



**GDG**

**State Management  
2020**



# Flutter

## State Management



Level 0



# GDG

# State Management



- Introduction
- Think declaratively
- Stateful and stateless widgets
- Flutter life cycle
- Ephemeral vs app state
- Flutter design pattern



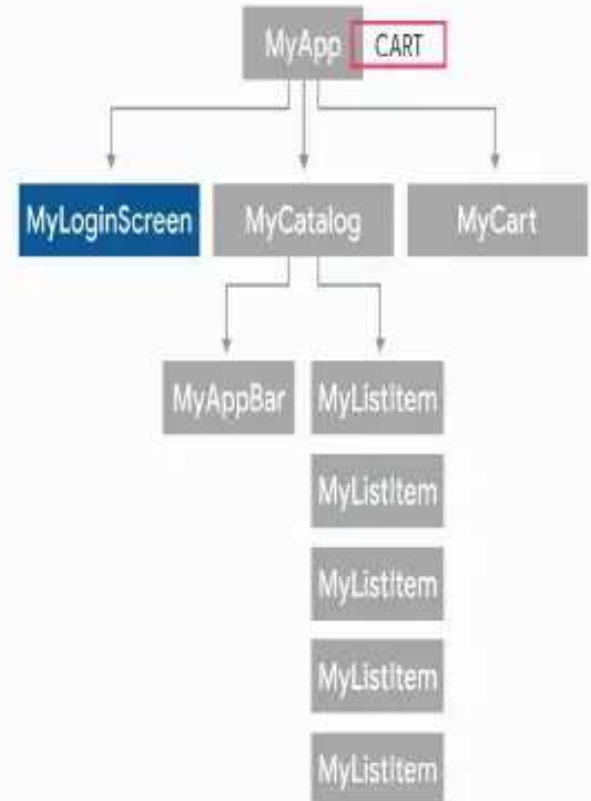
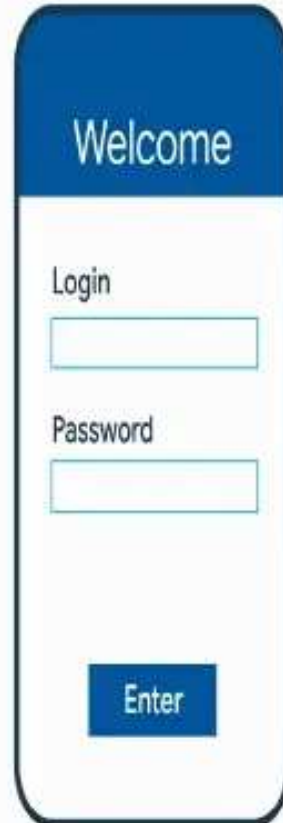
# Introducción

---



# Estado

- A medida que explora Flutter, llega un momento en que necesita compartir el estado de la aplicación entre pantallas, a través de su aplicación.
- Hay muchos enfoques que puede tomar, y muchas preguntas para pensar.





## Flutter IU Declarativa

ViewA a

ViewB b

ViewC c1

ViewC c2



ViewA a

ViewB b

ViewC c3

## Imperativo

```
// Imperative style  
b.setColor(red)  
b.clearChildren()  
ViewC c3 = new ViewC(...)  
b.add(c3)
```

## Declarativo

```
// Declarative style  
return ViewB(  
  color: red,  
  child: ViewC(...),  
)
```



# UI = f( state )

The layout  
on the screen

Your  
build  
methods

The application state



main.dart

```
29
30 void _incrementCounter() {
31   setState(() {
32     _counter++;
33   });
34 }
35
36 @override
37 Widget build(BuildContext context) {
38   return new Scaffold(
39     appBar: new AppBar(
40       title: new Text(widget.title),
41     ), // AppBar
42     body: new Center(
43       child: new Text(
44         'Button tapped $_counter times',
45         style: Theme.of(context).textTheme.display1,
46       ), // Text
47     ), // Center
48     floatingActionButton: new FloatingActionButton(
49       onPressed: _incrementCounter,
50       tooltip: 'Increment'
```



Button tapped 4 times

# State Management



`int age;`

`final title;`

`Stateful Widget`

`var title;`

**State = Data**

`Stream <String> user;`

`String title;`

Your app **Data** related to Domain like . **TODO List**. **Cart Items** . etc..



# Declarative UI



$= f(\text{state})$



# Stateless y Stateful widgets

---



# StatelessWidget

constructor

build

Stateless

Icon

Chip

Text

```
class MenuWidgetImmutable extends StatelessWidget {  
  const MenuWidgetImmutable({Key key}) : super(key:  

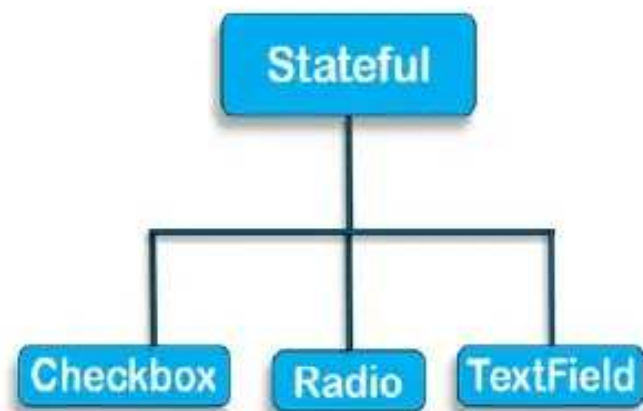
```

override

```
Widget build(BuildContext context) {
```



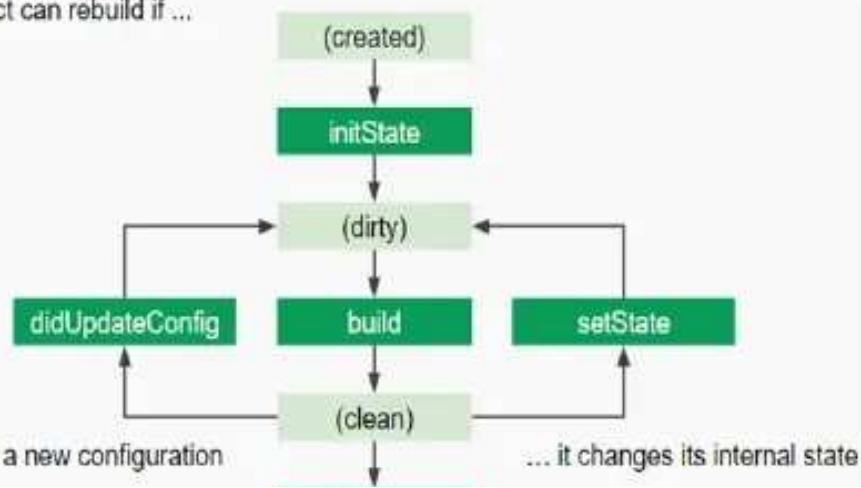
```
return Text('I will not have my status cha
```



```
setState(() {  
    firstName = 'Juan';  
});  
  
setStateWith(() {  
    firstName = 'Juan';  
});  
  
setStateAfter(() {  
    firstName = 'Juan';  
});
```



A State<T> object can rebuild if ...



## StatefulWidget

constructor

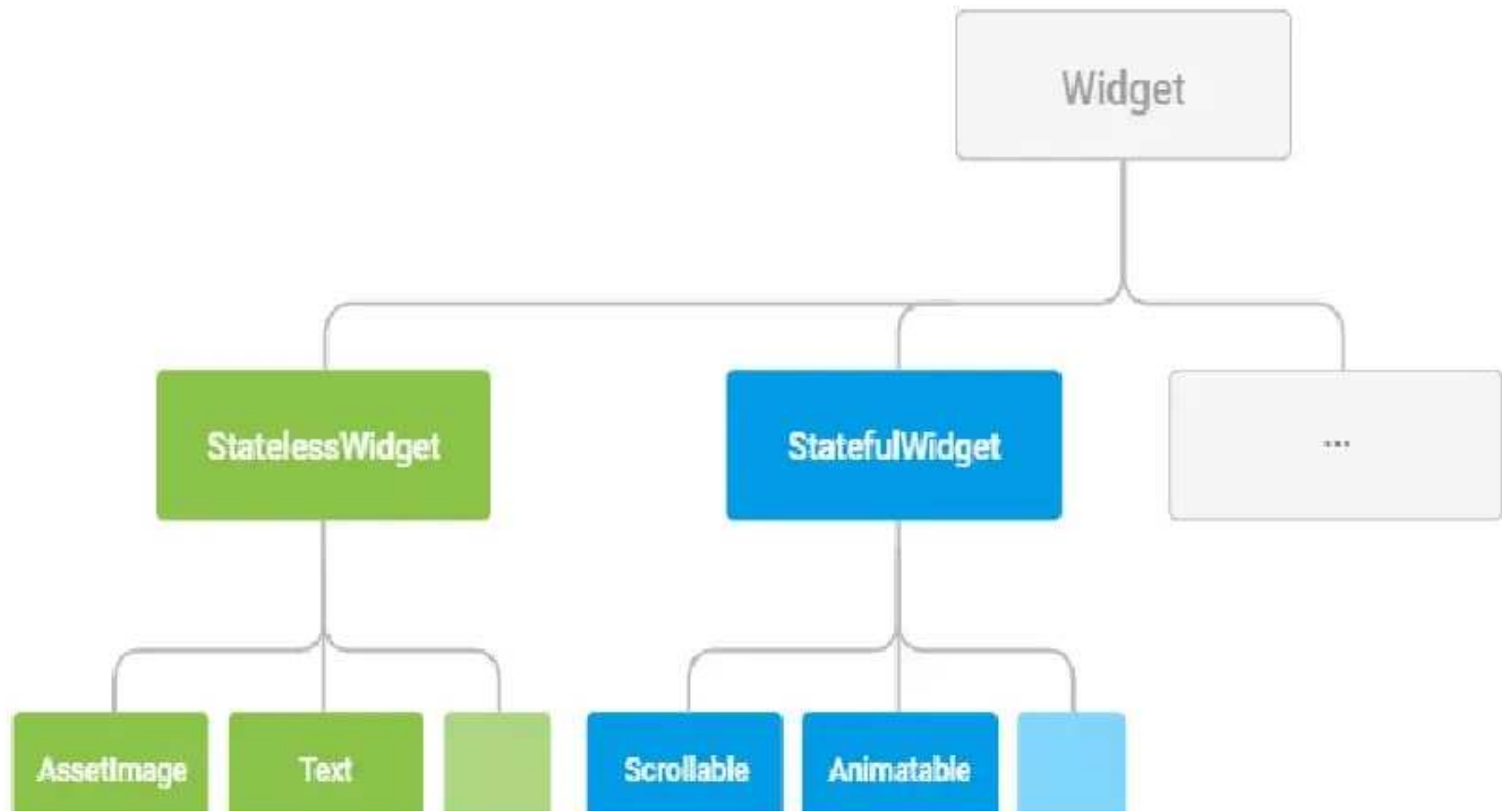
dispose

createState

sample\_page.dart

```
1      import 'package:flutter/material.dart';
2
3      class SamplePage extends StatefulWidget{
4          @override
5          State<StatefulWidget> createState() {
6              return new SamplePageState();
7          }
8
9      }
10
11     class SamplePageState extends State<SamplePage>{
12         @override
13         Widget build(BuildContext context) {
14             // TODO: Implement build
15         }
16
```

StatefulWidget



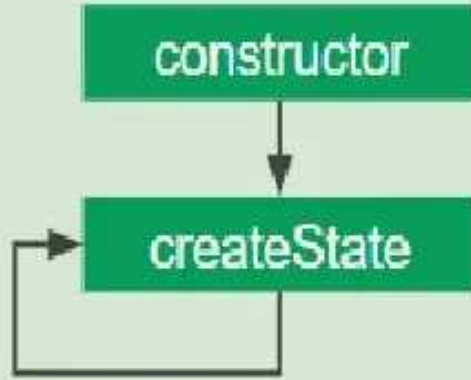


# State Management

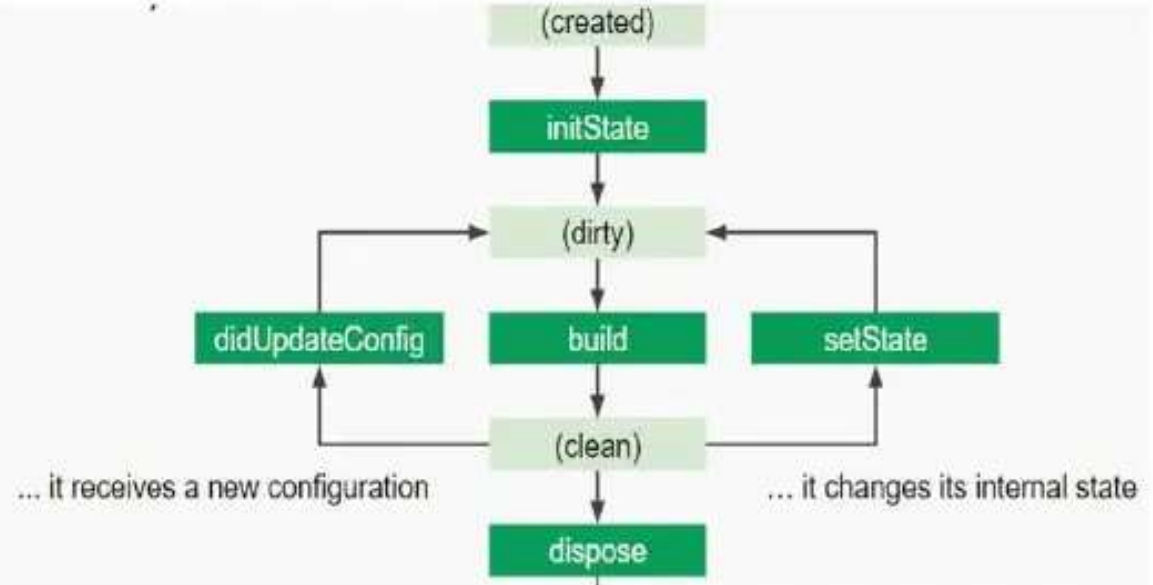
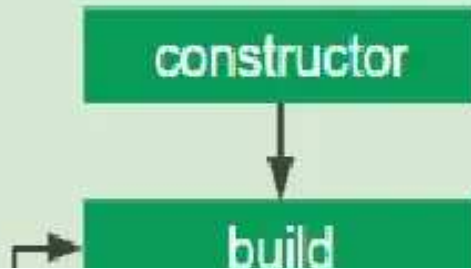


**State = Data**


## StatefulWidget



## StatelessWidget



Ciclo de Vida de Flutter



# Estado efímero VS Estado de aplicación

---



## Estado Efímero

El estado efímero (a veces llamado estado de IU o estado local ) es el estado que puede contener perfectamente en un solo widget.

- página actual en un PageView
- progreso actual de una animación compleja
- pestaña seleccionada actual en un BottomNavigationBar,
- Calculadora

InheritedWidget & InheritedModel

StatefulWidget

MobX

Provider & ScopedModel

## Estado de la Aplicación

El estado que no es efímero, que desea compartir en muchas partes de su aplicación y que desea mantener entre sesiones de usuario, es lo que llamamos estado de aplicación (a veces también llamado estado compartido).

- Preferencias del usuario
- Información de inicio de sesión
- Notificaciones en una aplicación de red social
- El carrito de compras en una aplicación de comercio electrónico
- Estado leído / no leído de artículos en una aplicación de noticias

# Data (State )Management

Mobx

Scoped\_model

Inherited Widget

Provider

Redux

Fish\_Redux

Stateful Widget

Flux

Bloc



**"La regla de oro es: hacer lo que sea menos incómodo".**



**¿Por qué el estado necesita ser administrado?** A medida que su aplicación crece en complejidad, es probable que encuentre errores directamente relacionados con la forma en que los datos fluyen a través de su aplicación a través de la entrada del usuario. Administrar los cambios de estado con cuidado lo ayuda a evitar errores que destruyen el alma que solo suceden en tiempo de ejecución y también pueden ayudar a optimizar el rendimiento.

Analizaremos una variedad de widgets empaquetados en Flutter que se utilizan para administrar el estado.

- ***StatefulWidget***
- ***StatefulBuilder***
- ***StreamBuilder***



Estado Efimero



GDG

```
class MyHomePage2 extends StatelessWidget {
```

```
  int _counter = 0;
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return StatefulBuilder(
```

```
      builder: (ctx, StateSetter setState) =>
```

```
        Scaffold(
```

```
          body: Text('$ _counter'),
```

```
          floatingActionButton: FloatingActionButton(
```

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

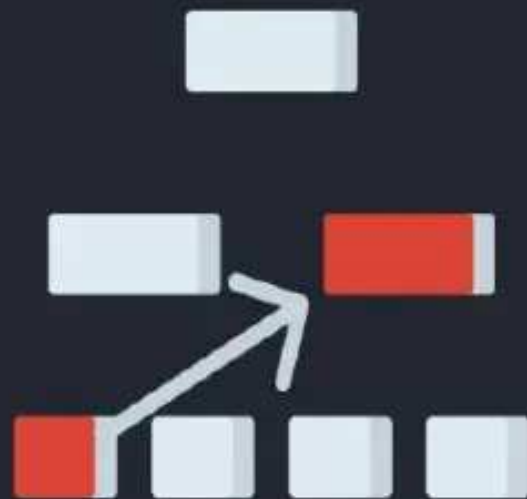
```
        ),
```

# StatefulBuilder





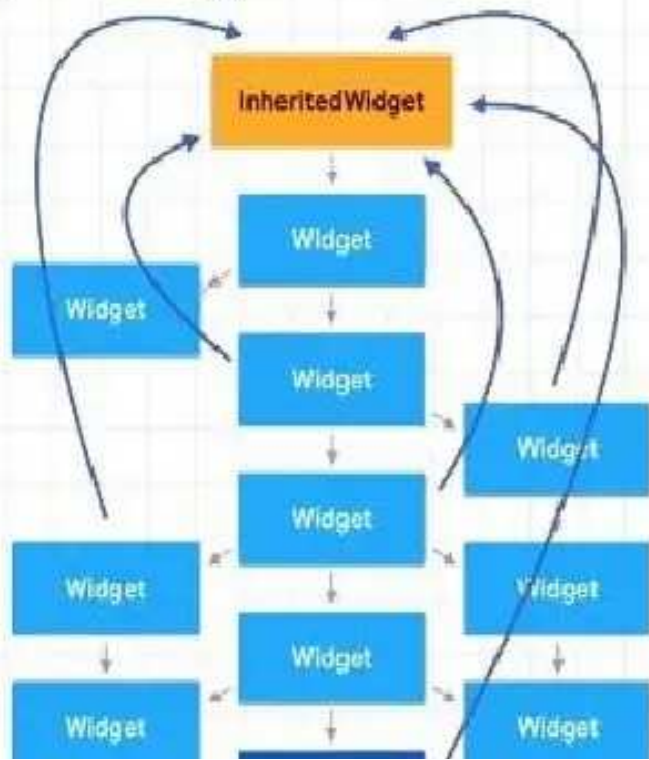
CONSEJO: en Flutter, los datos se mueven de arriba a abajo. Si tiene datos en un widget secundario que desea enviar a un elemento primario, debe usar uno de los métodos de administración de estado global.



# Estado de aplicación



# Data (State) Management





## InheritedWidget

Flutter proporciona un widget InheritedWidget que puede definir proporcionar contexto a cada widget debajo del árbol.

Si bien esto es bueno en teoría, puede ver que se necesita bastante código para obtener un ejemplo básico conectado. Afortunadamente, hay bibliotecas como Bloc, Redux y Scoped Model que resumen esta complejidad.

```
// The InheritedWidget
class InheritedCounter extends InheritedWidget {
  final Map _counter = { 'val': 0 };
  final Widget child;

  InheritedCounter({ this.child }) : super(child: child);

  increment() {
    _counter['val']++;
  }

  get counter => _counter['val'];
}
```

```
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: InheritedCounter( child: MyHomePage3() ),
    );
  }
}
```

```
class MyHomePage3 extends StatelessWidget {

  @override
  Widget build(BuildContext context) {

    return StatefulBuilder(
      builder: (BuildContext context, StateSetter setState) {

        int counter = InheritedCounter.of(context).counter;
        Function increment = InheritedCounter.of(context).increment;
```



```
override  
bool updateShouldNotify(InheritedCounter oldWidget) => true;
```

```
return Scaffold(
```

# Data (State )Management

## Inherited Widget

`Theme.of(context).textTheme.headline`

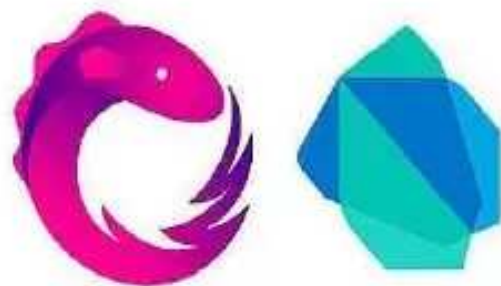
`MediaQuery.of(context).size.width`

`Provider.of<DbProvider>(context).getSomeData()`

`Navigator.of(context).pushNamed(routeName)`

`Scaffold.of(context).showSnackBar()`





# RxDart

**Using** Subject  
(Publish, Behavior, Replay)



# StreamBuilder + RxDart BehaviorSubject

```
// StreamBuilder Widget
class MyHomePage4 extends StatelessWidget {

  @override
  Widget build(BuildContext context) {

    return Scaffold(
      body: StreamBuilder(
        stream: counterService.stream$,
        builder: (BuildContext context, AsyncSnapshot snap) {
          return Text('${snap.data}');
        }
      )
    );
  }
}
```

```
import 'package:get_it/get_it.dart';

GetIt getIt = new GetIt();

void main() {
  getIt.registerSingleton<Counter>(Counter());
  runApp(MyApp());
}

class MyHomePage4 extends StatelessWidget {
  final counterService = getIt.get<Counter>();

  // ...
}
```

```
import 'package:rxdart/rxdart.dart';

// Global Variable
Counter counterService = Counter();

// Data Model
class Counter {

  BehaviorSubject _counter = BehaviorSubject.seeded(0);

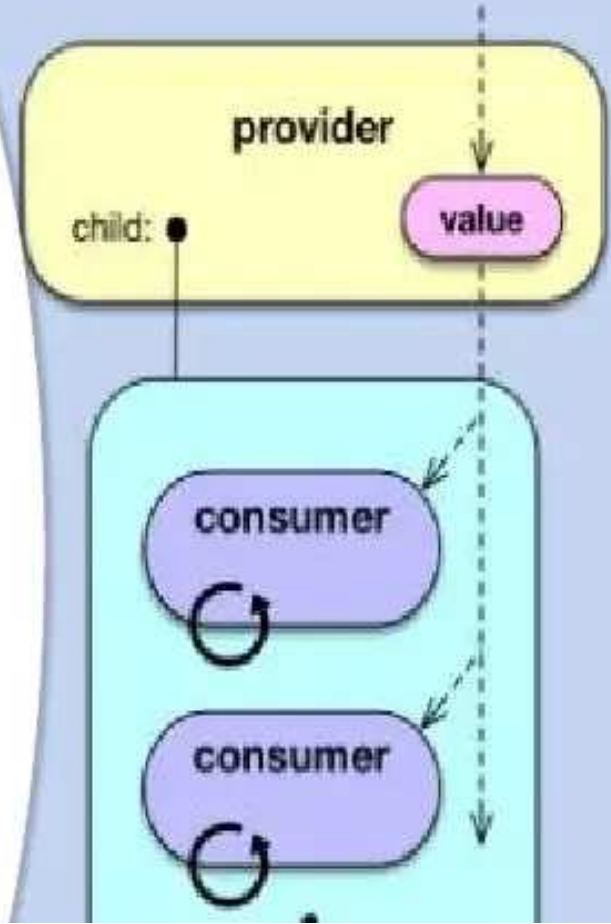
  Observable get stream$ => _counter.stream;
```

),

```
observeValue get { return ... }  
int get current => _counter.value;
```



# Provider

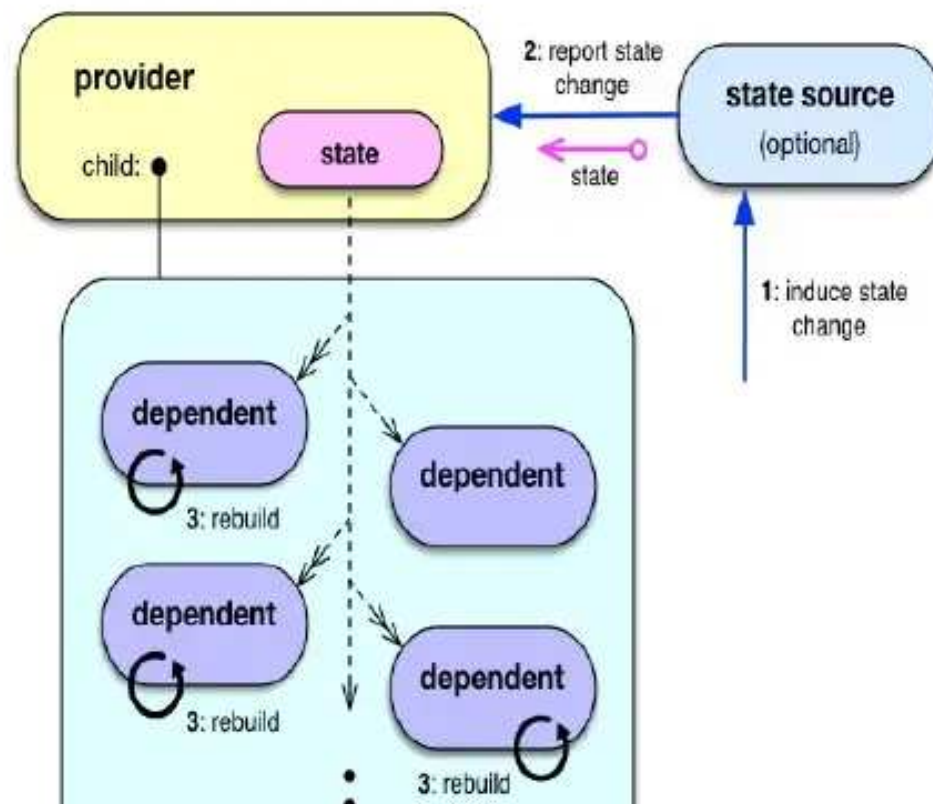


# PROVIDER



**Provider** es una **biblioteca de terceros**. Es un azúcar sintáctico de InheritedWidget. Hay 3 clases importantes para usar esta biblioteca:

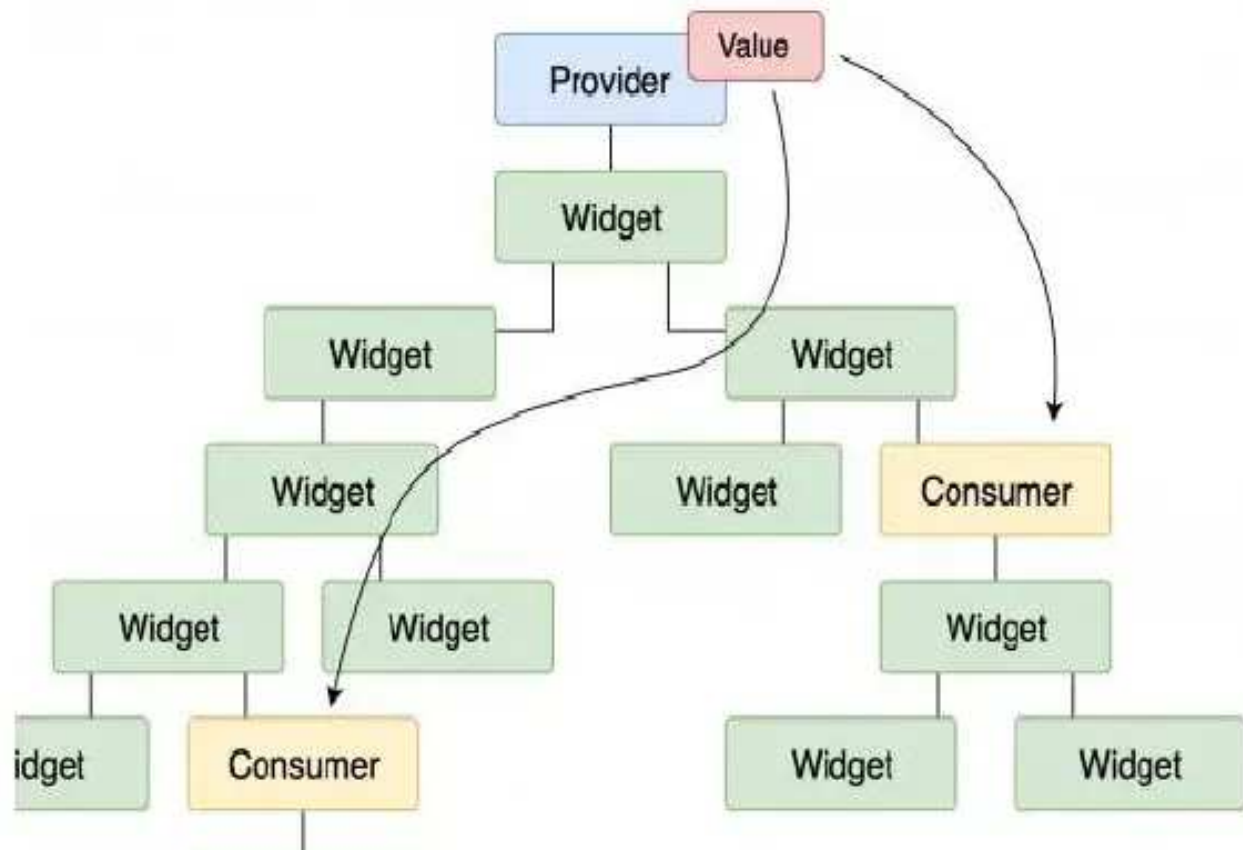
1. **ChangeNotifier** clase integrada
2. **ChangeNotifierProvider**
3. **Consumer**



State

Model

!



Provider



## Widget

```
class ValueNotifierModel extends ValueNotifier {  
  String stringThatChanges = 'testing';  
  
  ValueNotifierModel(value) : super(value);  
  
  void changeTheString(String input) {  
    stringThatChanges = input;  
  
    notifyListeners();  
  }  
}
```

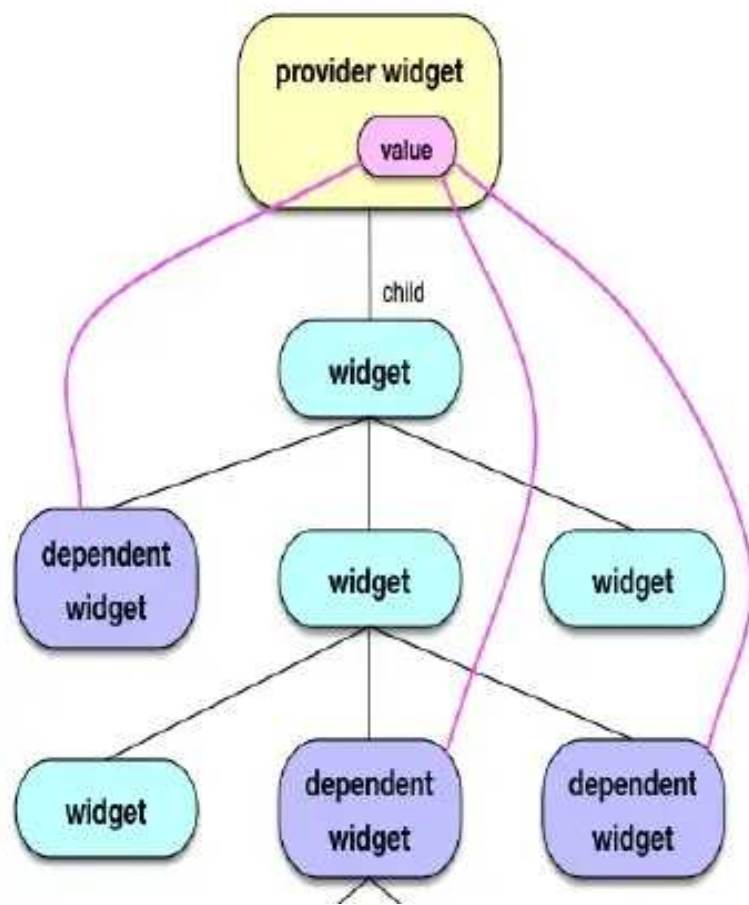
```
class ChangeNotifierModel extends ChangeNotifier {  
  String stringThatChanges = 'testing';  
  
  void changeTheString(String input) {  
    stringThatChanges = input;  
  }  
}
```

```
return Provider<ChangeNotifierModel>(  
  builder: (BuildContext context) => ChangeNotifierModel(),  
  child: MultiProvider(  
    providers: [  
      ChangeNotifierProvider<ChangeNotifierModel>(  
        builder: (BuildContext context) => ChangeNotifierModel(),  
      ),  
      ValueListenableProvider<ValueNotifierModel>.value(  
        value: _valueNotifierModelInstance.value,  
      ),  
      /// Others include:  
      /// FutureProvider  
      /// StreamProvider  
      /// ListenableProvider  
      /// InheritedProvider  
    ],  
  ),  
)
```

/// We need another build method for Context  
child: UseTheProvidersInThisWidget(),

```
notifyListeners();  
}
```

```
);
```



```
class UseTheProvidersInThisWidget extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    ...  
    /// Approach 3 of 3) Use a Consumer creates its own Context and widget tree.  
    Consumer<ChangeNotifierModel>{  
      builder: (context, madeUpNameForObjectInstance, _) {  
        return Text(  
          'The value of MyObject.stringThatChanges is  
          ${madeUpNameForObjectInstance.stringThatChanges}';  
        ),  
      },  
    )  
  }  
}
```

```
class UseTheProvidersInThisWidget extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    /// Approach 1 of 3) Make an instance of MyModelObject above the return and use it below  
    final providerOfAccessedObject = Provider.of<ChangeNotifierModel>(context);  
    ...  
  }  
}
```

# Patrón de Diseño

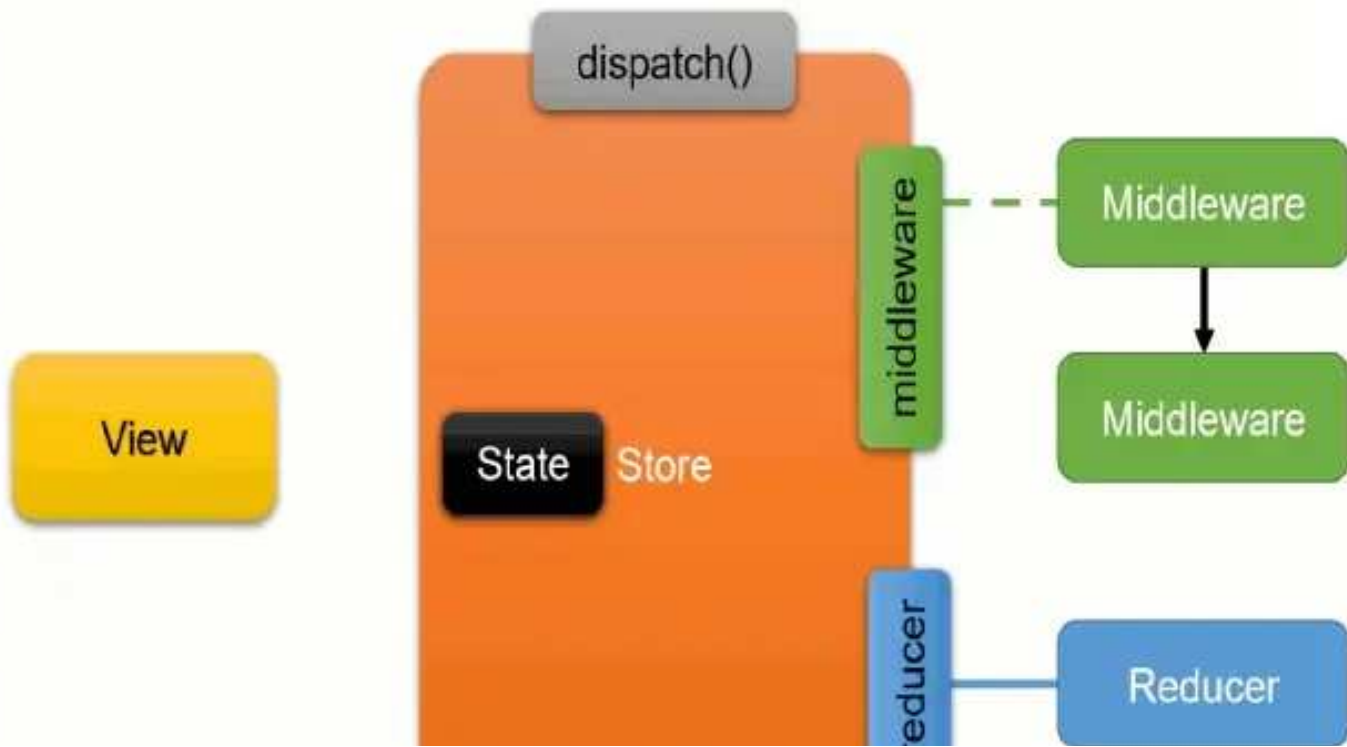
```
return container(...  
...Text(  
    /// This is where we're reusing the above variable
```

# Lista de enfoques de gestión de estado

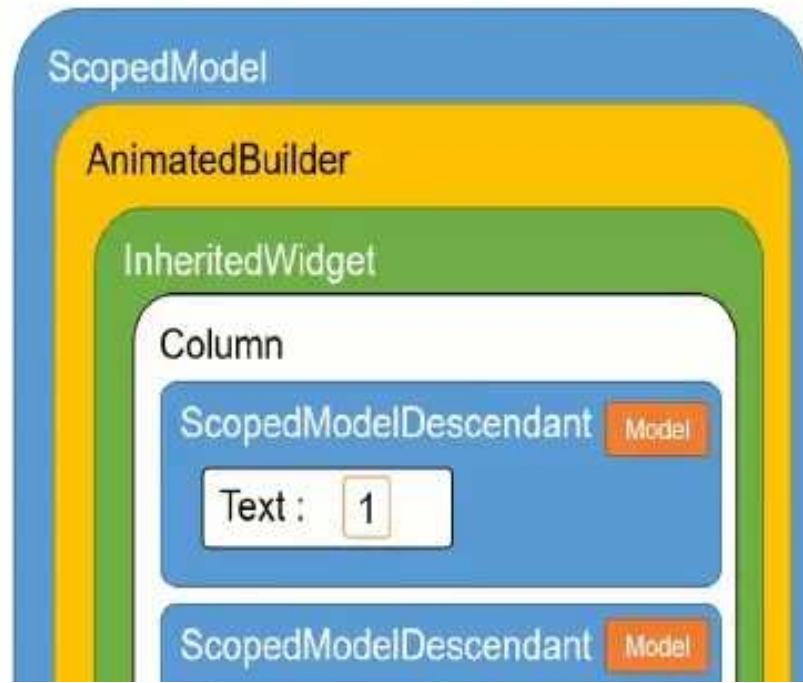
- InheritedWidget & InheritedModel
- Provider & Scoped Model
- Redux
- BLoC / Rx
- MobX



# REDUX



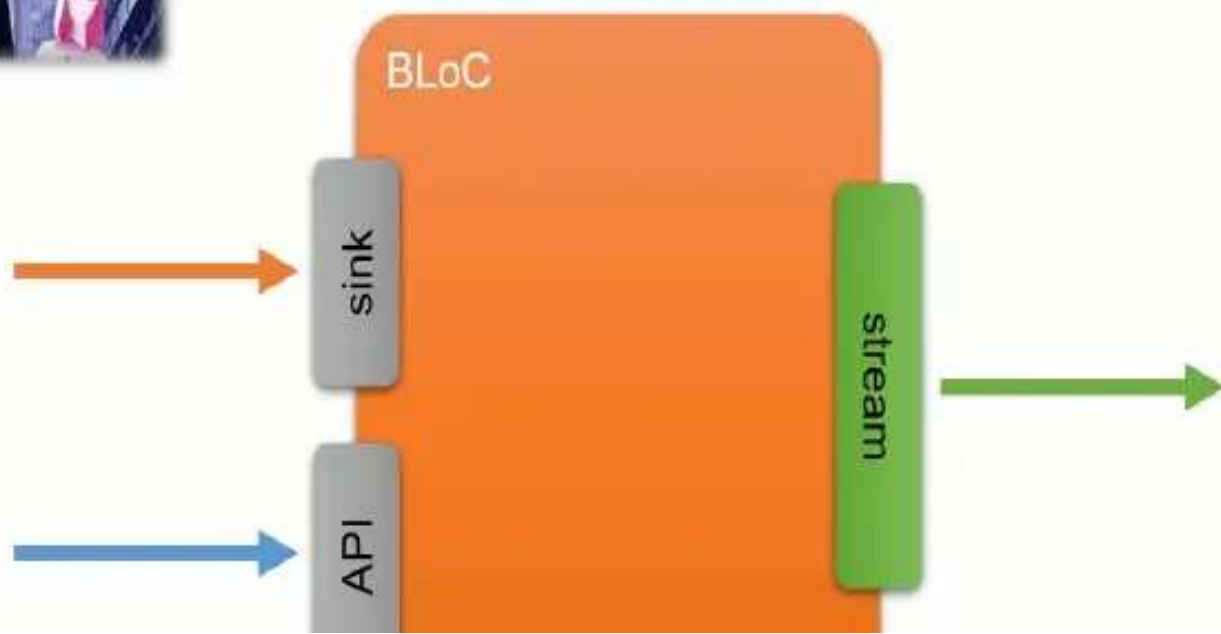
# SCOPED MODEL







# BLOC





# State Management



**State = Data**



# Flutter Dominicana



@flutterdominicana



flutterdominicana



@flutterDominic



flutterdominicana



<https://www.eventbrite.com/o/28290902701/>



flutterdominicana

LinkedIn

<https://www.linkedin.com/in/flutter-dominicana-7a533a198/>



GDG  
Santo Domingo



- Mobile
- Web
- Desktop
- Embedded





Referencia.html