Day 1:
Welcome to Mobile Application

Mobile App Introduction

A **mobile application** (or mobile app) is a software program designed to run on mobile devices such as smartphones and tablets.
Mobile apps are typically downloaded from app stores and can provide a wide range of functionalities, from productivity tools to entertainment and social networking.

1. **Faster Development:** Build and maintain just one codebase for all platforms, speeding up the development process.
2. **Lower Costs:** Reduce expenses by managing a single project instead of separate ones for each platform.
3. **Consistency:** Deliver a uniform user interface and consistent features across all platforms.

Mobile App Development Frameworks

- **Flutter:** By Google, uses Dart for fast, expressive UIs and native performance.
- **React Native:** By Meta, uses JavaScript/React to build native-like apps.
- **Xamarin:** By Microsoft, uses C# for cross-platform apps within .NET.
- **FireMonkey:** By Embarcadero, uses Delphi for visually rich, multi-platform apps.

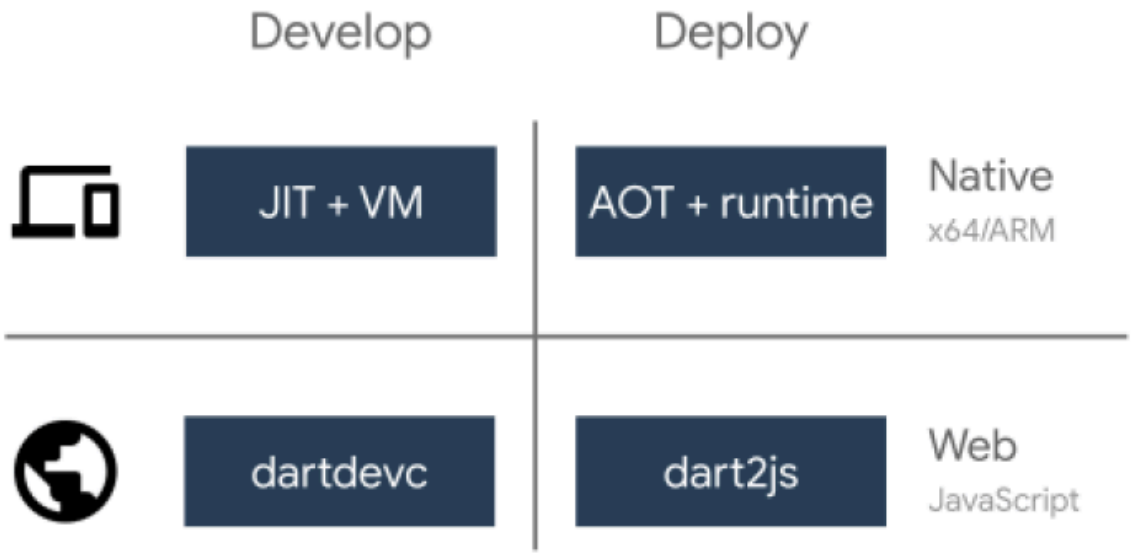
Dart Introduction

Supported Platforms:

- **Desktop & Mobile:** Run Dart applications natively on Windows, macOS, Linux, Android, and iOS.
- **Web:** Compile Dart code to JavaScript for seamless browser support.
- **Stand-alone:** Execute Dart programs directly from the command line.

AOT (Ahead-Of-Time) compilation compiles Dart code to native machine code before execution, resulting in faster startup times and improved performance for production applications.

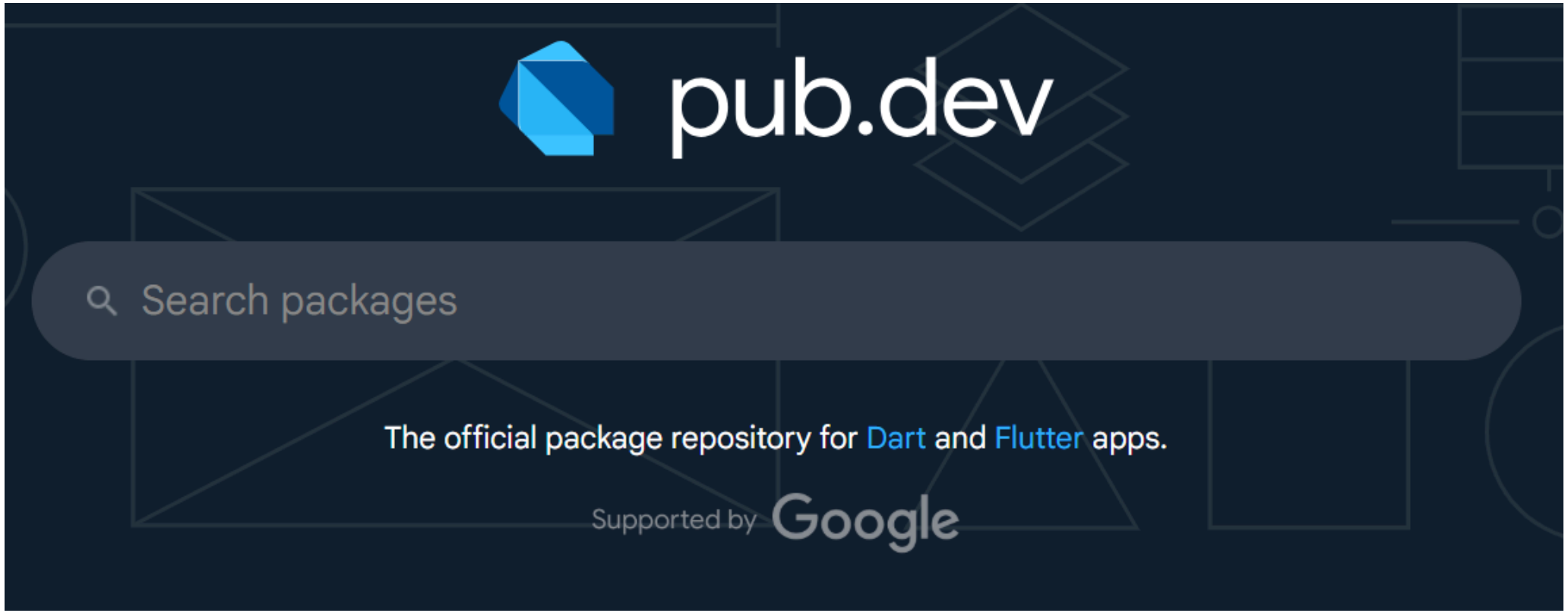
- APK (Android Package)
- AAB (Android App Bundle)
- IPA (iOS App Store Package)



- **JIT (Just-In-Time)** compilation allows for rapid development and debugging by compiling code on the fly.
- **VM (Virtual Machine)** execution provides a platform-independent environment for running Dart code efficiently.
- **AOT (Ahead-Of-Time)** compilation improves performance by compiling Dart code to native machine code before execution.

Dart Package system

Dart uses a package manager called "pub" to manage libraries and dependencies.



For more information, visit the https://dart.dev/guides.

Dart Hello World

```dart
void main() {
  print('Hello, World!');
}
```

The `void main()` is the entry point of a Dart application.
The `print` function outputs text to the console.

Dart Basic Syntax

Comments are used to explain code and improve readability.

```
// This is a single-line comment
/* This is a
   multi-line comment */
```

Variable is used to store data values.

```
// Variable declaration
int age = 25;
String name = 'John Doe';
bool isStudent = true;
double height = 5.9;
```

Constants are fixed values that cannot be changed during program execution.

```
// Constant declaration
const double pi = 3.14; // const must be known at compile-time
final String country = 'USA'; // final can be set only once
```

Array is a collection of items stored at contiguous memory locations.

```
// Array declaration
List<int> numbers = [1, 2, 3, 4, 5];
List<String> fruits = ['Apple', 'Banana', 'Orange'];
// Accessing array elements
void main() {
  print(numbers[0]); // Output: 1
  print(fruits[1]); // Output: Banana
}
```

Enumerated is a special data type that consists of a set of named values called elements or members.

```
// Enum declaration
enum Day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }
// Using enum
void main() {
  Day today = Day.Monday;
  if (today == Day.Monday) {
    print('Start of the week!');
  }
}
```

Arithmetic operators are used to perform mathematical operations.

```
// Arithmetic operations
int sum = a + b; // Addition
int difference = a - b; // Subtraction
int product = a * b; // Multiplication
double quotient = a / b; // Division
int remainder = a % b; // Modulus
int intQuotient = a ~/ b; // Integer Division
```

Relational operators are used to compare two values.

```
// Relational operations
bool isEqual = a == b; // Equal to
bool isNotEqual = a != b; // Not equal to
bool isGreater = a > b; // Greater than
bool isLess = a < b; // Less than
bool isGreaterOrEqual = a >= b; // Greater than or equal to
bool isLessOrEqual = a <= b; // Less than or equal to
```

Logical operators are used to combine multiple boolean expressions.

```
bool result = (a > b) && (c < d); // Logical AND
bool result = (a > b) || (c < d); // Logical OR
bool result = !(a > b); // Logical NOT
```

Type test operators are used to check the type of an object.

```
bool isString = obj is String; // Check if obj is a String
String str = obj as String; // Cast obj to String
```

Bitwise operators are used to perform bit-level operations.

```
int andResult = a & b; // Bitwise AND
int orResult = a | b; // Bitwise OR
int xorResult = a ^ b; // Bitwise XOR
int notResult = ~a; // Bitwise NOT
int leftShift = a << 2; // Left shift
int rightShift = a >> 2; // Right shift
```

Nullable types are types that can hold a null value, indicating the absence of a value.

```
int? nullableNumber = null;
String? nullableString = null;

void main() {
  print(nullableNumber); // Output: null
  print(nullableString); // Output: null
}
```

Bang operator (!) is used to assert that a nullable expression is non-null.

```
int? nullableNumber = null;
String? nullableString = null;

void main() {
  print(nullableNumber!); // Throws an error if nullableNumber is null
  print(nullableString!); // Throws an error if nullableString is null
}
```

Conditional statements are used to perform different actions based on different conditions.

```
// If-else statement
int number = 10;
if (number > 0) {
  print('Positive number');
} else if (number < 0) {
  print('Negative number');
} else {
  print('Zero');
}

// ternary operator
String result = (number % 2 == 0) ? 'Even' : 'Odd
```

Loop is used to repeat a block of code multiple times.

```
// For loop
for (int i = 0; i < 5; i++) {
  print('Iteration: $i');
}
// While loop
int count = 0;
while (count < 5) {
  print('Count: $count');
  count++;
}
// Do-while loop
int num = 0;
do {
  print('Number: $num');
  num++;
} while (num < 5);
```

Function is a reusable block of code that performs a specific task.

```
// Function declaration
int add(int a, int b) {
  return a + b;
}
// Function call
void main() {
    int sum = add(5, 10);
    print('Sum: $sum');
}
```

Named parameters allow you to specify arguments by name when calling a function.

```
// Function with named parameters
void greet({required String name, int age = 0}) {
  print('Hello, $name! You are $age years old.');
}
// Function call with named parameters
void main() {
  greet(name: 'Alice', age: 25);
  greet(name: 'Bob'); // age will use default value 0
}
```

Typedef is used to create an alias for a function type.

```
// Typedef declaration
typedef MathOperation = int Function(int a, int b);
// Function using typedef
int multiply(int a, int b) {
  return a * b;
}
// Function call using typedef
void main() {
  MathOperation operation = multiply;
  int result = operation(5, 10);
  print('Result: $result');
}
```

Annonymous function is a function without a name, often used as a callback.

```
var numbers = [1, 2, 3, 4, 5];
var doubledNumbers = numbers.map((n) { return n * 2; });
```

Lambda function is a concise way to represent a function.

```
var numbers = [1, 2, 3, 4, 5];
var squaredNumbers = numbers.map((n) => n * n);
```

Dart OOP Concepts

Class is a model or blueprint of code.

```
// Class declaration
class Person {
  String name; // Attribute
  Person(this.name); // Constructor
  void introduce() {   // Method
    print('Hello, my name is $name.');
  }
}
```

Object is an instance of a class.

```
Person person = Person('Alice');
person.introduce();
```

Encapsulation is the bundling of data and methods that operate on that data within a single unit, typically a class.

```
class BankAccount {
  String _accountNumber; // Private variable
  double _balance; // Private variable
  BankAccount(this._accountNumber, this._balance);
  double get balance => _balance; // Getter
  void deposit(double amount) {
    _balance += amount;
  }
}
```

Inheritance is a mechanism where a new class inherits the properties and behavior of an existing class.

```
class Animal {
  void speak() {
    print('Animal speaks');
  }
}
class Dog extends Animal {
  @override
  void speak() {
    print('Dog barks');
  }
}
```

Polymorphism is the ability of different classes to be treated as instances of the same class through a common interface.

```
class Shape {
  void draw() {
    print('Drawing a shape');
  }
}
class Circle extends Shape {
  @override
  void draw() {
    print('Drawing a circle');
  }
}
```

Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object.

```
abstract class Vehicle {
  void start();
  void stop();
}
class Car extends Vehicle {
  @override
  void start() {
    print('Car started');
  }
  @override
  void stop() {
    print('Car stopped');
  }
}
```

Interface is a contract that defines a set of methods that a class must implement.

```
abstract class Flyable {
  void fly();
}
class Bird implements Flyable {
  @override
  void fly() {
    print('Bird is flying');
  }
}
```

Exception Handling is used to handle errors gracefully.

```
void main() {
  try {
    int result = 10 ~/ 0; // This will throw an exception
    print('Result: $result');
  } catch (e) {
    print('Error: $e');
  } finally {
    print('Execution completed.');
  }
}
```

Assert statement is used to test a condition during development.

```
assert(number != null, 'Number must not be null');
```

Mixins is a way to reuse a class's code in multiple class hierarchies.

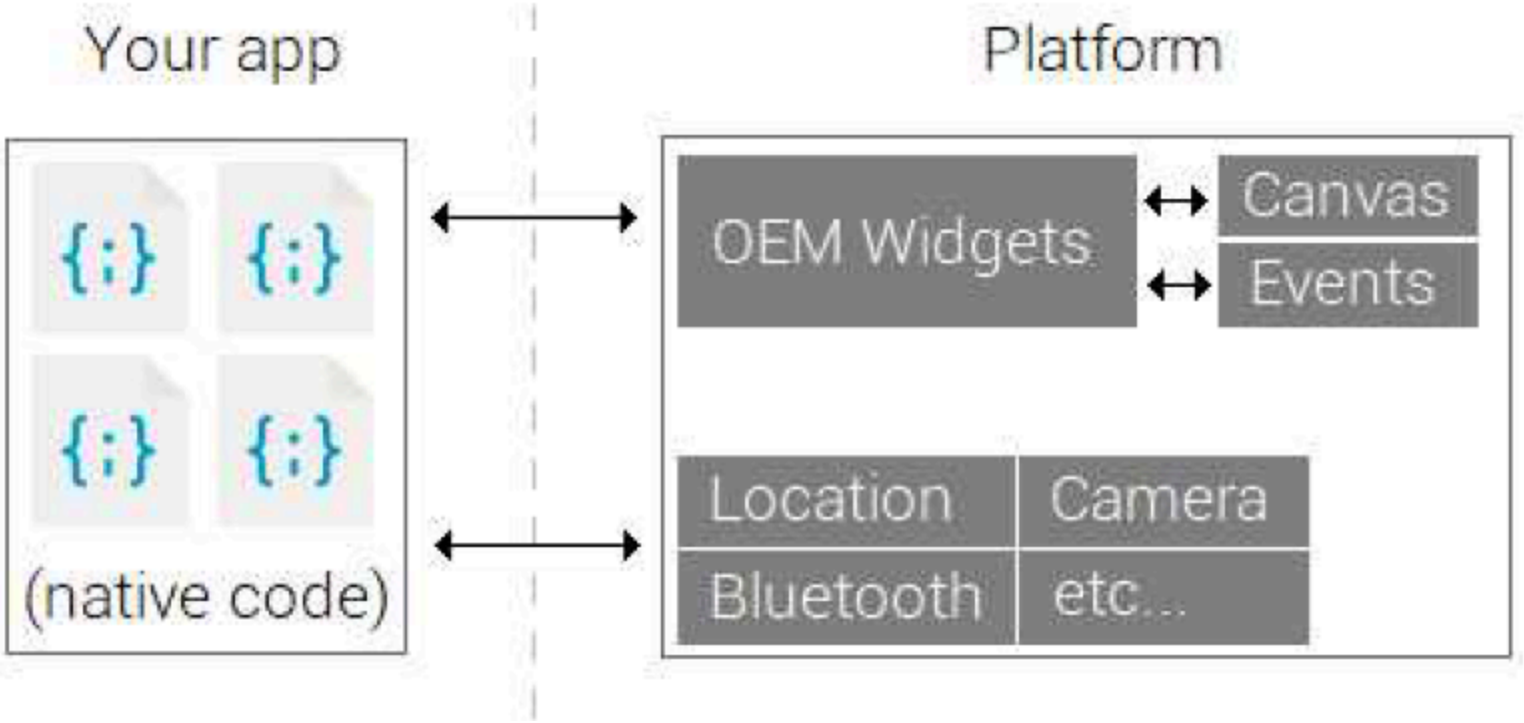
```
mixin Logger {
  void log(String message) {
    print('Log: $message');
  }
}
class Person with Logger {
  String name;
  Person(this.name);
  void introduce() {
    log('Introducing $name');
    print('Hello, my name is $name.');
  }
}
void main() {
  Person person = Person('Alice');
  person.introduce();
}
```

Flutter Introduction

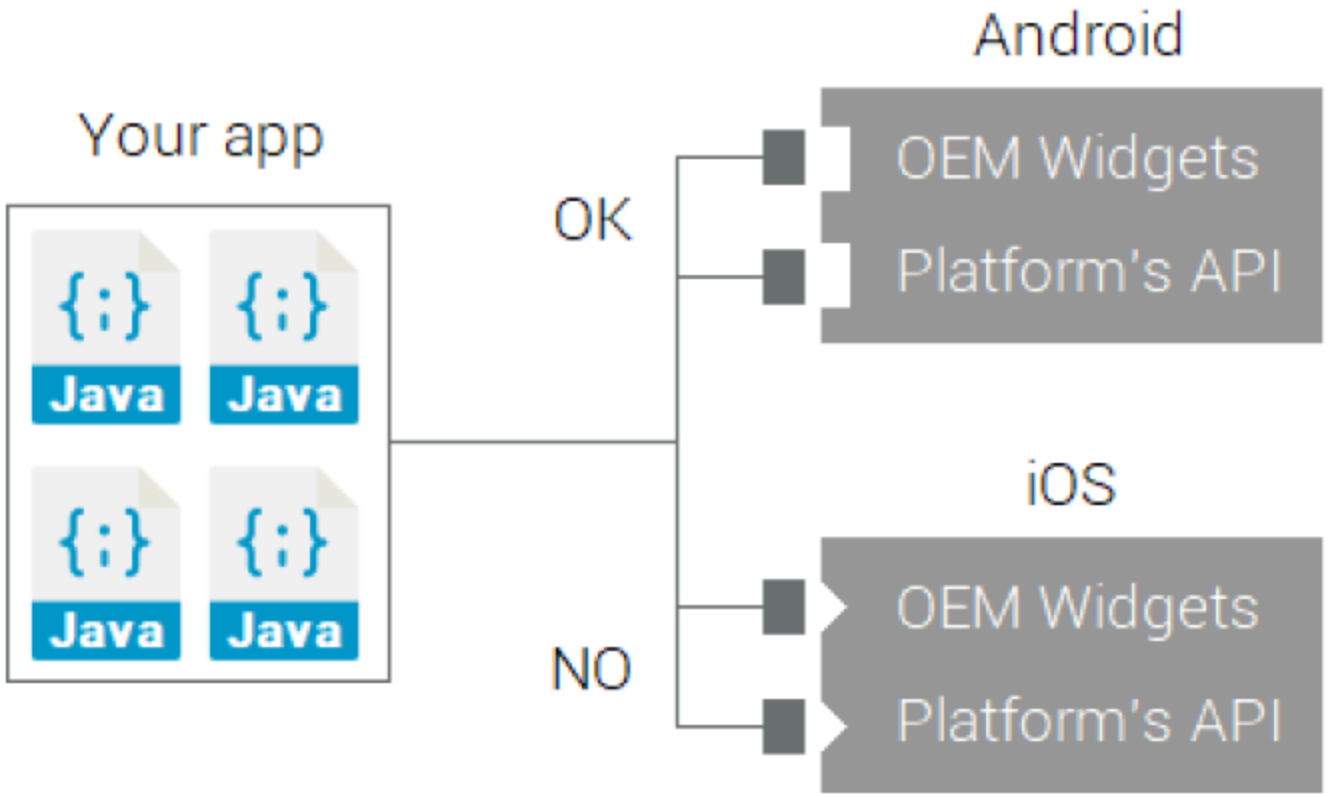
Flutter is an open-source UI software development kit (SDK) created by Google.
It is used to build natively compiled applications for mobile, web, and desktop from a single codebase.

- **Why developers like Flutter:**
  - Hot reload for instant updates during development
  - Rich set of customizable widgets
  - Strong community support and extensive documentation
  - Single codebase for multiple platforms
  - Excellent performance and native-like experience

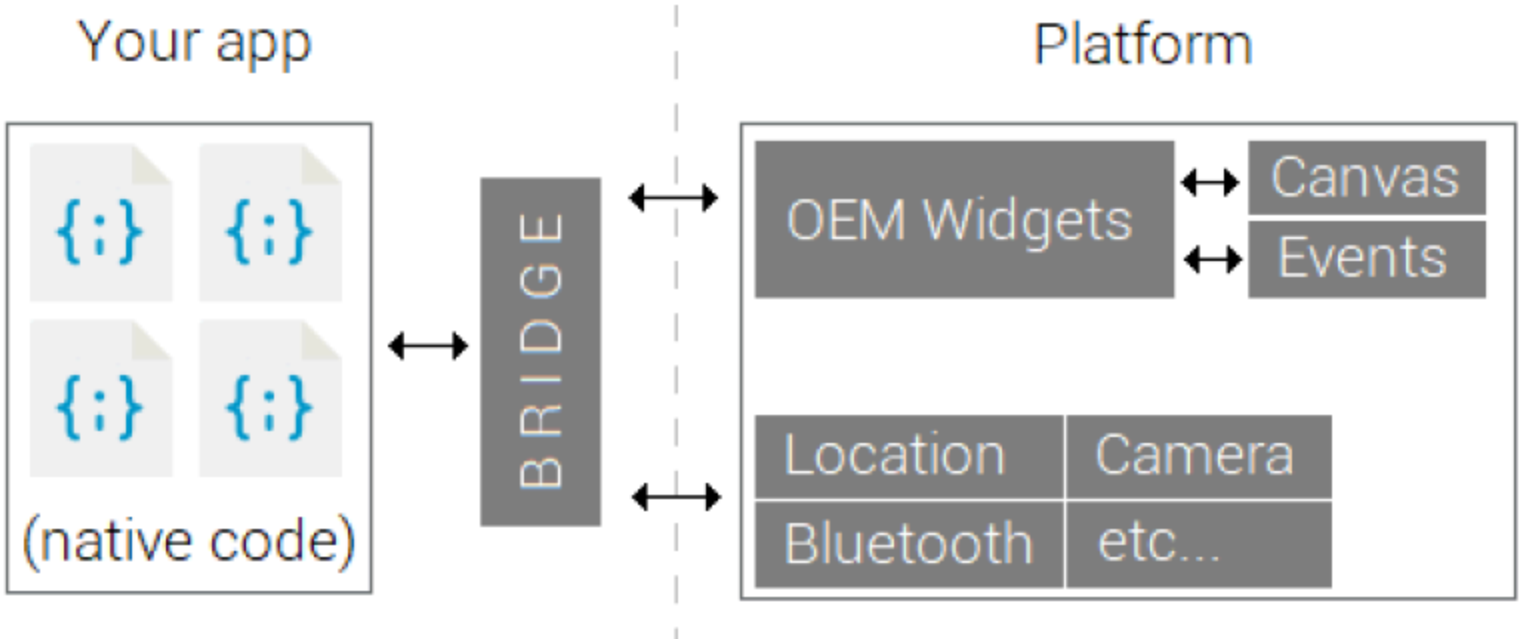
Flutter Architecture



- **OEM widgets:** Flutter uses its own rendering engine to draw widgets directly on a canvas provided by the platform, bypassing OEM widgets.



- **Platform channels:** Flutter uses platform channels to communicate with native code when necessary, allowing access to platform-specific features.



- **Bridge:** The Flutter framework communicates with the underlying platform through a bridge, enabling seamless integration with native components when needed.

Flutter and Dart

Why Flutter uses Dart?

- **OOP style:**
  - Dart is an OOP (object-oriented programming) language, which aligns well with Flutter's widget-based architecture.
- **Performance:**
  - Dart is designed for high performance, with features like ahead-of-time (AOT) compilation that allows Flutter apps to run quickly and efficiently.
- **Productivity:**
  - Dart's syntax is easy to learn and use, which helps developers be more productive when building Flutter applications.

Flutter IDE (Integrated Development Environment)

- **Android Studio:** https://developer.android.com/studio
  Comprehensive IDE for Android development with built-in tools for Flutter.
- **Visual Studio Code:** https://code.visualstudio.com/
  Lightweight, fast editor with excellent Flutter and Dart extensions.

Flutter Hello World

```dart
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text("Flutter app!"),
        ),
      ),
    );
  }
}
```

Flutter Project Setup

Create a new Flutter project

```
flutter create my_app
```

To create a Flutter project with specific platforms

```
flutter create my_app --platforms=android,ios,web
```

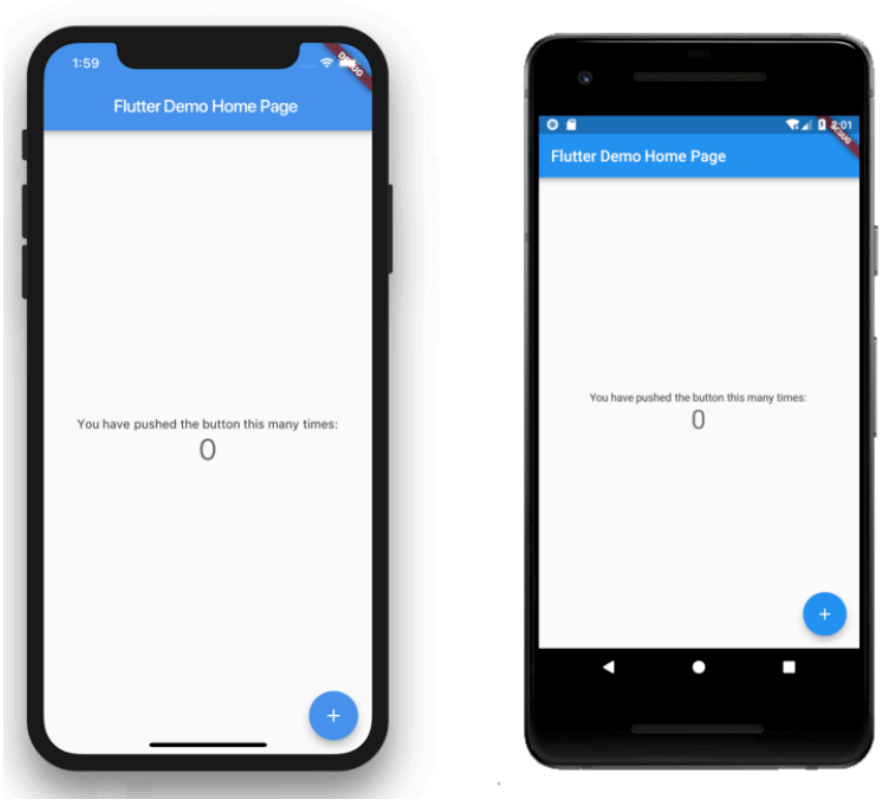
Run the Flutter app

```
flutter run
```

Build the Flutter app

```
flutter build apk
```

Flutter Demo



In Flutter, everything is a widget.

End of Day 1