Anexo III TypeScript y Web Components

1.1 TypeScript

Antes de comenzar a aprender Angular, vamos a conocer una serie de pinceladas del lenguaje en el que esta basado, TypeScript.

TypeScript es un lenguaje de programación de distribución y código abierto desarrollado por Microsoft. Su propio fabricante lo define como un *superset* (superconjunto) de JavaScript que, en esencia, mejora JavaScript añadiéndole funciones de tipado y de programación orientada a objetos.

Además, la sintaxis y desarrollo en TypeScript incorpora las mejoras y características de ECMAScript 6, estandarización de JavaScript.

Debido a estas nuevas especificaciones, el *framework* Angular utiliza TypeScript como lenguaje de programación para la lógica de las aplicaciones. Como las características de ECMAScript 6 aún no son soportadas por los navegadores web, el propio *framework* Angular 'transpila' el código TypeScript directamente a JavaScript para que pueda ser interpretado en los clientes web.

1.2 Instalación de TypeScript

Antes de instalar TypeScript y debido a que también es necesario para Angular, debemos instalar Node Js y NPM. Al final del libro dispones de un anexo que explica los pasos para su instalación.

Para instalar TypeScript, podemos emplear el gestor de paquetes NPM. De esta manera, en la consola del equipo completamos:

```
npm install -g typescript
```

Para comprobar la correcta instalación o la versión de TypeScript en el caso de que ya estuviera instalado, completamos en la consola:

```
tsc -v
```

Que nos devolverá la versión instalada y así tendremos nuestro equipo listo para trabajar.

1.3 Tipos de datos en TypeScript

Ya hemos dicho que una de las principales características de TypeScript es que proporciona tipos de datos, siendo la sintaxis de declaración, tipo e inicialización de variables globales, la siguiente:

```
let nombrevariable: tipodedato = valor;
```

De esta manera TypeScript soporta los siguientes tipos básicos de datos:

- Strings

  ```
  let texto: String = 'Cadena de caracteres';
  ```

  Que además, permite multilínea con comillas *backsticks*:

  ```
  let textomulti: String = ` ACME S.A.
                            Paseo de la Castellana, 100
                              28.010 Madrid ` ;
  ```

- Números

  ```
  let decimal: number = 6;
  let hexadecimal: number = 0xf00d;
  let binario: number = 0b1010;
  let octal: number = 0o744;
  ```

- *Arrays*.

  Que se declaran simplemente con corchetes y el tipo de dato de sus elementos mediante:

  ```
  let lista: number[] = [2,4,8];
  ```

  o bien, de manera más expresiva mediante:

  ```
  let lista: Array<number> = [1, 2, 3];
  ```

- *Any*

Permite que la variable tome cualquier tipo de dato:

```
let cualquiera: any = 12;

let cualquiera = "¡Hola Mundo!";
```

- Booleanos.

Obliga a seleccionar los valores booleanos true o false:

```
let booleano: true = 12;
```

Para más información sobre otros tipos de datos y su empleo, podemos consultar la documentación de TypeScript en:

https://www.typescriptlang.org/docs/handbook/basic-types.html.

Vamos a practicar un poco de TypeScript en nuestro editor.

Si quieres utilizar Visual Studio Code, mi editor favorito, al final del libro aprendemos a instalarlo y configurarlo.

En primer lugar, para comprobar los tipos de datos en TypeScript creamos un directorio de nombre typestest, por ejemplo, en nuestro equipo. A continuación, vamos a crear dos archivos de ejemplo. El primero, un clásico archivo HTML denominado index.html con el siguiente código:

index.html

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title></title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1"
  </head>
```

```
    <body>
        <h1 id="encabezado"></h1> ①
        <script type="text/javascript" src="test.js"></script> ②
    </body>
</html>
```

En el cual:

① Hemos introducido un simple h1 con el id "encabezado".

② Y referenciamos en una etiqueta script a un archivo JavaScript test.js.

Creamos un segundo archivo TypeScript denominado test.ts, con en el que comenzamos escribiendo el siguiente código:

test.ts

```
let nombre: string = "Carlos"; ①

function saludo(nombre){
    return "Hola "+nombre;   ②
}
document.getElementById("encabezado").innerHTML = saludo(nombre);  ③
```

En el cual:

① Declaramos la variable nombre y la inicializamos con el valor "Carlos".

② Creamos una función saludo con el parámetro nombre que devuelve un "Hola" más el nombre.

③ Y finalmente la inyectamos en el elemento "encabezado" del html con el método JavaScript.

El fichero TypeScript como tal no lo podemos emplear en el html, de hecho en el script hemos indicado un archivo js, ya que los navegadores no lo reconocerían. Por tanto, lo tenemos que 'transpilar' a JavaScript.

Para ello, en la consola de nuestro equipo, dentro de la carpeta typetest tecleamos:

```
tsc test.ts
```

Y comprobamos en el editor como se ha creado un archivo JavaScript, con algunos cambios respecto al archivo original, por ejemplo, observamos como la palabra reservada let se ha convertido a var.

Ahora, cuando carguemos nuestro archivo index.html en el navegador verificamos como imprime por pantalla el valor correctamente.

Hola Carlos

Vamos a continuación a realizar una serie de cambios, y para no tener que 'transpilar' manualmente. Añadimos, en la consola del equipo y en el directorio que hemos creado para el ejemplo, typestest, el siguiente comando:

```
tsc –w test.ts
```

Podemos cambiar el valor de la variable nombre para comprobar como 'transpila' automáticamente.

¡Continuamos! Vamos a modificar ahora el código de test.ts de la siguiente manera:

test.ts

```
function imprPantalla(a,b){
    return a+b;
}

let a: number = 10;

let b: number = 5;

document.getElementById("encabezado").innerHTML = imprPantalla(a,b);
```

Lógicamente este código imprimirá por pantalla el número 15. Pero si ahora modificamos:

```
let a: number = “Jorge”;
```

La consola nos devolverá un error pues el valor Jorge no es un número.

```
test.ts(9,1): error TS2322: Type '"Jorge"' is not assignable to type 'number'.
15:13:29 - Compilation complete. Watching for file changes.
```

Otra de las particularidades de TypeScript, es que se puede establecer el tipo de dato que devolverá una función.

La sintaxis será la siguiente:

```
function nombreFuncion(parametros): tiposdedato {
        //código de la función
}
```

Un sencillo ejemplo sería:

```
function devuelveEdad ( edad ): number {
        let miEdad = edad;
        return miEdad;
}
```

1.4 Variables en TypeScript

Ya hemos adelantado en el apartado anterior, que en TypeScript las variables se declaraban con la palabra reservada *let*.

Pero además, TypeScript permite también emplear la palabra reserva *var* como en JavaScript, entonces… ¿cuándo emplear una u otra forma?

Pues dependiendo del alcance de la variable. Si necesitamos declarar la variable de forma local al bloque de código, declaración o expresión donde se ejecuta emplearemos *let*. En cambio si la declaración se necesita realizar de manera global, emplearemos *var*.

Veamos un ejemplo a continuación.

Vamos a editar nuestro archivo TypeScript sustituyendo todo el código por el siguiente:

test.ts

```
let a = 10; ①

if (true){

    let a = 15 ②
    console.log("Dentro del if a tiene un valor de "+a);
    } else {
    //
    }

console.log("Fuera del if a tiene un valor de "+a);
```

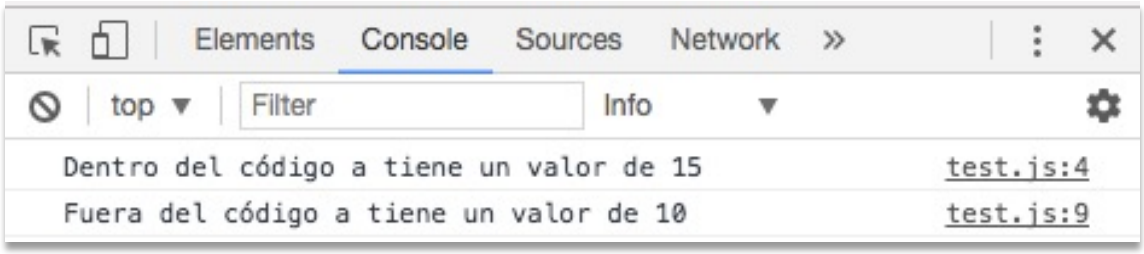
En el cual:

① Declaramos la variable a y la inicializamos con valor 10.

② Creamos una estructura if-else donde cambiamos el valor de a a 15 y lo imprimimos en la consola dentro del if (el else lo dejamos vacío).

③ Y finalmente, imprimimos el valor de a fuera del if-else.

Comprobaremos en la consola del navegador, como arroja dos valores diferentes de acuerdo al código que hemos programado:

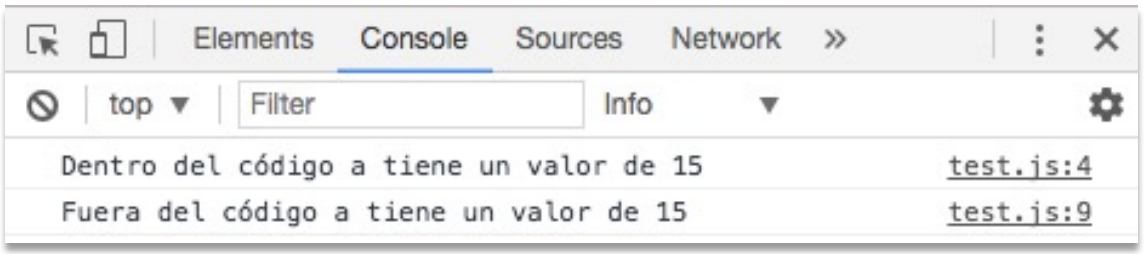


Si ahora modificamos la declaración de la variable dentro del *if* utilizando la palabra reservada *var:*

test.ts

```
var a = 10;

if (true){

    var a = 15
    console.log("Dentro del if a tiene un valor de "+a);
    } else {
    //
    }

console.log("Fuera del if a tiene un valor de "+a);
```

Como la variable pasa a ser global, su valor dentro del *if* también se utiliza fuera de esta y por tanto ambos *logs* muestran el mismo valor en la consola.



1.5 Clases en TypeScript

En TypeScript disponemos de clases para crear objetos, consiguiendo de esta manera que este lenguaje pueda ser orientado a objetos.

La sintaxis para crear clases será la siguiente:

```
class NombreClase {
        public/private nombrepropiedad: tipo de dato;
        ….

        public/private nombremetodo() {
                //código del método
        }
        …
}
```

Por ejemplo, en nuestro archivo test.ts sustituimos todo el código para añadir:

test.ts

```
class Curso {
        public titulo: string;                ①
        public descripcion: string;
        public horas: number;
        public inscritos: number;

        public getInscritos() {               ②
                return this.inscritos;
        }

        public setInscritos(inscritos: number) {
                this.inscritos = inscritos;
        }

        public addInscrito(){
                this.inscritos++;
         }
```

```
public remInscrito() {
        this.inscritos--;
}

var  cursoAngular = new Curso();              ③

cursoAngular.setInscritos(9);
cursoAngular.addInscrito();

console.log(cursoAngular.getInscritos());     ④
```

En el cual:

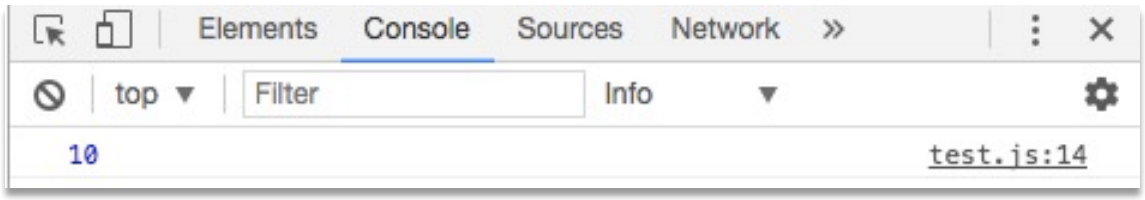
① Declaramos cuatro propiedades públicas.

② Creamos cuatro métodos públicos para devolver el número de inscritos, establecer el número de inscritos, añadir un nuevo inscrito y eliminar un inscrito.

③ Creamos un objeto cursoAngular de la clase Curso, establecemos el valor 9 en la propiedad inscritos con el método setInscritos y añadimos uno más con el método addInscrito.

④ Imprimimos en la consola la cantidad de inscritos del cursoAngular.

Comprobamos el resultado esperado, en la consola del navegador:



1.6 Constructor en TypeScript

También TypeScript permite el uso de constructores, siendo estos, métodos para inicializar las propiedades o atributos de un clase.

La sintaxis es la siguiente:

```
class NombreClase {
        public/private nombrepropiedad: tipo de dato;
        ….
        constructor () {
                this.nombrepropiedad = valor;
                …
        }

        public/private nombremetodo() {
                //código del método
        }
        …
}
```

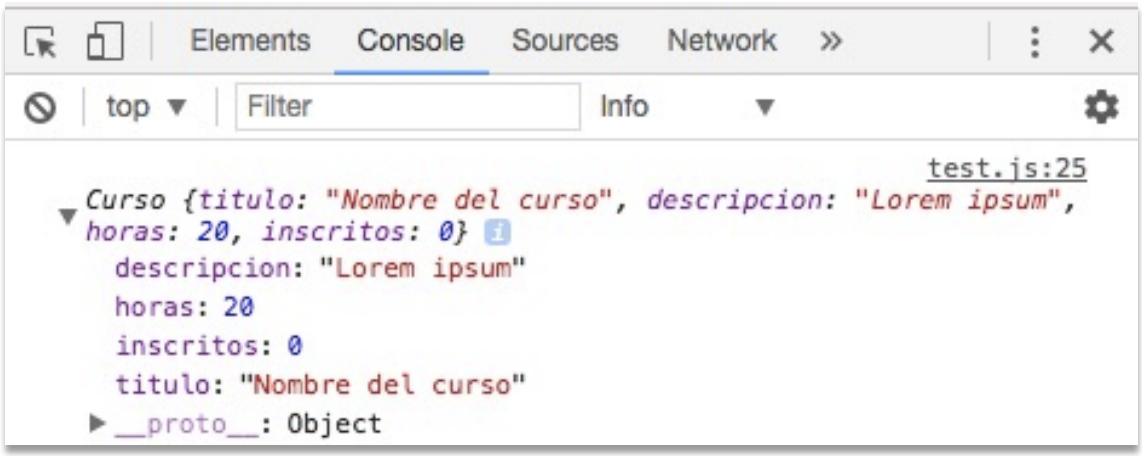
Por ejemplo, podemos sustituir la clase anterior Curso por el siguiente código:

test.ts

```
class Curso {
        public titulo: string;
        public descripcion: string;
        public horas: number;
        public inscritos: number;
   }

constructor() {
      this.titulo = "Nombre del curso";
      this.descripcion = "Lorem ipsum";
      this.horas = 20;
      this.inscritos = 0;
   }
```

```
public getInscritos() {
                return this.inscritos;
        }

public setInscritos(inscritos: number) {
                this.inscritos = inscritos;
        }

public addInscrito(){
       this.inscritos++;
   }

public remInscrito() {
       this.inscritos--;
   }

}

var  cursoAngular = new Curso();

console.log(cursoAngular);
```

Comprobaremos en la consola del navegador como se ha creado este objeto de la clase curso y sus variables han sido inicializadas con los valores por defecto del constructor.



También podemos emplear parámetros en el constructor. Para ello modificamos por ejemplo el constructor de la siguiente forma.

```
  …
  constructor( titulo, descripcion, horas ) {
      this.titulo = titulo;
      this.descripcion = descripcion;
      this.horas = horas;
      this.inscritos = 0; //en este caso se establecerán con el método
   }
…
```

Y sustituimos la línea de creación del objeto cursoAngular:

```
…
var  cursoAngular = new Curso("Curso Angular","Lorem...",100);
…
```

De esta forma, el objeto se crea con los parámetros introducidos para cada propiedad.

Si no disponemos de algún parámetro a la hora de crear el objeto, para evitar errores de compilación, se puede emplear como valor *undefined* o *null*, de tal manera que se introduzcan todos los parámetros y en el orden especificado.

1.7 Interfaces en TypeScript

Para concluir con esta introducción a TypeScript, completamos el conocimiento de la sintaxis básica para la creación de objetos con interfaces.

Las interfaces, nos permitirán obligar a definir clases con unas determinadas propiedades o métodos.

Su sintaxis es:

```
interface NombreInterfaz {

        nombrepropiedad: tipodedato;
        nombreMetodo(): tipodedato;

}
```

Y se implementan en la clase:

```
class Nombredelaclase implements NombreInterfaz {
 …
}
```

En el ejemplo anterior, añadimos antes de la clase en el archivo test.ts, el siguiente código:

test.ts

```
interface DatosMaestros {
        titulo: string;
        addInscrito();
}
…
```

Modificamos la primera línea de la definición de la clase de la siguiente forma:

```
…
class Curso implements DatosMaestros {
…
}
```

Y eliminamos la propiedad titulo en el constructor:

```
…
constructor( descripcion, horas ) {
        this.descripcion = descripcion;
        this.horas = horas;
        this.inscritos = 0; //en este caso se establecerán con el método
    }
…
```

Como la interfaz obliga a implementar la clase con todas sus propiedades, la consola nos devolverá un error:

```
test.ts(6,7): error TS2420: Class 'Curso' incorrectly implements interface 'DatosMaes
tros'.
  Property 'addInscrito' is missing in type 'Curso'.
4:52:07 PM - Compilation complete. Watching for file changes.
```

El resto de características básicas de TypeScript pueden ser consultadas en su documentación:

https://www.typescriptlang.org/docs/home.html

1.8 Web Components

Los web components son un estándar del W3C, que permiten componer el desarrollo de aplicaciones web a partir de contenedores dedicados a una cierta funcionalidad. Estarán compuestos por archivos con el lenguaje de marcado HTML5, con código JavaScript para la lógica de negocio y con hojas de estilo CSS para la presentación.

Cada uno de estos componentes se puede implementar en el código de las páginas HTML5 a través de una etiqueta específica, que renderizará el componente y su funcionalidad, con una sintaxis muy simple:

```
<nombre-del-componente></nombre-del-componente>
```

Los web components pueden estar compuestos de 4 elementos:

- Templates. Son plantillas HTML5 con el contenido del componente.
- Custom Elements. Permiten la definición de elementos HTML propios.
- Shadow DOM. Podría definirse como un elemento DOM ajeno al DOM de la página web el cual encapsula el componente.
- HTML Imports. Líneas de código para en HTML, importar los ficheros HTML que contengan los componentes.

Angular, desde su primera versión a la actual, utiliza los web components como tecnología para el desarrollo de aplicaciones web, como veremos a lo largo de todo el libro.

Para más información sobre esta tecnología, su documentación se encuentra en:

https://developer.mozilla.org/es/docs/Web/Web_Components