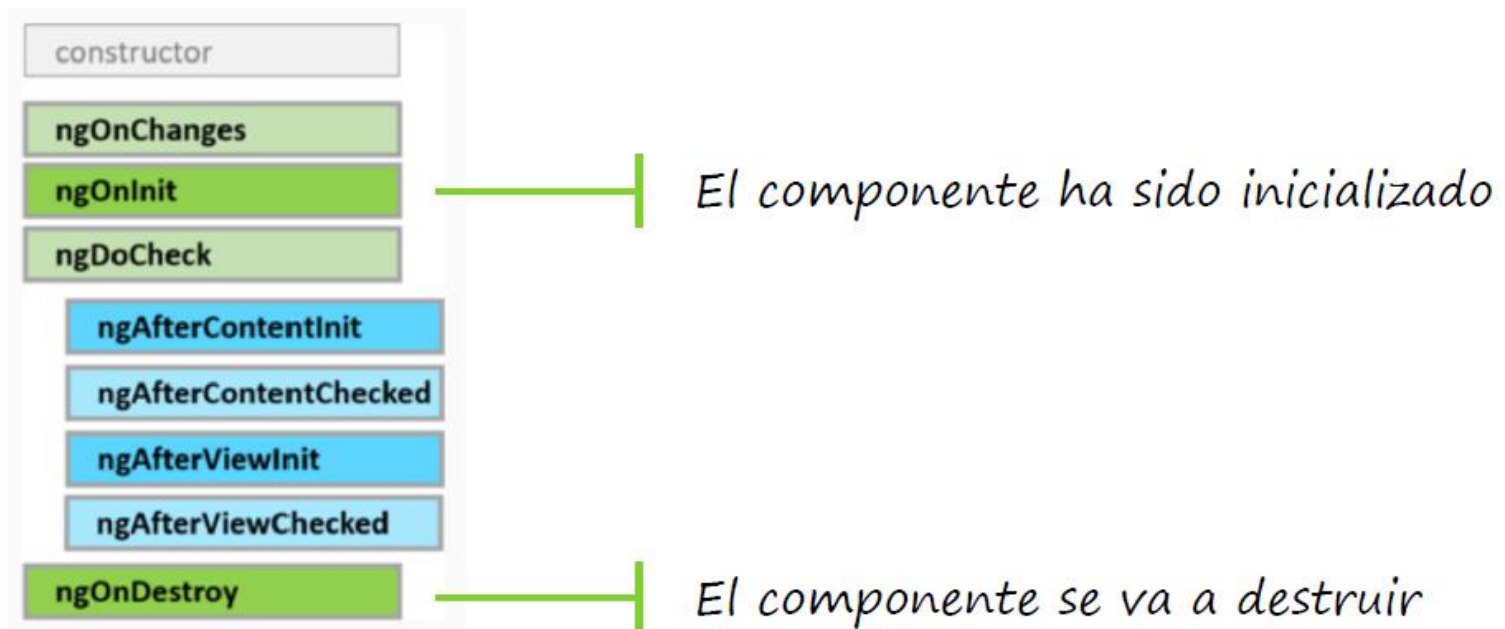


COMPONENTES

»» Ciclo de Vida

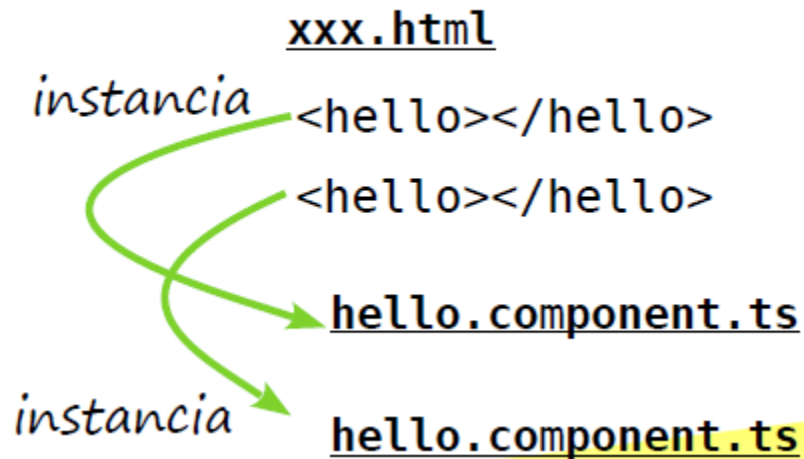
Componentes. Ciclo de Vida

- ▶ Un componente pasa por distintas fases en su vida.
- ▶ Podemos fijar **hooks** o **handlers** sobre esas fases:
 - Registraremos un manejador en el **controlador del componente** para que sea notificado en cada fase.



Componentes. Ciclo de Vida

- ▶ Cuando se crea un componente, Angular **instancia su controlador**.
- ▶ Cada vez que insertamos un tag de un componente en una plantilla (ej: `<app-libro>`) **se crea una instancia nueva de ese componente**
 - Que **tendrá su propio ciclo de vida diferente al resto de las otras instancias**.



¿Qué ocurre cuando se instancia un componente?

1º) Se llama a su **constructor**

2º) Se llama a una serie de **métodos** o **hooks** o **manejadores** (siempre y cuando los hayamos definido en el interior del controlador) y siguiendo el siguiente **orden**:

- **ngOnChanges**

- **ngOnInit**

- **ngDoCheck**

- ...

- **ngOnDestroy**

ngOnInit se llamará cuando nuestro componente haya sido correctamente inicializado, es decir,

- Todas las **ENTRADAS** del componente ya se han establecido correctamente.
- Por tanto, nos permite acceder de forma segura a nuestros datos de entrada.
- Cuando se llama al **CONSTRUCTOR**, las **ENTRADAS** aún **no han sido fijadas**, serían undefined.
- Este método, al igual que el constructor, sólo se llama una vez.

ngOnDestroy se llamará antes de que se destruya el componente:

- Se usa para liberar recursos: *peticiones asíncronas en curso, conexiones,...*

- Los más importantes son: **ngOnInit** y **ngOnDestroy**

Componentes. Ciclo de Vida

- ▶ Para registrar manejadores o hooks para esos puntos del ciclo de vida, Angular define varios **INTERFACES**:
 - Cada interfaz modela un único momento del ciclo de vida del componente
 - Cada interfaz contienen un solo método
- ▶ Por ejemplo, si quiero definir el **manejador ngOnInit**, debemos implementar la **interfaz OnInit**. Para ello:

1º) se importa la interfaz

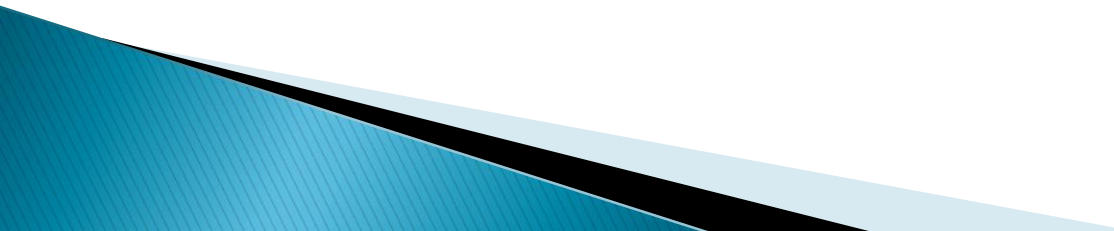
```
import { OnInit } from '@angular/core';
```

2º) se implementa

```
export class HelloWorldComponent implements OnInit {  
  
  constructor() { }  
  
  ngOnInit() {  
  }  
  
}
```

Componentes. Ciclo de Vida

Por **defecto**, al crear un componente ya implementa e importa la interfaz **ngOnInit**.



Constructor vs ngOnInit

- ▶ Ambos sirven para inicializar el componente.
- ▶ **Constructor:**
 - Necesario para “**inyección de dependencias o DI**” (*lo veremos en los servicios*).
 - Debe ser **lo más simple posible**:
 - Sólo *inicializar propiedades de la clase a valores por defecto*.
 - **Evitar**: *lógica compleja, manejo de excepciones, obtener datos de forma local o remota de una BD o API*.
 - **IMPORTANTE**: aquí, **las propiedades de tipo “Input”** del componente **aún no están definidas** (no podremos referenciarlas).

Constructor vs ngOnInit

► ngOnInit:

- Se llama **después del constructor**.
- **Permite acceder a las propiedades de tipo “Input”** puesto que ya estarán definidas aquí.
- Se debe usar para:
 - **Realizar inicializaciones complejas**
Ej: *obtener datos por medio de un servicio* para usarlos dentro del componente.
 - **Definir variables de instancia** *basadas en las propiedades de tipo Input.*

Constructor vs ngOnInit

► Ejemplo de uso:

```
import {Component, OnInit} from '@angular/core';

@Component({selector: 'list-cmp', template: `...`})
export class ListComponent implements OnInit {
  errorMessage: string;
  products: Product[];

  constructor (private productService: ProductService) {}

  ngOnInit() { this.getProducts(); }

  getProducts() {
    this.productService.getProducts()
      .subscribe(
        products => this.products = products,
        error => this.errorMessage = <any>error);
  }
}
```