

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 terza)

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





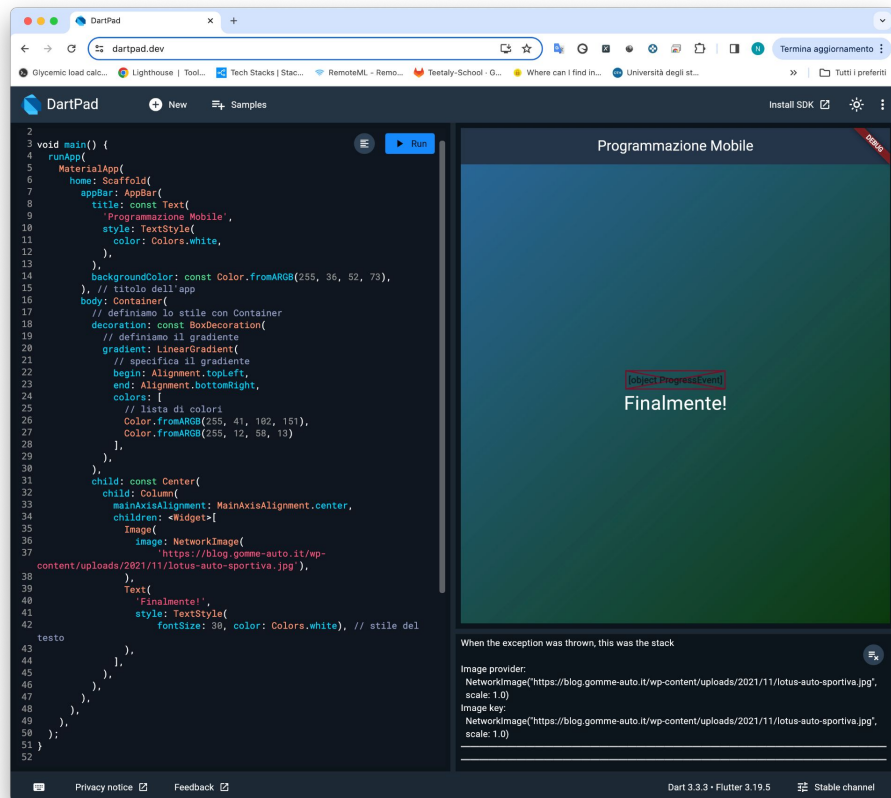
Domande ricevute
dalle studentesse e
dagli studenti

Problemi e quesiti che mi sono stati posti durante l'ultima esercitazione

- Ho problemi a visualizzare un'immagine presa da internet, quale può essere il problema?
- Come personalizzo l'Icona e la Splash Screen della mia App?
- Come installo la mia App su un device fisico?
- L'Hot Reload non funziona



Errore nella visualizzazione dell'immagine

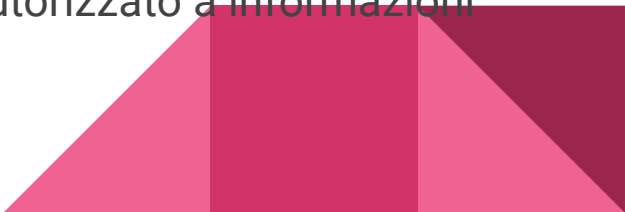


CORS (Cross-Origin Resource Sharing)

Le **CORS** (Cross-Origin Resource Sharing) sono politiche di sicurezza implementate dai browser web per proteggere gli utenti da richieste di risorse (come script, font, immagini, etc.) da parte di origini diverse dal sito web che stanno attualmente visualizzando.

Le **web app** sono effettivamente soggette alle politiche CORS, proprio come i siti web. Tuttavia, le applicazioni native, incluse le **App per dispositivi mobili**, non sono soggette alle stesse restrizioni CORS perché non sono eseguite all'interno di un browser web.

Le politiche CORS sono fondamentali per mantenere la sicurezza (servono ad evitare un determinato tipo di attacchi) e la privacy degli utenti sul web, limitando l'accesso delle risorse solo a siti web affidabili e prevenendo l'accesso non autorizzato a informazioni sensibili.



CORS (Cross-Origin Resource Sharing)

Quando un browser fa una richiesta HTTP per ottenere una risorsa da un altro dominio, esso include informazioni aggiuntive nell'header della richiesta, come l'origine del sito che sta facendo la richiesta. Questo consente al server di rispondere alla richiesta in base alle politiche CORS configurate.

Un sito web non può dire con certezza se una richiesta proviene da un'App o da un browser web standard. Tuttavia, può utilizzare alcuni metodi per identificare possibili comportamenti sospetti o non autorizzati, come l'analisi dell'User-Agent o altri attributi della richiesta.



CORS - Come risolvere?

Se un sito “protegge” le proprie risorse con delle CORS non è possibile acquisirle in maniera diretta.

Per fare dei prototipi è possibile usare dei siti web per creare placeholder di immagini (come ad esempio <https://picsum.photos/>), mentre per il progetto o per dei lavori professionali è possibile affidarsi ad API per le quali si ha l'autorizzazione oppure ad un backend proprio.



Icona dell'App

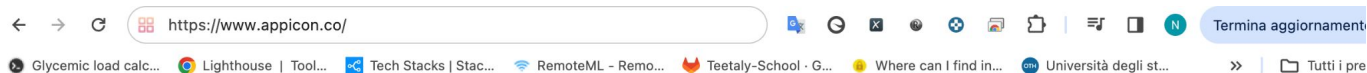
 App Icon

 Image Sets

 [Donate](#)



Click or drag image file (1024 x 1024)

OR

Generate app icon using [Appicons.ai](https://appicons.ai)



App icon Generator

Drag or select an app icon image (1024x1024) to generate different app icon sizes for all platforms

iOS and macOS

- ✓ **iPhone** - 11 different sizes and files
- ✓ **iPad** - 13 different sizes and files
- ✓ **watchOS** - 8 different sizes and files
- ✓ **macOS** - 11 different sizes and files

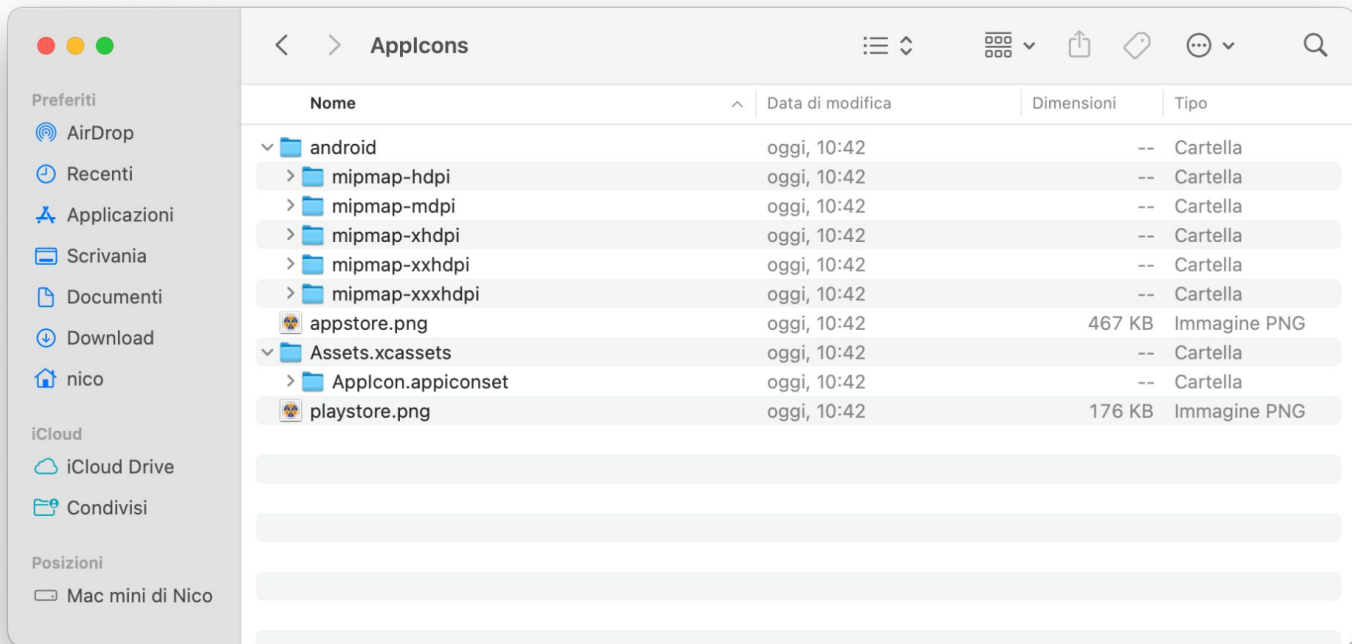
Android

- ✓ **Android** - 4 different sizes and files

File name	ic_launcher.png
-----------	-----------------

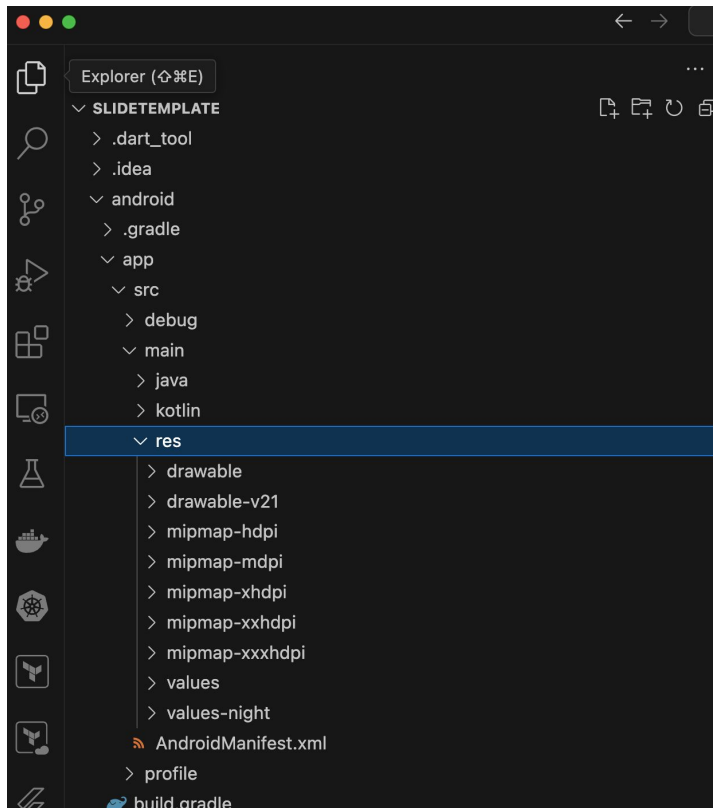
Change file name for all generated Android images

Icona dell'App



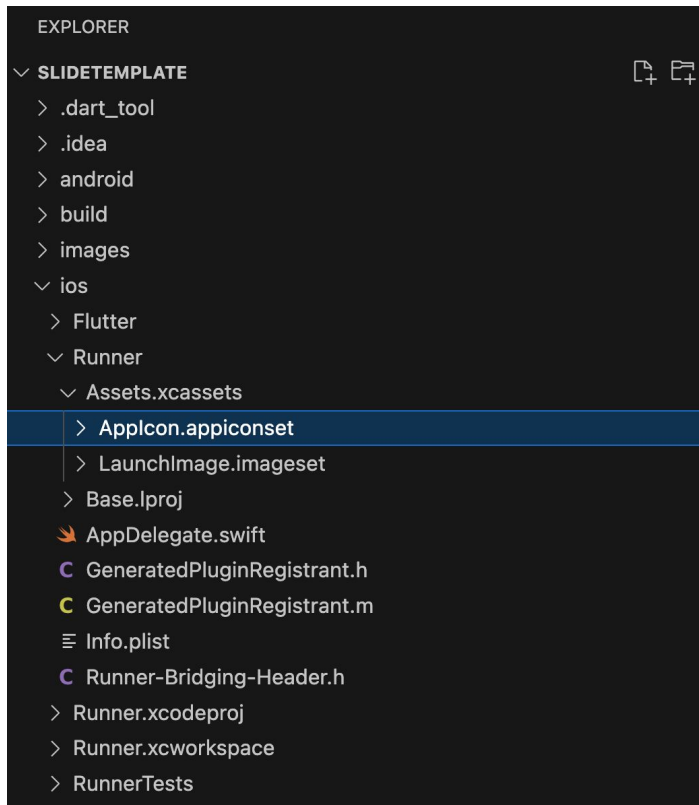
Icona dell'App

Per Android,
sostituire le directory
presenti in
android/src/main/res



Icona dell'App

Per iOS, sostituire la directory
AppIcon.appiconset
presente in
ios/Runner/Assets.xcassets



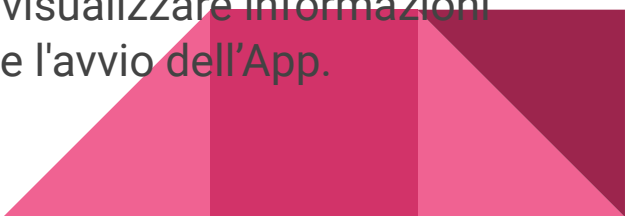
Splash Screen

La gestione della Splash Screen tra le applicazioni iOS e Android ha delle differenze riguardanti la gestione e la personalizzazione.

Su iOS, la Splash Screen (da linee guida iOS, Launch Screen) è controllata dal sistema operativo e viene mostrata automaticamente quando l'applicazione viene avviata.

L'immagine mostrata è definita nella directory `ios/Runner/Assets.xcassets/LaunchImage.imageset` e può essere personalizzata con un'immagine statica.

D'altra parte, su Android, la Splash Screen è più flessibile e personalizzabile. È possibile usare un'immagine statica ma anche fornire delle animazioni, visualizzare informazioni aggiuntive o implementare altre logiche personalizzate durante l'avvio dell'App.



Splash Screen (riepilogo)

- **Scelta dell'immagine di splash:** È necessario avere un'immagine di splash adatta alle dimensioni e alla risoluzione per entrambe le piattaforme. Se si tratta di immagini statiche, è possibile sostituirle nella rispettiva directory del progetto Flutter;
- **Utilizzo di un package come ad esempio [flutter_native_splash](#):** È possibile utilizzare questo package per generare automaticamente le splash screen native per iOS e Android. Questo package permette di personalizzare l'aspetto della splash screen e gestire le dimensioni e la posizione dell'immagine.



Installazione su dispositivi fisici (Android)

- **Abilitare la modalità sviluppatore su Android:** È necessario accedere ai settaggi (Settings) → About → Poi cercare Build Number. Non sempre è nella stessa posizione, in alcuni dispositivi è sotto Settings → About → Software information → More → Build Number. Una volta trovato bisogna effettuare 7 tocchi su Build Number per abilitare la modalità sviluppatore su quel dispositivo;
- **Abilitare USB debugging sul dispositivo:** Una volta abilitata la modalità sviluppatore all'interno di Settings sarà presente una nuova voce: Developer Options. All'interno di questa voce è necessario abilitare USB debugging;
- **Collegare il dispositivo al computer con un cavo dati:** Una volta collegato il dispositivo dovrebbe essere visto correttamente dall'IDE e sarà possibile installare applicazioni.



Installazione su dispositivi fisici (iOS)

- **Creazione di un Apple ID**
- **Installazione di Xcode**
- **Avere un dispositivo iPhone/iPad:** il dispositivo deve avere a sua volta una versione del sistema operativo compatibile con la versione di Xcode installata;
- **Installare homebrew:** Dopo l'installazione di homebrew (<https://brew.sh/>), installare ideviceinstaller, ios-deploy, cocoapods;
- **Configurare l'Apple ID su Xcode**
- **Creare un team di development**
- **Collegare il dispositivo**
- **Creare dei certificati di sviluppo**
- **Verificare che tutto sia correttamente installato eseguendo “flutter doctor”**

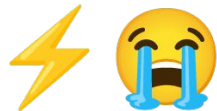
Più in generale seguire la guida ufficiale di Flutter:

<https://docs.flutter.dev/get-started/install/macos/mobile-ios#configure-ios-development>



L'Hot reload non funziona

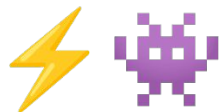
La cosa è andata più o meno così: “L'Hot Reload non funziona! Professore, lei ci ha mentito!”



L'Hot reload non funziona

Per far funzionare l'Hot Reload, ci vuole qualcosa in più.

Lo vedremo in questa lezione.

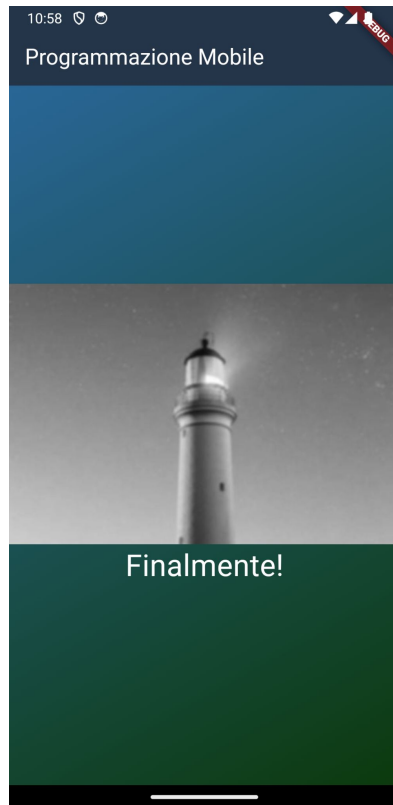




Dove eravamo
rimasti?

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
```



AssetImage

! pubspec.yaml ×

! pubspec.yaml > {} flutter > [] assets >  0

```
38   dev_dependencies:
47     flutter_lints: ^3.0.0
48
49   # For information on the generic Dart part of this file, see the
50   # following page: https://dart.dev/tools/pub/pubspec
51
52   # The following section is specific to Flutter packages.
53   flutter:
54     # The following line ensures that the Material Icons font is
55     # included with your application, so that you can use the icons in
56     # the material Icons class.
57     uses-material-design: true
58
59     # To add assets to your application, add an assets section, like this:
60     assets:
61       - images/
62
```

Comandi eseguiti dall'IDE in fase di build

flutter clean

Elimina tutti i file generati durante le precedenti compilazioni. Viene eseguito per assicurarsi che la build parta da uno stato pulito

flutter pub get

Scarica tutte le dipendenze definite nel file pubspec.yaml e le installa localmente nel progetto. È necessario per assicurarsi che tutte le dipendenze siano presenti e aggiornate

flutter build

Effettua la compilazione effettiva del progetto. A seconda delle opzioni specificate, può generare un'app per una piattaforma specifica (ad esempio, iOS o Android) o può generare un'app per tutte le piattaforme supportate

Il nostro codice sta crescendo in altezza...


```
41         style: TextStyle(  
42             fontSize: 30, color: Colors.white  
43         ), // Text  
44     ], // <Widget>[]  
45 ), // Column  
46 ), // Center  
47 ), // Container  
48 ), // Scaffold  
49 ), // MaterialApp  
50 );  
51 }
```



Stateless Widget

Un **StatelessWidget** è un widget che non ha stato interno. Questo significa che non può essere modificato dopo essere stato creato. È utile per widget che mostrano informazioni statiche o che non cambiano nel tempo. Ad esempio, un'icona, un testo statico o un pulsante che non necessita di aggiornamenti durante l'esecuzione dell'App.

Si contrappone agli **StatefulWidget** (che non vedremo oggi) e serve per scrivere codice più efficiente, chiaro e facilmente manutenibile, contribuendo alla creazione di App robuste e performanti.

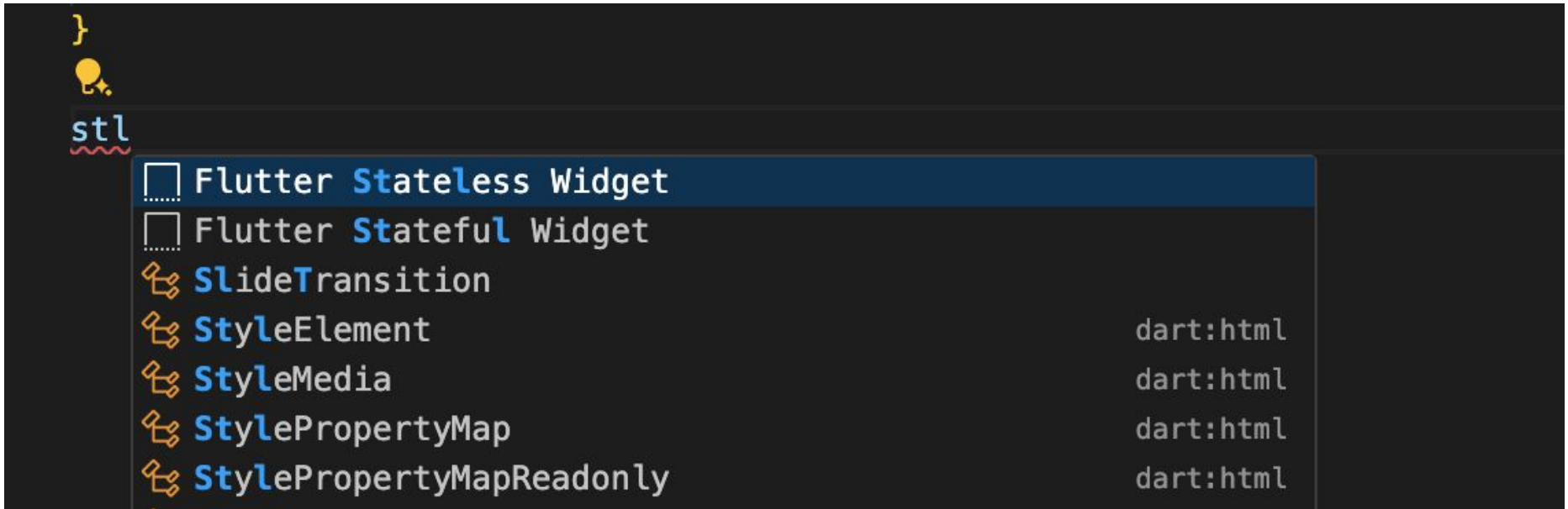


Stateless Widget

- **Efficienza delle prestazioni:** Gli StatelessWidget sono immutabili, il che significa che una volta creati, non possono essere modificati. Questo li rende molto efficienti in termini di prestazioni perché non è necessario gestire eventuali aggiornamenti di stato o ricalcolare l'interfaccia utente quando non ci sono modifiche;
- **Riutilizzo del codice:** Possono essere facilmente riutilizzati in diverse parti dell'App senza dover gestire lo stato condiviso. Questo promuove una progettazione modulare e favorisce la creazione di componenti UI altamente riusabili. Inoltre usarli può rendere il codice più chiaro e più semplice da comprendere;
- **Prevenzione di errori:** Gli StatelessWidget non possono essere modificati dopo la creazione, di conseguenza viene ridotta la probabilità di errori dovuti a modifiche accidentali dello stato o a comportamenti imprevisti.



Un aiuto dal plugin Flutter



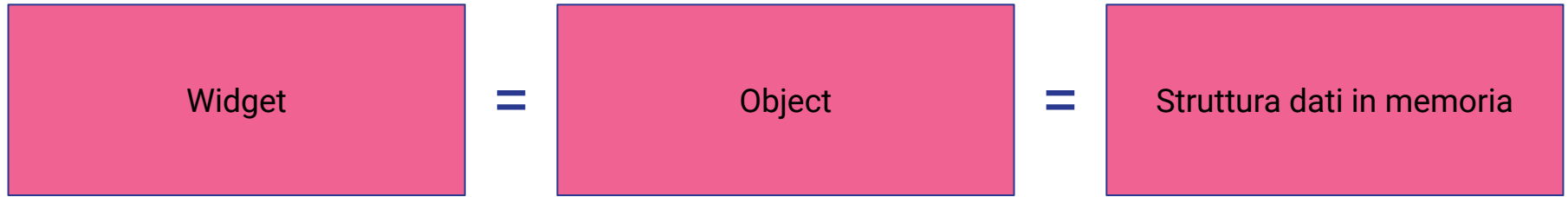
The screenshot shows an IDE with a dark theme. A code completion menu is open for the prefix 'stl'. The menu lists several Flutter-related classes and widgets. The first two items, 'Flutter StatelessWidget' and 'Flutter StatefulWidget', are preceded by a small square icon. The remaining items are preceded by a small icon representing a widget or class. The text 'dart:html' appears to the right of the last three items in the list.

```
}  
stl  
Flutter StatelessWidget  
Flutter StatefulWidget  
SlideTransition  
StyleElement dart:html  
StyleMedia dart:html  
StylePropertyMap dart:html  
StylePropertyMapReadOnly dart:html
```

Un aiuto dal plugin Flutter

```
class MyWidget extends StatelessWidget {  
  const MyWidget({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

Widget = Object



Classi

Dart è un linguaggio object-oriented

Primitive

Text
'Ciao Unimol'

Numbers
1, 150, 11.32

Numbers
1, 150, 11.32

**Valori più
complessi**

**es. Widget, Color,
etc.**

Elementi creati a
partire da un
blueprint: **Le Classi**



Classi

Dart è un linguaggio object-oriented

Primitive

Text

'Ciao Unimol'

Numbers

1, 150, 11.32

Numbers

1, 150, 11.32

Le classi servono a definire gli oggetti. Questi oggetti possono essere Dati o Funzioni, che al loro volta saranno composti da variabili e metodi. Usiamo questo approccio per organizzare le nostre informazioni e per creare una separazione logica del nostro lavoro. Quindi una migliore organizzazione.

amenti creati a
artire da un
rint: **Le Classi**

Definizione di una nuova classe

```
class MyWidget extends StatelessWidget {}  
  const MyWidget({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

Questa è la dichiarazione di una nuova classe chiamata `MyWidget` che estende `StatelessWidget` (e quindi ne erediterà le proprietà). `StatelessWidget` è un widget che descrive parte dell'interfaccia utente che può dipendere solo dalla configurazione iniziale e non può cambiare nel tempo.

Definizione di una nuova classe

```
class MyWidget extends StatelessWidget {}  
  const MyWidget({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

Questo è un decorator che indica che il metodo successivo sovrascrive un metodo della superclasse.

Definizione di una nuova classe

```
class MyWidget extends StatelessWidget {}  
  const MyWidget({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

In questo caso implementare il metodo build è obbligatorio quando usiamo StatelessWidget. Questo metodo della superclasse, che restituisce un Widget, viene sovrascritto, ma a noi interessa perché ha la responsabilità di “descrivere” cosa dovrebbe “disegnare” il Widget. Prende un BuildContext in input che contiene informazioni sull'albero dei Widget.

Definizione di una nuova classe

```
class MyWidget extends StatelessWidget {\n  const MyWidget({super.key});\n\n  @override\n  Widget build(BuildContext context) {\n    return const Placeholder();\n  }\n}
```

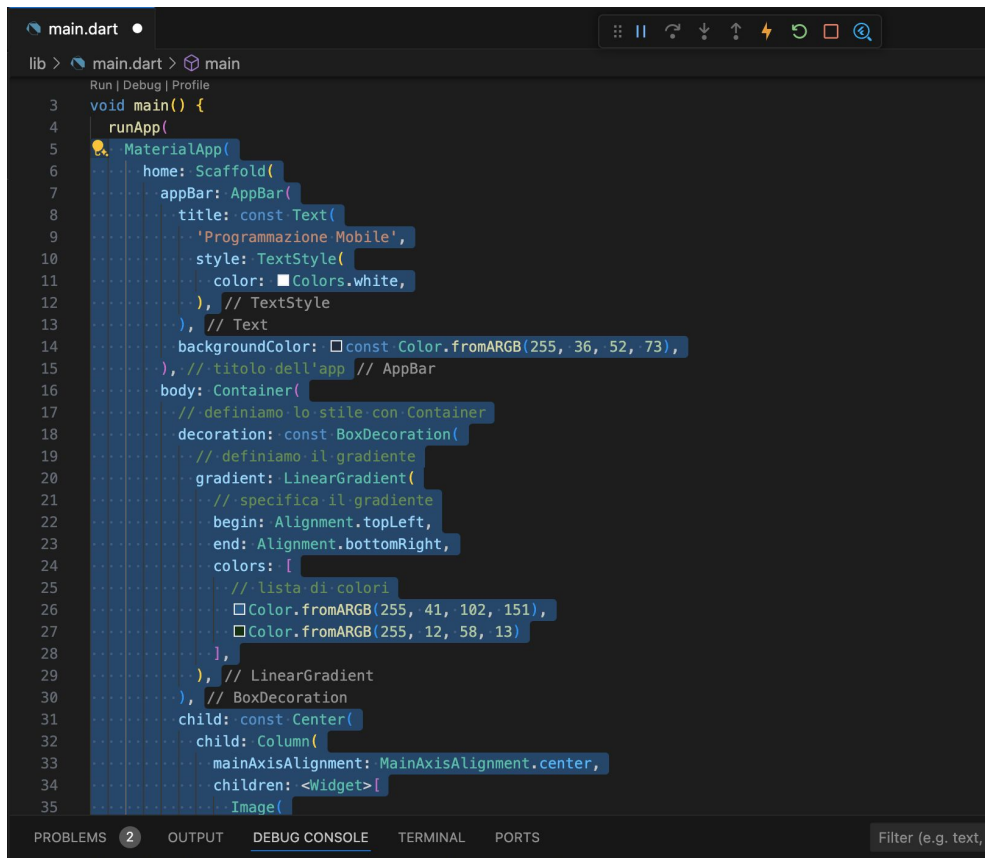
Questo è il costruttore per la classe `MyWidget`. Il costruttore è una funzione speciale che viene chiamata quando creiamo un'istanza di una classe. In questo caso, il costruttore accetta un parametro opzionale **key** che viene passato al costruttore della superclasse attraverso **super.key**.

Definizione di una nuova classe

```
class MyWidget extends StatelessWidget {}  
  const MyWidget({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

Questo è il corpo del metodo build. In questo caso, il widget restituisce un widget Placeholder, che è un widget che disegna una scatola durante lo sviluppo. Avendo const davanti al costruttore, significa che il widget Placeholder è immutabile, cioè non cambierà nel tempo. Ma a noi Placeholder() interessa poco. Noi vogliamo spostare in questa classe le nostre logiche...

Definizione di una nuova classe



```
main.dart
lib > main.dart > main
Run | Debug | Profile
3 void main() {
4   runApp(
5     MaterialApp(
6       home: Scaffold(
7         appBar: AppBar(
8           title: const Text(
9             'Programmazione Mobile',
10            style: TextStyle(
11              color: Colors.white,
12            ), // TextStyle
13          ), // Text
14          backgroundColor: const Color.fromARGB(255, 36, 52, 73),
15        ), // titolo dell'app // AppBar
16        body: Container(
17          // definiamo lo stile con Container
18          decoration: const BoxDecoration(
19            // definiamo il gradiente
20            gradient: LinearGradient(
21              // specifica il gradiente
22              begin: Alignment.topLeft,
23              end: Alignment.bottomRight,
24              colors: [
25                // lista di colori
26                Color.fromARGB(255, 41, 102, 151),
27                Color.fromARGB(255, 12, 58, 13)
28              ],
29            ), // LinearGradient
30          ), // BoxDecoration
31          child: const Center(
32            child: Column(
33              mainAxisAlignment: MainAxisAlignment.center,
34              children: <Widget>[
35                Image(
```

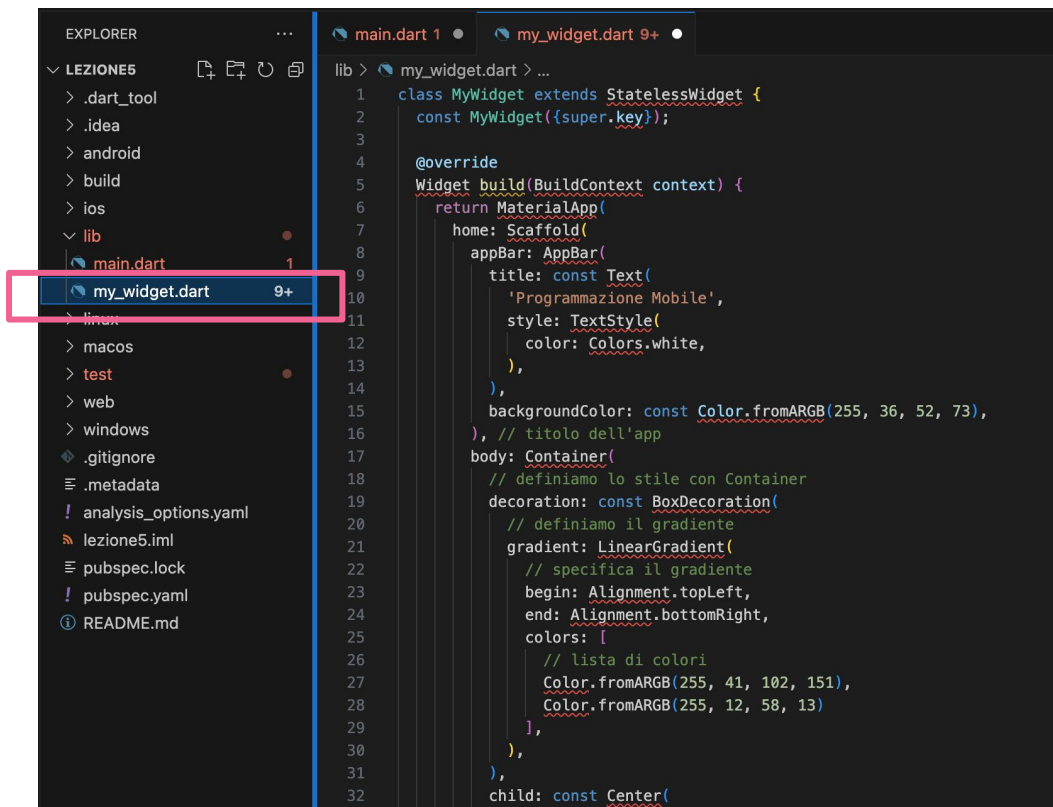
Definizione di una nuova classe

```
main.dart 2 ●
lib > main.dart > MyWidget > build
1  import 'package:flutter/material.dart';
2
3  Run | Debug | Profile
4  void main() {
5      runApp(
6      );
7  }
8
9  class MyWidget extends StatelessWidget {
10     const MyWidget({super.key});
11
12     @override
13     Widget build(BuildContext context) {
14         return ;
15     }
16 }
```

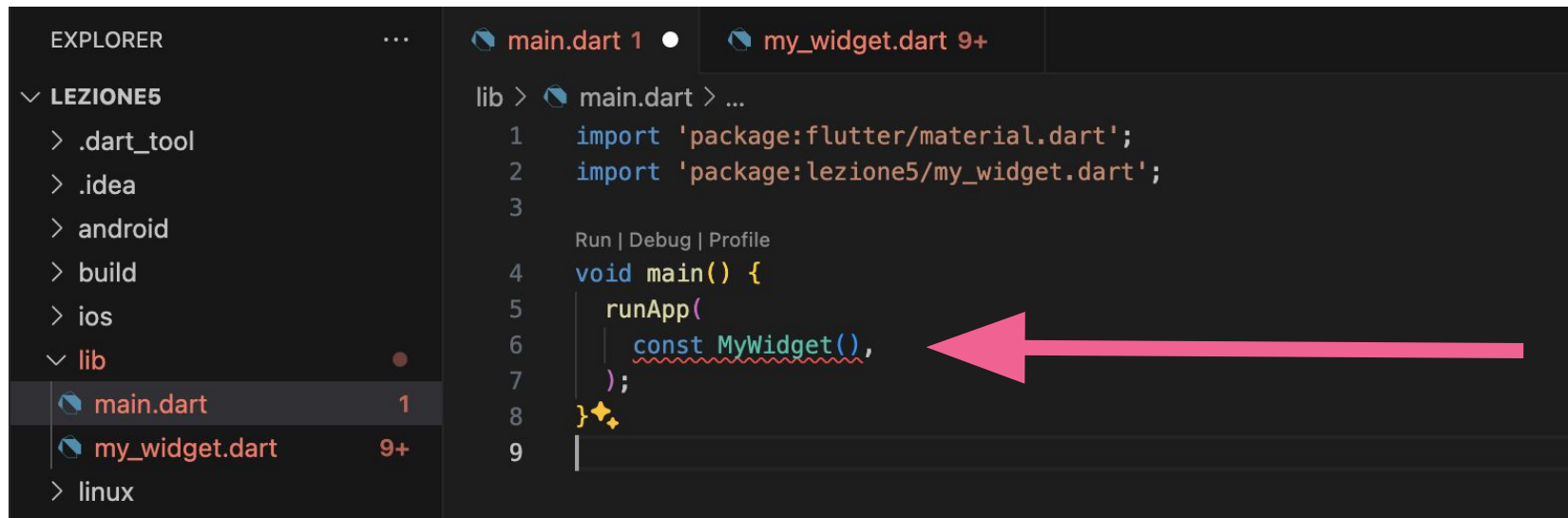
Definizione di una nuova classe

```
main.dart • [Icons] [Run] [Debug] [Profile] [Download] [Up Arrow]
lib > main.dart > main
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(
5     const MyWidget(),
6   );
7 }
8
9 class MyWidget extends StatelessWidget {
10   const MyWidget({super.key});
11
12   @override
13   Widget build(BuildContext context) {
14     return MaterialApp(
15       home: Scaffold(
16         appBar: AppBar(
```

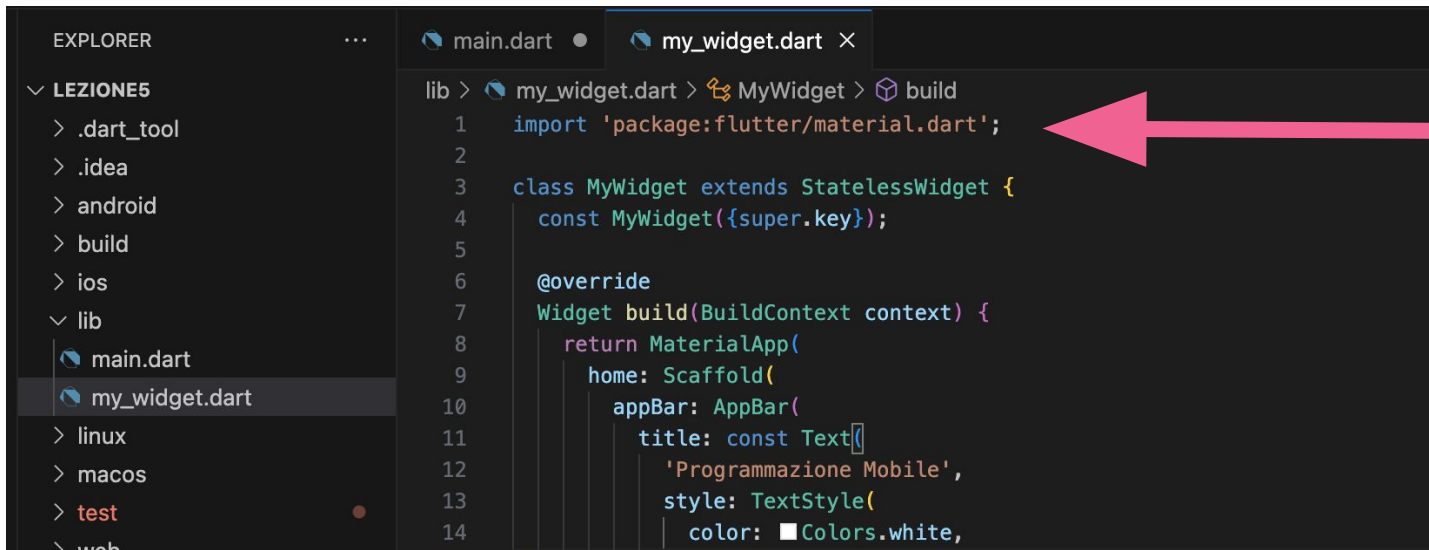
Dividere le classi in file differenti



Dividere le classi in file differenti



Dividere le classi in file differenti



Esercizio 1

Partendo dal codice sviluppato durante la scorsa lezione, dividete tutto in almeno due classi e spostate le classi in file differenti (almeno due file oltre il main.dart)

Utilizzo di variabili

```
import 'package:flutter/material.dart';
var startAlign = Alignment.topLeft;
var endAlign = Alignment.bottomRight;
class MyWidget extends StatelessWidget {
  const MyWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      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(
          // definiamo lo stile con Container
          decoration: const BoxDecoration(
            // definiamo il gradiente
            gradient: LinearGradient(
              // specifica il gradiente
              begin: startAlign,
              end: endAlign,
              colors: [
                // lista di colori
                Color.fromARGB(255, 41, 102, 151),
                Color.fromARGB(255, 12, 58, 13)
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Utilizzo di variabili - Errore

Errore: Non posso usare delle variabili (che potrei riassegnare) per un costruttore definito con const (in questo caso BoxDecoration)

```
import 'package:flutter/material.dart';
var startAlign = Alignment.topLeft;
var endAlign = Alignment.bottomRight;
class MyWidget extends StatelessWidget {
  const MyWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      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(
          // definiamo lo stile con Container
          decoration: const BoxDecoration(
            // definiamo il gradiente
            gradient: LinearGradient(
              // specifica il gradiente
              begin: startAlign,
              end: endAlign,
              colors: [
                // lista di colori
                Color.fromARGB(255, 41, 102, 151),
                Color.fromARGB(255, 12, 58, 13)
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Utilizzo di variabili - Errore

Una possibile
soluzione è
rimuovere
const da
BoxDecoration
...

```
import 'package:flutter/material.dart';
var startAlign = Alignment.topLeft;
var endAlign = Alignment.bottomRight;
class MyWidget extends StatelessWidget {
  const MyWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      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(
          // definiamo lo stile con Container
          decoration: const BoxDecoration(
            // definiamo il gradiente
            gradient: LinearGradient(
              // specifica il gradiente
              begin: startAlign,
              end: endAlign,
              colors: [
                // lista di colori
                Color.fromARGB(255, 41, 102, 151),
                Color.fromARGB(255, 12, 58, 13)
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Utilizzo di variabili - Errore

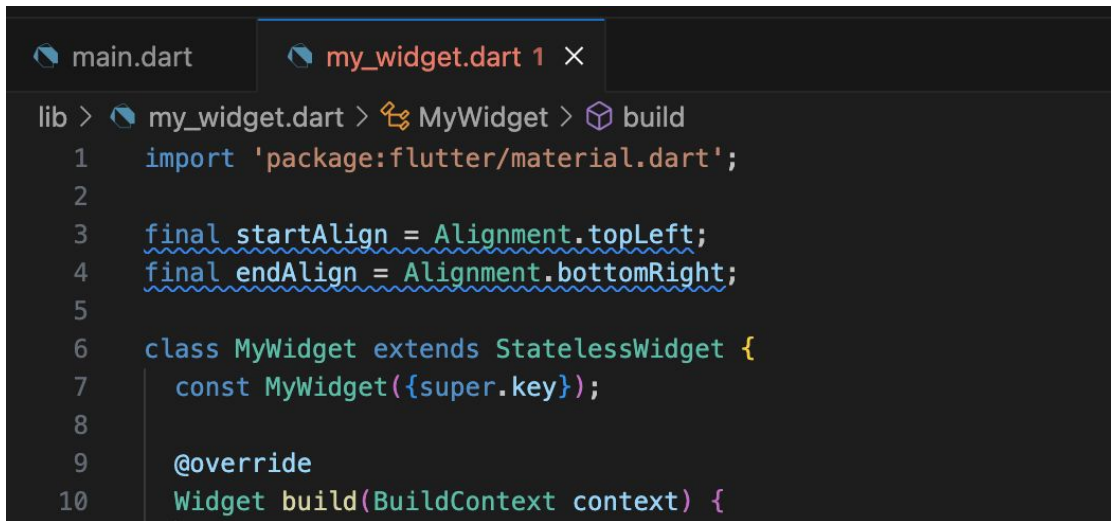
Un'altra è quella di cambiare il tipo alle variabili per non renderle più "variabili"

```
import 'package:flutter/material.dart';
var startAlign = Alignment.topLeft;
var endAlign = Alignment.bottomRight;
class MyWidget extends StatelessWidget {
  const MyWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      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(
          // definiamo lo stile con Container
          decoration: const BoxDecoration(
            // definiamo il gradiente
            gradient: LinearGradient(
              // specifica il gradiente
              begin: startAlign,
              end: endAlign,
              colors: [
                // lista di colori
                Color.fromARGB(255, 41, 102, 151),
                Color.fromARGB(255, 12, 58, 13)
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

Final

Si usa “final” quando sappiamo che un contenitore di informazioni non riceverà mai nuovi valori, anche per essere maggiormente restrittivi ed evitare che vengano sovrascritte le informazioni, ma forse si può fare di meglio...



```
main.dart  my_widget.dart 1 x
lib > my_widget.dart > MyWidget > build
1  import 'package:flutter/material.dart';
2
3  final startAlign = Alignment.topLeft;
4  final endAlign = Alignment.bottomRight;
5
6  class MyWidget extends StatelessWidget {
7    const MyWidget({super.key});
8
9    @override
10   Widget build(BuildContext context) {
```

Const

Come final, ma il contenuto è noto a tempo di compilazione

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

const startAlign = Alignment.topLeft;
const endAlign = Alignment.bottomRight;

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      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(
          // definiamo lo stile con Container
          decoration: const BoxDecoration(
            // definiamo il gradiente
            gradient: LinearGradient(
              // specifica il gradiente
              begin: startAlign,
              end: endAlign,
              colors: [
                // definiamo i colori
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```


Tipi di variabile

var

Viene utilizzato per dichiarare una variabile il cui tipo viene inferito dal valore assegnato ad essa al momento della dichiarazione. In altre parole, il tipo della variabile viene determinato in fase di compilazione in base al tipo del valore assegnato. Una volta che il tipo è stato inferito, non può essere cambiato durante l'esecuzione del programma

final

Viene utilizzato per dichiarare una variabile che può essere assegnata solo una volta e deve essere inizializzata prima dell'accesso. Una volta assegnato un valore, non può essere cambiato

const

Vene utilizzato per dichiarare una variabile costante il cui valore è noto a tempo di compilazione. Le variabili **const** sono implicitamente **final**. Il che significa che una volta assegnato un valore, non può essere cambiato

Esercizio 2

Cos'è il type dynamic? Perché tendiamo a non usarlo? Const è sempre meglio di final? Allora perché usiamo final?

Uso di variabili di istanza

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

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

  @override
  Widget build(BuildContext context) {
    return Text(
      'Finalmente!!!',
      style: TextStyle(fontSize: 30, color: Colors.white), // stile del testo
    ); // Text
  }
}

// Path: lib/my_widget.dart
```

Creo una classe con il testo “stilizzato”
usato nella nostra App

Uso di variabili di istanza

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

class MyText extends StatelessWidget {
  const MyText(String text, {super.key});

  @override
  Widget build(BuildContext context) {
    return Text(
      text,
      style: TextStyle(fontSize: 30, color: Colors.white), // stile del testo
    ); // Text
  }
}
```

Aggiungo un parametro posizionale nel costruttore e chiedo che sia usato al posto della stringa statica

Uso di variabili di istanza

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

class MyText extends StatelessWidget {
  const MyText(this.text, {super.key});

  // Aggiungiamo una variabile di classe per il testo
  String text;

  @override
  Widget build(BuildContext context) {
    return Text(
      text,
      style: const TextStyle(fontSize: 30, color: Colors.white), // stile del testo
    ); // Text
  }
}
```

Aggiungo una variabile di istanza

Uso di variabili di istanza

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

class MyText extends StatelessWidget {
  const MyText(this.text, {super.key});

  // Aggiungiamo una variabile di classe per il testo
  final String text;

  @override
  Widget build(BuildContext context) {
    return Text(
      text,
      style: const TextStyle(fontSize: 30, color: Colors.white), // stile del testo
    ); // Text
  }
}
```

La definisco final, perché?

Uso di variabili di istanza

```

    // lista di colori
    Color.fromARGB(255, 41, 102, 151),
    Color.fromARGB(255, 12, 58, 13)
  ],
), // LinearGradient
), // BoxDecoration
child: const Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Image(
        image
      ), // I
      MyText[],
    ], // <Widget>[]
  ), // Column
), // Center
), // Container
), // Scaffold
); // MaterialApp
}
}
```

1 positional argument expected by 'MyText.new', but 0 found.
Try adding the missing argument. dart([not_enough_positional_arguments](#))

View Problem (⌘F8) Quick Fix... (⌘.)

Inserisco il valore per il parametro posizionale, la stringa che voglio mostrare

L'Hot reload non funziona

Per gli scettici, adesso funziona. *



Esercizio 3

Provate a passare i colori come parametri nella classe che contiene LinearGradient. Attenzione, vogliamo che i colori siano solo 2 e non un'intera lista...