Flutter

Stateful y Stateless

Carmelo Villegas Cruz

Creando layouts

En esta presentación vamos a centrarnos en el siguiente tema.

- ***Stateful*** y *stateless widgets*.
- Catálogo de widgets.
- Entender layout widgets.
- Crear widgets personalizados

Stateful vs Stateless

Las UI(Interfaces de usuarios) en la mayor parte de los casos no son estáticas.

Es por ello que necesitamos tener dos tipos de widgets, donde uno de ello reaccione ante los posibles eventos que pueden ocurrir.

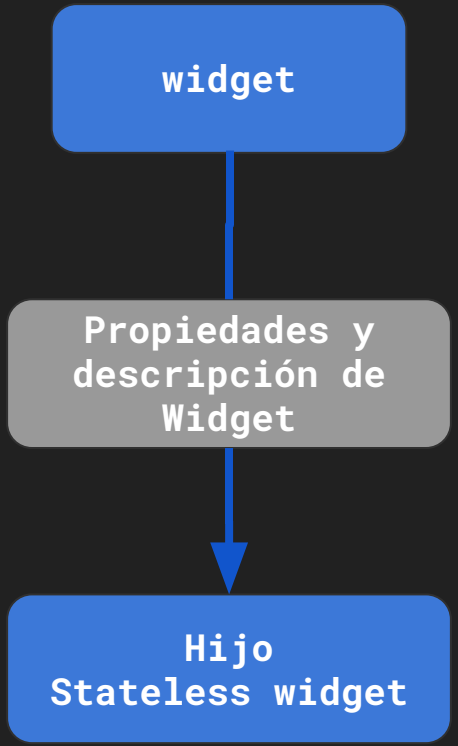
Stateless widget

Son aquellos que nunca van a cambiar sus propiedades después de ser instanciados.

No tienen estado, esto significa que no cambian por ellos mismos por ninguna acción o comportamiento.

La única forma de cambiar sería a causa de que algún evento en sus padres provoque que se construya de nuevo. Por lo que podemos decir que los stateless widgets ceden a sus padres el control de cómo ellos son creados.

Stateless widget



Stateless widget

En resumen, el widget hijo recibirá su descripción del padre y no cambiará por sí mismo.

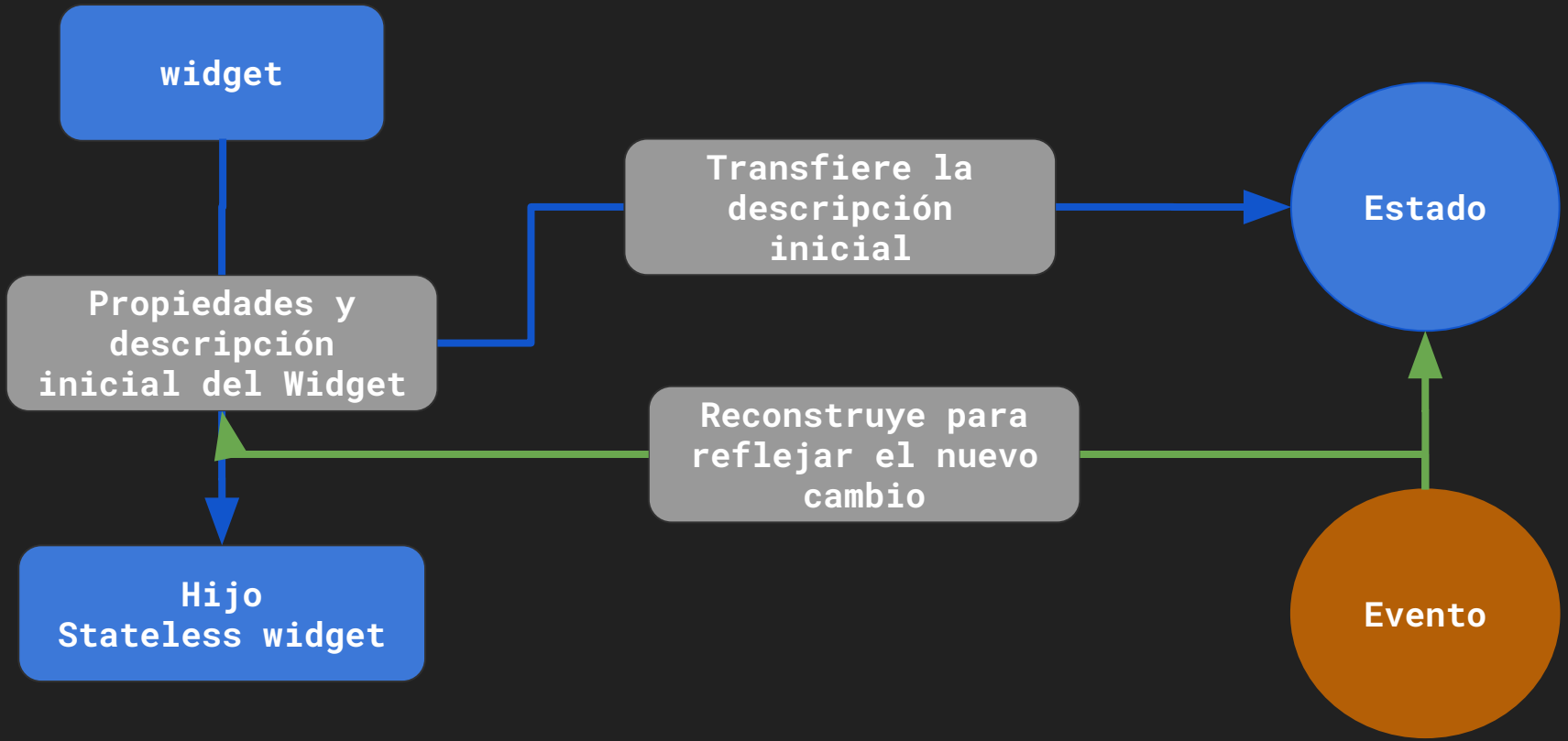
¿Qué modificador tendrá la definición de sus propiedades?
final

Stateful widget

Este tipo de widget está destinado a que su descripción cambie dinámicamente a lo largo del ciclo de vida de la aplicación.

Disponen de una clase llamada *State* que representa o contiene el estado actual. Es esta la que gestionará cómo , cuándo y qué acciones/eventos se deben producir para provocar un cambio en el widget asociado.

Stateful widget



Stateless en código

Para entender mejor los conceptos mencionados anteriormente es hora de verlo en código.

Para ello abrimos el proyecto por defecto que creamos la clase anterior.

```dart
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        //...
        primarySwatch: Colors.red,
        //...
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ), // ThemeData
      home: MyHomePage(title: 'Prueba Inicial'),
    ); // MaterialApp
  }
}
```

Stateless en código

A reseñar en el código anterior es que **MyApp** extiende de ***statelessWidget*** y sobreescribe el método ***build(BuildContext)***.

build(BuildContext), este método se encarga de describir la UI,construye los widgets situados más abajo en el árbol.

BuildContext sirve para interactuar con los elementos del árbol sobretodo para acceder a los ancestros

En el ejemplo anterior el elemento raíz sería MyApp que tendría un hijo de tipo MaterialApp.

Stateful en código

```
class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  //...

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

Stateful en código

Comentamos anteriormente que los stateful widget tienen asociado un objeto ***State***, ***_MyHomePageState***.

Al extender de statefulWidget nos vemos obligado a sobreescribir createState, este método tiene que devolver un ***State***.

Las clases **State** asociadas suelen ser implementadas en el mismo fichero y además suelen ser privadas también, ya que los elementos externos no necesitan interactuar con ella.

Stateful en código

```dart
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      //...
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {...}
}
```

Stateful en código

Como podemos observar un widget state extiende de la clase State.

En este ejemplo el estado es definido por una simple propiedad, ***_counter***. Este almacena el número de veces que el botón ha sido pulsado.

¿Quien construye el botón?
En este caso es el propio widget state el encargo de ello.

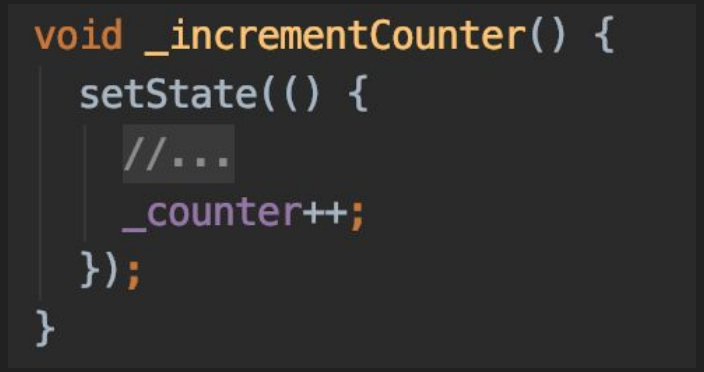
Stateful en código

```
floatingActionButton: FloatingActionButton(
  onPressed: _incrementCounter,
  tooltip: 'Increment',
  child: Icon(Icons.add),
), // This trailing comma makes auto-formatting
```

Podemos observar en el código tanto la definición del botón como la asignación de una acción al evento onPressed.

_incrementCounter es el método asociado al evento onPressed, veamos su código.

Stateful en código



```
void _incrementCounter() {
  setState(() {
    //...
    _counter++;
  });
}
```

Simplemente aumenta el valor de la variable que define el estado _counter, pero lo envuelve con setState(), ¿qué es?

Pues es la manera que tenemos de decir que es necesario hacer un rebuild y volver a pintar todo con los nuevos valores.

Stateful en código

```
Text(
  '$_counter',
  style: Theme.of(context).textTheme.headline4,
), // Text
```

Cuando se acciona el botón se invoca a ***_incrementCounter*** y está a su vez provoca un rebuild cambiando el valor de _counter, por lo que se llamaría de nuevo al método build y sus descendientes.

Inherited widgets

Además de stateful y stateless widgets existe otro tipo llamado ***InheritedWidget***

A veces necesitamos tener acceso a datos situados en widget situados en más arriba en nuestra jerarquía de árbol y para evitar que ese datos sea replicado en cada

Inherited widgets