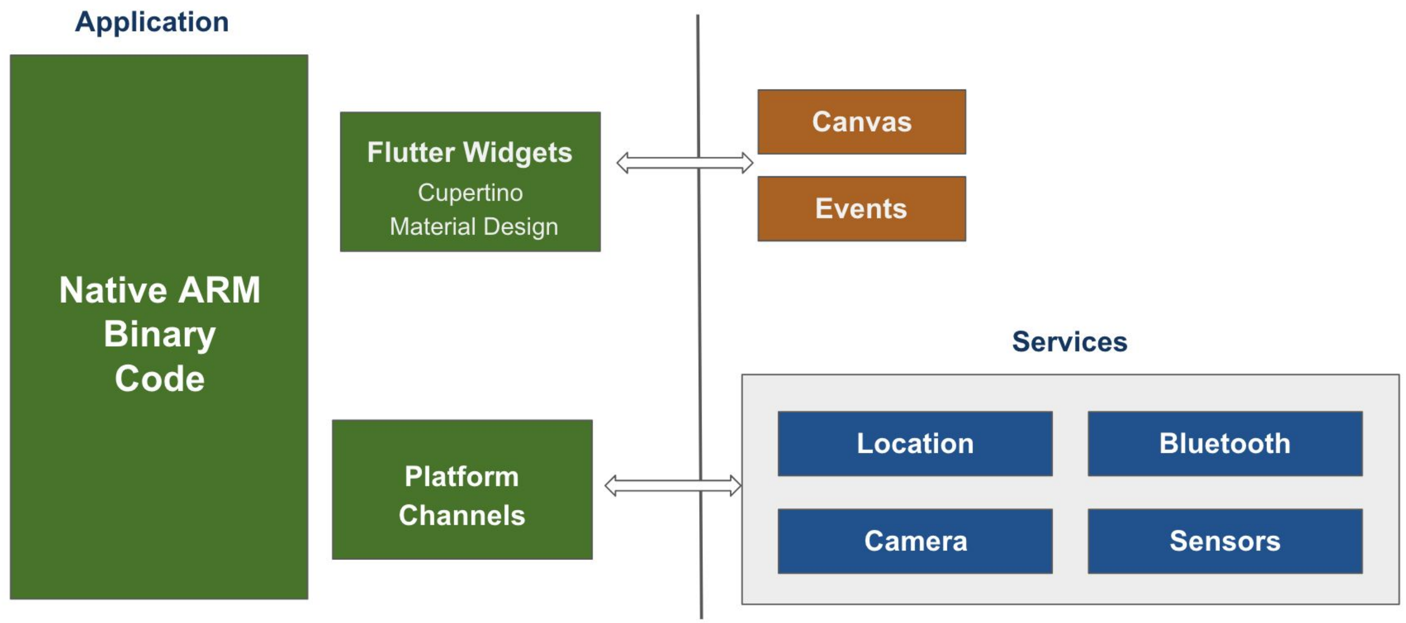
Flutter

- **Motivation**
  - Open-source UI software development kit created by Google
  - Motto: “Build beautiful native apps in record time”


The dilemma of mobile apps development
Develop a native app for each device and maintain several projects
Use a unique framework (Phonegap, Adobe Air, Appcelerator) and maintain only one project
CommitStrip.com

Flutter

- **Components**
  - **Flutter Engine :** High performance 2D graphics engine
  - **Foundation Library:** All the basic building blocks (link)
  - **Widgets:** An immutable description of part of the UI (link)
  - **Design Specific Widgets:** Two sets of widgets (Material Design for Android side and Cupertino Style for iOS side)

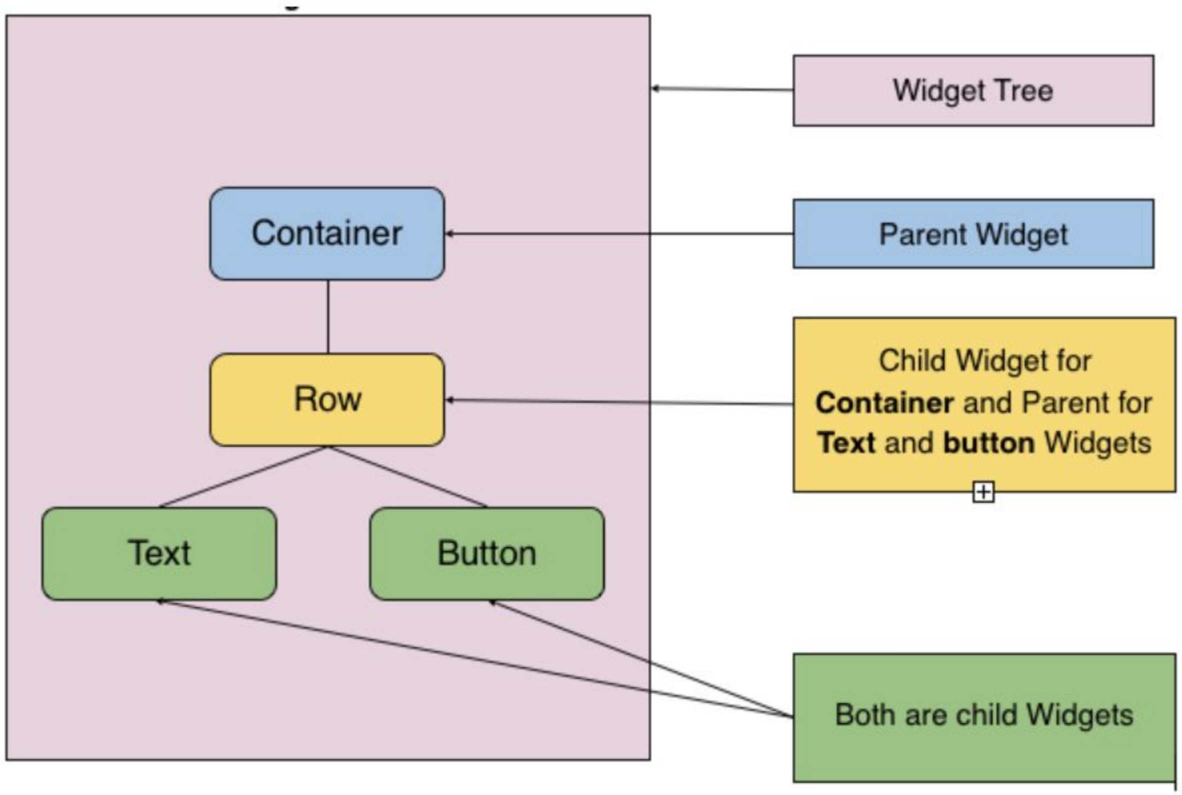


https://medium.com/gradeup/mobile-development-approaches-and-flutter-architecture-flutter-part-i-a7e08838c97a

Flutter

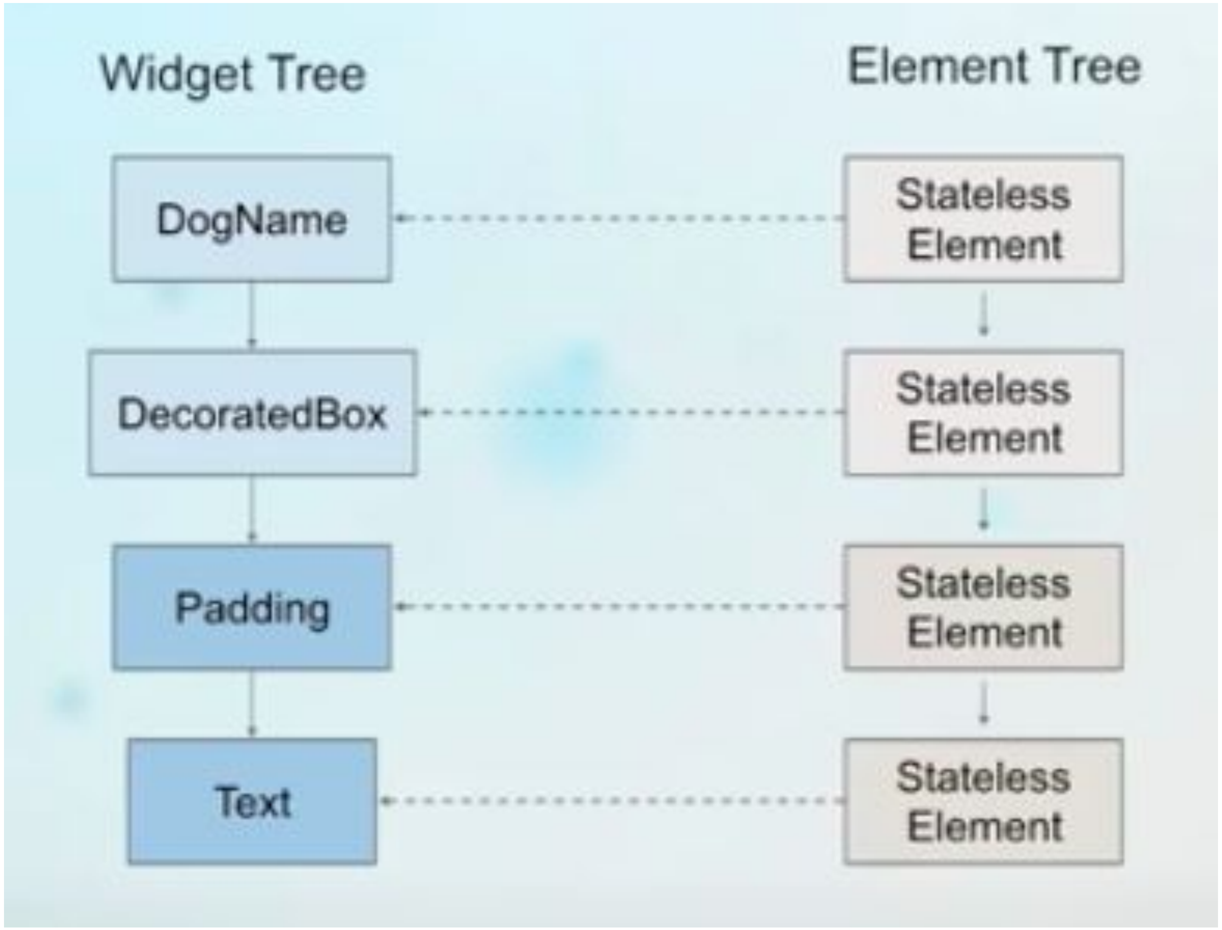
- **Widgets**

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return ...
    }
}
```


Widget Tree
Container
Parent Widget
Row
Child Widget for **Container** and Parent for **Text** and **button** Widgets
Text
Button
Both are child Widgets

Flutter

- **Widgets**



Flutter

- **Stateless Widgets**

```
class ItemCounter extends StatelessWidgets {
    final String name;
    final int count;

    ItemCounter({this.name, this.count});

    @override
    Widget build(BuildContext context) {
        return Text('$name: $count')
    }
}
```

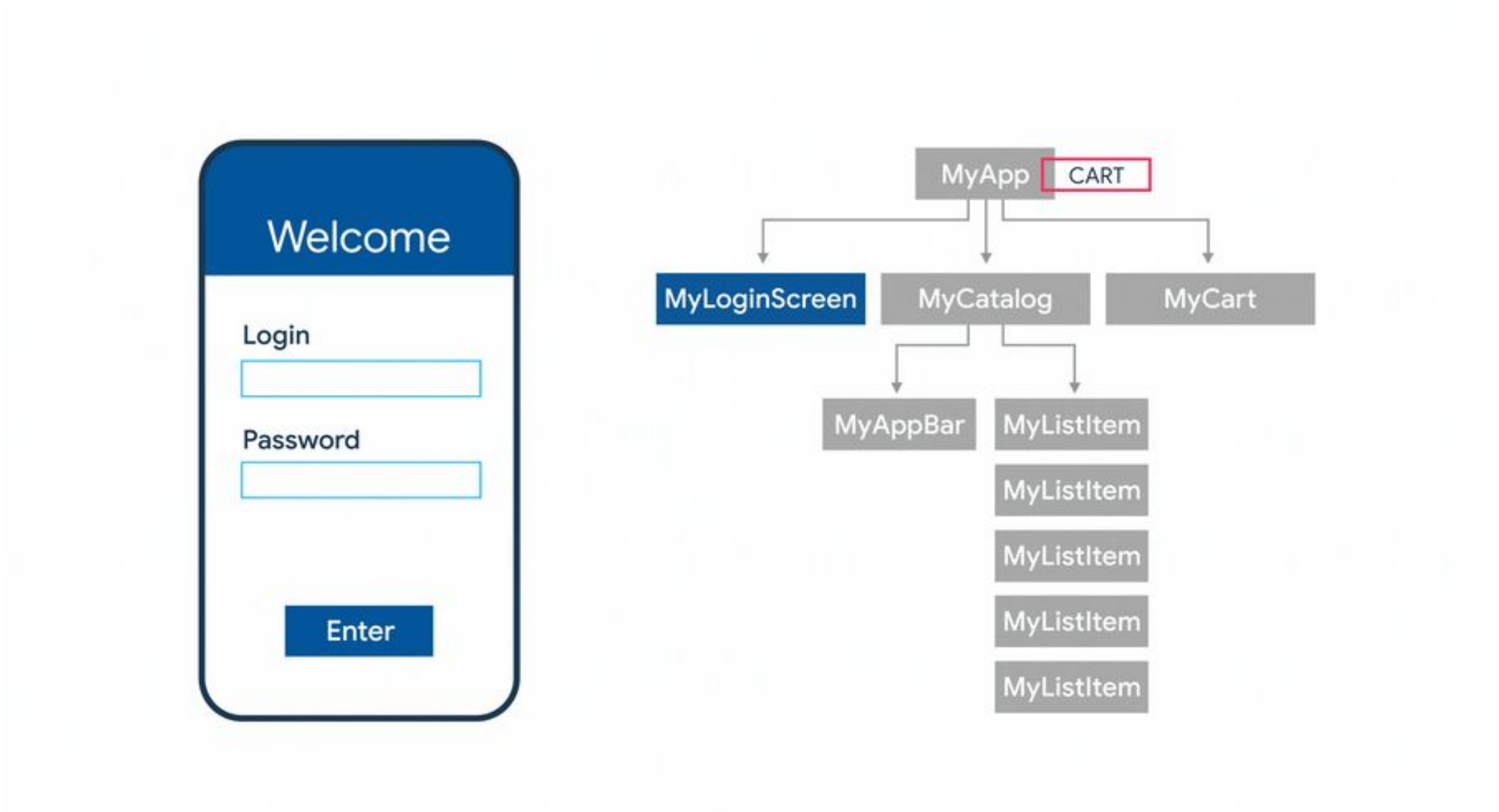
Flutter

- **Stateful Widgets**

```
class RandomWords extends StatefulWidget {
  @override
  _RandomWordsState createState() => _RandomWordsState();
}

class _RandomWordsState extends State<RandomWords> {
    @override
    Widget build(BuildContext context) {
      final wordPair = WordPair.random();
      return Text(wordPair.asPascalCase);
    }
  }
```

Flutter

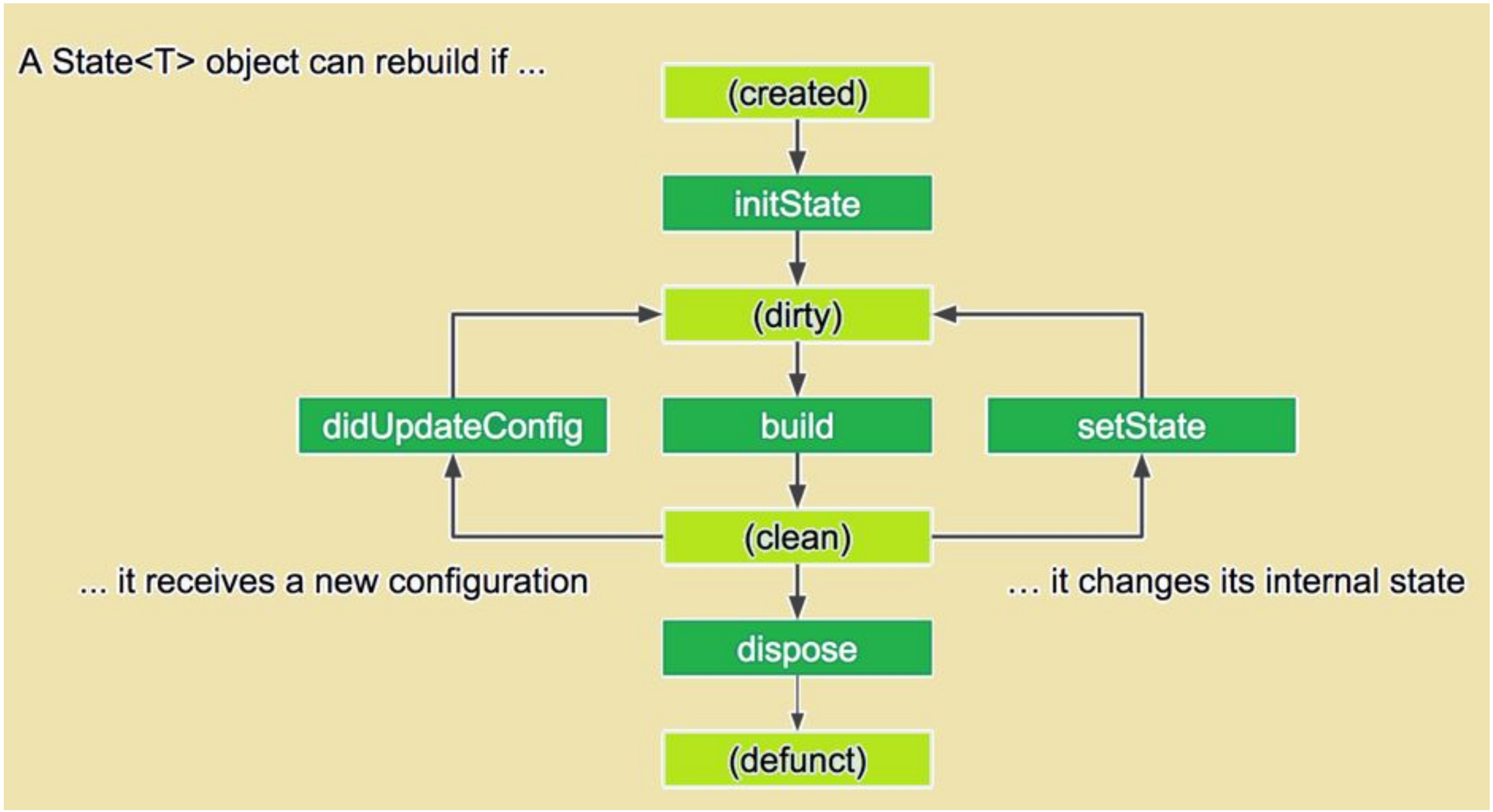
- **Widgets**



https://flutter.dev/docs/development/data-and-backend/state-mgmt/intro

Flutter

- **Widgets & states**
  - Only stateful widget can hold a state


A State<T> object can rebuild if ...
(created)
initState
(dirty)
didUpdateConfig
build
setState
(clean)
... it receives a new configuration
... it changes its internal state
dispose
(defunct)

https://www.developerlibs.com/2019/12/flutter-lifecycle-widgets.html

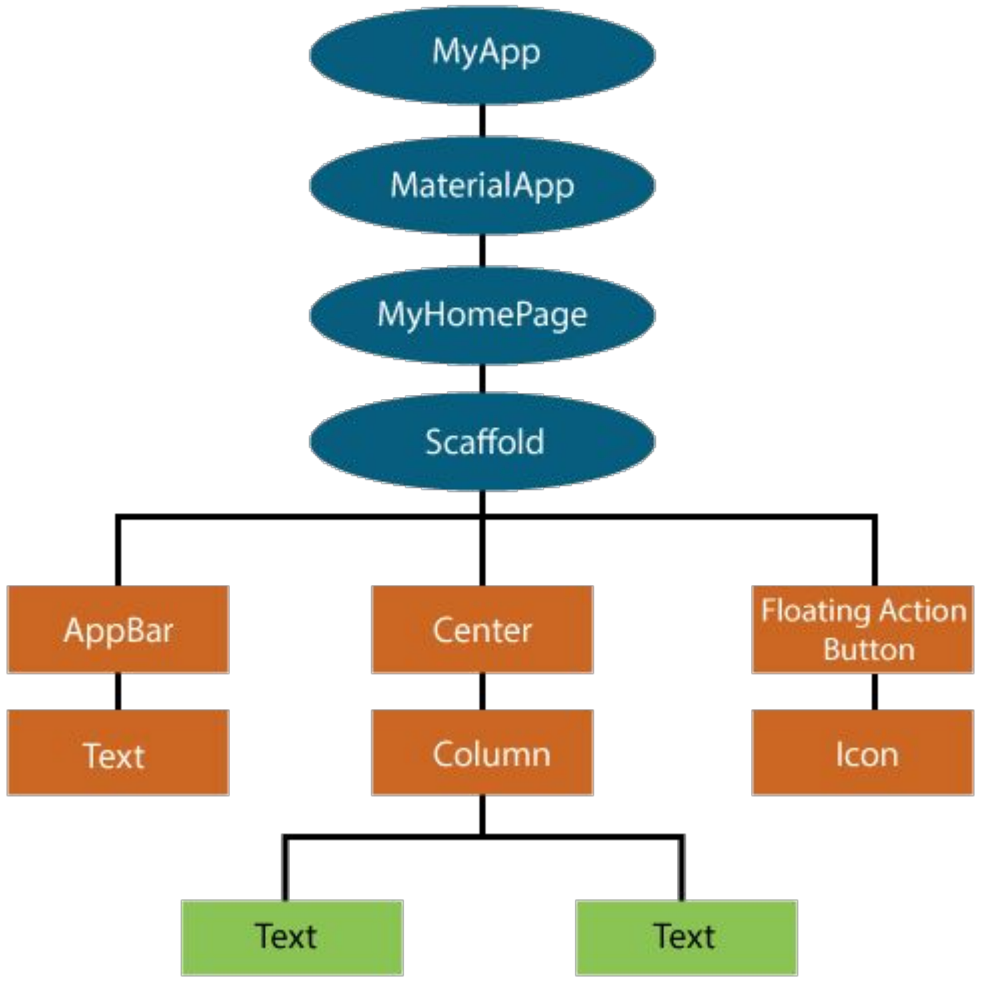
Flutter

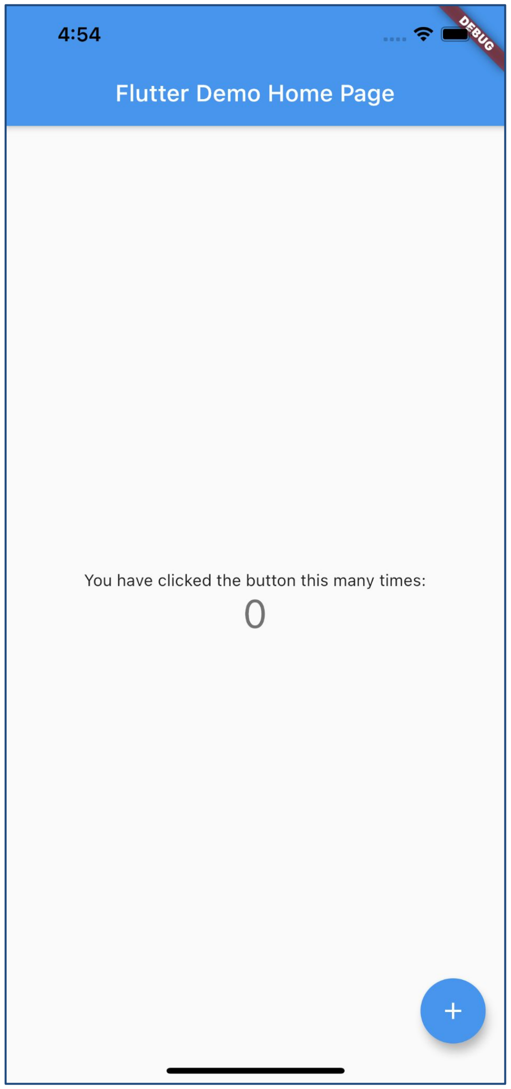
- **Widgets & states**
  - Almost everything is a widget in Flutter
  - Although it's convenient, it's not recommended to put an API call in a build() method (use initState or Constructor).

- **Async & await**
  - `Future` is a core Dart class for working with async operations.
  - The `http.Response` class contains the data received from a successful http call
  - "await" is used to wait for the result of async call

Flutter

- **Demo**





You have clicked the button this many times:

0

Q/A