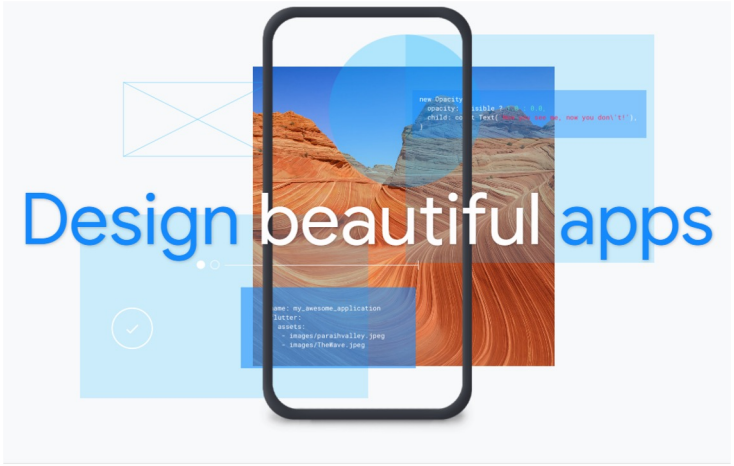
# Flutter

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Profesor: Carlos Andrés Parra
E-mail: ca.parraa@javeriana.edu.co

# Flutter

- Flutter es un framework de desarrollo móvil,web y de escritorio (windows) creado por Google.
- Presentado en 2015. La version 1 fue lanzado en diciembre de 2018.
- Basado en Material Design.



# Flutter

- Se utiliza el lenguaje de programación Dart de  Google
- Una base de código para las dos plataformas Android y iOS
- Generación de codigo **nativo!**
  - No se depende de un browser
  - No está basado en JavaScript ni HTML 5
- Se puede desarrollar con Linux, Windows o Mac usando Android Studio o Visual Studio Code
- Se pueden usar los mismos emuladores creados con Android Studio para Android, y los de XCode para iOS
  - *Si se desea probar la aplicación en iOS es necesario usar un computador con MacOS y XCode*

# Flutter vs otros frameworks

| | Flutter | React Native | Ionic | Xamarin | PhoneGap |
|---|---|---|---|---|---|
| **Desarrollado por:** | Google | Facebook | Ionic | Microsoft | Apache - Adobe |
| **Nativo** | Si | Si (UI + javascript) | No | No | No |
| **Usa un Browser** | No | No, Motor de JS | Si | No | Si |
| **Single-Code Base** | Si | Si | Si | No *(hasta 75% del código común)* | Si |
| **Lenguajes** | Dart | JavaScript/ TypeScript | JavaScript / TypeScript | C# | JavaScript - Html CSS |
| **Año** | 2018 | 2015 | 2013 | 2011 | 2009 |

# Flutter – Instalación

Windows | macOS | Linux

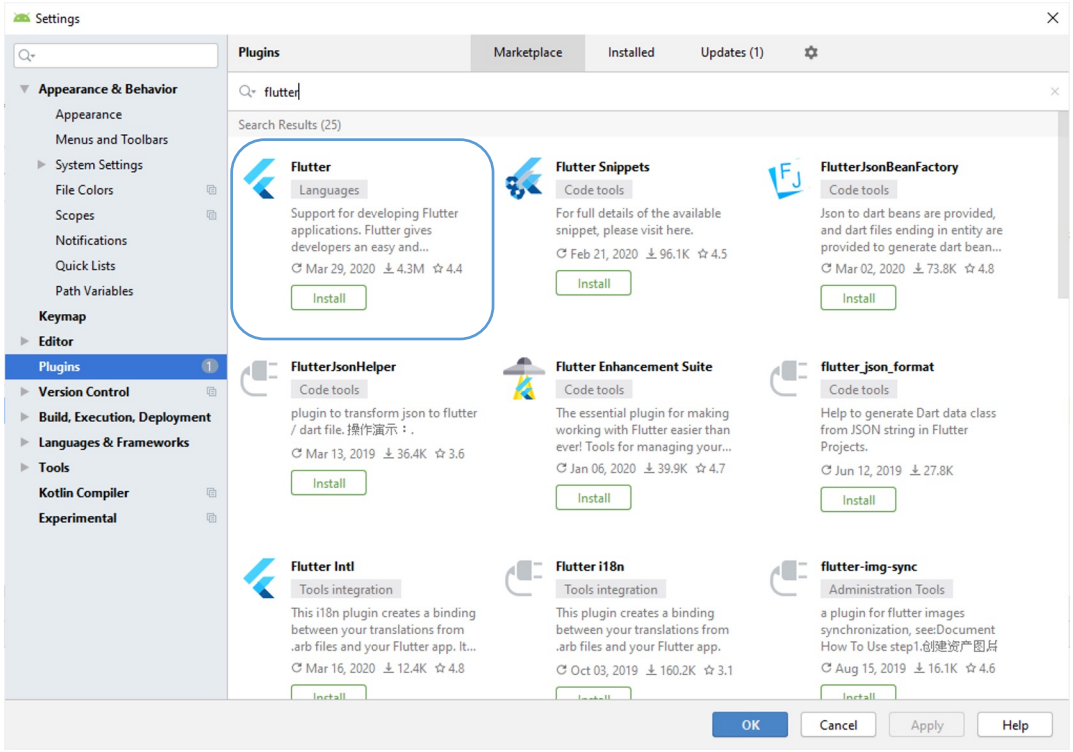
- Descargar Flutter SDK
- Extraer el contenido del SDK a una carpeta específica.
  - Alternativamente puede bajar la última versión estable de github
```
C:\src>git clone https://github.com/flutter/flutter.git -b stable
```
- Crear una entrada en el Path a *flutter/bin*
- Correr: `flutter doctor`

```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, v1.12.13+hotfix.8, on Microsoft Windows [Version 10.0.18362.720], locale en-US)

[√] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
[√] Android Studio (version 3.5)
[√] VS Code (version 1.43.2)
[√] Connected device (1 available)

• No issues found!

! Doctor found issues in 4 categories.
```

https://flutter.dev/docs/get-started/install/windows

# Flutter – Instalación Android Studio

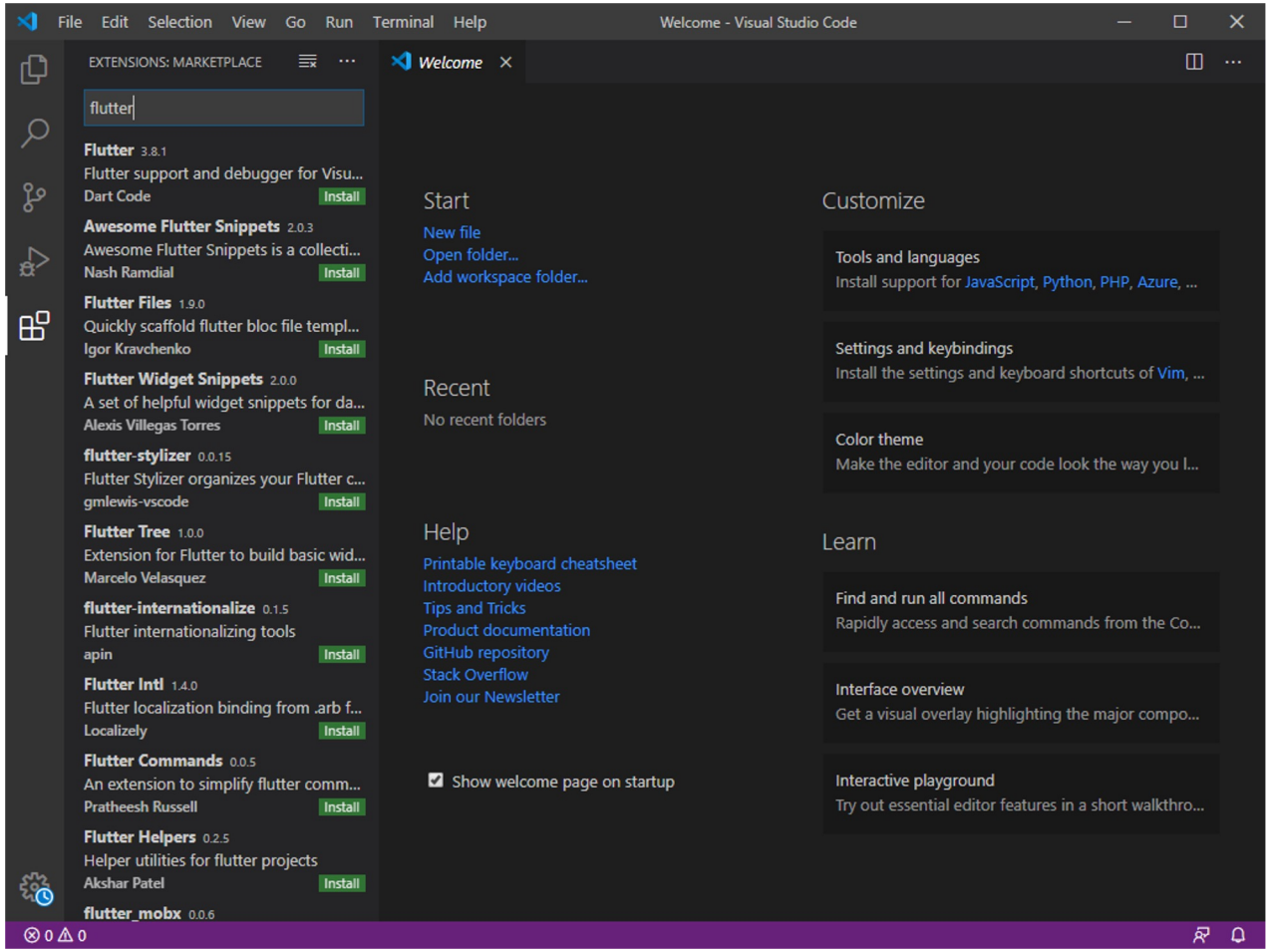
- Ir a File->Settings ->MarketPlace




- Aceptar la instalación de las dependencias (Dart) y reiniciar Android Studio

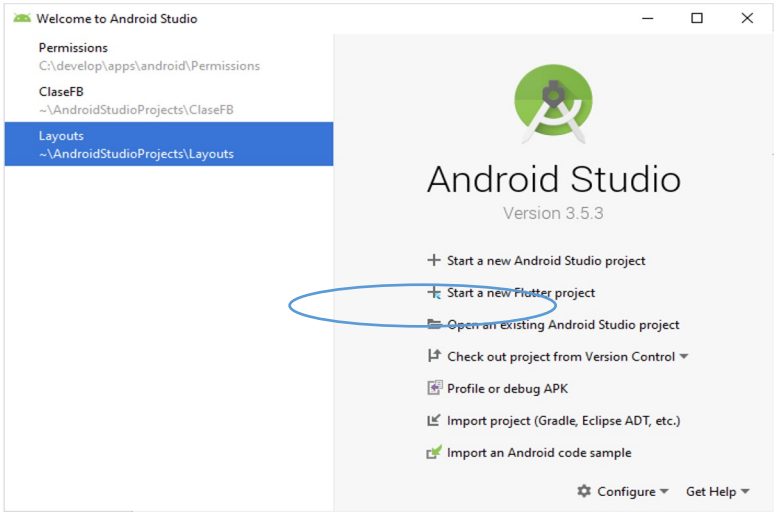
# VisualStudio Code

- Ir a Extensions
- Buscar Flutter e instalar
- Aceptar las instrucciones del asistente



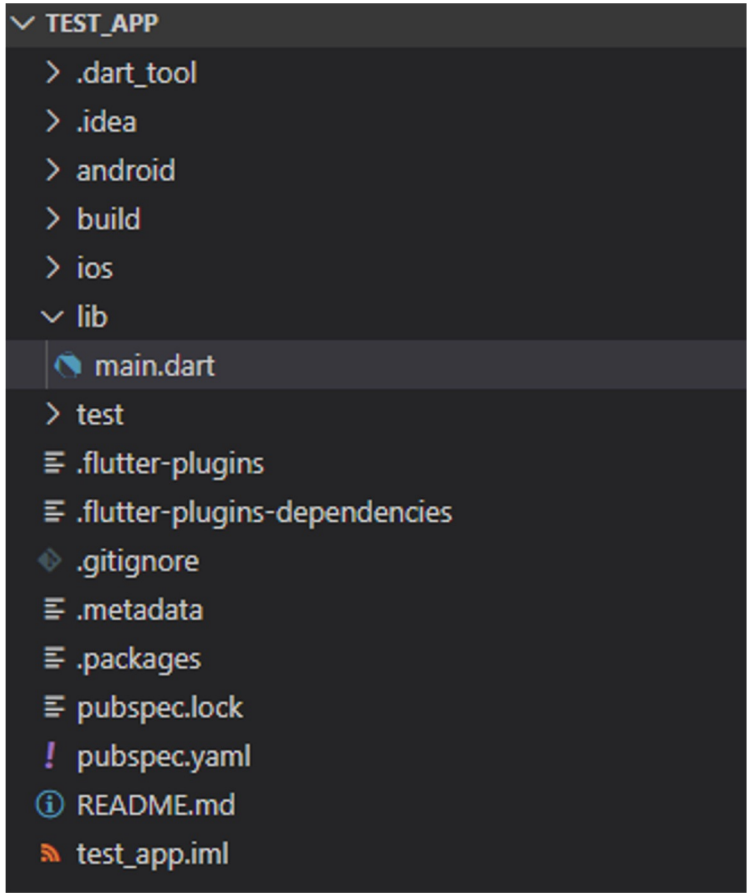
# Crear primera app

- En Android Studio:
  - Crear un nuevo proyecto de flutter y seguir las instrucciones




- Par VS Code, desde el directorio deseado correr:
  - `flutter create nombreApp`
  - Abrir la carpeta generada desde VSCode

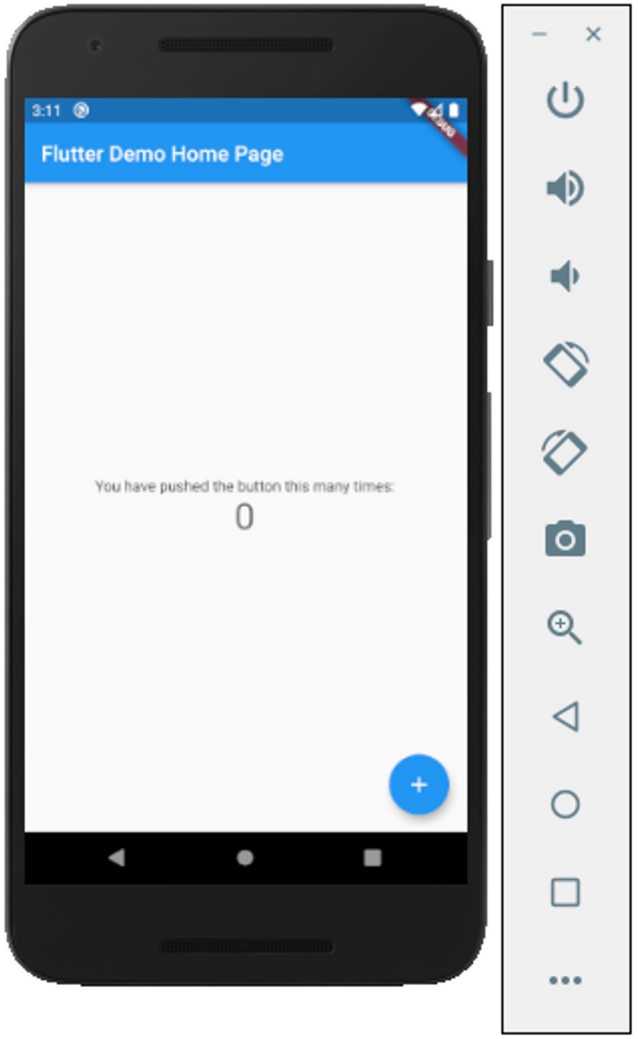
# Estructura del proyecto




- **lib**
  - Carpeta con el código fuente
- **test**
  - Carpeta con el código fuente de las pruebas
- **pubscpec.yaml**
  - Archivo de configuración del proyecto
- *android e ios*
  - *Carpetas generadas automáticamente para cada plataforma destino*
- *.*
  - *Archivos de configuración del proyecto*

# Correr el proyecto

- Desde Android Studio
  - Escoger o crear un emulador e iniciarlo
  - Ir al menú Run y seleccionar run.
- Desde Visual Studio Code
  - La primera vez iniciar el emulador desde Android Studio
  - Ir a la opción Run -> Run without debugging
  - El plugin de VS code detecta los emuladores disponibles y se puede seleccionar el que se desee (iOS, Android o Chrome)



# Dart – Conceptos básicos

- Playground web para Dart
  - https://dartpad.dev/
- Furtemente tipado
  - Inferencia y tipos

```dart
int a=6; //declaración normal
var b=5; //inferencia del tipo
final c=7; //no modificable pero asignable en ejecución
const d=1; //no modificable y sólo asignado al inicio del programa
print(a+b); //impresión normal
print("$a - $b - $c - $d"); //impresión con uso de cadena
```

```
11
6 - 5 - 7 - 1 - 1
```

# Dart – Conceptos básicos

- Existe un tipo de dato dinámico que puede cambiar en ejecución. Para esto es necesario usar la palabra reservada dynamic

```
dynamic dinamico = 8;
print(dinamico);
dinamico = "hola";
print (dinamico);
```

```
8
hola
```

# Dart – Conceptos básicos

- Constructor
  - Al igual que en java el constructor de la clase lleva el mismo nombre de la clase y los parámetros a inicializar. Sin embargo, se puede abreviar:

```
class Dog {
  String name;
  int age;
  void Dog(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

```
class Dog {
  String name;
  int age;
  Dog({this.name, this.age});
}
```

# Dart – Conceptos básicos

- Las funciones en Dart pueden tener parámetros con nombre y opcionales:

```
void bark(String language, int times){
   int counter = 0;
   print(language);
   while (counter < times){
     if(language == "English"){
       print("wofff");
     }  else {
       print("guau");
     }
     counter++;
   }
}
```

```
void barkTwo({String language, int times}){
   int counter = 0;
   print(language);
   while (counter < times){
         if(language == "English"){
         print("wofff");
      } else {
         print("guau");
      }
      counter++;
   }
}
```

- Invocar la función

```
dog.bark("English", 5);
```

```
dog.barkTwo(language: "English", times: 5);
dog.barkTwo(times: 5, language: "English");
dog.barkTwo(times: 5,);
```

# Dart – Conceptos básicos

- Referencias a funciones. Dart es un lenguaje funcional y permite tener referencias a funciones y pasarlas como parámetro:

```
floatingActionButton: FloatingActionButton(
    onPressed: buttonPressed, //referencia a la función
    tooltip: 'Increment',
    child: Icon(Icons.add),
),
```

```
void buttonPressed() {
  _counter++;
}
```

# Dart – Conceptos Básicos

- Funciones anónimas
  - Son funciones con un bloque de Código que no tienen un nombre. Si en el ejemplo anterior no se quisiera definir una función, el atributo onPressed se podría definir así:

```
floatingActionButton: FloatingActionButton(
        onPressed: () {
          _counter++;
                }, //fin función anónima
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ),
```

  - O usando el operador => para funciones que sólo tienen una línea:

```
onPressed: () => _counter++,
```

# Dart – Conceptos básicos

- List, es un conjunto de elementos

```
List<String> cadenas = ['hola', 'mundo', 'hola', 'mundo'];
print(cadenas[0]);
cadenas.add('valor');
print(cadenas);
```

```
hola
[hola, mundo, hola, mundo, valor]
```

- Map, conjunto llave valor

```
Map mapa = {'hola': 'mundo', 'entero': 5, 'double': 56.4};
print(mapa['double']);
print(mapa);
Map<String, String> mapaParametrizado = {'hola': 'mundo', 'hello': 'world'};
print(mapaParametrizado['hello']);
print(mapaParametrizado);
```

```
56.4
{hola: mundo, entero: 5, double: 56.4}
world
{hola: mundo, hello: world}
```

# Flutter

- Todo es un widget!!
  - Cada Widget tiene un método build asociado
- Los widgets pueden ser:
  - Stateless
    - No tienen estado, solo se corre el método ***build*** una vez y si tienen datos estos no pueden variar durante la ejecución (***final***)
  - Statefull
    - Tiene un estado asociado
    - Se corre el método ***build*** cada vez que cambia el estado
    - Se debe usa el método ***setState*** para actualizar el estado

# Flutter

- Flutter funciona de forma declarativa.
- Se define un árbol o jerarquía de widgets que deben aparecer en la pantalla.
- El método build se encarga de recorrer el árbol y dibujar los elementos que encuentre.
  - Si es un stateless widget, se hace una vez
  - Si es un stateful widget, se hace cada vez que cambie el estado

# Flutter - Widgets

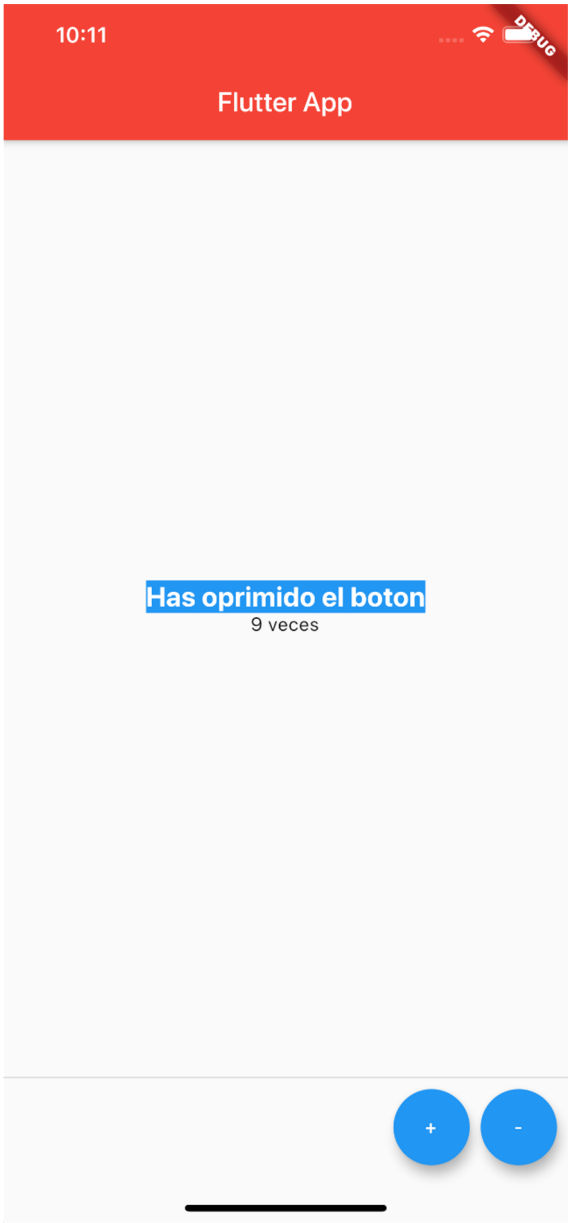
Flutter - *Android* - *iOS*

- Text (TextView, Label)
- TextField (EditText, TextField)
- Buttons
  - RaisedButton, FlatButton, FloatingActionButton
- Scaffold
- AppBar
- Column (LinearLayout vertical, StackView)
- Row (LinearLayout horizontal, StackView)
- ListView
- TileList
- Image

https://flutter.dev/docs/reference/widgets

# Primera App en Flutter

- Hola Mundo Flutter!
    1. Crear proyecto desde Android Studio o Terminal para VSCode
    2. Probar app generada
    3. Crear un AppBar y un Body (Column) con dos etiquetas, un campo de texto y un botón
    4. Programar la acción de los botones
    5. Probar en emulador Android
    6. Probar en emulador iOS



# Primera App Flutter

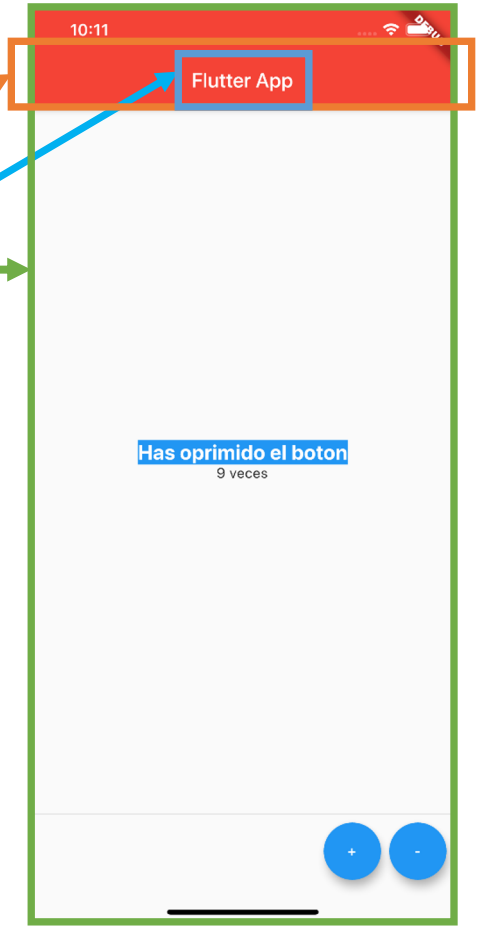
- Crear un widget StateFull (el estado es el contador) e inicializar la aplicación

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

void main() => runApp(MaterialApp(
            title: "Flutter App",
            home: MyWidget(),
));

class MyWidget extends StatefulWidget {
            @override
            _MyWidgetState createState() => _MyWidgetState();
}

class _MyWidgetState extends State<MyWidget> {…
```

# Primera App en Flutter

- En el método build del widget se programa la interfaz:

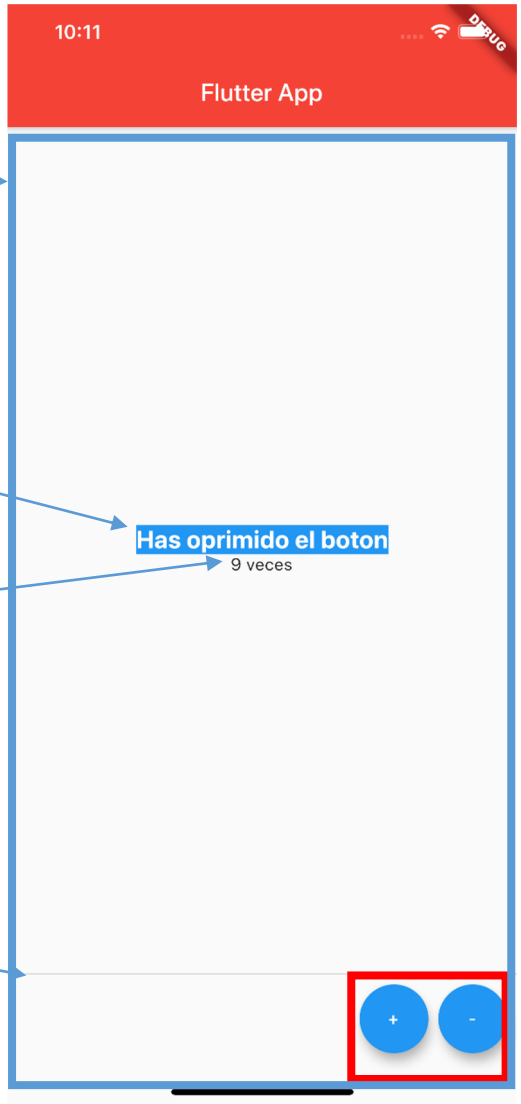
```
@override
Widget build(BuildContext context) {
return Scaffold(
            appBar: AppBar(
            title: Text("Flutter App"),
            backgroundColor: Colors.red,
),
//…
```



# Primera App en Flutter

- Body tiene el contenido de la “actividad”

```
body: Center(
   child: Column(
   mainAxisAlignment: MainAxisAlignment.center,
   children: <Widget>[
      Text("Has oprimido el boton",
          style: TextStyle(
                  backgroundColor: Colors.blue,
                  color: Colors.white,
                  fontSize: 20,
                  fontWeight: FontWeight.bold,
                ),
          ),
      Text("$counter veces"),
   ],
  ),
),
persistentFooterButtons: <Widget>[
   FloatingActionButton(child: Text("+"),  onPressed: incrementar,),
              FloatingActionButton(child: Text("-"),  onPressed: disminuir,),
],
```



# Primera App en Flutter

- Los métodos modifican el estado para que se vuelva a llamar a build y se actualice la presentación:

```
int counter=0;

void incrementar(){
        print("boton oprimido");
        setState(() {
                counter++;
        });
}
void disminuir(){
        setState(() {
                counter--;
        });
}
```

# Ejercicio

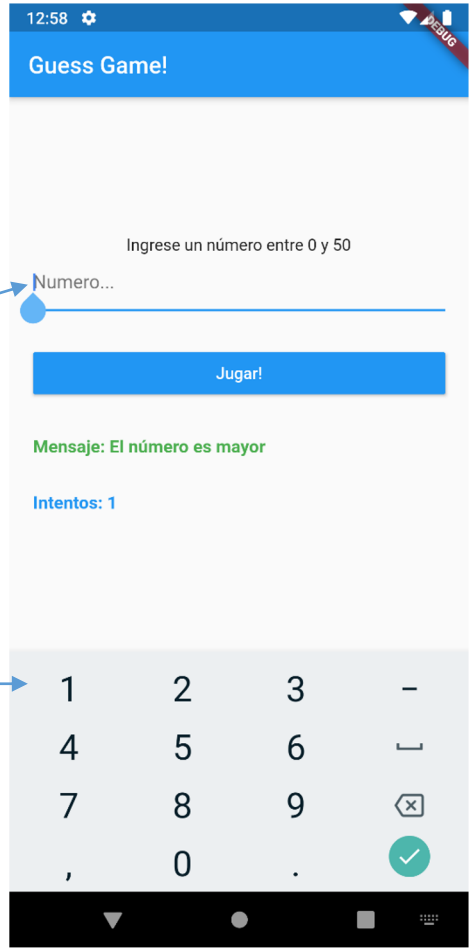
- Construir la aplicación demo en Flutter

# TextFields y controladores

- Para acceder al valor de un textfield en flutter, es necesario definir un controlador, y asignarlo al textfield

```
TextEditingController controller;
// = TextEditingController();
@override
void initState() {
        controller = TextEditingController();
        super.initState();
}
```

```
TextField(
        controller: this.controller,
        keyboardType: TextInputType.number,
        decoration: InputDecoration(
        hintText: "Numero...",),
),
```

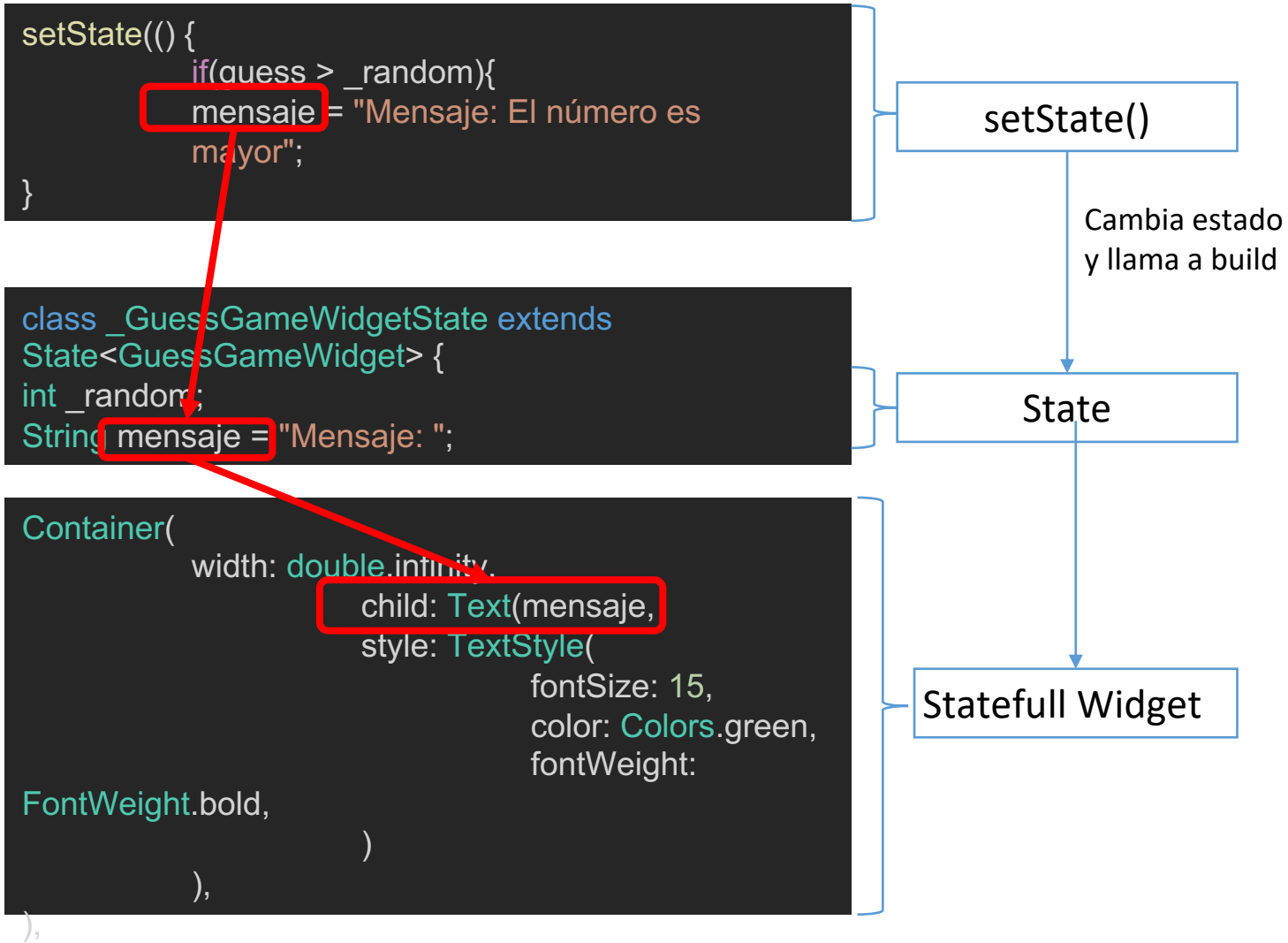


# TextFields y Controladores

- Para saber lo que el usuario ingresó en el TextField se accede a la propiedad text del controlador

```
if(controller.text.isNotEmpty){
                print(controller.text);
                int guess = int.parse(controller.text);
                //…
```

# Manejo del Estado



# GuessGame

- Construir la aplicación de GuessGame en Flutter!