Dart

By Nishat Quyoum, Giselle Avila, Micheal O’Neill, Christian Anderson, Devin Khun

Dart Background

- What is it?
  - Object-oriented, class-based language with garbage collection
  - Used to build fast and responsive applications across web, mobile, and desktop platforms
- Who created it?
  - Developed in 2011 by Google and designed by Lars Bark and Kasper Lund
- Why was it created?
  - Improves JavaScript's performance and maintainability for large projects.(ex: Gmail and Google Maps)
  - Dart does this by offering optional static typing, a more advanced virtual machine, and easier-to-read code

Google

Dart designers Lars Bark and Kasper Lund

Dart Background

- Additional Info:
  - Dart struggled to gain popularity at first
  - JavaScript was already well established, and Typescript became popular
  - Dart became well known with the introduction of Flutter, a UI toolkit, developed by Google in 2015
  - Flutter used Dart as its language for its speed and strong developer's tools.
  - Flutter's success boosted Dart's adoption and popularity.

Syntax

Trusted Source:

(https://dart.dev/resources/language/spec/versions/DartLangSpec-v2.10.pdf)

Formal Syntax/grammar description

- Used Backus-Naur Form (BNF) and Extended Backus-Nuar Form (EBNF) to define the language constructs

BNF Examples:

```
<libraryDeclaration> ::= <scriptTag>? <libraryName>? <importOrExport>*
<partDirective>*
              (<metadata> <topLevelDeclaration>)* <EOF>

<scriptTag> ::= '#!' (~<NEWLINE>)* <NEWLINE>

<libraryName> ::= <metadata> library <dottedIdentifierList> ';'

<importOrExport> ::= <libraryImport> | <libraryExport>

<dottedIdentifierList> ::= <identifier> ('.' <identifier>)*
```

```
<methodSignature> ::= <constructorSignature> <initializers>?
            | <factoryConstructorSignature>
            | static? <functionSignature>
            | static? <getterSignature>
            | static? <setterSignature>
            | <operatorSignature>
```

```
<classDeclaration> ::= abstract? class <identifier>
<typeParameters>?
             <superclass>? <interfaces>?
             '{' (<metadata> <classMemberDeclaration>)* '}'

<classMemberDeclaration> ::= <declaration> ';'
                 | <methodSignature> <functionBody>
```

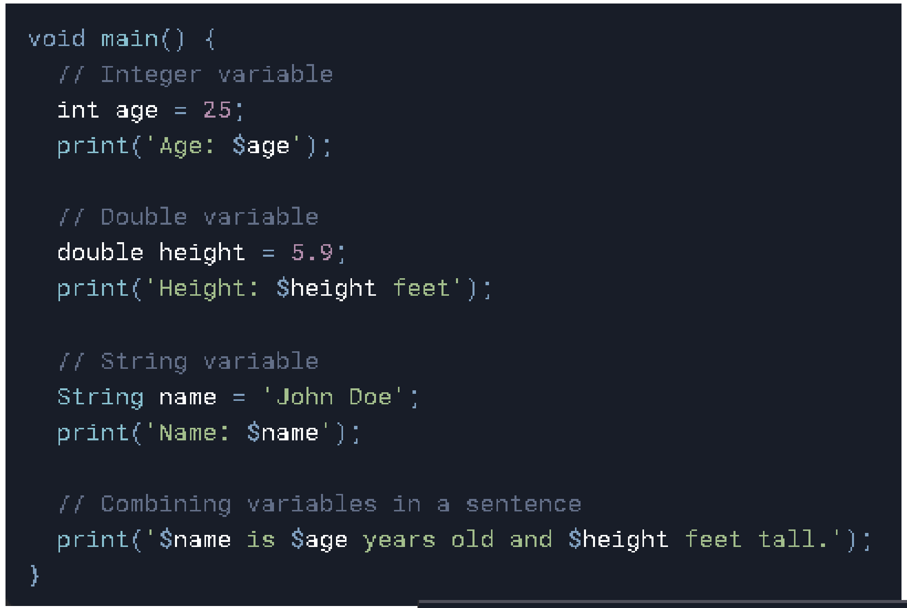
```
<getterSignature> ::= <type>? get <identifier>

<setterSignature> ::= <type>? set <identifier>
<formalParameterList>
```

Syntax Basics

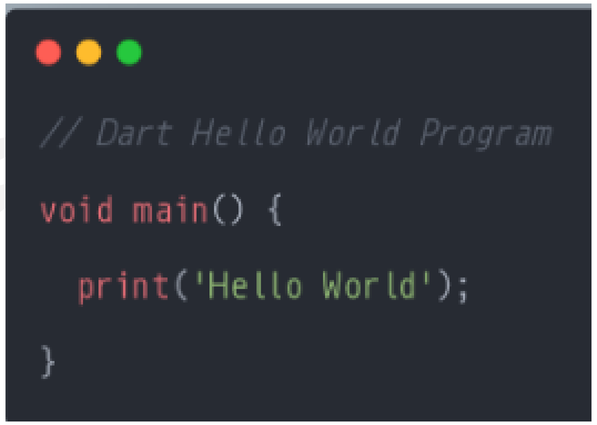
Trusted Source: (https://dart.dev/language)

Main Features

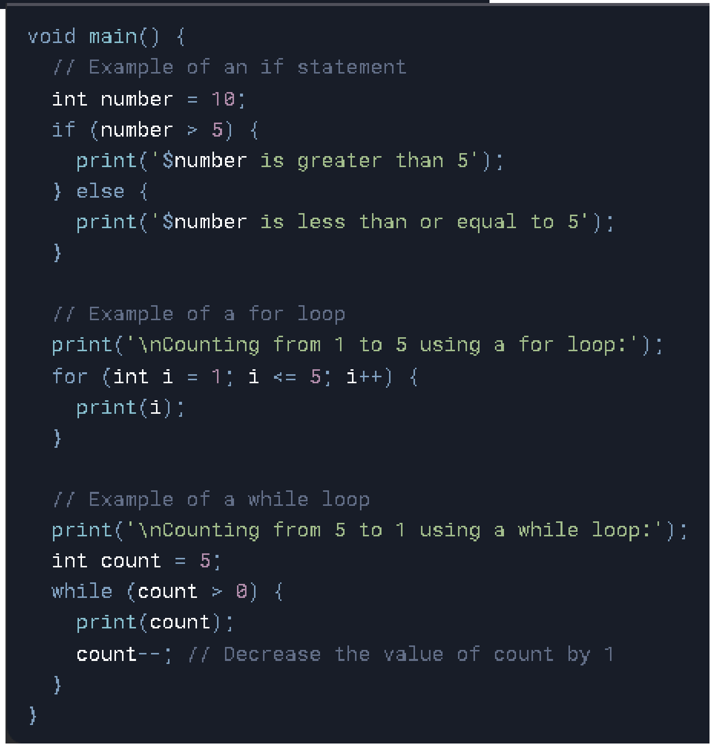
- Hello World
  - Every app starts with main
- Variables
  - Use var for type inference or explicitly declare types, type safe
- Control Flow
  - Like if, for, while
- Function
  - Explicit types recommended
- Comments
  - Single-line Comments (//)
  - Multi-line Comments (/* … */)
  - Documentation Comments (/// or /** … */)



```dart
void main() {
  // Integer variable
  int age = 25;
  print('Age: $age');

  // Double variable
  double height = 5.9;
  print('Height: $height feet');

  // String variable
  String name = 'John Doe';
  print('Name: $name');

  // Combining variables in a sentence
  print('$name is $age years old and $height feet tall.');
}
```



```dart
// Dart Hello World Program

void main() {

  print('Hello World');

}
```

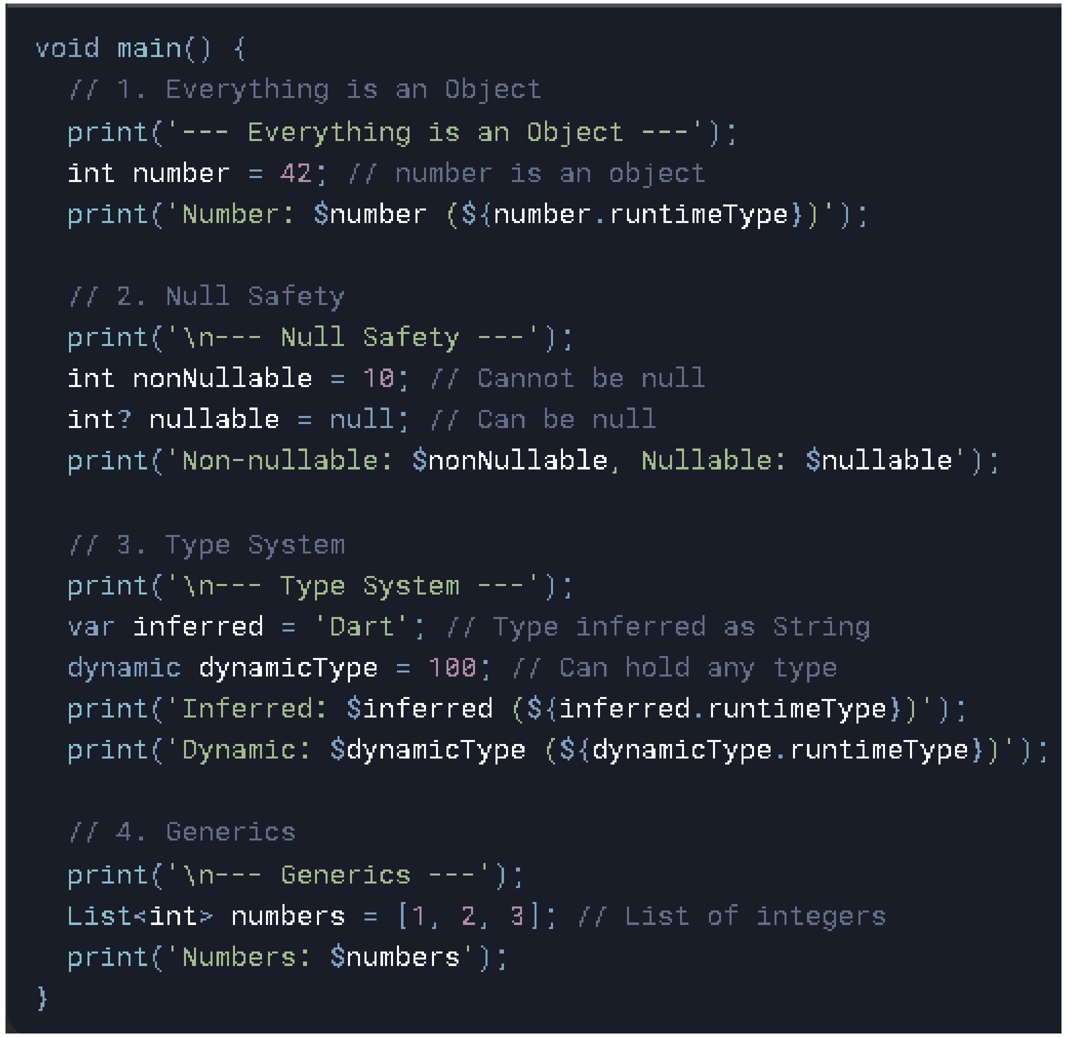


```dart
void main() {
  // Example of an if statement
  int number = 10;
  if (number > 5) {
    print('$number is greater than 5');
  } else {
    print('$number is less than or equal to 5');
  }

  // Example of a for loop
  print('\nCounting from 1 to 5 using a for loop:');
  for (int i = 1; i <= 5; i++) {
    print(i);
  }

  // Example of a while loop
  print('\nCounting from 5 to 1 using a while loop:');
  int count = 5;
  while (count > 0) {
    print(count);
    count--; // Decrease the value of count by 1
  }
}
```

Syntax Basics

- Everything is an Object
    - Everything you can assign to a variable is an object, including numbers, functions, and null
    - All objects are instances of a class, inheriting from the Object class
- Null Safety
    - Variables are non-nullable by default
    - Use ? for nullable types
- Type System
    - Strongly typed with type inference
    - Use dynamic to defer type checking until runtime
- Generics
    - Ensure type safety with generics

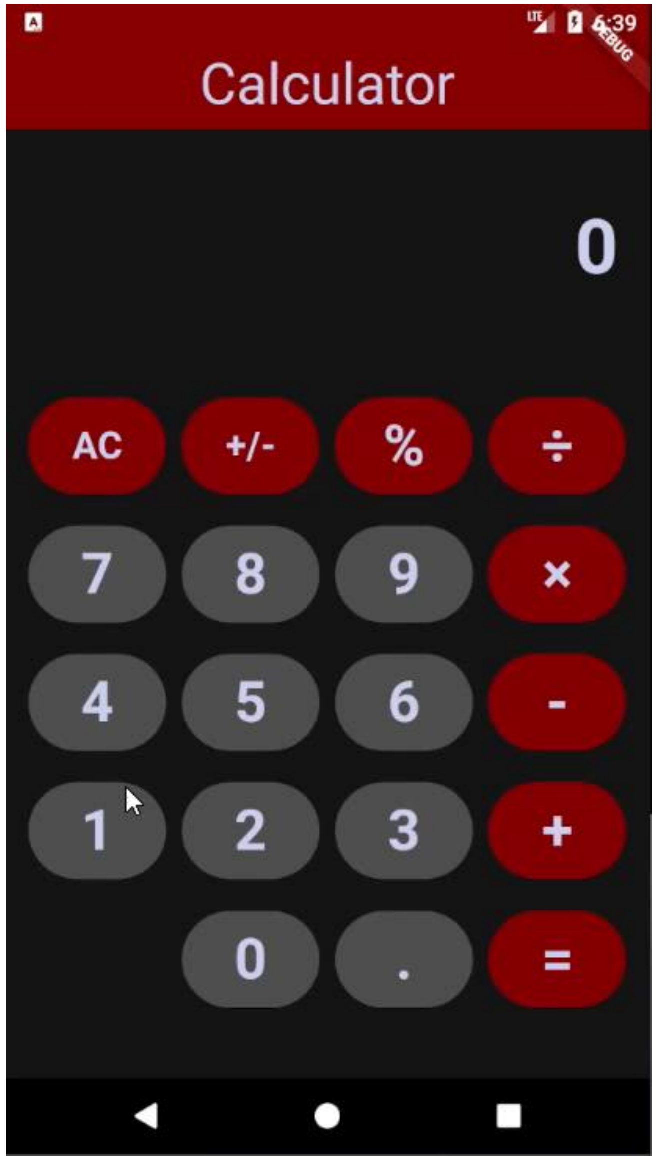
And much more!



```dart
void main() {
  // 1. Everything is an Object
  print('--- Everything is an Object ---');
  int number = 42; // number is an object
  print('Number: $number (${number.runtimeType})');

  // 2. Null Safety
  print('\n--- Null Safety ---');
  int nonNullable = 10; // Cannot be null
  int? nullable = null; // Can be null
  print('Non-nullable: $nonNullable, Nullable: $nullable');

  // 3. Type System
  print('\n--- Type System ---');
  var inferred = 'Dart'; // Type inferred as String
  dynamic dynamicType = 100; // Can hold any type
  print('Inferred: $inferred (${inferred.runtimeType})');
  print('Dynamic: $dynamicType (${dynamicType.runtimeType})');

  // 4. Generics
  print('\n--- Generics ---');
  List<int> numbers = [1, 2, 3]; // List of integers
  print('Numbers: $numbers');
}
```

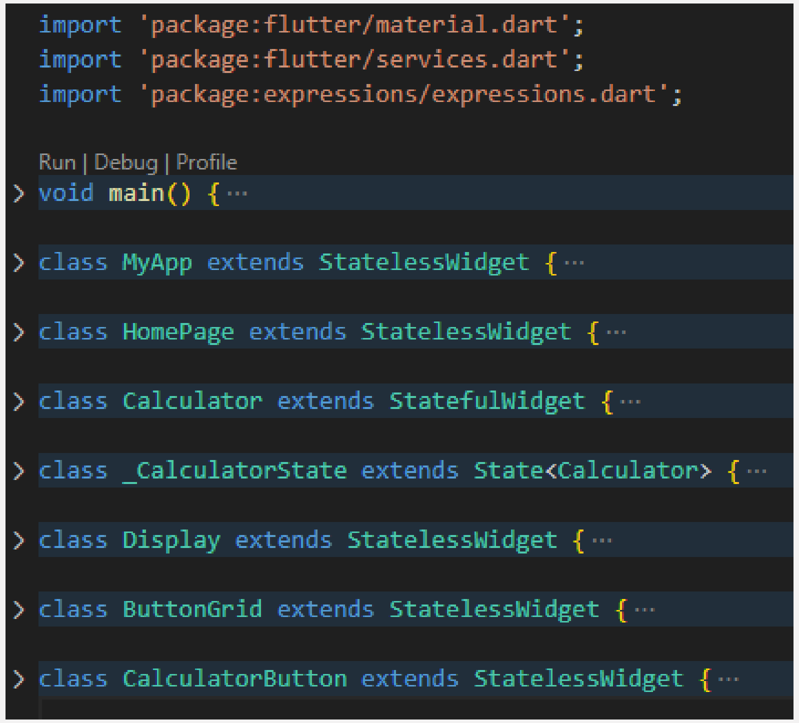
Sample Program

- Built using the Flutter framework
- Deployable across multiple platforms
- https://github.com/kamikazeem1/CS4080Demo



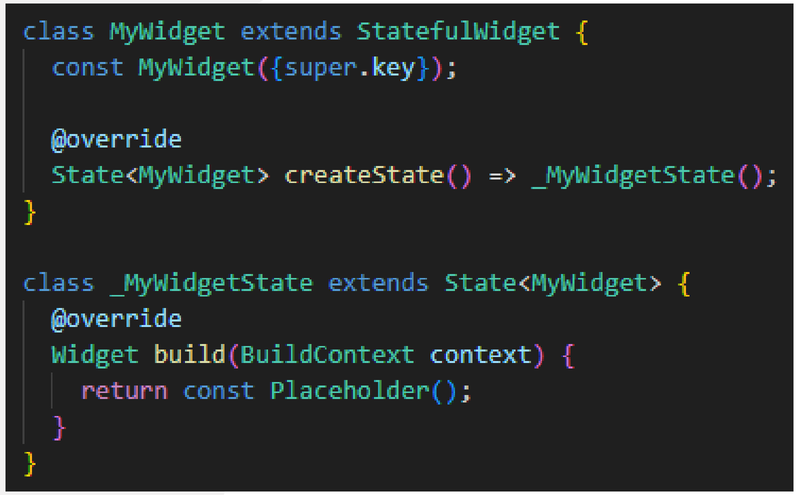
Program Structure

Calculator program structure



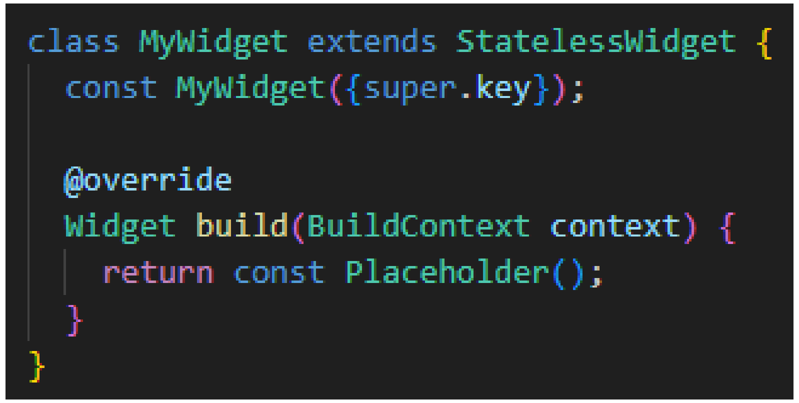
```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:expressions/expressions.dart';

Run | Debug | Profile
void main() {…

class MyApp extends StatelessWidget {…

class HomePage extends StatelessWidget {…

class Calculator extends StatefulWidget {…

class _CalculatorState extends State<Calculator> {…

class Display extends StatelessWidget {…

class ButtonGrid extends StatelessWidget {…

class CalculatorButton extends StatelessWidget {…
```

Stateful Widget structure



```
class MyWidget extends StatefulWidget {
  const MyWidget({super.key});

  @override
  State<MyWidget> createState() => _MyWidgetState();
}

class _MyWidgetState extends State<MyWidget> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Stateless Widget structure



```
class MyWidget extends StatelessWidget {
  const MyWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

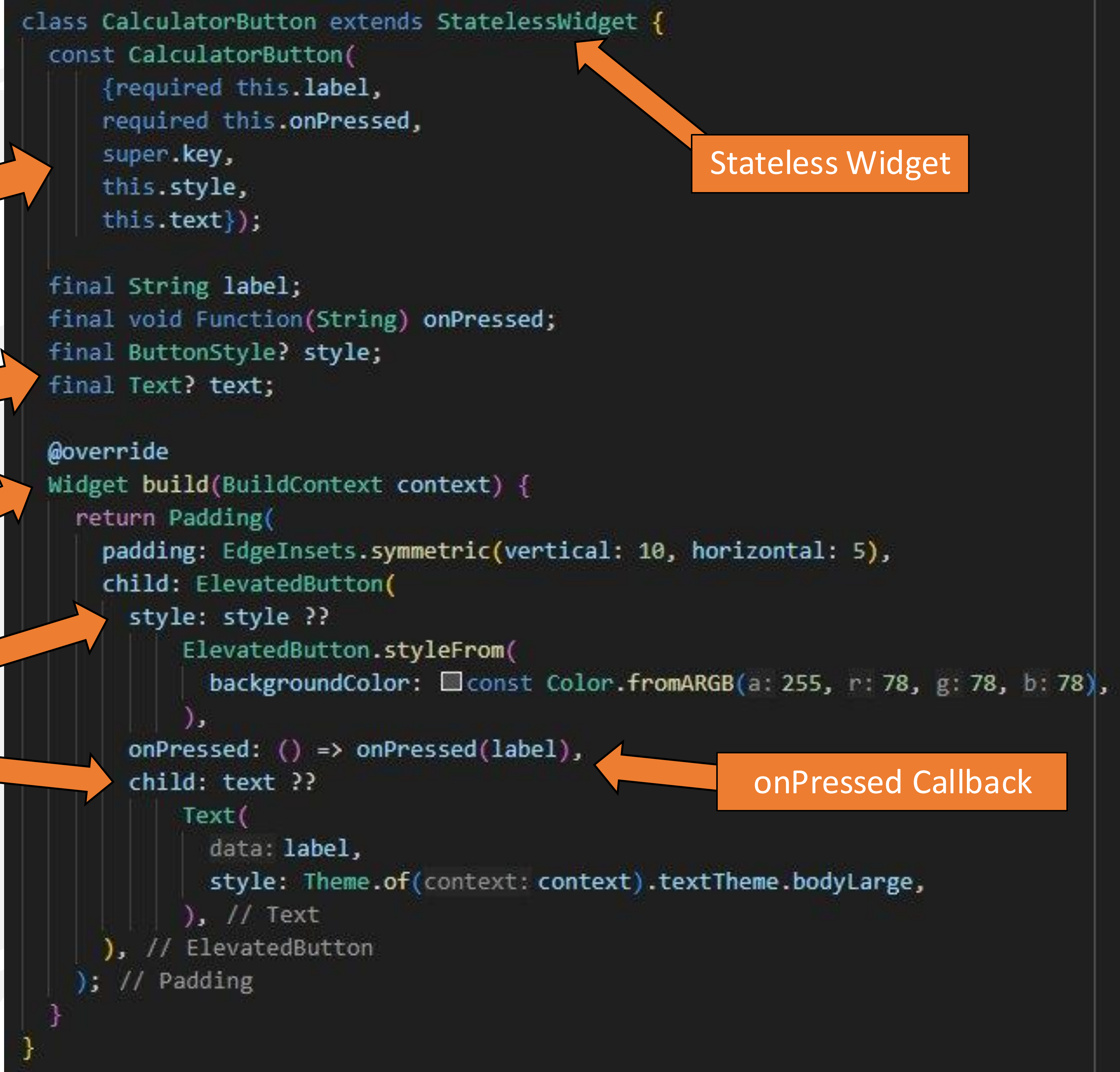
Example Class

Constructor: label and onPressed are required, style and text are optional

Instance Variables: final keyword assures values cannot change after assignment
? Allows variables to be null

build Method: core function for every widget, returns widget tree

?? (null-coalescing operator): provides a default value if style/text is null



Stateless Widget

onPressed Callback

```
class CalculatorButton extends StatelessWidget {
  const CalculatorButton(
      {required this.label,
      required this.onPressed,
      super.key,
      this.style,
      this.text});

  final String label;
  final void Function(String) onPressed;
  final ButtonStyle? style;
  final Text? text;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(vertical: 10, horizontal: 5),
      child: ElevatedButton(
        style: style ??
            ElevatedButton.styleFrom(
              backgroundColor: const Color.fromARGB(a: 255, r: 78, g: 78, b: 78),
            ),
        onPressed: () => onPressed(label),
        child: text ??
            Text(
              data: label,
              style: Theme.of(context: context).textTheme.bodyLarge,
            ), // Text
      ), // ElevatedButton
    ); // Padding
  }
}
```

Resources

1. https://medium.com/@author2000.1225/the-history-and-rules-of-dart-language-f25e09a58530
2. https://www.deusinmachina.net/p/the-dart-programming-language-is
3. https://dart.dev/language