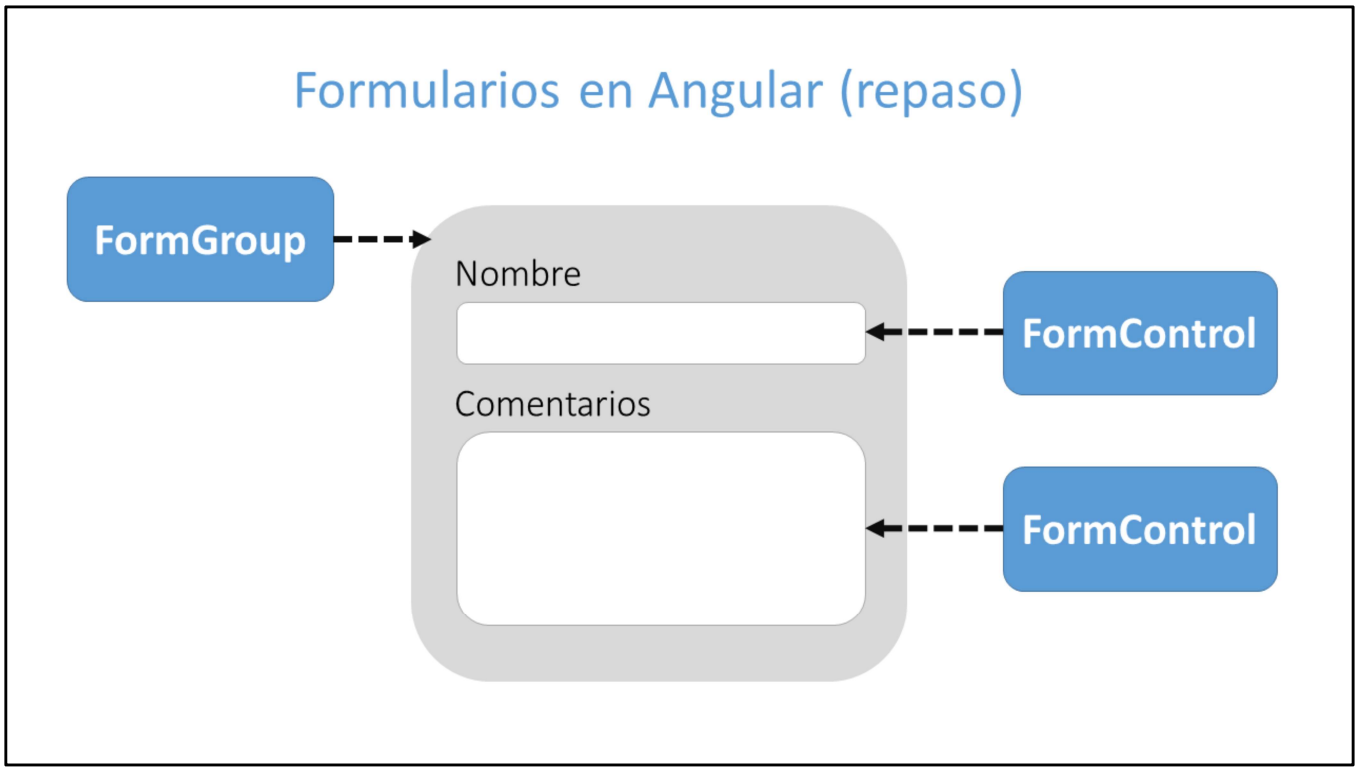
Tema 9 – Formularios reactivos

- Tipos de formularios en Angular (repaso)
- Jerarquía de tipos de objetos para formularios
- Creación de un formulario reactivo
- Grupos de controles
- Inicialización y recogida de valores
- Validaciones básicas y personalizadas

En este tema vamos a ver los formularios reactivos, la otra forma que tiene Angular de gestionar los formularios, además de la que ya hemos visto anteriormente.



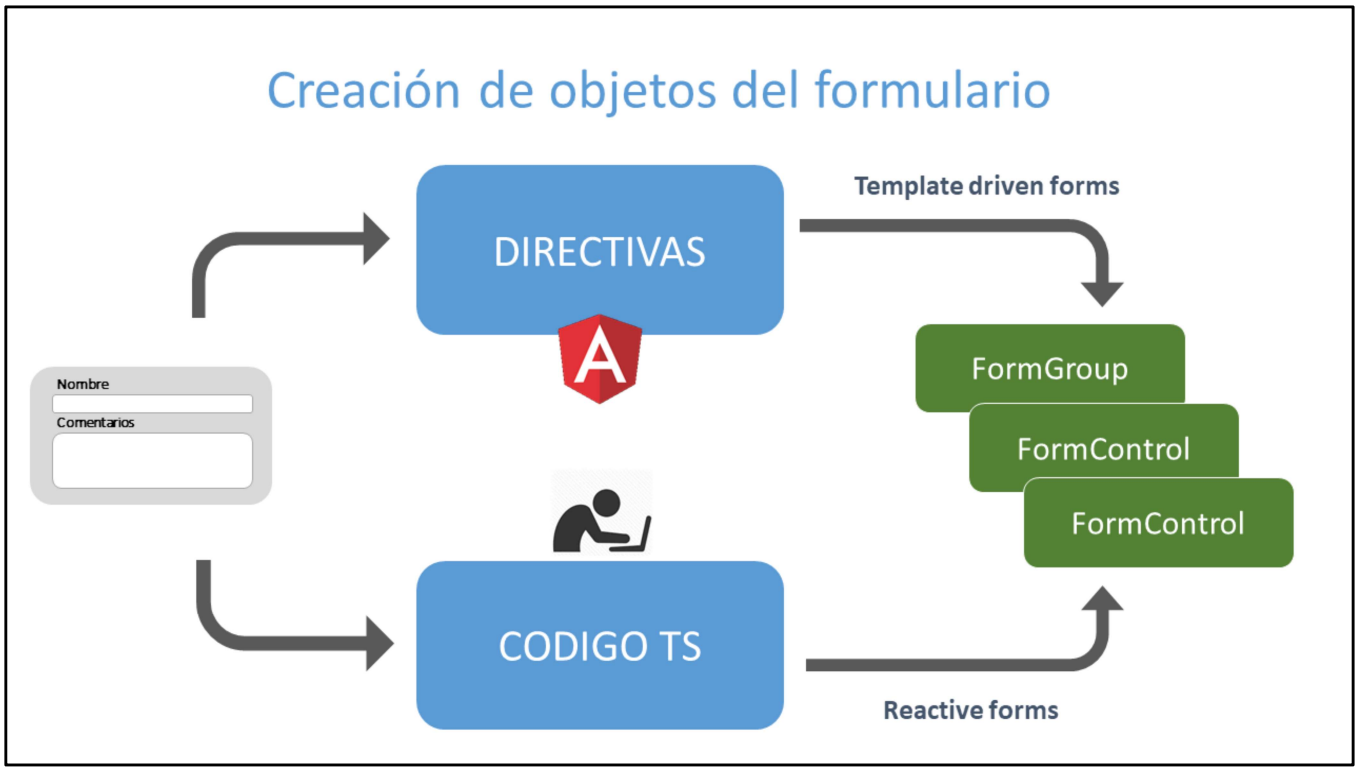
Repaso del tema 6 (Formularios dirigidos por plantilla)

Los formularios en Angular no dejan de ser formularios HTML.

La diferencia es que tanto el propio formulario como los elementos del mismo, deben asociarse a objetos a los que accederemos en nuestro componente, para controlar el formulario y acceder a la información del mismo.

- Un formulario HTML se asocia a un objeto de tipo FormGroup;
- Cada control del formulario se asocia a un objeto de tipo FormControl.

Estos objetos, a través de sus métodos y propiedades, nos permiten acceder y controlar el formulario en el componente.



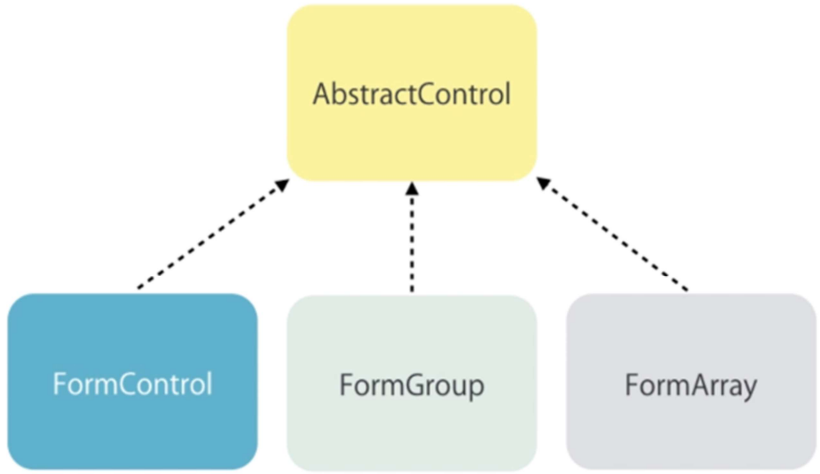
Como vimos en el tema 6, Angular tiene 2 formas de crear los objetos FormGroup y FormControl:

- Mediante directivas en la plantilla HTML. Se denominan formularios dirigidos por plantilla, o "template-driven forms". Están recomendados para formularios simples con validaciones básicas.

- Mediante código TS en el componente. Se denominan Formularios Reactivos, o "reactive forms" . Están recomendados para formularios y validaciones más complejos.

En este tema vamos a ver la segunda forma de crear los objetos FormGroup y FormControl.

Jerarquía de tipos para formularios



En este diagrama vemos la jerarquía de los tipos generales de objetos que forman parte de los formularios en Angular.

- FormControl, como sabemos, es el tipo base para todos los controles de formulario: cuandros de texto, listas desplegables, checkbox, etc.

- FormGroup, que también conocemos, sirve para representar al propio formulario, y agrupa los controles incluidos en el mismo. Además también puede usarse para crear estructuras complejas de grupos de controles anidados.

- FormArray, que es nuevo, sirve, como su nombre indica, para gestionar grupos de controles como si fueran parte de un array. Se usa normalmente en formularios con listas de campos a editar.
    - No lo vimos en el tema de formularios dirigidos por plantilla, porque solo se pueden usar con formularios reactivos.

El hecho de que todos los tipos deriven de AbstractControl, nos indica que todos comparten propiedades comunes, como por ejemplo si es válido o no, lo que simplifica su tratamiento.

También permite flexibilizar la estructura de un formulario, ya que un formulario o grupo

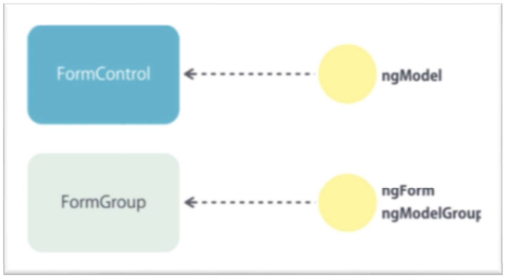
puede contener cualquier tipo de AbstractControl, bien sean controles, arrays u otros grupos anidados.

Creación de formularios dirigidos por plantilla

```
<form ngForm >
  <input
      ngModel
      name="nombre"
      type="text"
  >
</form>
```

La directiva **ngForm** asocia al form con un FormGroup. Es Opcional.

La directiva **ngModel** asocia al control con un FormControl



Cuando vimos los formularios dirigidos por plantilla, vimos que únicamente debíamos actuar sobre la plantilla HTML del componente.

Mediante las directivas ngForm, ngModel y ngModelGroup de Angular indicábamos en HTML cómo debía crear los objetos FormGroup y FormControl.

Creación de un formulario reactivo

1. Importar ***ReactiveFormsModule*** en AppModule (*o en el módulo del formulario*)

TS

```
import { ReactiveFormsModule} from '@angular/forms'
```

2. Crear el formulario y los controles en el componente

TS

```
export class FormRegistroComponent {
 formulario = new FormGroup({
   usuario: new FormControl(),
   clave: new FormControl()
 });
```

3. Crear el formulario en la plantilla HTML y asociar los objetos con las directivas

HTML

```
<form [formGroup]="formulario"> // OJO, USA PROPERTY BINDING
   <div>
       <label for="usuario">Usuario</label>
       <input
           formControlName = "usuario" // AQUI NO USA PROPERTY BINDING
           id="usuario"
           type="text">
        ...
        ...
   </div>
```

Un componente que contenga un formulario reactivo, por el contrario, primero debe crear los objetos asociados al formulario en el código del componente, y luego asociar cada objeto en la plantilla HTML a su control o parte del formulario.

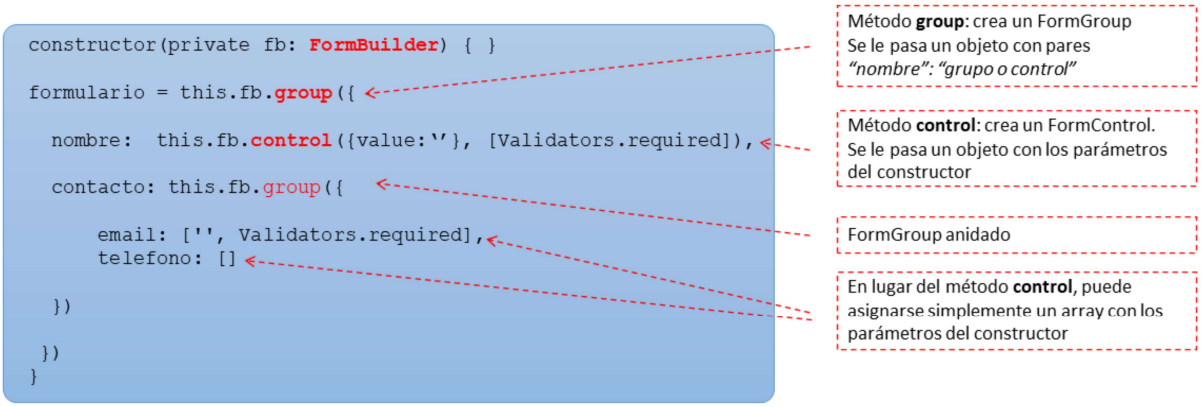
- Vemos que lo primero que cambia es que debemos importar el módulo ***ReactiveFormsModule***, en lugar de FormsModule.
- El formulario lo creamos como de tipo FormGroup.
- El constructor de FormGroup, acepta un objeto, en el que cada atributo es un objeto derivado de AbstractControl, es decir, puede ser un FormControl, un FormGroup o un FormArray.
- El nombre del atributo al que se asigna cada control, es el nombre que se utiliza para registrar el control en el formulario, y es el que debemos usar en el futuro para referirnos a él.
- En la plantilla, usamos:
    - la directiva “formGroup” para asociar la etiqueta form al objeto FormGroup principal
    - la directiva “formControlName” para asociar cada control al objeto que lo

representa en el código.

- Tener en cuenta que la directiva “formGroup” se asigna con binding de propiedades (con corchetes), mientras que la directiva “formControlName” se asigna directamente, sin binding.

Creación del formulario con FormBuilder

- ***FormBuilder*** es una clase "factoría" que crea formularios y controles con una sintaxis más limpia que el uso de "new"
- Se inyecta en el constructor del componente.



En lugar de usar la sintaxis normal para construir un form, mediante "new", podemos usar la clase FormBuilder dentro del constructor del componente

Creación del formulario con FormBuilder

- Comparativa de código con o sin FormBuilder

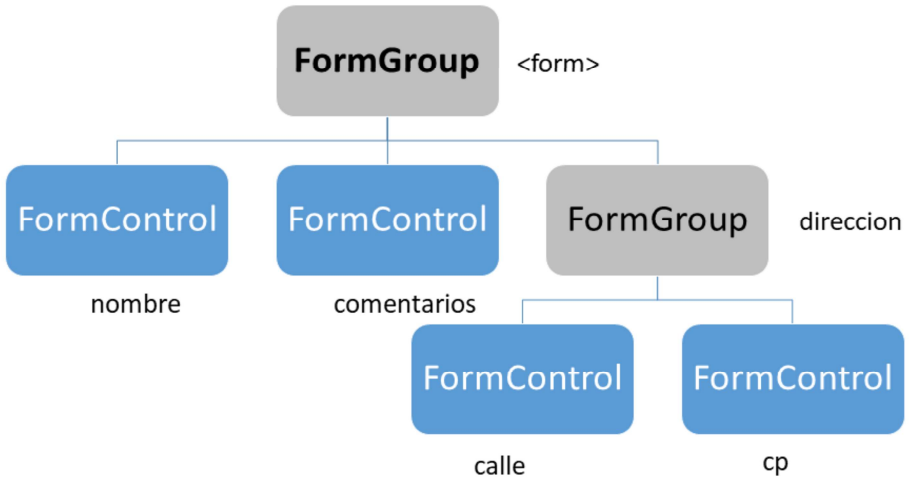
```
formulario = new FormGroup({
   usuario: new FormControl('', [Validators.minLength(3)]),
   clave: new FormControl()
});
```

Sin FormBuilder

```
formulario = this.fb.group({
   usuario: ['', [Validators.minLength(3)]],
   clave: ['']
 })
```

Con FormBuilder

Grupos de controles



De la misma forma que en los formularios dirigidos por plantilla, podemos agrupar controles en un formulario reactivo.

Grupos de controles anidados

- Componente

```
formulario = new FormGroup({
   nombre: new FormControl(),
   direccion: new FormGroup({
      calle: new FormControl(),
      cp: new FormControl()
   }),
   comentarios: new FormControl()
});
```

- Plantilla HTML

```
<form [formGroup]="formulario"> //

 <input formControlName="nombre">

   <div formGroupName="direccion">
      <input formControlName="calle">
      <input formControlName="cp">
   </div>

 <input formControlName="comentarios">
```

Para crear grupos de controles, debemos crear en el componente objetos de tipo FormGroup, y asociarlos a un bloque en la plantilla HTML mediante la directiva **formGroupName**

Inicialización y recogida de valores

- Para inicializar un control, se pasa el valor como parámetro del constructor:

TS

```
formulario = new FormGroup({
   usuario: new FormControl('Miguel Angel')
})
```

- Para acceder al objeto desde el componente o desde la plantilla, podemos usar el método "get()" del formulario, o su atributo "value"

TS

```
let nombreUsuario = formulario.get("usuario").value; // El get devuelve el propio objeto FormControl
let clave = formulario.value.clave;   // "value" devuelve un objeto solo con los valores
```

HTML

```
<form [formGroup]="formulario">

    <input formControlName = "usuario" (keyup.enter)="cargarUsuario(formulario.value.usuario)">
    ...
    <input formControlName = "clave" (keyup.enter)="cargarClave(formulario.get('clave'))">
```

Validaciones básicas

- No se usan validaciones estándar HTML 5
- Validadores predefinidos: métodos estáticos de la clase **Validators** (https://angular.io/api/forms/Validators):
  - min, max, required...
- Se pasan en un array como 2º parámetro del constructor del control:

TS

```
import { Validators } from '@angular/forms'
...
formulario = new FormGroup({
   usuario: new FormControl('Miguel Angel', [Validators.required, Validators.minLength(3)])
})
```

Con reactive forms, no se usan las validaciones estandar HTML 5, como hacíamos en los formularios dirigidos por plantilla.

Es necesario asignar los validadores expresamente al FormControl cuando se crea en el componente, mediante los miembros estáticos de la clase ***Validators***

Validaciones personalizadas

- Ventaja de formularios reactivos: permiten usar validadores personalizados
- Qué es un Validador:
  - Es una **función** que implementa la interfaz ***ValidatorFn***
  - Recibe el control a validar (*AbstractControl*)
  - Debe devolver un objeto ***ValidationErrors*** (control inválido), o *null* (control válido)
    - El objeto de error contiene pares *nombreError : { información del error }*

Validaciones personalizadas

- Crear un validador para comprobar que no hay espacios en el valor del control:

TS

```ts
import { AbstractControl, ValidationErrors } from '@angular/forms';

export class MisValidadores {

    static sinEspacios(control: AbstractControl) : ValidationErrors | null {
        if ((control.value as string).indexOf(" ") >= 0) { // control inválido
            return { sinEspacios: true } // formato: nombreError : valor simple u objeto con más datos
        }
        else // control válido
            return null;
    }
}
```

- Agregar el validador personalizado a un control:

TS

```ts
formulario = new FormGroup({
   usuario: new FormControl('Miguel Angel', [MisValidadores.sinEspacios])
})
```

- Mostrar un mensaje de error en el formulario:

HTML

```html
<mat-error>
          <div *ngIf="formulario.get('usuario').errors?.sinEspacios">
              El nombre no debe contener espacios
          </div>
</mat-error>
```

Un validador puede ser una función cualquiera, mientras siga la interfaz de ValidatorFn, es decir:

- Debe aceptar un parámetro de tipo *AbstractControl*
- Debe devolver un objeto de tipo *ValidationErrors*, o bien el valor *null*

No obstante, por seguir el mismo sistema que los validadores predefinidos en Angular, se suele crear una clase para los validadores personalizados, y cada uno de ellos se crean como funciones estáticas de la clase.

Cuando el validador comprueba que el control pasado como parámetro no pasa la validación, debe devolver un objeto con un atributo que indique el nombre del error, y un valor que puede ser simplemente un booleano “true”, o bien un objeto más complejo que proporcione más información al consumidor del validador.

Si el control pasa la validación correctamente, el validador debe devolver *null.*

En la plantilla HTML, podemos acceder a los errores del control a través de su propiedad “errors”.