What is Flutter

•••

Mera presentation

Overview

- Google - owner and maintainer
- Dart - language
- Flutter architecture - layers
- Code reusable for all supported platforms, up to 95%
- New technology - active development
- Widgets - everything's a Widget
- Business - reduce cost of deliverable product
- Documentation - excellent documentation from Google

What is Flutter ?

- Multi-platform development SDK for mobile (Android & iOS) and other platforms
- High performance
- Based on Dart
- Inspiration from React
- First alpha version late 2015
- Open source (can be ported to any platform with any programming language)
- Custom 2D UI-rendering engine (not used native platform components)
- Access to, and control over, all layers of host system

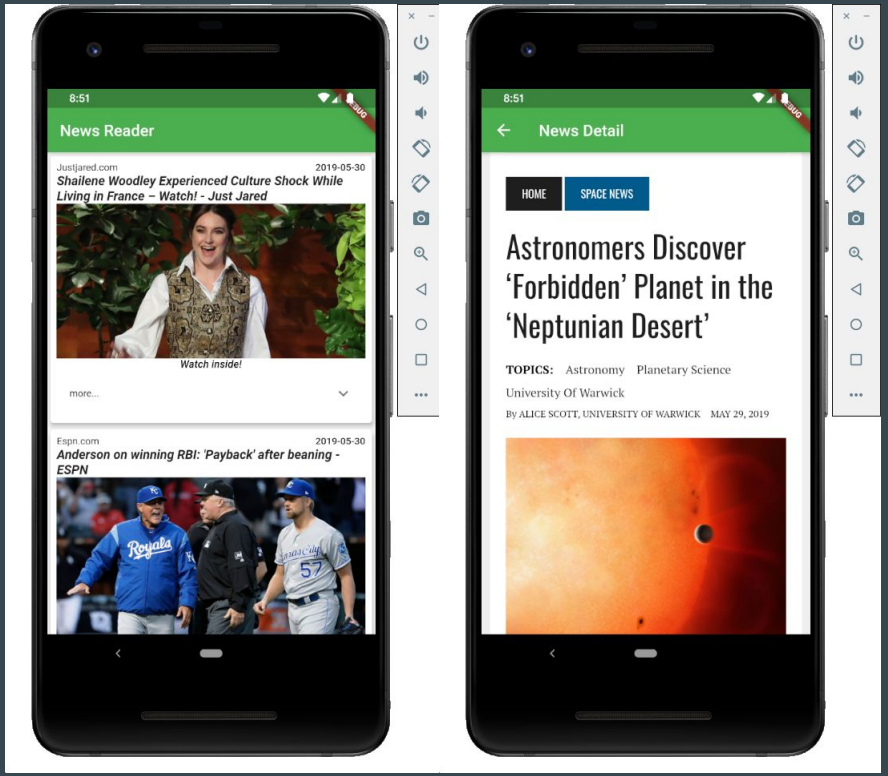
Features

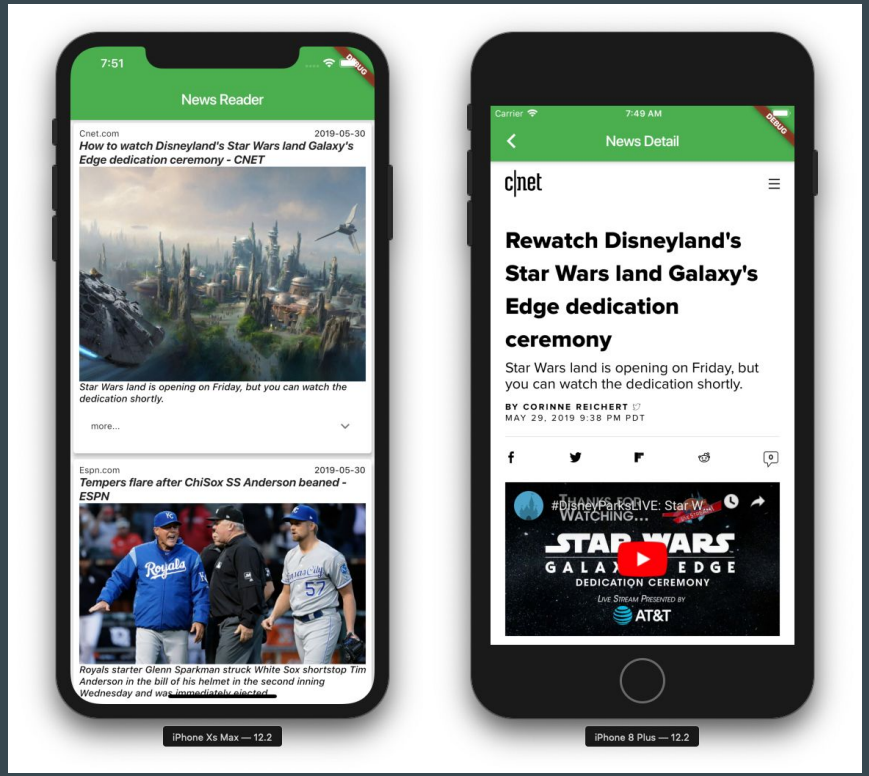
- Declarative
- A functional-reactive framework (There is RxDart, but it's for convenience only)
- Hot Reload
- Async/await
- Everything is Widget
- Rendering engine - optimized Chrome without html+css, faster >20 times
- Unit tests for UI

Technology limitations

- Android: Jelly Bean 4.1.x or newer, API v16
- iPhone: iOS 8 or newer
- Android emulator / iOS simulator simultaneously
- Performance: constant 60 fps
- Can be ported to any platform/system able to compile C++ engine and Dart tools
- (Fuchsia OS, Project Z, macOS, Windows, Linux)
- Desktop, web, embedded - under development

Flutter News Reader





Editor or IDE

IntelliJ Idea

Android Studio

Visual Studio Code

Console with any other editor

Flutter Distribution Channels

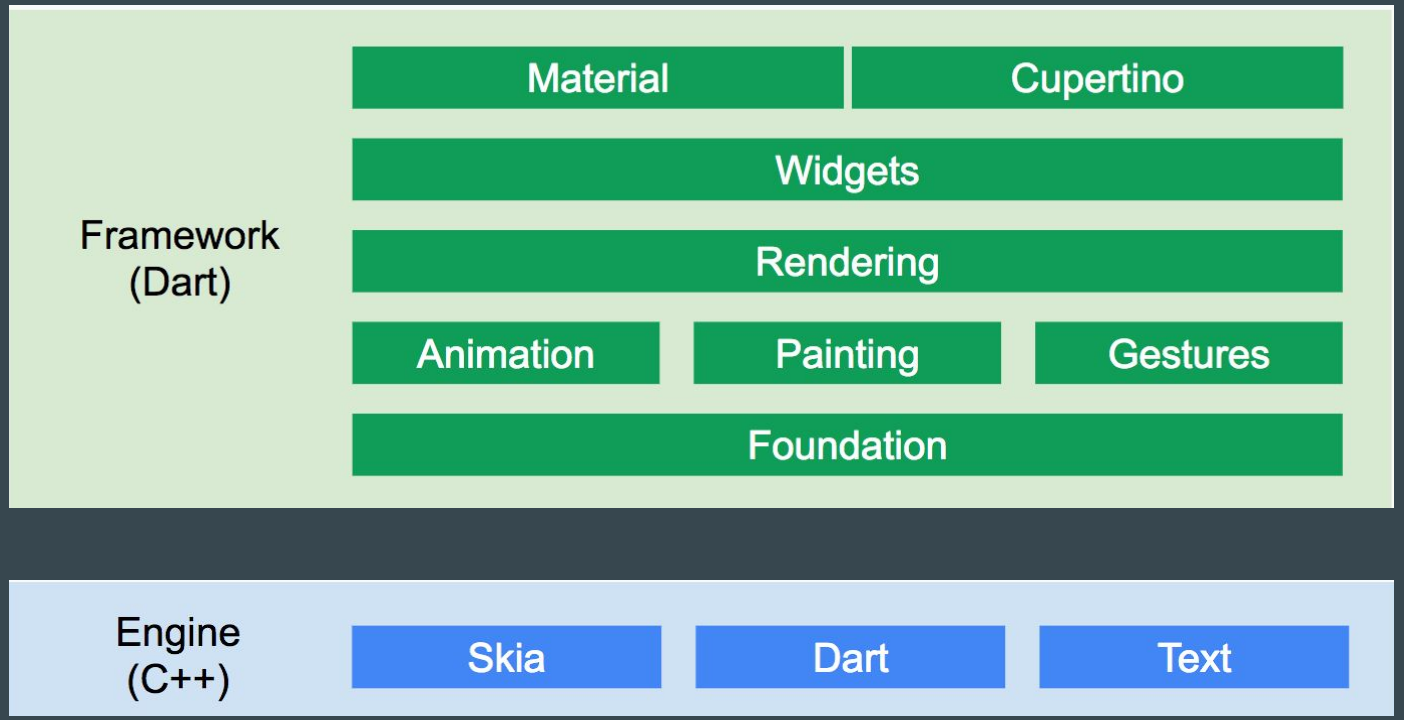
- Flutter doctor
- Flutter release channels

Flutter has four release channels:

- stable
- beta
- dev
- master

Google recommends using the stable channel unless you need a more recent release.

Flutter architecture



Everything's a Widget

Widget is an immutable declaration of part of the user interface.

Unified object model: the Widget, no any separate Views, ViewControllers, Layouts, Activities, Fragments, XML, Storyboards, xib.

A widget can define:

- a structural element (like a button or menu)
- a stylistic element (like a font or color scheme)
- an aspect of layout (like padding)

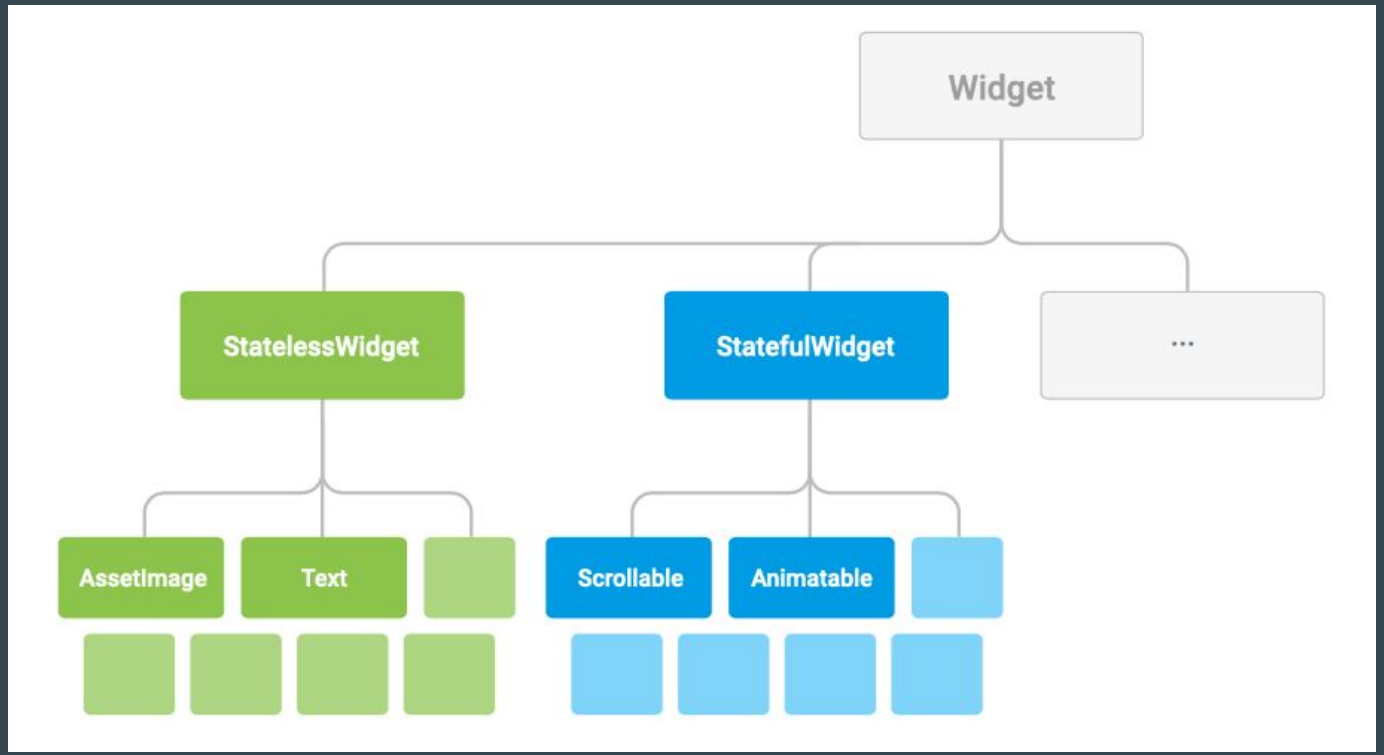
and so on...

Composition > inheritance

Widgets inside widgets

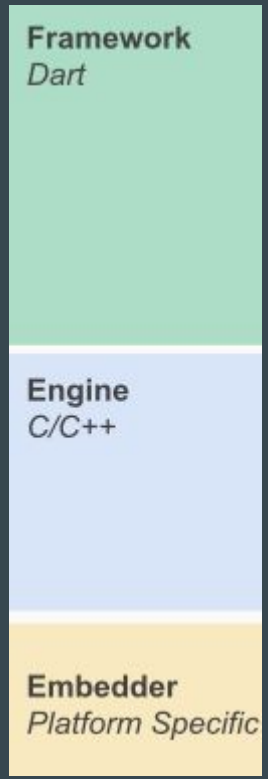
StateLess
immutable

StateFull
has State Object

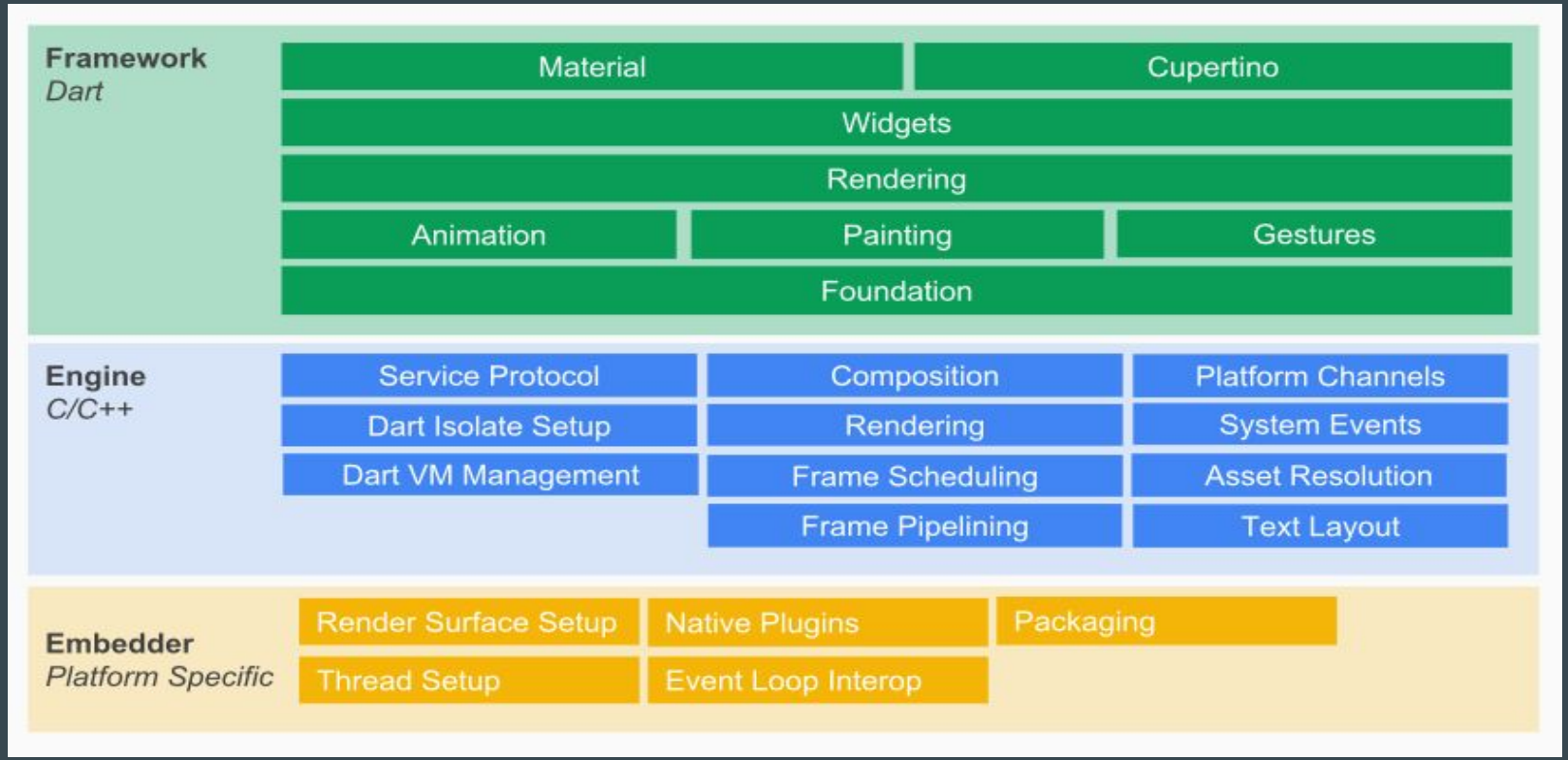


Layer structure of functionality

The Flutter framework is organized
into a series of layers,
with each layer building upon the previous layer.



Flutter system overview details



What is Dart ?

Dart is an object-oriented language using C-style syntax that transcompliles optionally into JavaScript. It is developed by Google and is used to build mobile, desktop, backend and web applications.

Dart

But why Dart, not Kotlin or Swift?

- Decision was in early 2015
- Google owns Dart
- Dart and Flutter teams coordination
- Dart VM optimizations
- Google moves toward a future without Java
- Fuchsia OS experiment
- 2015: Swift not open-source yet, Kotlin doesn't run on iOS

Different between Kotlin and Dart

- **No data classes** mean you're back to generating equals and hashCode for every single data class you have. For immutability you also have to either manually write copy method or write hefty boilerplate to let BuiltValue generates those for you. And boy it feels like forever to do either, while in Kotlin you only add data keyword to your class.
- **No extension functions** makes you unable to extend somewhat limited standard library.
- **Lack of null-safe types**. Kotlin makes you document whether you expect something to be nullable or not — Dart doesn't, and it simply adds a lot of mental overhead to deduce the nullability.
- **Sealed classes** — in Dart (same as Java) if you contain everything in a single type, you have to juggle boolean values to know whether you have a value or not, all while opening yourself to exceptions and/or null values. If you use multiple classes, you're forced to return dynamic and you lose all the compile-time safety.

Dart vs Kotlin vs Swift

Variables and Constants

Dart

```
// Implicit type
var myVariable = 5;

// Explicit type
String name = 'John';

// Dynamic type
dynamic name = 'John';

// Constants
final name = 'John';
const name = 'John';
```

Kotlin

```
// Implicit type
var myVariable = 5

// Explicit type
let name: String = "John"

// Dynamic type (not for JVM)
var dyn: dynamic = ...

// Constants
val name = "John"
```

Swift

```
// Implicit type
var myVariable = 5

// Explicit type
var name: String = 'John'

// Dynamic type
N/A

// Constants
let name = "John"
```

Dart vs Kotlin vs Swift

Basic types

Dart

```dart
// String -> int
var one = int.parse('1');

// int -> String
String oneAsString = 1.toString();

// Boolean
bool boolValue = false

// Enum
enum Color { red, green, blue }

// Type aliases
N/A

// Check type
if (a is String)...

// Type casting
Movie movie = item as Movie
```

Kotlin

```kotlin
// String -> Int
var one = "1".toIntOrNull()

// Int -> String
var oneAsString = 1.toString()

// Boolean
var boolValue: Boolean = false

// Enum
enum class Color { RED, GREEN, BLUE }

// Type aliases
typealias NodeSet = Set<Network.Node>

// Check type
if (a is String)...

// Downcasting
val movie = item as? Movie
```

Swift

```swift
// String -> Int, 1 or nil
let one = Int("1")

// Int -> String
let oneAsString = String(1)

// Boolean
var boolValue: Bool = false

// Enum
enum Color { case red, green, blue }

// Type aliases
typealias AudioSample = UInt16

// Check type
if a is String...

// Downcasting
let movie = item as? Movie
```

Dart vs Kotlin vs Swift

Collection

Dart

```
// Arrays / List
var emptyList = <int>[];
List fixedLengthList = new List(3);
List growableList = new List();
var myList = [1,2,3];

// Sets
var emptySet = Set<String>();
mySet = Set<String>.from(['1', '2']);

// Maps
var emptyMap = Map<String, Int>();
var myMap = {'key1': value1, 'key2',
value2};
```

Kotlin

```
// Arrays / List
val emptyArray = arrayOf<String>()
val emptyList = listOf<String>()
val myArray = arrayOf("1", "2", "3")

// Sets
val emptySet = setOf<String>()
val mySet = setOf("1", "2"")

// Maps
val emptyMap = emptyMap<String, Int>()
val myMap = mutableMapOf("key1" to
"value1", "key2" to "value2")
```

Swift

```
// Arrays
var emptyArray = [Int]()
var myArray = [“1”, “2”, “3”]

// Sets
var emptySet = Set<String>()
var mySet = Set<String>(["1", "2"])

// Maps
var emptyMap = [String: Int]()
var myMap = ["key1": "value1", "key2":
"value2"]
```

Dart vs Kotlin vs Swift

Nullability & Optionals

Dart

```
// Declaration
int id = null;
id.abs(); // Exception
id?.abs(); // Safe call


// Null - aware operators
int id = null;
var userId = id ?? -1; // prints -1

// Optionals
int id;
var userId = id ?? -1; // prints -1
```

Kotlin

```
// Declaration
var id: Int? = null
id.inc() // Compilation error
id!!.inc() // Exception
id?.inc() // Safe call


// Elvis operator
val id: Int? = null
var userId = id ?: -1 // prints -1

// Optionals
val id: Int?
var userId = id ?: -1 // id must be
initialized
```

Swift

```
// Declaration
var id: Int? = nil
id.signum() // Compilation error
id?.signum() // Safe call


// nil-coalescing operator
let id: Int? = nil
var userId = id ?? -1 // prints -1

// Optionals
var id: Int? // optional
var id: Int = 1 // non-optional, must
be initialized
```

Dart vs Kotlin vs Swift

Functions

Dart

```dart
// Declaration
bool greet(String name) {
  return "Hey ${name}!";
}

// Optional named parameters
void greet(name: String) {}

// Named parameters
void greet({name: String}) {}


// Default parameters
void greet({String name = “John”}) {}

// Var args
N/A

// Tuple return
N/A, you can return array


// Closure / lambda
var square = (int x) => x * x;
print(square(4)); // prints 16


// Extensions
N/A
```

Kotlin

```kotlin
// Declaration
fun greet(name: String): String {
  return "Hey ${name}!"
}

// Optional named parameters
fun greet(name: String): String {}

// Named parameters
fun greet({name: String}) {}
greet(“John”)
greet(name = “John”)


// Default parameters
fun greet(name: String = “John”) {}

// Var args
fun greet(vararg names: String...) {}

// Tuple return
N/A, return class instead


// Closure / lambda
val square = { x:Int -> x * x }
print(square(4)) // prints 16


// Extensions
fun Int.square(): Int {
  return this * this
}

println(4.square()) //prints 16
```

Swift

```swift
// Declaration
func greet(name: String) -> String {
  return "Hey \(name)!"
}

// Optional named parameters
fun greet(_ name: String) {}

// Named parameters
fun greet(name: String) {}


// Default parameters
fun greet(name: String = “John”) {}

// Var args
fun greet(_ names: String...) {}

// Tuple return
func getRates() -> (Int, Int, Int) {
    return (15, 20, 30)
}

// Closure / lambda
var square = { (x: Int) -> Int in
  return x * x
}
print(square(4)) // prints 16


// Extensions
extension Int {
  func square() -> Int {
    return self * self
  }
}

print(5.square()) //prints 16
```

Dart vs Kotlin vs Swift

Classes

Dart

```dart
// Declaration
class Square {
  var color = 0
  String greet(name: String) {
      return "Hello $name"
  }
}

// Inheritance
class Employee extends Person {}

// Data class or struct
N/A, use regular class declaration


// Interface & Protocol
abstract class Nameable {
  String name() {}
}

void call<T extends Nameable>(T user) {
    print("Name is " + user.name());
}
```

Kotlin

```kotlin
// Declaration
class Square {
  var color = 0
  fun greet(name: String)
      = "Hello $name"
}

// Inheritance
class MyActivity: AppCompatActivity()

// Data class (reference type)
data class User(var name:String)

var user1 = User("John")
var user2 = user1
user1.name = "Bob"
print(user2.name) // prints Bob


// Interface & Protocol
interface Nameable {
    fun name(): String
}

fun call<T: Nameable>(user: T) {
    println("User is " + user.name())
}
```

Swift

```swift
// Declaration
class Square {
  var color = 0
  func greet(name: String) -> String {
      return "Hello \(name)"
  }
}

// Inheritance
class Controller: UIViewController {}

// Struct (value type)
struct User {
 var name: String
}

var user1 = User(name: "John")
var user2 = user1
user1.name = "Bob"
print(user2.name) // prints John

// Interface & Protocol
protocol Nameable {
    func name() -> String
}

func call<T: Nameable>(user: T) {
    print("User is " + user.name())
}
```

What is inside the Flutter SDK?

- Heavily optimized, mobile-first 2D rendering engine with excellent support for text
- Modern react-style framework
- Rich set of widgets for Android and iOS
- APIs for unit and integration tests
- Interop and plugin APIs to connect to the system and 3rd-party SDKs
- Headless test runner for running tests on Windows, Linux, and Mac
- Command-line tools for creating, building, testing, and compiling your apps

Compilers

Just In Time (interpreteur for development)

Ahead Of Time (byte-codes for production and small executable size)

JavaScript

OpenSource, Flutter team will accept any other compilers for any platforms

Dart tooling

- Dynamically typed, since Dart 2 - static typing
- Dart team try to simplify, and make Dart like Kotlin and Swift
- Pub - package manager
- Build in linter
- dartfmt - Dart code formatter
- Dart Analyzer - static code analyzer
- Dart Observatory - debugger in Chrome

Dart language features

- Everything is an object, even numbers, functions, nulls are objects

```
double a = 3.2;
var floor = a.floor();
(floor + 2).abs();
```

- Top level functions without classes, nested functions, unnamed functions

```
void email() {
  emailTransport.send(envelope)
  .then((success) => 'Email sent!')
  .catchError((e) => 'Error occured');
}
```

- Generics

```
class List<T> {
  T _head;

  T get head {
    return _head;
  }
}
```

Dart language features

- Interfaces (protocols), abstract classes

```dart
//class class_name implements interface_name
class Person {

  void walk() {
    print("Person can walk");
  }

  void talk() {
    print("Person can talk");
  }
}
class Jay implements Person {

  @override
  void walk() {
    print("Jay can walk");
  }

  @override
  void talk() {
    print("Jay can talk");
  }
}
```

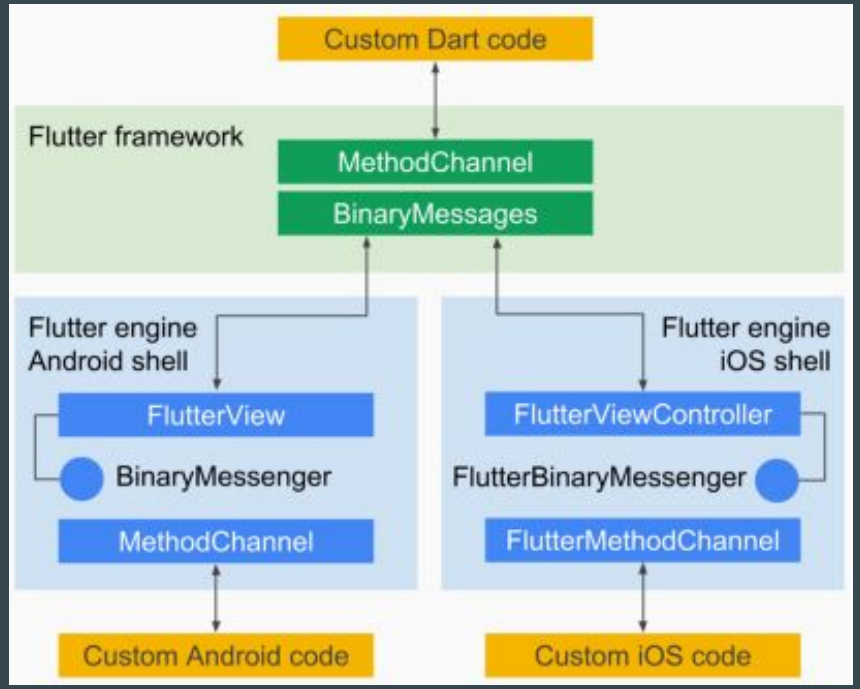
- Mixins

```dart
class AB extends P with A, B {}
class BA extends P with B, A {}
```

- Future

```dart
Future<void> printDailyNewsDigest() async {
  try {
    var newsDigest = await gatherNewsReports();
    print(newsDigest);
  } catch (e) {
    // Handle error...
  }
}
```

Flutter Platform Channels



Flutter Platform Channels (example)

```
// Invocation of platform methods, simple case.
// Dart side.
const channel = MethodChannel('foo');
final String greeting = await channel.invokeMethod('bar',
'world');
print(greeting);
```

```
// iOS side.
let channel = FlutterMethodChannel(
  name: "foo", binaryMessenger: flutterView)
 channel.setMethodCallHandler {
  (call: FlutterMethodCall, result: FlutterResult) -> Void in
            switch (call.method)
            {
              case "bar": result("Hello, \(call.arguments as! String)")
              default: result(FlutterMethodNotImplemented)
            }
}
```

```
// Android side.
val channel = MethodChannel(flutterView, "foo")
channel.setMethodCallHandler { call, result ->
  when (call.method) {
    "bar" -> result.success("Hello, ${call.arguments}")
    else -> result.notImplemented()
  }
}
```

Fuchsia OS

- No any official Google announcement
- Microkernel OS without Linux
- No app store yet
- No device fragmentation
- No any legacy code
- Modern architecture
- No public release date
- Influence on Android

Links

My blog:
https://svasilevkin.wordpress.com/

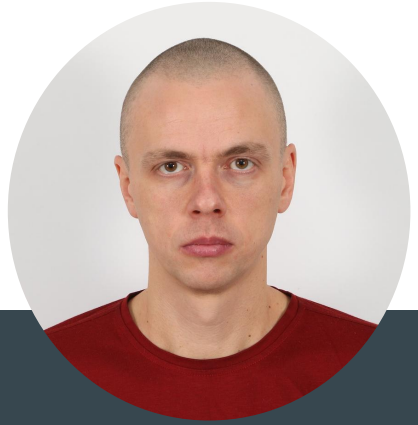
GitHub Flutter News Reader
https://github.com/vasilevkin/flutter_news_reader

GitHub Awesome Flutter

https://github.com/Solido/awesome-flutter

Twitter: Sergey Vasilevkin
@vasilevkin

The Team



Sergey Vasilevkin

Mobile apps developer

iOS developer, Mera

Averin Andrey

Mobile apps developer

iOS developer, Mera