Proyecto TypeScript Clases con Herencias

Gonzalo Pazos Sardella

TypeScript

Índice.

Concepto de clases.

TypeScript es un lenguaje de programación *orientado a objetos*. En estas clases podemos tener una agrupación de datos y funciones las cuales operan sobre dichos datos.

Dentro de estas clases podemos encontrar las llamadas **superclases** y las que se extienden de dicha, es decir su **herencia**.

SuperClase:

En mi caso tenemos una superclase llamada **armas**. Dentro de esta he declarado dos variables principales (tipo y precioBase), estas tendrán el funcionamiento de poder tener un tipo de arma y un precio base el cual tendrá esta arma. Depende del tipo de arma se le añadirá un valor extra.
Por ejemplo: Vemos que a un subfusil se le agrega al precio base un 1.8 de su valor base por ser ese tipo de arma específica. Esto se ve afectado al resto de tipos de armas. Todos estos valores son metidos en un método con el cual podemos recoger estos valores.

```
export class armas {       You, 6 minutes ago • Primer commit

   private _tipo: string
   public _precioBase: number

      constructor(tipo: string, precioBase: number){
         this._tipo = tipo,
         this._precioBase = precioBase
      }

      get tipoA(){
         return this._tipo
      }

      precio(){
         let precio: number = this._precioBase
         if (this._tipo == 'subfusil'){
            precio = this._precioBase * 1.8
         }
         if (this._tipo == 'fusil') {
            precio = this._precioBase * 1.4
         }
         if (this._tipo == 'escopeta'){
            precio = this._precioBase * 1.2
         }
         return precio
      }

      resumen() {
         return `El precio base es: ${this._precioBase} y el arma es: ${this._tipo}`;
         }
}
```

Herencias / Subclases:

En mi caso existen dos subclases (**cargadores** y **accesorios**), estas son un complemento de la clase principal "**Armas**". En mi caso estas tendrán el valor de añadir un mayor precio a las armas principales si así se desea.

Primero hemos importado a la subclase, la superclase para poder usar los valores de la superclase si nos es necesario. Estas son declaradas en el constructor.
En el método precio, ya existente en la superclase lo que hacemos es sobre escribirla con los nuevos valores añadidos a los tipos de armas iniciales. Para poder hacer uso de este método lo llamamos dentro del método precio con el **super.nombredelmétodo**.
Por ejemplo: Al accesorio mirilla, que se encuentra en la subclase **accesorios**, en caso de que la persona quiera tenerla pues se le añadirá al precio base un valor de 10.

```
import { armas } from "./armas";      You, 11 minutes ago • Primer com

export class accesorios extends armas {

   private _tipoAccesorio: string

   constructor(tipo:string, precioBase:number, tipoAccesorio:string){
      super(tipo, precioBase)
      this._tipoAccesorio = tipoAccesorio
   }

   get tipoAccesorioA(){
      return this._tipoAccesorio
   }

   get acceder(){
      return this._precioBase
   }

   precio(): number {
   let precio: number;
   precio = super.precio();
   if (this._tipoAccesorio == 'mirilla') {
      precio += 10;
   }
   if (this._tipoAccesorio == 'empuñadura') {
      precio += 15;
   }
   return precio;
   }

   resumen(){
   let resumen: string
   resumen = `${super.resumen()}, accesorio: ${this._tipoAccesorio}`
   return resumen
   }

}
```

Este mismo funcionamiento lo tiene la segunda subclase “**Cargadores**” la cual es un extendido de la clase “**Armas**”.

```
import { armas } from "./armas"        You, 21 minutes ago • Primer commit

export class cargadores extends armas{

   private _tipoCargador: string

      constructor(tipo:string, precioBase:number, tipoCargador:string){
         super(tipo, precioBase)
         this._tipoCargador = tipoCargador
      }

      get tipoCargadorA(){
         return this._tipoCargador
      }

      get acceder(){
         return this._precioBase
       }

      precio(): number {
         let precio: number;
         precio = super.precio();
         if (this._tipoCargador == 'extendido') {
           precio += 20;
         }
         if (this._tipoCargador == 'cinta') {
            precio += 40;
         }
         if (this._tipoCargador == 'rotativo') {
            precio += 59;
         }
         if (this._tipoCargador == 'tambor') {
            precio += 10;
         }
         return precio;
      }

      resumen(){
         let resumen: string
         resumen = `${super.resumen()}, cargador: ${this._tipoCargador}`
         return resumen
       }
}
```

Como vemos en el método precio lo hemos vuelto a sobrescribir, para que estos nuevos añadidos puedan complementar el valor principal de la **superclase**.

```
precio(): number {
    let precio: number;
    precio = super.precio();
    if (this._tipoCargador == 'extendido') {
      precio += 20;
    }
    if (this._tipoCargador == 'cinta') {
        precio += 40;
    }
    if (this._tipoCargador == 'rotativo') {
        precio += 59;
    }
    if (this._tipoCargador == 'tambor') {
        precio += 10;
    }
    return precio;
}
```

Todos los valores que le pongamos a tanto la **superclase**, como las **subclases** será recogido en un método llamado *resumen()*, el cual nos proporcionará información sobre lo que hemos puesto.

```
resumen(){
    let resumen: string
    resumen = `${super.resumen()}, cargador: ${this._tipoCargador}`
    return resumen
 }
```

Menú:

Para el menú he creado uno sencillo que cumple las necesidades. Primero deberemos importar “**leer**”, para que el usuario pueda escribir en la consola que opción quiere usar.

```
import { leer } from '../util/consolePrint'

export const menu = async () => {
    let n: number
    console.log('\n')
    console.log('1.- Crear objetos de distinto tipo y añadirlos al array.')
    console.log('2.- Listar el contenido del array')
    console.log('3.- Modificar información de un objeto del array')
    console.log('4.- Borrar algún objeto del array')
    console.log('4.- Ver la información de un objeto concreto')
    console.log('0.- Salir')
    n = parseInt( await leer('opción: '))
    return n
}       You, 37 minutes ago • Primer commit
```

Index:

Importar:

Como primer paso he importado al index todas las cosas necesarias para que este funcione correctamente. En mi caso la **superclase**, las **subclases** y el **menú**.

```
import { armas} from './classes/armas';
import { cargadores } from './classes/cargadores';
import { accesorios } from './classes/accesorios';
import { menu } from './util/menu';
```

Array:

Un array es una colección de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre.

Nuevamente crearemos un array donde guardaremos los objetos que crearemos.

```
let armasA: Array<armas> = new Array<armas>();
```

Funciones:

Función nuevo:

Esta función tiene el propósito de que una vez que la llamemos cree los objetos dentro de las arrays asignadas. Para guardar estos objetos he creado un array nuevo donde guardar esta información, que después igualamos al array principal es decir a "**armasA**". Vemos que en armas, asignaremos primero el tipo y después el precio base. Pero en accesorios o en cargadores tenemos que declarar los valores de la superclase y después los de la subclase para que le agregue al valor base los valores secundarios.

```
const nuevo = async () =>  {
    let armasF: Array<armas> = new Array<armas>();
    armasF[0] = new armas('escopeta', 1000)
    armasF[1] = new armas('fusil', 1400)
    armasF[2] = new cargadores('fusil', 1400, 'cinta')
    armasF[3] = new accesorios('escopeta', 1000, 'mirilla')
    return armasF
}
```

Función main:

En esta función llamaremos al menú el cual usaremos a través de un switch para que cuando el usuario ingrese un valor entre 0-5, este cumpla una función. En el caso 1, llama a la función nuevo el cual añadirá los valores al array. Y en el caso 4 podemos ver como con **delete** podemos borrar un objeto específico del array. Para que esto funcione debemos llamar a la función main y así ejecute este bloque de código.

```
const main = async () => {
    let n: number
    let arma: armas | undefined

do {
  n = await menu()
  switch(n){
  case 1:
        armasA = await nuevo()
        console.log(armasA)
        break
  case 2:
        console.log(armasA)
        break
  case 3:
        armasA[0]._precioBase = 1100
        console.log(armasA)
        break
  case 4:
        delete armasA[3]
        console.log(armasA)
  case 5:       You, an hour ago • Primer commit
        console.log(armasA[0])
  case 0:
console.log('\nAdios')
}
}while (n != 0)
}
main()
```