

59 - TypeScript: decoradores

En TypeScript, un decorador es una función especial que se utiliza para modificar o extender el comportamiento de clases, métodos, propiedades o parámetros.

Las funciones decoradoras permiten agregar metadatos y comportamientos adicionales.

Las funciones decoradoras se ejecutan en Angular en tiempo de compilación de la aplicación, recordemos que el framework Angular compila el código TypeScript, HTML y CSS, generando JavaScript que es lo que puede interpretar un navegador web.

Veremos con dos ejemplos cuál es la sintaxis para crear funciones decoradoras en TypeScript y en los próximos conceptos veremos las funciones decoradoras que vienen en el framework de Angular.

Función decoradora de clase sin parámetros.

Podemos agregar el código siguiente a un proyecto que tengamos en Angular:

```
app.component.ts
```

```

import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

function MostrarMensajeDeCreacion(constructor: Function) {
  console.log(constructor.toString());
  const prototipo = constructor.prototype;
  const nombresMetodos = Object.getOwnPropertyNames(prototipo)
    .filter(nombre => typeof prototipo[nombre] === 'function');
  console.log('Métodos:', nombresMetodos.join(', '));
}

// Aplicamos el decorador a una clase
@MostrarMensajeDeCreacion
class MiClase {

  constructor() {
    console.log('Objeto creado');
  }

  metodo1() {
    console.log('Metodo 1 ejecutado');
  }

  metodo2() {
    console.log('Metodo 2 ejecutado');
  }
}

@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'prueba';
}

```

Para aplicar una función decoradora a una clase debemos anteceder al nombre de la clase el caracter @ seguido del nombre de la función decoradora:

```

@MostrarMensajeDeCreacion
class MiClase {

```

La función decoradora `MostrarMensajeDeCreacion` llega una función (el constructor de la clase) como parámetro. En este caso, la función imprime la representación en cadena de la clase y los nombres de los métodos de la clase:

```
function MostrarMensajeDeCreacion(constructor: Function) {
  console.log(constructor.toString());
  const prototipo = constructor.prototype;
  const nombresMetodos = Object.getOwnPropertyNames(prototipo)
    .filter(nombre => typeof prototipo[nombre] === 'function');
  console.log('Métodos:', nombresMetodos.join(', '));
}
```

No hemos definido objetos de la clase 'MiClase' y podemos ver que el compilador de Angular ejecuta el decorador cuando compila la aplicación:



Es solo un ejemplo elemental para entender que la función decoradora se ejecuta independientemente a que definamos objetos de la clase. El decorador se ejecuta en tiempo de compilación y se utiliza para modificar o extender la funcionalidad de la clase, nosotros solo hemos hecho unas pocas salidas por la consola.

Función decoradora de clase con parámetros.

Podemos pasar parámetros a una función decoradora, veamos un ejemplo donde le pasamos un objeto de una determinada interface (en forma similar el framework Angular utiliza los decoradores para añadir funcionalidades por ejemplo a las componentes):

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

// Interfaz para el parámetro del decorador
interface DetallesDeCreacion {
  selector: string;
  templateUrl: string;
  styleUrls: string;
}

// Decorador de Clase para Mostrar Detalles de Creación con Parámetro
// de la Interfaz
function MostrarDetallesDeCreacion(detalles: DetallesDeCreacion) {
  return function (constructor: Function) {
    console.log(constructor.toString());
    // Mostrar información adicional sobre métodos y propiedades
    const prototipo = constructor.prototype;
    const nombresMetodos = Object.getOwnPropertyNames(prototipo)
```

```

        .filter(nombre => typeof prototipo[nombre] === 'function');
        console.log('Métodos:', nombresMetodos.join(', '));

        console.log('Template URL:', detalles.templateUrl);
        console.log('Style URL:', detalles.styleUrl);
    };
}

// Aplicamos el decorador a una clase con un parámetro de la interfaz
@MostrarDetallesDeCreacion({
    selector: 'app-mi-clase',
    templateUrl: 'mi-clase.component.html',
    styleUrl: 'mi-clase.component.css',
})
class MiClase {

    constructor() {
        console.log('Objeto creado');
    }

    metodo1() {
        console.log('Metodo 1 ejecutado');
    }

    metodo2() {
        console.log('Metodo 2 ejecutado');
    }
}

@Component({
    selector: 'app-root',
    imports: [RouterOutlet],
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent {
    title = 'prueba';
}

```

```

interface DetallesDeCreacion {
    selector: string;
    templateUrl: string;
    styleUrl: string;
}

```

Se define una interfaz llamada DetallesDeCreacion que especifica tres propiedades: selector, templateUrl, y styleUrl. Es solo un ejemplo y los nombres de las propiedades pueden ser cualquiera.

```
function MostrarDetallesDeCreacion(detalles: DetallesDeCreacion) {
  return function (constructor: Function) {
    console.log(constructor.toString());
    // Mostrar información adicional sobre métodos y propiedades
    const prototipo = constructor.prototype;
    const nombresMetodos = Object.getOwnPropertyNames(prototipo)
      .filter(nombre => typeof prototipo[nombre] === 'function');
    console.log('Métodos:', nombresMetodos.join(', '));

    console.log('Template URL:', detalles.templateUrl);
    console.log('Style URL:', detalles.styleUrl);
  };
}
```

Se define una función llamada `MostrarDetallesDeCreacion` que toma un parámetro `'detalles'` de tipo `DetallesDeCreacion`. Esta función devuelve otra función que actúa como el decorador real.

Dentro de la función del decorador, se imprime la representación en cadena del constructor de la clase, así como los nombres de los métodos de la clase.

Luego, imprime las propiedades del parámetro `'detalles'` que fueron proporcionadas al aplicar el decorador.

```
// Aplicamos el decorador a una clase con un parámetro de la interfaz
@MostrarDetallesDeCreacion({
  selector: 'app-mi-clase',
  templateUrl: 'mi-clase.component.html',
  styleUrls: 'mi-clase.component.css',
})
class MiClase {

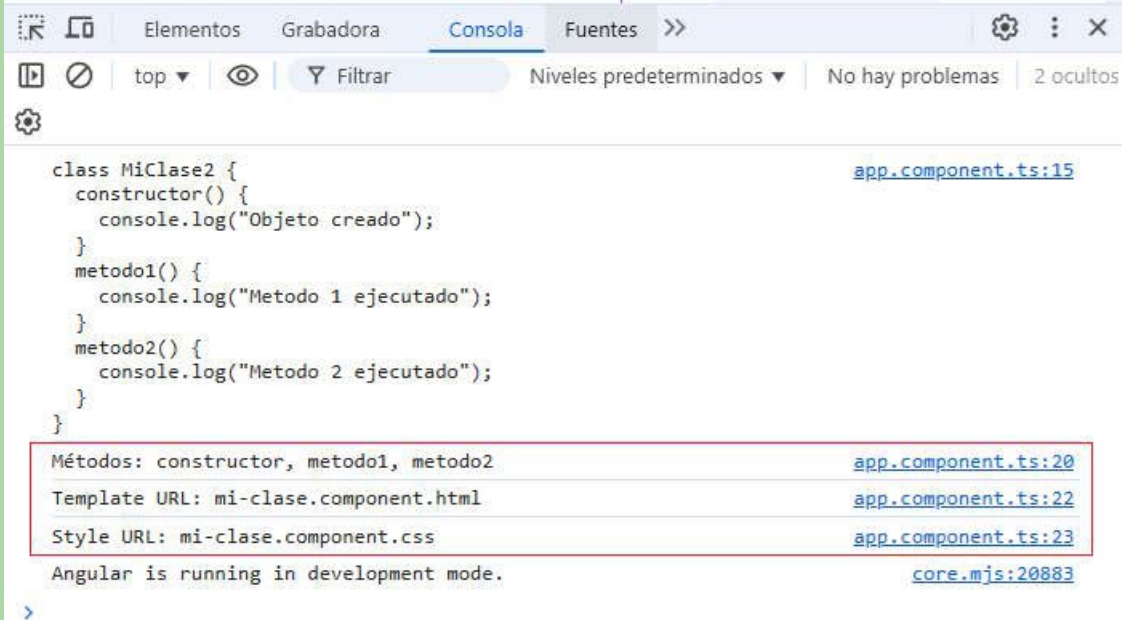
  constructor() {
    console.log('Objeto creado');
  }

  metodo1() {
    console.log('Metodo 1 ejecutado');
  }

  metodo2() {
    console.log('Metodo 2 ejecutado');
  }
}
```

Se aplica el decorador `MostrarDetallesDeCreacion` a la clase `MiClase` con un objeto que cumple con la interfaz `DetallesDeCreacion`. Los valores proporcionados (como `selector`, `standalone`, `templateUrl`, y `styleUrl`) se utilizarán dentro del decorador para mostrar información adicional.

Tengamos en cuenta que no creamos objetos de la clase `'MiClase'` y el compilador de Angular ejecuta la función decoradora aplicada a la clase:



The screenshot shows the Angular IDE interface with the 'Consola' (Console) tab selected. The console displays the compilation output for a component class named 'MiClase2'. The output includes the class definition, its methods, and the URLs for its template and styles. A red box highlights the summary information: 'Métodos: constructor, metodo1, metodo2', 'Template URL: mi-clase.component.html', and 'Style URL: mi-clase.component.css'. The console also shows that Angular is running in development mode.

```
class MiClase2 {
  constructor() {
    console.log("Objeto creado");
  }
  metodo1() {
    console.log("Metodo 1 ejecutado");
  }
  metodo2() {
    console.log("Metodo 2 ejecutado");
  }
}

Métodos: constructor, metodo1, metodo2
Template URL: mi-clase.component.html
Style URL: mi-clase.component.css
Angular is running in development mode.
```

Este ejemplo nos debe dar una idea como el compilador de Angular procesa el código de la componente AppComponent:

```
@Component({
  selector: 'app-root',
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'prueba';
}
```

Lo primero que hace es ejecutar la función decoradora `@Component` y le pasa como parámetro un objeto, en este caso inicializando 4 propiedades: `selector`, `imports`, `templateUrl` y `styleUrls`. Todos estos datos complementan a la clase `AppComponent` para acceder a los archivos: `app.component.html` y `app.component.css`

Retornar