Intro a Typescript e inyección de dependencias

Coding Potions –3

May 29, 2018

Introducción

IMPORTANTE Con ‘Controllers’ no me refiero al mismo concepto que en AngularJS. Con controllers me refiero a los archivos .ts en sí, es decir a la lógica del componente. Aunque en realidad a estos archivos se les llama Component, yo los llamo controllers para no confundirlos con todo el componente en conjunto, es decir, a todos los archivos que componen dicho componente.

Controllers

Los controladores se encargan de ‘controlar’ la vista. Es decir se encargan de atender los datos de la vista. Los controladores pueden especificar que datos se pueden mostrar en la vista, cuales no, y modificar sus propiedades así como los datos en sí, de tal forma que, en la vista, los datos se refrescan con la nueva información ya actualizada.

Por ejemplo, imagina que tenemos un array de strings en los que guardamos cada nota que el usuario crea en nuestra aplicación de tomar notas. Si desde el controlador actualizamos el array de notas, y por ejemplo, añadimos un nuevo valor, automáticamente en la vista html, se refresca la parte de mostrar las notas con la nueva nota sin tener que actualizar la página.

Estructura de los controllers

La estructura básica de los componentes la vimos por encima en este artículo. Si has creado el componente con angular cli, te habrás dado cuenta de que al comienzo importa OnInit, por ejemplo para el componente que he creado para mostrar una lista de items:

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


@Component({
  selector: 'app-item-list',
  templateUrl: './item-list.component.html',
```

```
  styleUrls: ['./item-list.component.css']
})
export class ItemListComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Y que dentro de la clase del controller, llama al método ngOnInit() vacío. Pues bien, todo lo que pongamos dentro de

ngOnInit() se ejecutará justo al cargar la página que contiene el componente, es decir, se ejecuta al inicio.

Pero también hay un método llamado constructor(), que también se ejecuta al cargar la página. El constructor también se utiliza para la inyección de dependencias.

¿Qué diferencia hay entre ngOnInit() y el constructor() en Angular?

Como otros lenguajes, typescript también tiene un constructor de clase, en este caso

el constructor se ejecuta antes que el ngOnInit(). Normalmente se usa el constructor para inicializar variables, y el ngOnInit para inicializar o ejecutar tareras que tienen que ver con Angular. Todo esto lo podemos poner directamente en el constructor y funcionaría de la misma manera, pero no está de más tener más separado el código para que sea más mantenible.

Sintaxis TypeScript

Typescript es un lenaguaje desarrollado y mantenido por Microsoft. Typescript extiende a Javascript, es decir, puedes insertar código JS dentro de archivos TS, que va a funcionar. Entre sus características, destacan:

Tipado estático

Typescript añade tipado estático a Javascript, es decir, puedes crear variables con un tipo fijado (string, números, etc). El tipado estático es opcional por lo que deja al desarrollador si quiere utilizarlos.

Los tipos que vienen con typscript son:

- boolean: (true/false)
- number: integers, floats, Infinito y not a number

- string: caractéres o lista de caracteres
- []: Arrays of other types, like number[] or boolean[]
- {}: Objeto literal
- undefined

Las ventajas que ofrece tener tipado estático son que el compilador detecta errores más fácilmente y el código es mantenible y menos propenso a tener errores.

Programación orientada a objetos con clases en Typescript

Para declarar las clases se utiliza la misma sintaxis que Javascript ES6, por ejemplo:

```
export class Rectangle() {
    private width:number:
    private height:number;
}
```

También he declarado dos atributos de clase (a diferencia de Javascript que no los puedes declarar directamente). Como pasa en otros lenguajes, aquí también se pueden

declarar variables ‘private’ es decir, no se puede acceder a estas variables desde fuera de la clase en la que están declaradas

Para crear funciones se hace igual que en ES6:

```
calcArea() {
    return this.height * this.width;
  }
```

Para la herencia también se hace como en ES6:

```
class Dog extends Animal {
```

```
}
```

Interfaces

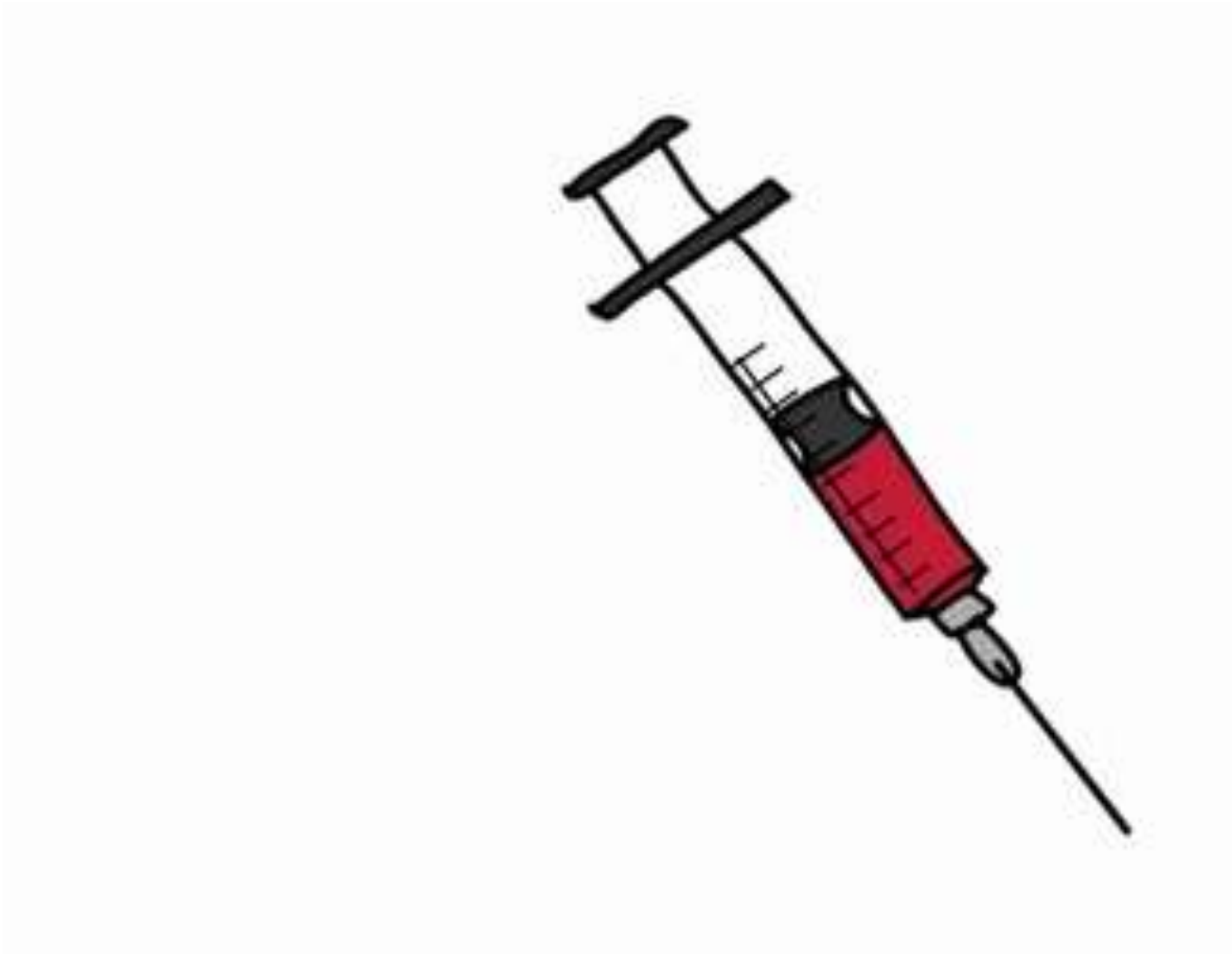
Igual que en Java, Typescript también tiene interfaces:

```
interface Person {
    firstName: string;
    lastName: string;
}
```

Obviamente el código Typescript se puede usar sin tener que usar Angular, ya que simplemente se compila a código Javascript entendible por todos los navegadores.

El resto de cartacterísticas (por ejemplo los arrays) son exactamente igual que en ES6.

Inyección de dependencias



Inyección de dependencias es un patrón de diseño orientado a objetos que se trata de suministrar las dependencias directamente a los objetos.

Imagina que inicializamos las dependencias de una clase de la manera tradiccional:

```
public Example() {
    object_test = Factory.getObject();
}
```

A la hora de testear esta clase es mucho más complicado porque si quieres comprobar la llamada a Factory o

interceptarla no puedes, para ello lo que se hace es inicializar la dependencia directamente desde el constructor, de esta forma lo que conseguimos es que al testear la aplicación, podamos crear el objeto Factory desde fuera:

```
public Example(OtherClass object) {
    object_test = Factory.getObject();
}
```

Una alternativa a este patrón es definir las dependencias desde los métodos Setters.

Este código pertenece a Java, para hacerlo en Typescript tenemos que declarar las dependencias (previamente importadas al comienzo del archivo) en el constructor. Por ejemplo:

```
constructor(itemService: ItemService)
```

En este caso dentro del ItemService tenemos que poner la anotación @Injectable(), esta anotación define que el servicio puede ser inyectado mediante inyección de dependencias.

Conclusiones

Typescript suena muy diferente pero no deja de ser ES6 con unas cuantas cosas más. En mi opinión es un lenguaje en el que todo el mundo se siente cómodo porque es un poco una mezcla de Java y de Javascript. Como siempre, esto es una pequeña muestra de todo lo que Typescript puede ofrecer. Si te has quedado con las ganas de aprender más sobre Typescript visita su página oficial:

http://www.typescriptlang.org/index.html