

# Métodos

Clases y Patrones de diseño



Yosafat Coronel  
GDSC ESCOM IPN  
GitHub: YosafatM



Developer Student Clubs  
ESCOM - IPN

# Itinerario

- Heredados
- De instancia
- Estáticos
- Ejemplos
- Sobre el SDK con Android Studio



# Heredados



# Heredado

Hay veces en las que una clase que hacemos **funciona como base para muchas otras clases**. Por ejemplo, Widget, en Flutter, para todo lo visual:

```
class DataCard extends StatelessWidget {  
  final IconData icon;  
  final String text;  
  
  const DataCard({  
    @required this.icon,  
    @required this.text,  
  });
```

*En este caso, para indicarle a Dart que queremos heredar, usamos la palabra reservada **extends***

# Heredado

Cuando heredamos, le indicamos a Dart que esa **clase será un hijo (subtipo)** de la clase que heredamos. Flutter maneja Widgets, y DataCard, lo es:

```
class DataCard extends StatelessWidget {  
  final IconData icon;  
  final String text;  
  
  const DataCard({  
    @required this.icon,  
    @required this.text,  
  });
```

*Con esto, podemos usar DataCard como un Widget, sin embargo, así como nos otorga beneficios nos pide obligaciones*

# Heredado

Se le llama herencia, porque las propiedades y métodos del que hereda. Además, tiene permitido sobreescribir los métodos marcados con `@protected`:

*Con esto, podemos usar DataCard como un Widget, sin embargo, así como nos otorga beneficios nos pide obligaciones*

```
/// * [StatelessWidget], which contains the discussion on performance considerations.  
@protected  
Widget build(BuildContext context);
```

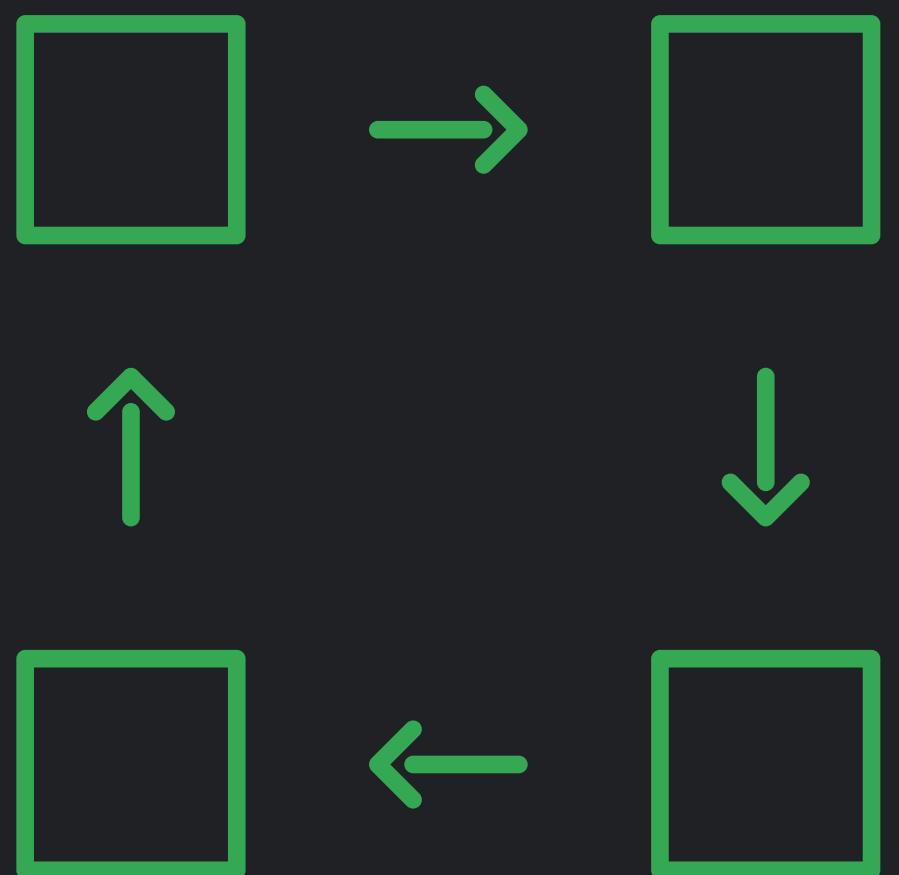
# Heredado

En este caso, es abstracto, por lo que DEBE implementarlo la clase que hereda. Con esto decimos: 'No sé cómo lo hagan, pero sé que lo hacen'

```
@override  
Widget build(BuildContext context) {  
  return Card(  
    color: Colors.white,  
    margin: EdgeInsets.symmetric(vertical: 10.0, horizontal: 30.0),  
    child: Padding(...), // Padding  
); // Card  
}
```

*Flutter, no sabe cómo son los Widgets,  
pero sabe que todos tienen el método  
build para pintarlos*

# Instancia



# Insta\_ncia

```
@override  
Widget build(BuildContext context) {  
  return Card(  
    color: Colors.white,  
    margin: EdgeInsets.symmetric(vertical: 10.0, horizontal: 30.0),  
    child: Padding(...), // Padding  
); // Card  
}
```

Un método es de instancia, cuando solo se puede acceder por una instancia. Una instancia es un objeto creado de la clase

*build no solo es heredado, sino que se necesita de un objeto DataCard (en general un Widget) para llamarlo*

# Insta\_ncia

```
@override  
Widget build(BuildContext context) {  
  return Card(  
    color: Colors.white,  
    margin: EdgeInsets.symmetric(vertical: 10.0, horizontal: 30.0),  
    child: Padding(...), // Padding  
); // Card  
}
```

¿Por qué se necesita de una instancia?, porque son específicos para cada objeto. Por ejemplo, dispose (método que indica que un Widget ya no tiene estado).

*build no solo es heredado, sino que se necesita de un objeto DataCard (en general un Widget) para llamarlo*

# Instancia

```
void _redirectToHome() {  
    Get.back();  
    showNotifier(  
        type: notification_type.warning,  
        message: 'Lo sentimos la categoría no se encuentra disponible',  
    );  
}
```

Un método de instancia puede ser privado si a su identificador se le pone guión bajo al comienzo. De esta forma, solo podrá ser accedido dentro del marco del archivo:

*No tendría caso poder llamar este método desde fuera del archivo, la clase por dentro debe ser la única que acceda a él*

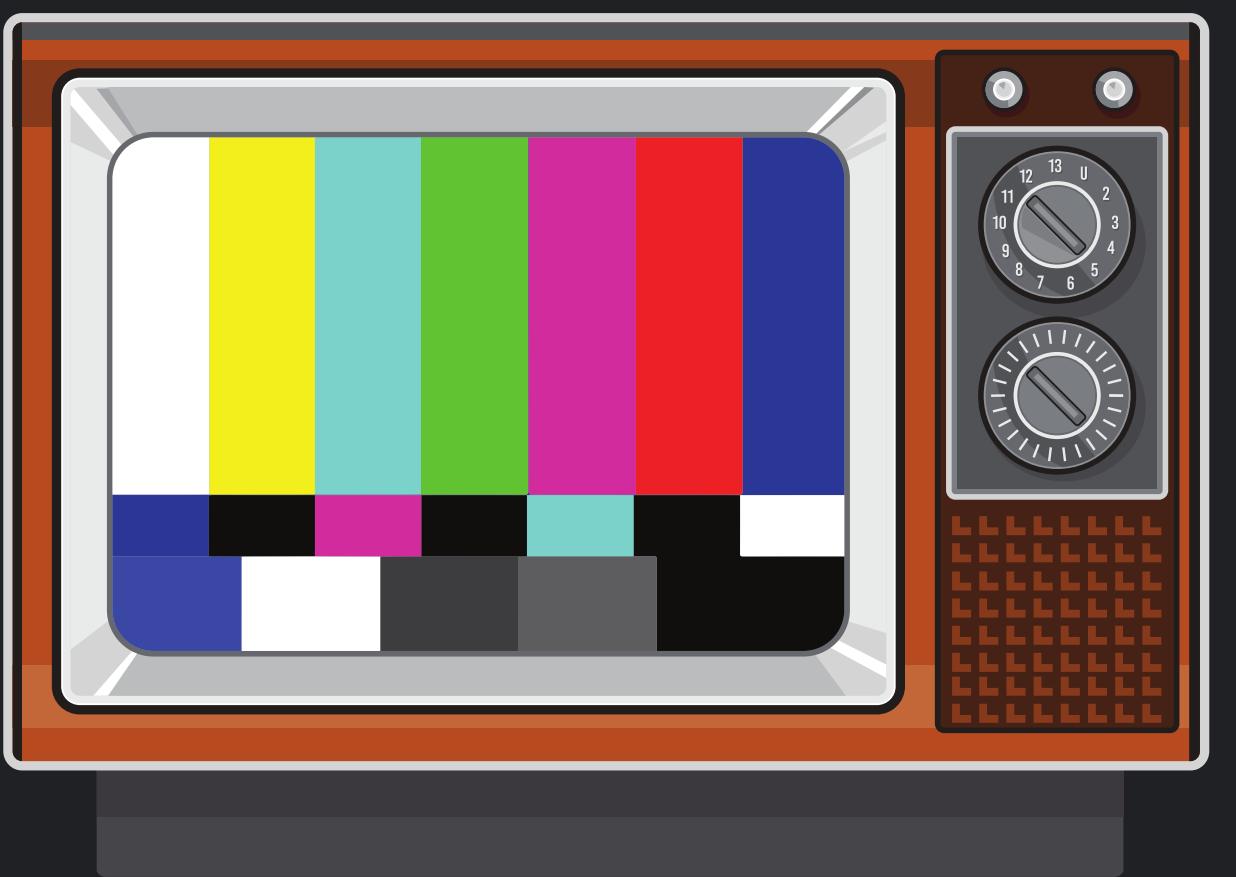
# Instancia

Esto resulta útil cuando usamos una porción de código en varios lados, pero queremos que sea de forma interna en la clase. Así queda más organizado:

```
double _calculateTotalPrice() {  
    totalPrice = 0;  
  
    for (var cartItem in cartItems) {  
        totalPrice += cartItem.totalPrice;  
    }  
  
    update(['totalPrice']);  
  
    return totalPrice;  
}
```

*Calcular el precio total de un carrito no debería poder ser accedido por fuera de esta clase. Sin embargo, lo usamos mucho porque puede cambiar de muchas forma el carrito*

# Estáticos



# Estáticos

Cuando no se requiere una instancia para acceder al método, más que el nombre de la clase, entonces ese método es estático:

*No tendría sentido tener que hacer un objeto para saber qué versión de Dart estamos corriendo. Solo con la clase*

```
/// The version of the current Dart runtime.  
///  
/// The value is a [semantic versioning](http://semver.org)  
/// string representing the version of the current Dart runtime,  
/// possibly followed by whitespace and other version and  
/// build details.  
  
static String get version => _version;
```

*Platform.version, basta*

# Estáticos

En este caso, la palabra reservada get nos dice que asocia la lectura de `_version` de forma pública (más no la escritura). Podemos leer la versión pero no cambiarla.

```
/// The version of the current Dart runtime.  
///  
/// The value is a [semantic versioning](http://semver.org)  
/// string representing the version of the current Dart runtime,  
/// possibly followed by whitespace and other version and  
/// build details.  
  
static String get version => _version;
```

*De esta forma aseguramos solo la lectura, y la escritura solo por dentro de la clase*

**¡Muchísimas  
gracias!**



Developer Student Clubs  
ESCOM - IPN