

Programmazione Mobile

Nicola Noviello

nicola.noviello@unimol.it

Corso di Laurea in Informatica
Dipartimento di Bioscienze e Territorio
Università degli Studi del Molise
Anno 2023/2024

Lezione: Flutter e Dart Foundation (parte seconda)

- Basi di Flutter
- Sintassi di Dart
- Comprendere il funzionamento del framework
- Primi approcci alla creazione di un progetto
- Widgets

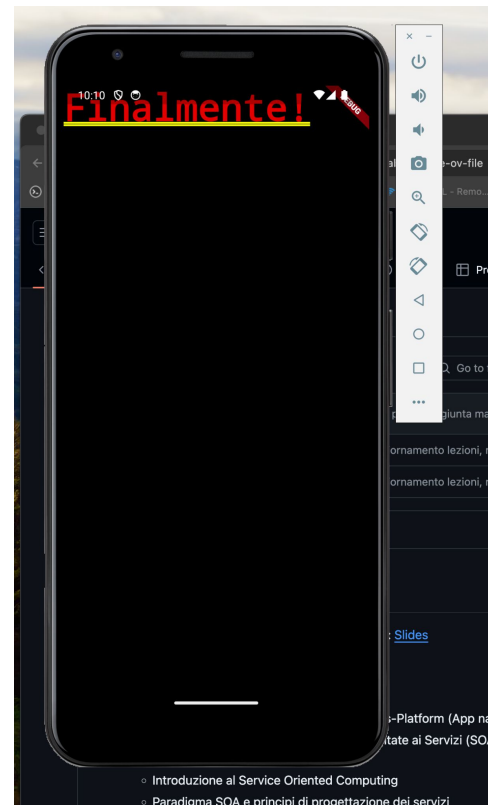




Dove eravamo
rimasti?

La nostra App

```
main.dart ●  
lib > main.dart > ...  
1 import 'package:flutter/material.dart';  
   Run | Debug | Profile  
2 void main() {  
3   runApp(MaterialApp(home: Text('Finalmente!')));  
4 }  
5  
6
```



- Introduzione al Service Oriented Computing
- Paradigma SOA e principi di progettazione dei servizi

Parametri posizionali e nominali

Nella precedente lezione, abbiamo visto come passare argomenti / parametri “posizionali” e “nominali” ad una funzione.

In generale, i parametri / argomenti delle funzioni sono un concetto chiave.

Si usano gli argomenti per passare dei valori a una funzione. La funzione può quindi “trattare” questi valori per farne qualcosa come ad esempio, per visualizzarli sullo schermo, usarli in un calcolo o inviarli a un'altra funzione.



Parametri posizionali

La posizione dell'argomento determina quale parametro riceverà il valore

```
1 void add(a, b) {  
2   print(a + b);  
3 }  
4  
5 add(5, 10); // 5 è il valore di a, 10 di b
```



Parametri nominali

Il nome dell'argomento determina quale parametro riceverà il valore

```
1 void add({a, b}) {  
2   print(a + b);  
3 }  
4  
5 add(b: 5, a: 10); // 5 è il valore di b,  
6                      // 10 quello di a
```

La differenza

Oltre alle casistiche di utilizzo la vera differenza sostanziale tra le due tipologie di parametri è che se vengono definiti dei parametri posizionali, sarà obbligatorio utilizzarli quando si invoca una funzione.

I parametri nominali invece sono opzionali.

È possibile però lavorare a grana fine se abbiamo bisogno di una gestione più precisa dei parametri.



Parametri opzionali e valori di default

```
1 void add(a, [b]) { // b è opzionale
2   print(a + b);
3 }
4
5 void add(a, [b = 5]) { // b è opzionale, 5 è valore di default
6   print(a + b);
7 }
8 add(10); // b vale 5
9 add(10, 6); // b vale 6
```



Run

Parametri opzionali e valori di default

```
1 void add({a, b = 5}) { // b è 5 di default
2   print(a + b);
3 }
4
5 add(b: 10); // b vale 10, a non è obbligatorio
```

Parametri opzionali e valori di default

```
1 void add({required a, required b}) {  
2     print(a + b);  
3 }  
4  
5 // in questo caso che succede?  
6 // a cosa equivale questa definizione?
```

Parametri obbligatori e dove trovarli

```
main.dart ×  
lib > main.dart > main  
1 import 'package:flutter/material.dart';  
  Run | Debug | Profile  
2 void main() {  
3   runApp(MaterialApp(home: Center(child: Text('Finalmente!')),));  
4 }  
5  
6  
7  
8  
9
```

Try adding the 'const' keyword to the constructor invocation. dart([prefer_const_constructors](#))

Use 'const' with the constructor to improve performance.
Try adding the 'const' keyword to the constructor invocation. dart([prefer_const_constructors](#))

```
(new) Text Text(  
  String data, {  
  Key? key,  
  TextStyle? style,  
  StrutStyle? strutStyle,  
  TextAlign? textAlign,  
  TextDirection? textDirection
```

Uso di const

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

Run | Debug | Profile
void main() {
  runApp(MaterialApp(home: Center(child: Text('Finalmente!')),));
}

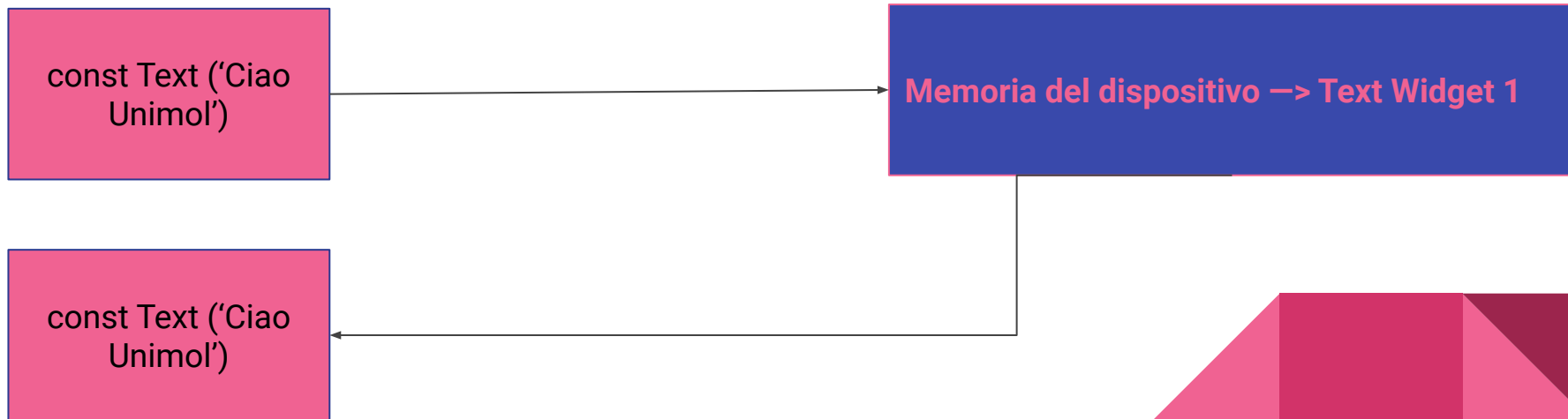
Use 'const' with the constructor to improve performance.
Try adding the 'const' keyword to the constructor
invocation. dart(prefer_const_constructors)

(new) MaterialApp MaterialApp({
  Key? key,
  GlobalKey<NavigatorState>? navigatorKey,
  GlobalKey<ScaffoldMessengerState>? scaffoldMessengerKey,
```

È presente ancora un “errore” (che in realtà un errore non è, ma si tratta di un’ottimizzazione), ma è sufficiente aggiungere **const** con il costruttore per risolverlo

“const”

const è uno strumento di Dart per
ottimizzare le performance al runtime



Senza errori, Finalmente!

main.dart > ...

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

Run | Debug | Profile

```
void main() {  
  runApp(const MaterialApp(home: Center(child: Text('Finalmente!'),),),);  
}
```



Scaffold

```
lib > main.dart > ...  
1  import 'package:flutter/material.dart';  
2  
   Run | Debug | Profile  
3  void main() {  
4    runApp(const MaterialApp(home: Scaffold(body: Center(child: Text('Finalmente!'),),),),);  
5  }  
6
```

Finalmente!

Scaffold ha migliorato l'estetica della nostra App, ma come abbiamo integrato Scaffold? Guardate in autonomia la documentazione dall'IDE oppure approfondite il funzionamento del widget qui:

<https://api.flutter.dev/flutter/material/Scaffold-class.html>

Esercizio 1

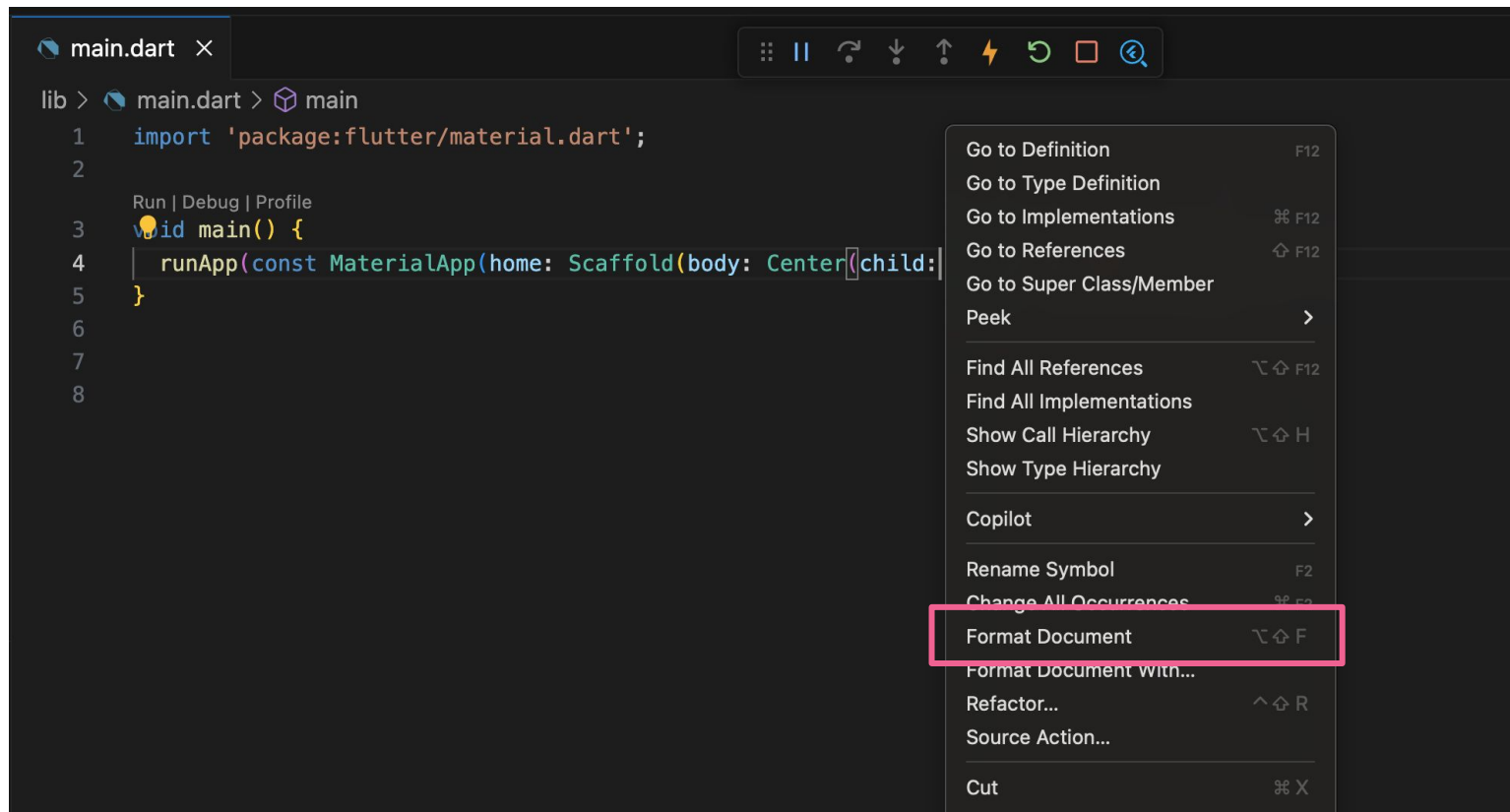
Se non l'avete fatto ieri, 10 minuti per leggere in autonomia la documentazione di Scaffold e scoprite come usare un gradiente come background

dart format

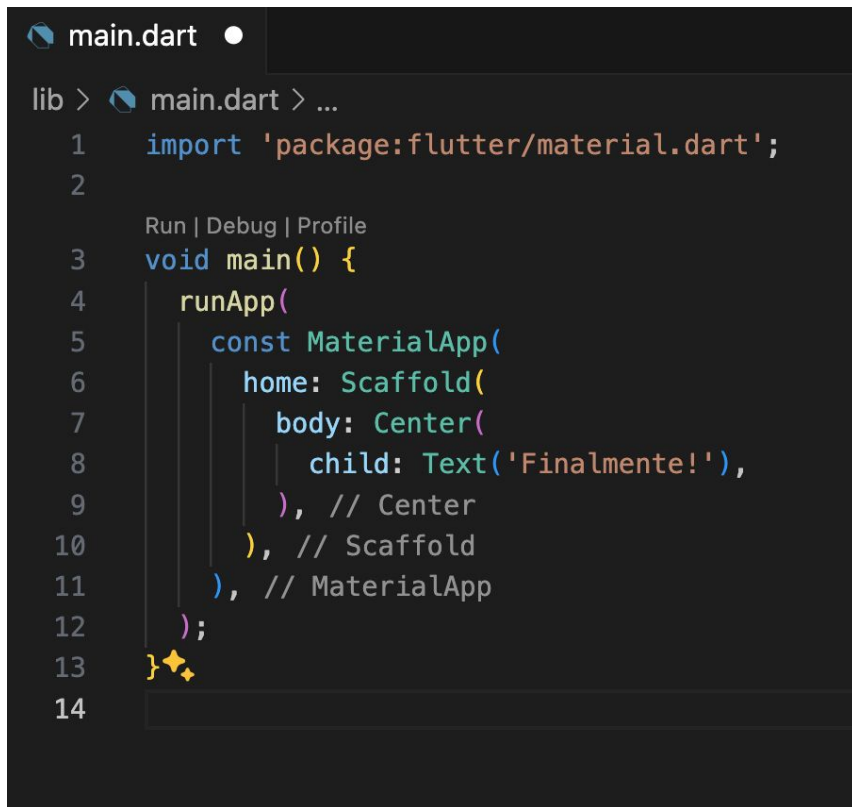
```
ente!' ), ), ), ), );
```

Perché tendo ad aggiungere una virgola dopo ogni parentesi? O meglio, alla chiusura delle parentesi di un widget?

dart format



dart format

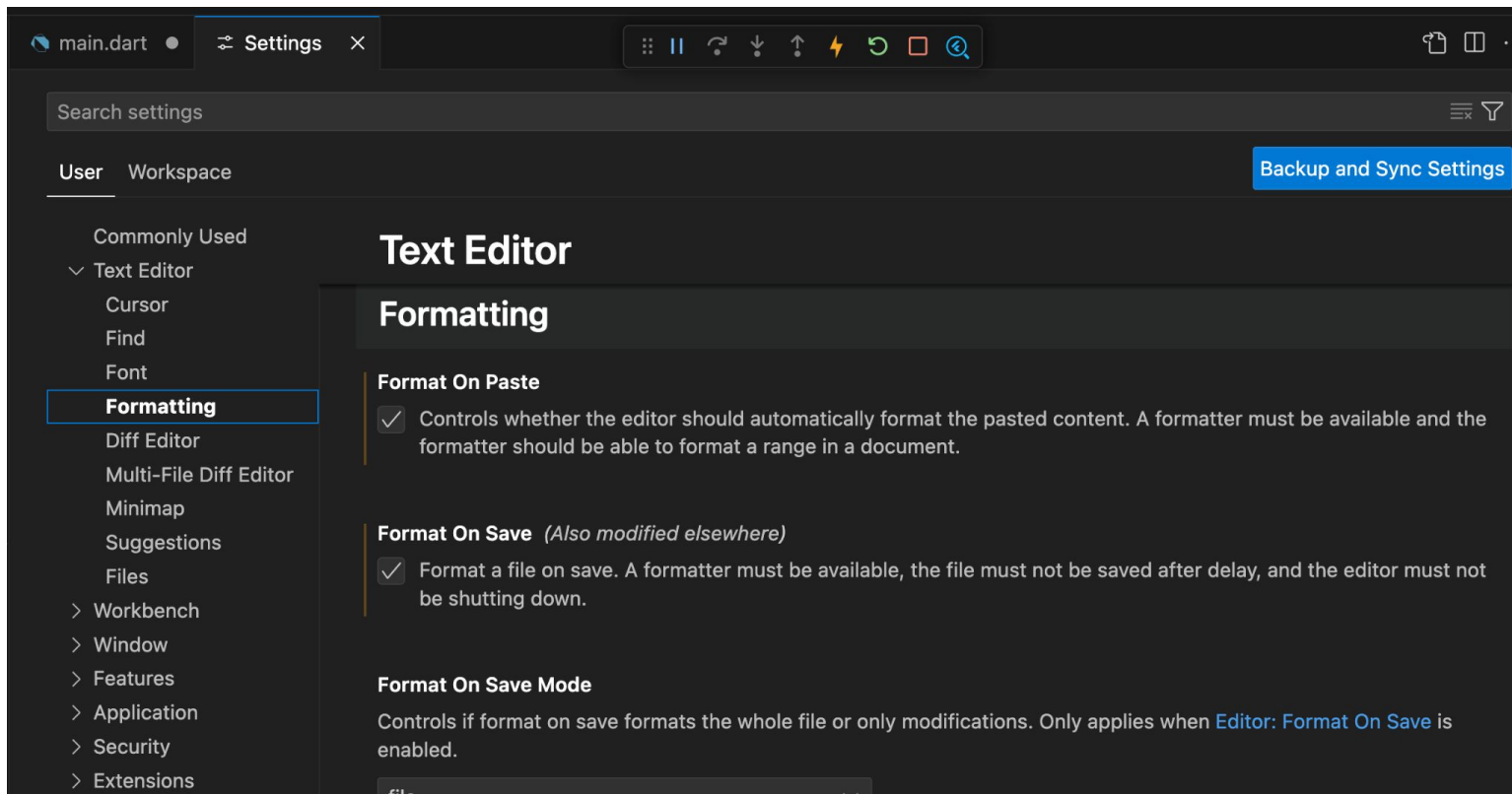


The screenshot shows a code editor window titled 'main.dart'. The code is as follows:

```
lib > main.dart > ...  
1  import 'package:flutter/material.dart';  
2  
   Run | Debug | Profile  
3  void main() {  
4      runApp(  
5          const MaterialApp(  
6              home: Scaffold(  
7                  body: Center(  
8                      child: Text('Finalmente!'),  
9                      ), // Center  
10                 ), // Scaffold  
11             ), // MaterialApp  
12         );  
13     } ✨  
14
```

The code is formatted with standard Dart conventions: imports are at the top, followed by the `main` function. The `runApp` function call is indented, and its arguments are also indented. Comments like `// Center` and `// Scaffold` are placed on the same line as the closing parenthesis of the nested function calls. The `const` keyword is used for the `MaterialApp` widget. The `Text` widget's content is a string. The `runApp` function call ends with a semicolon. The `main` function is enclosed in curly braces. A yellow star icon is visible at the end of line 13.

dart format



type-safe

dart è un linguaggio **type-safe**

Tutti gli elementi appartengono ad un certo **Tipo**

'Ciao Unimol'

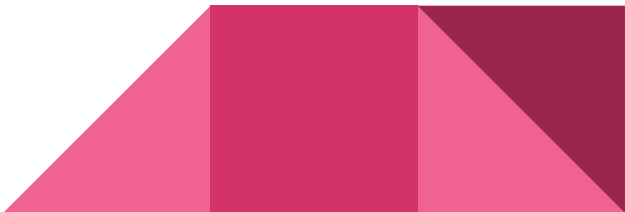
String

84

int

MaterialApp

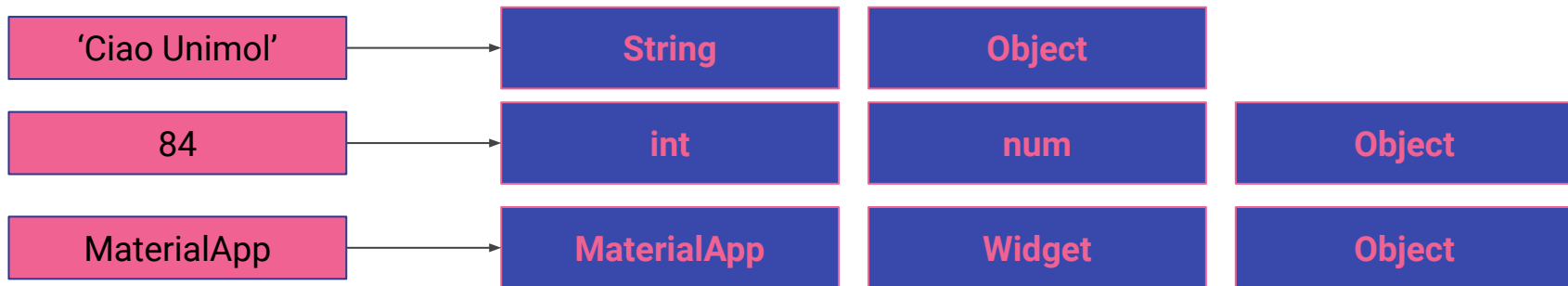
MaterialApp



type-safe

dart è un linguaggio **type-safe**

Tutti i valori appartengono ad un certo **Tipo**



Più di un tipo è possibile. I tipi possono essere definiti dal linguaggio, da terze parti oppure definiti dall'utente

type-safe

```
1 void add(num1, num2) {  
2     print(num1+num2);  
3 }  
4  
5 void main(){  
6     add(8,4);  
7 }
```



▶ Run

12

type-safe

```
1 void add(num1, num2) {  
2     print(num1+num2);  
3 }  
4  
5 void main(){  
6     add('8',4);  
7 }
```



▶ Run

Script error.

type-safe

```
1 void add(int num1, num2) {  
2     print(num1+num2);  
3 }  
4  
5 void main(){  
6     add('8',4);  
7 }
```

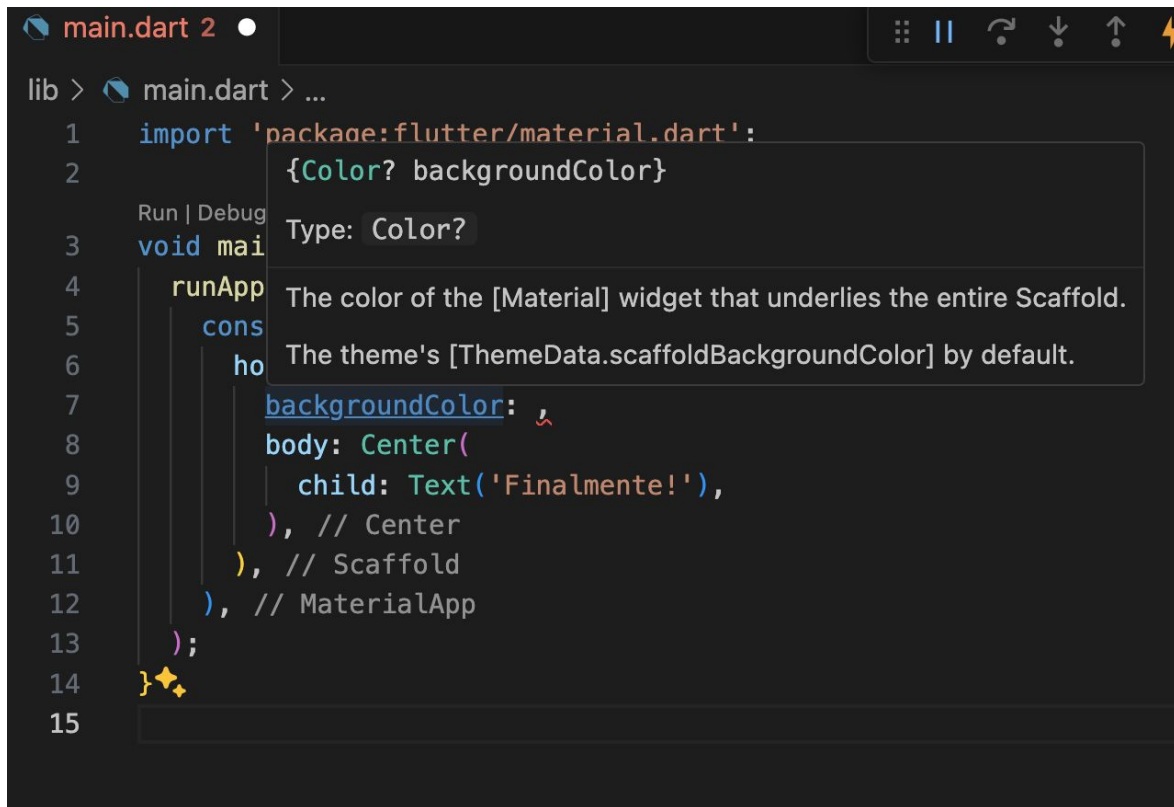


Run

Color

```
> main.dart > ...  
1  import 'package:flutter/material.dart';  
2  
   Run | Debug | Profile  
3  void main() {  
4      runApp(  
5          const MaterialApp(  
6              home: Scaffold(  
7                  backgroundColor: ,  
8                  body: Center(  
9                      child: Text('Finalmente!'),  
0                      ), // Center  
1                  ), // Scaffold  
2              ), // MaterialApp  
3          );  
4      }  
5
```

Color



```
main.dart 2 •  
lib > main.dart > ...  
1 import 'package:flutter/material.dart':  
2   {Color? backgroundColor}  
3 void main() {  
4   runApp(  
5     const MaterialApp(  
6       home: Scaffold(  
7         backgroundColor: Colors.white,  
8         body: Center(  
9           child: Text('Finalmente!'),  
10        ), // Center  
11      ), // Scaffold  
12    ), // MaterialApp  
13  );  
14 }  
15
```

Run | Debug
Type: Color?
The color of the [Material] widget that underlies the entire Scaffold.
The theme's [ThemeData.scaffoldBackgroundColor] by default.

Color

```
import 'package:flutter/material.dart';  
import 'package:flutter/widgets.dart';
```

Run | Debug | Profile

```
void main() {  
  runApp(  
    const MaterialApp(  
      home: Scaffold(  
        backgroundColor: Colors.blueGrey, // potevamo usare Color(0xFF000000) per esempio  
                                           // invece abbiamo preferito Colors (classe astratta)  
                                           // che contiene tutti i colori predefiniti di Flutter  
        body: Center(  
          child: Text('Finalmente!'),  
        ), // Center  
      ), // Scaffold  
    ), // MaterialApp  
  );  
}
```

11:01



Finalmente!

type-safe

dart è un linguaggio **type-safe**

Tutti i valori appartengono ad un certo **Tipo**

'Ciao Unimol'

String

Object

84

int

num

Object

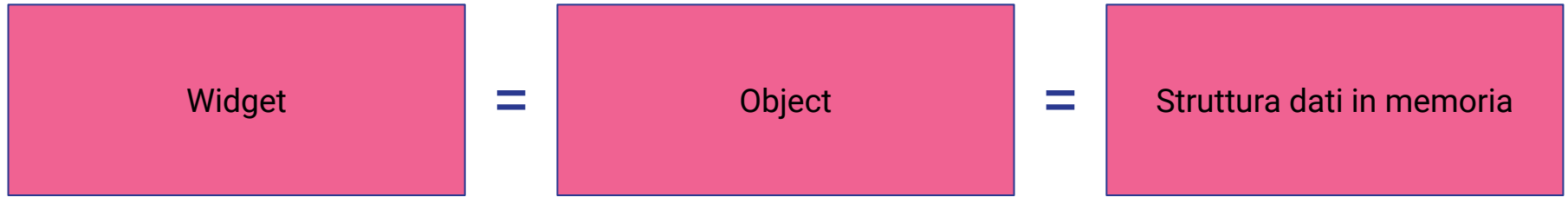
MaterialApp

MaterialApp

Widget

Object

Widget = Object



Un gradiente come background

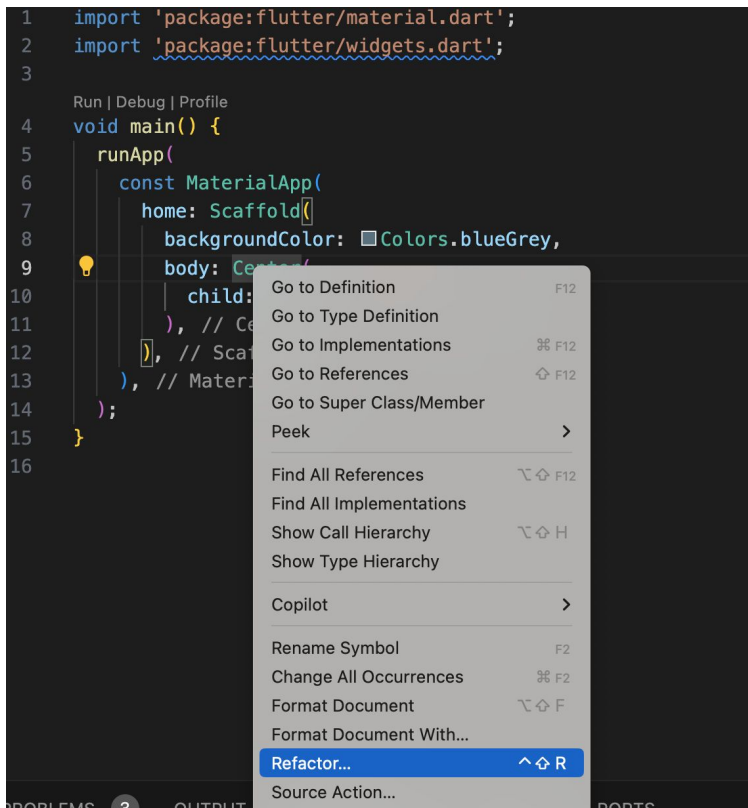
E se volessimo aggiungere un gradiente come background?

Analizzando Scaffold, non esiste una proprietà che mi permette di usare un gradiente, quindi che si fa?

Cosa si può fare se abbiamo bisogno di qualcosa per definire uno stile più preciso e completo nella nostra App? Aggiungiamo un Widget!



Refactor



Refactor



Container

Guardate in autonomia come l'IDE vi presenta l'errore: il problema è che il Widget Container non supporta const, di conseguenza dobbiamo spostare const al costruttore "figlio"

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

Run | Debug | Profile

```
void main() {  
  runApp(  
    const MaterialApp(  
      home: Scaffold(  
        body: Container(  
          child: Center(  
            child: Text('Finalmente!'),  
          ), // Center  
        ), // Container  
      ), // Scaffold  
    ), // MaterialApp  
  );  
}
```

Container

Per l'IDE il nostro Container al momento è inutile, perché? Perché non abbiamo ancora definito gli attributi.

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

Run | Debug | Profile

```
void main() {
```

```
  runApp(
```

```
    MaterialApp(
```

```
      home: Scaffold(
```

```
        body: Container(
```

```
          child: const Center(
```

```
            child: Text('Finalmente!'),
```

```
          ), // Center
```

```
        ), // Container
```

```
      ), // Scaffold
```

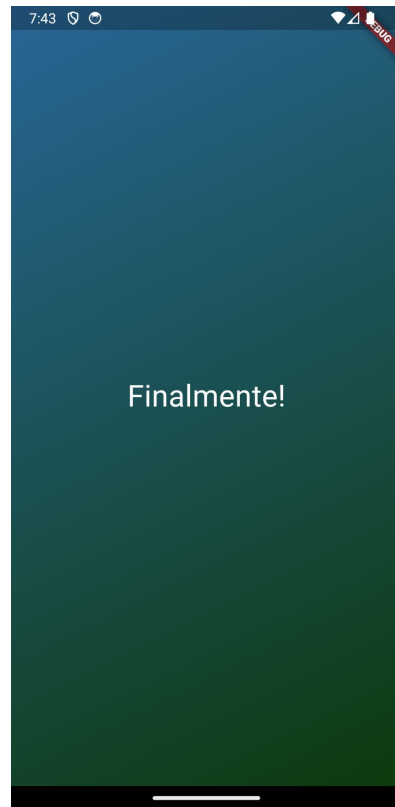
```
    ), // MaterialApp
```

```
  );
```

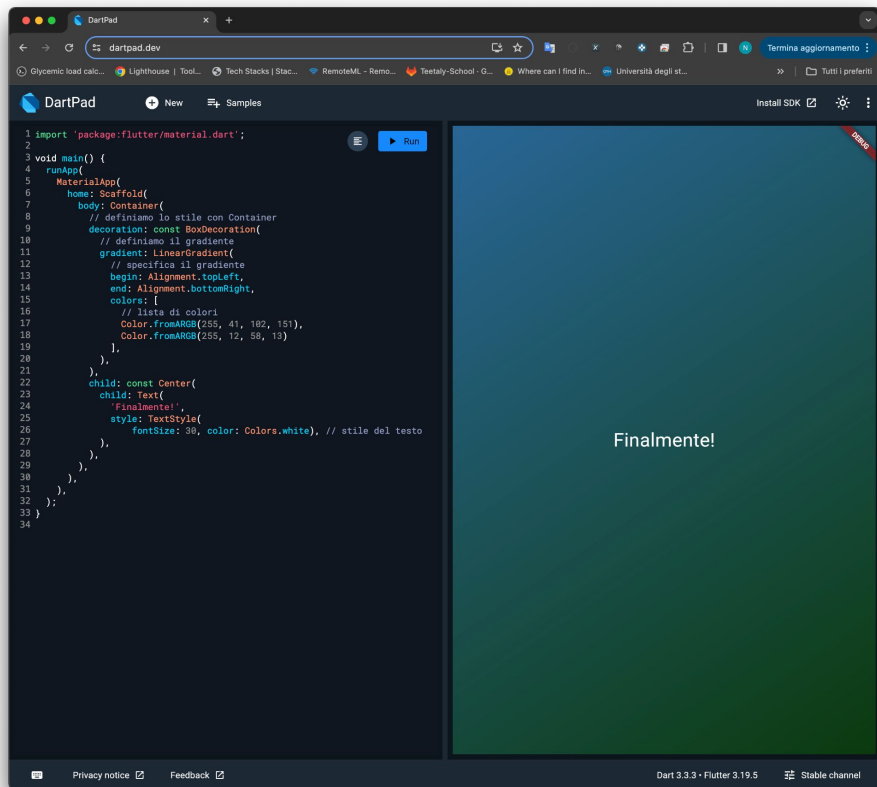
```
}
```

BoxDecoration, LinearGradient, TextStyle

```
void main() {  
  runApp(  
    MaterialApp(  
      home: Scaffold(  
        body: Container( // definiamo lo stile con Container  
          decoration: const BoxDecoration( // definiamo il gradiente  
            gradient: LinearGradient( // specifica il gradiente  
              begin: Alignment.topLeft,  
              end: Alignment.bottomRight,  
              colors: [ // lista di colori  
                Color.fromARGB(255, 41, 102, 151),  
                Color.fromARGB(255, 12, 58, 13)  
              ],  
            ), // LinearGradient  
          ), // BoxDecoration  
        child: const Center(  
          child: Text(  
            'Finalmente!',  
            style: TextStyle(fontSize: 30, color: Colors.white), // stile del testo  
          ), // Text  
        ), // Center  
      ), // Container  
    ), // Scaffold  
  ), // MaterialApp  
);  
}
```



dartpad.dev



dartpad.dev

Uno strumento utile per provare snippet di codice in maniera rapida (così da non passare tutte le ore di lezione a configurare l'IDE 😜).

Inoltre supporta i GitHub Gist...

Share with a gist

DartPad looks for a file named `main.dart` within your gist.

To share your code with others:

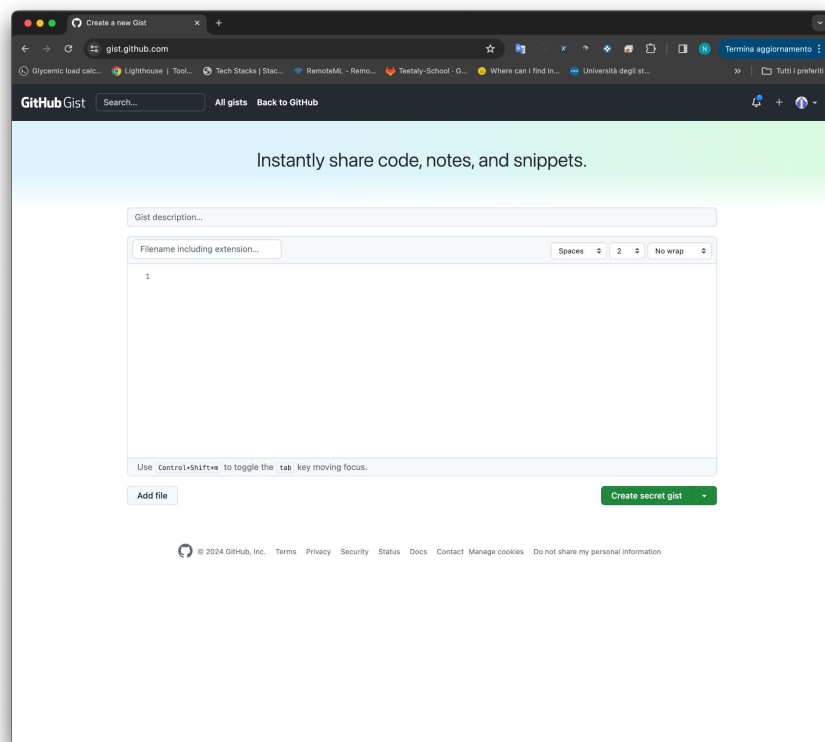
1. Create a gist with a `main.dart` file.
2. Direct DartPad to the gist ID, without any username specified. For example, to view gist.github.com/5c0e154dd50af4a9ac856908061291bc in DartPad, use the URL dartpad.dev/?id=5c0e154dd50af4a9ac856908061291bc.

Some example gists:

- gist.github.com/4a68e553746602d851ab3da6aeafc3dd
- gist.github.com/a1d5666d6b54a45eb170b897895cf757

Happy sharing!

GitHub Gist



GitHub Gist

Deve chiamarsi
per forza così

Il contenuto che
vogliamo provare

Codice da usare come base per l'Esercizio 2 della Lezione 4

main.dart

Spaces

2

No wrap

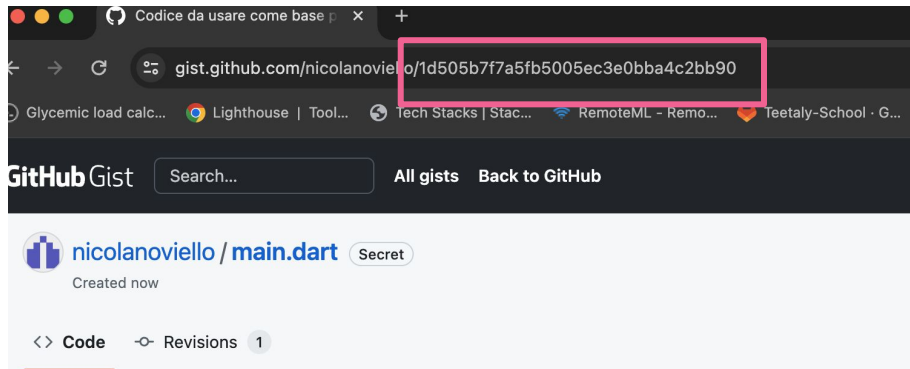
```
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(
5     MaterialApp(
6       home: Scaffold(
7         body: Container(
8           // definiamo lo stile con Container
9           decoration: const BoxDecoration(
10            // definiamo il gradiente
11            gradient: LinearGradient(
12              // specifica il gradiente
13              begin: Alignment.topLeft,
14              end: Alignment.bottomRight,
15              colors: [
16                // lista di colori
17                Color.fromARGB(255, 41, 102, 151),
```

Use `Control+Shift+m` to toggle the `tab` key moving focus.

Add file

Create secret gist

GitHub Gist



Codice da usare come base per l'Esercizio 2 della Lezione 4

 main.dart

```
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(
5     MaterialApp(
6       home: Scaffold(
7         body: Container(
8           // definiamo lo stile con Container
9         ),
10      ),
11    ),
12  );
13 }
```

dartpad.dev/?id=1d505b7f7a5fb5005ec3e0bba4c2bb90

Esercizio 2

Container, BoxDecoration, LinearGradient, TextStyle

A cosa servono? Cercate le rispettive documentazioni e provate a personalizzare il codice appena mostrato. Cosa notate di particolare in TextStyle?

List<Color>

```
void main() {  
  runApp(  
    MaterialApp(  
      home: Scaffold(  
        body: Container(  
          decoration: BoxDecoration(  
            gradient: LinearGradient(  
              colors: [ // lista di colori  
                Color.fromARGB(255, 41, 102, 151),  
                Color.fromARGB(255, 12, 58, 13)  
              ],  
            )  
          )  
        )  
      )  
    )  
  )  
}
```

Type: List<Color>

Creates a linear gradient.

If [stops] is non-null, it must have the same length as [colors].

AppBar

<https://api.flutter.dev/flutter/material/AppBar-class.html>

AppBar class



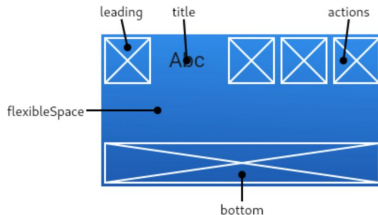
A Material Design app bar.

An app bar consists of a toolbar and potentially other widgets, such as a [TabBar](#) and a [FlexibleSpaceBar](#). App bars typically expose one or more common [actions](#) with [IconButton](#)s which are optionally followed by a [PopupMenuButton](#) for less common operations (sometimes called the "overflow menu").

App bars are typically used in the [Scaffold.appBar](#) property, which places the app bar as a fixed-height widget at the top of the screen. For a scrollable app bar, see [SliverAppBar](#), which embeds an [AppBar](#) in a sliver for use in a [CustomScrollView](#).

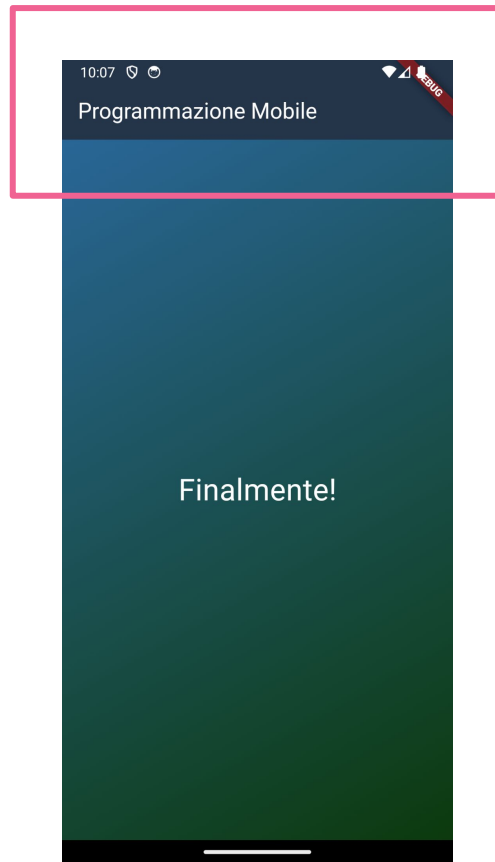
The AppBar displays the toolbar widgets, [leading](#), [title](#), and [actions](#), above the [bottom](#) (if any). The [bottom](#) is usually used for a [TabBar](#). If a [flexibleSpace](#) widget is specified then it is stacked behind the toolbar and the bottom widget. The following diagram shows where each of these slots appears in the toolbar when the writing language is left-to-right (e.g. English):

The [AppBar](#) insets its content based on the ambient [MediaQuery](#)'s padding, to avoid system UI intrusions. It's taken care of by [Scaffold](#) when used in the [Scaffold.appBar](#) property. When animating an [AppBar](#), unexpected [MediaQuery](#) changes (as is common in [Hero](#) animations) may cause the content to suddenly jump. Wrap the [AppBar](#) in a [MediaQuery](#) widget, and adjust its padding such that the animation is smooth.



AppBar

```
home: Scaffold(  
  appBar: AppBar(  
    title: const Text(  
      'Programmazione Mobile',  
      style: TextStyle(  
        color: Colors.white,  
      ), // TextStyle  
    ), // Text  
    backgroundColor: const Color.fromARGB(255, 36, 52, 73),  
  ), // titolo dell'app // AppBar  
  body: Container(  
    // corpo dell'app
```

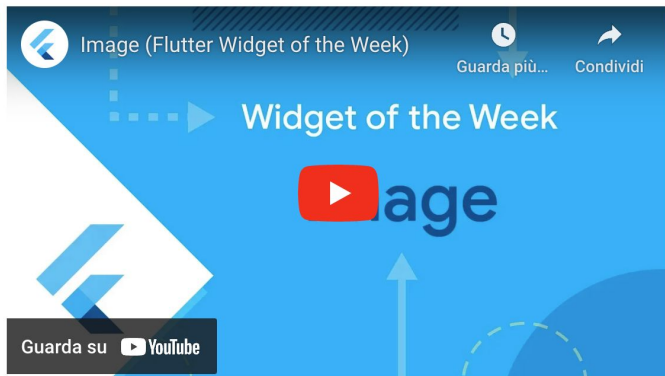


Image

<https://api.flutter.dev/flutter/widgets/Image-class.html>

Image class

A widget that displays an image.

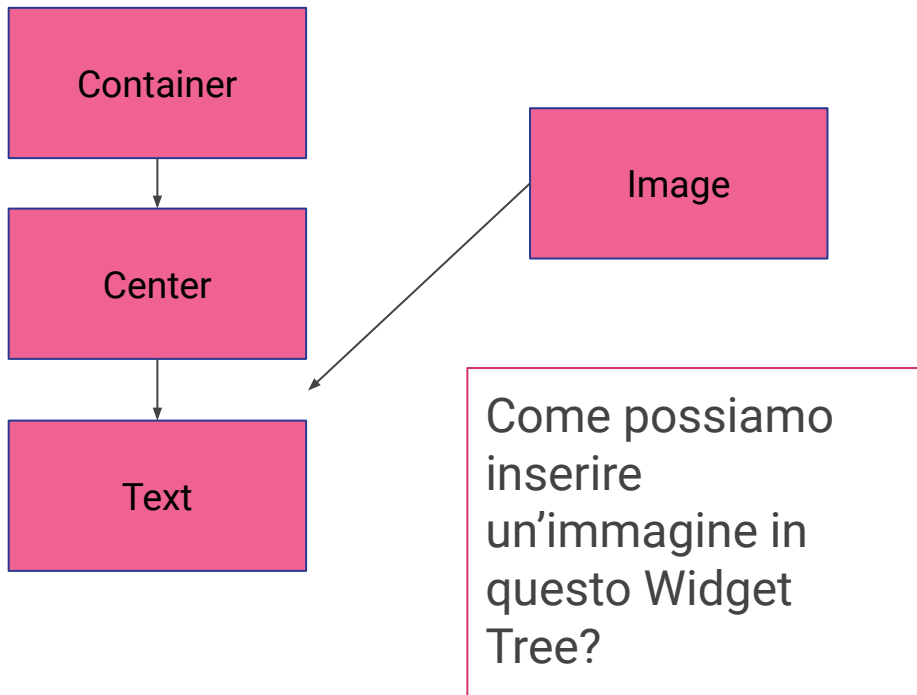
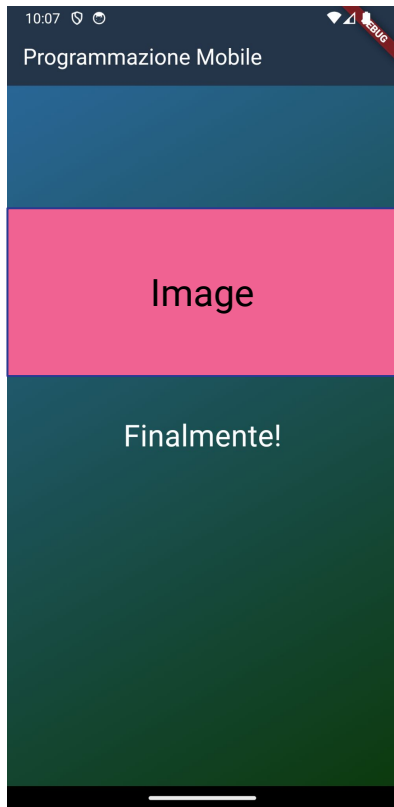


Several constructors are provided for the various ways that an image can be specified:

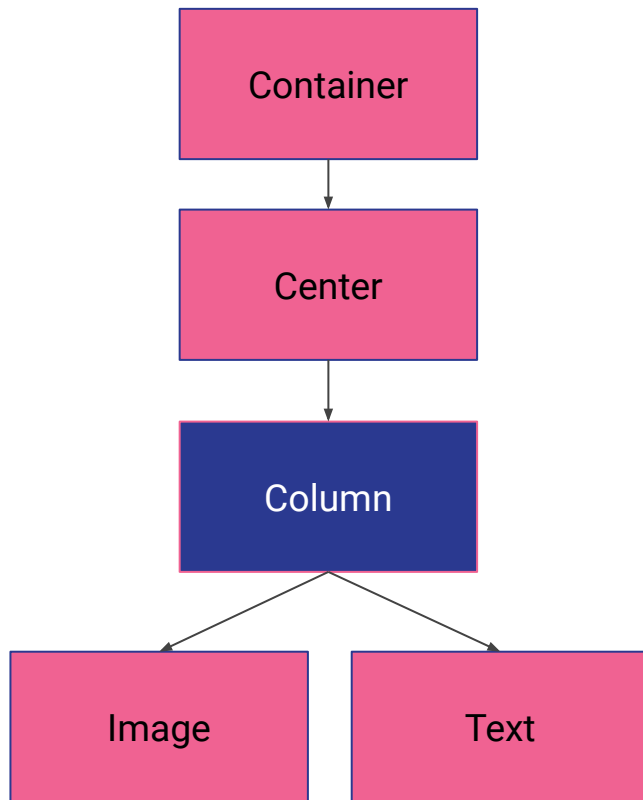
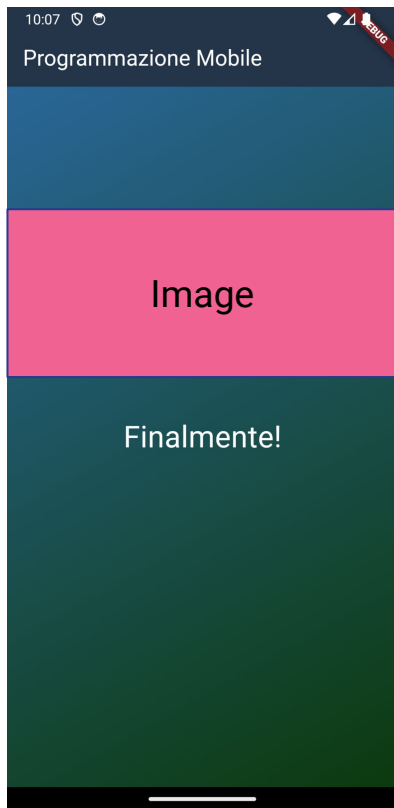
- `Image.new`, for obtaining an image from an `ImageProvider`.
- `Image.asset`, for obtaining an image from an `AssetBundle` using a key.
- `Image.network`, for obtaining an image from a URL.
- `Image.file`, for obtaining an image from a `File`.
- `Image.memory`, for obtaining an image from a `Uint8List`.

L'utilizzo avanzato di questo Widget richiede un maggiore approfondimento che avverrà nelle lezioni successive.

Image



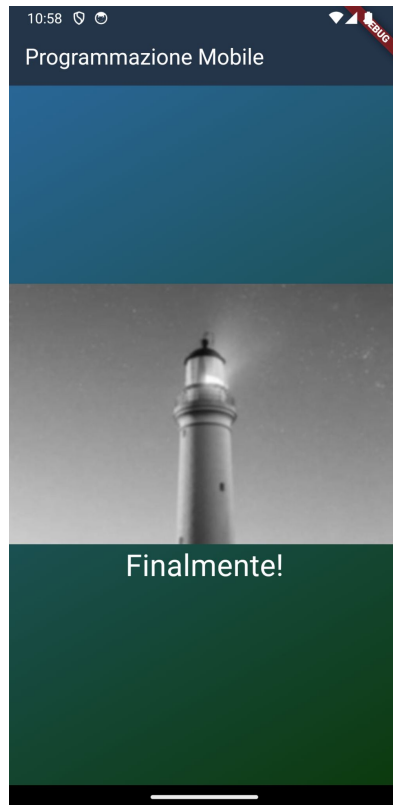
Image



Dove trovo la documentazione di Column?

Column + Image

```
child: const Center(  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.center,  
    children: <Widget>[  
      Image(  
        image: NetworkImage(  
          'https://fastly.picsum.photos/id/870/600/400.jpg?bl',  
        ), // Image  
      Text(  
        'Finalmente!',  
        style: TextStyle(  
          fontSize: 30, color: Colors.white), // stile del  
        ), // Text  
    ], // <Widget>[]  
  ), // Column  
, // Center
```



Esercizio 3

Partendo dal codice fornito su GitHub (`esercizio3.dart`) provare ad usare un asset locale per l'immagine invece di un url. (SUGGERIMENTO: Usare `AssetImage`, ma attenzione, c'è un'insidia)