

# **Clases, herencias, overwriting, polimorfismo TypeScript**

·Introducción.....pág 3.

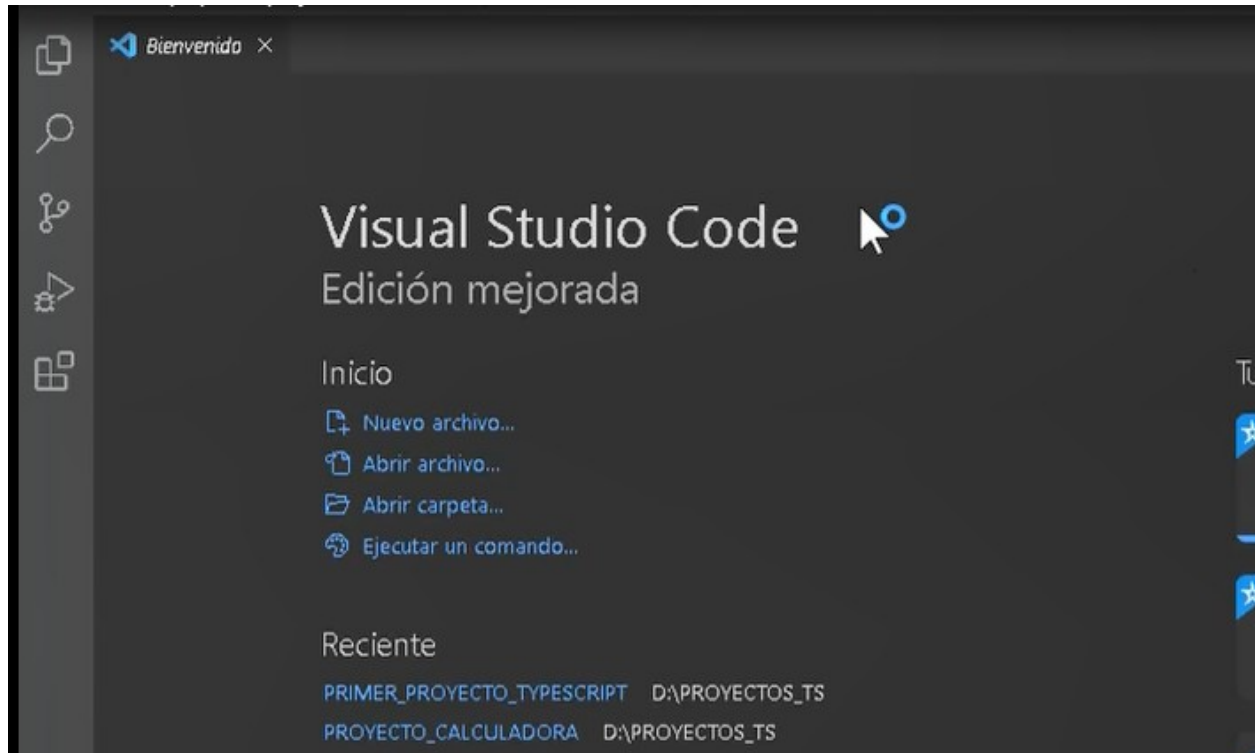
·Pasos.....pág 3

## INTRODUCCIÓN

En este documento vamos a explicar, paso a paso, el montaje de un proyecto typescript en el cual utilizaremos clases, en concreto una superclase, de la cual se extenderán otra clase más, vía herencia. Como aportación personal, también incluiremos un caso de polimorfismo y otro de sobreescritura de método (overwriting). Finalmente incluiremos un menú con distintas opciones.

## PASOS

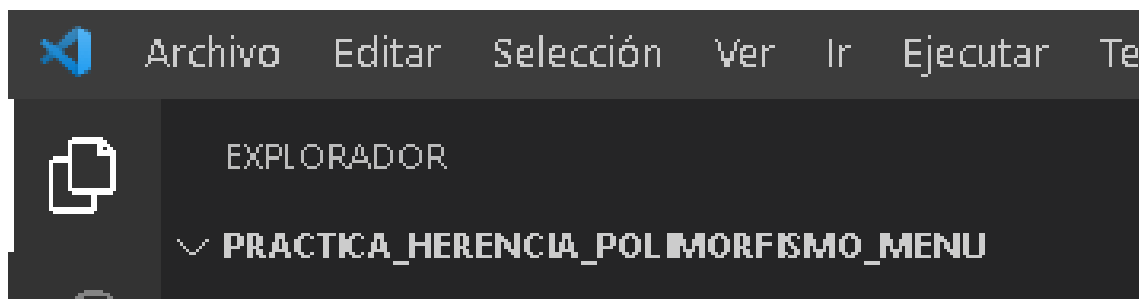
Con todo ello, procedemos a crear nuestro proyecto TypeScript. Abrimos VSC.



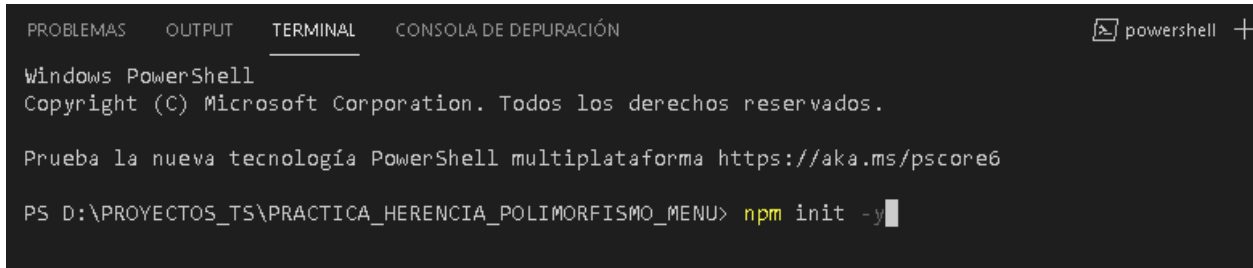
A continuación, creamos la carpeta de proyecto



Arrastramos la carpeta a VSC y nos queda así:



En la terminal del proyecto se crea el proyecto node con: **npm init -y**



```
PROBLEMAS  OUTPUT  TERMINAL  CONSOLA DE DEPURACIÓN
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS D:\PROYECTOS_TS\PRACTICA_HERENCIA_POLIMORFISMO_MENU> npm init -y
```

Se nos crea el **package.json**



Ahora escribimos en la terminal **npm install typescript -D**

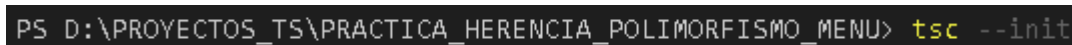


```
PS D:\PROYECTOS_TS\PRACTICA_HERENCIA_POLIMORFISMO_MENU> npm install typescript -D
```

Y nos crea esto:



A continuación escribimos en la terminal **tsc - -init**



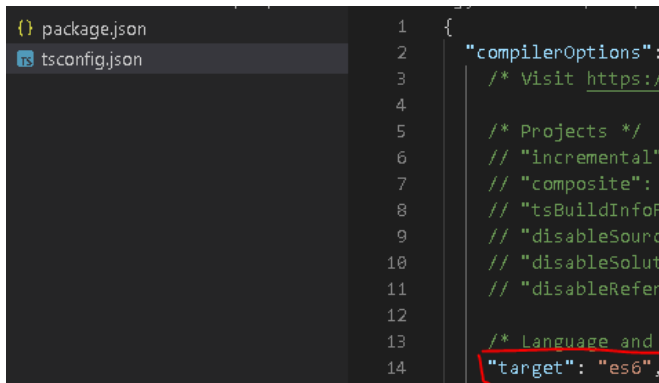
```
PS D:\PROYECTOS_TS\PRACTICA_HERENCIA_POLIMORFISMO_MENU> tsc --init
```

Y nos crea el archivo **tsconfig.json**



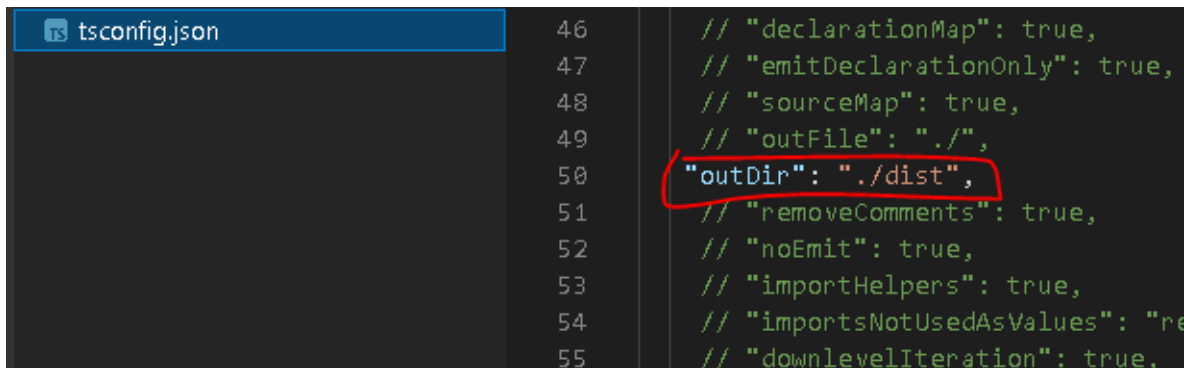
Vamos a dicho archivo para ver el contenido y hacemos los siguientes cambios:

Cambiamos "target:" a "es6"



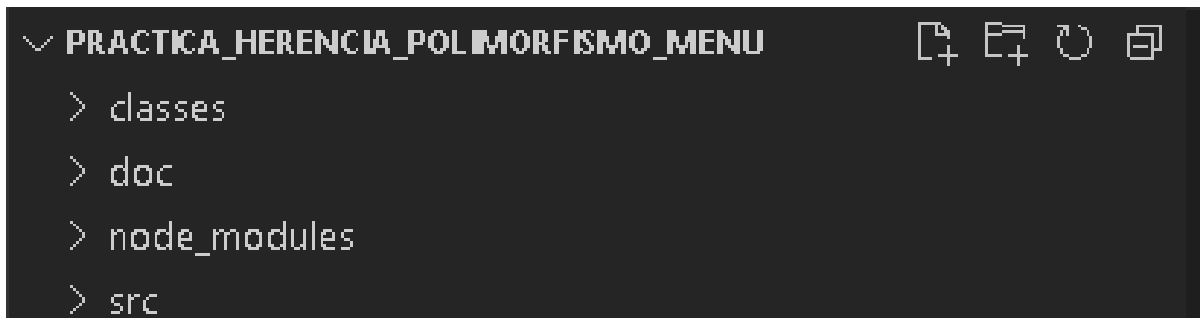
```
1 {
2   "compilerOptions":
3     /* Visit https://
4
5     /* Projects */
6     // "incremental"
7     // "composite":
8     // "tsBuildInfoF
9     // "disableSourc
10    // "disableSolut
11    // "disableRefer
12
13    /* Language and
14    "target": "es6",
```

Cambiamos "outDir": "./dist",



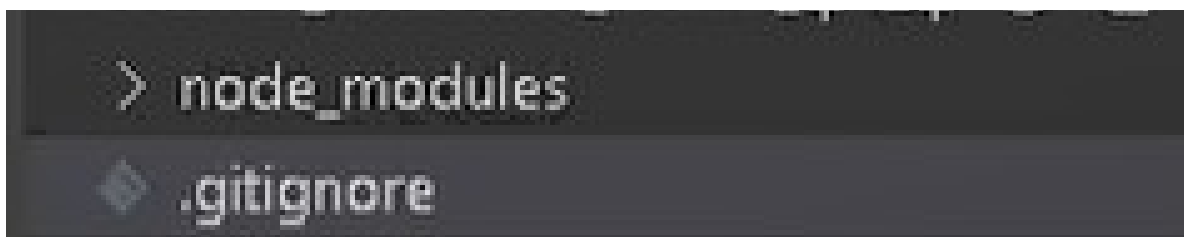
```
46 // "declarationMap": true,
47 // "emitDeclarationOnly": true,
48 // "sourceMap": true,
49 // "outFile": "./",
50 "outDir": "./dist",
51 // "removeComments": true,
52 // "noEmit": true,
53 // "importHelpers": true,
54 // "importsNotUsedAsValues": "re
55 // "downlevelIteration": true,
```

Creamos las carpetas **classes**, **doc** y **src**



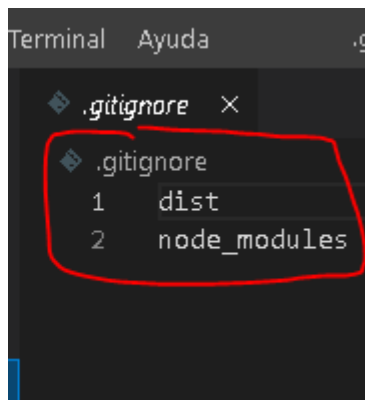
```
✓ PRACTICA_HERENCIA_POLIMORFISMO_MENU
  > classes
  > doc
  > node_modules
  > src
```

Creamos un archivo **.gitignore**

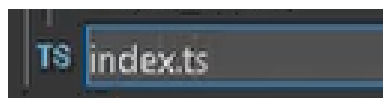


```
> node_modules
  .gitignore
```

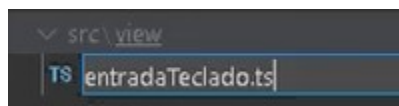
Y este es su contenido (para no subir a GitHub lo que ahí se especifica)



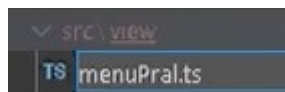
Dentro de carpeta **src** creamos el archivo **index.ts**



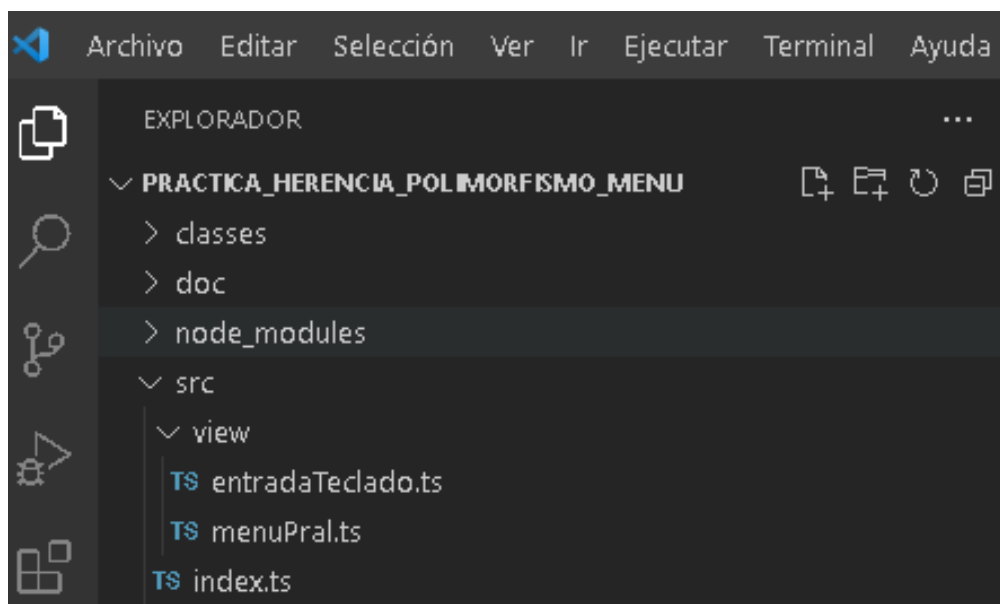
En **src** también creamos la carpeta **view** para poner dentro el archivo **entradaTeclado.ts**



Y en la misma carpeta ponemos otro archivo llamado **menuPral.ts**



Nos quedaría la siguiente estructura:



Ahora escribimos nuestro código en cada archivo \*.ts

Empezamos con el archivo **entradaTeclado.ts** (código que permite entrada de datos vía teclado)

```
src > view > TS entradaTeclado.ts > leerTeclado
1  import readline from 'readline'
2  let readlineI: readline.Interface
3
4  let leeLinea = (prompt: string) => {
5      readlineI = readline.createInterface({
6          input: process.stdin,
7          output: process.stdout,
8      })
9      return new Promise<string>((resuelta: any, rechazada: any) => {
10         readlineI.question(`${prompt}: `, (cadenaEntrada: string) => {
11             resuelta(cadenaEntrada)
12         })
13     })
14 }
15
16 export let leerTeclado = async (prompt: string) => {
17     let valor: string
18     valor = await leeLinea(prompt)
19     readlineI.close()
20     return valor
21 }
```

Después el archivo **menuPral.ts** (código que importa datos vía teclado y los aplica al menú)

```
TS menuPral.ts x
src > view > TS menuPral.ts > menuPral
1  import { leerTeclado } from '../view/entradaTeclado'
2
3  export const menuPral = async () => {
4      let seleccionado: number
5      console.log('\n')
6      console.log('1.- Precio base de nuevo vehículo')
7      console.log('2.- Comparador de vehículo (Coche vs Todoterreno)')
8      console.log('3.- Listar vehículos creados')
9      console.log('4.- MODIFICAR potencia del vehículo')
10     console.log('5.- Ver los datos de un vehículo concreto')
11     console.log('6.- BORRAR vehículo de la lista creada')
12     console.log('0.- Salir')
13     seleccionado = parseInt(await leerTeclado('opción '))
14     return seleccionado
15 }
```

En la carpeta **classes**, ponemos los siguientes archivos:

- **automovil.ts** (que será la superclase)
- **todoTerreno.ts** (que estenderá de la superclase vía herencia)

Veamos cada uno por separado.

El archivo **automovil.ts** (que será la superclase), contiene las siguientes líneas de las que cabe destacar, como ejemplos reseñables:

```
classes > TS automovil.ts > Automovil
1  export class Automovil {
```

La cual permite exportar la clase, que posteriormente será importada por los \*.ts necesarios.

```
private _precioBase: number;
```

Se usa “private” (encapsulamiento), lo cual nos obliga a crear un método para poder acceder:

```
get precioBase() {
  return this._precioBase;
```

Y con los distintos parámetros, creamos el constructor, que utilizaremos posteriormente:

```
constructor(precioBase: number, potenciaMotor: number) {
  this._precioBase = precioBase;
  this._potenciaMotor = potenciaMotor;
}
```

Gracias a todo lo anterior, podemos crear **precio()**

```
precio(): number {
  let precio: number;
  precio = this._precioBase;
  if (this._potenciaMotor > 150) {
    precio += 0.2 * precio;
  }
  return precio;
}
```

Y también **todo()**

```
todo() {
  return `Precio base: ${this._precioBase}, potencia: ${this._potenciaMotor}`;
}
```



El archivo **todoTerreno.ts** (que será una clase que extiende de la superclase vía herencia), contiene las siguientes líneas de las que cabe destacar, como ejemplos reseñables:

```
classes > TS todoTerreno.ts > TodoTerreno
1   import { Automovil } from './automovil';
```

Como se ve , antes exportamos **automovil**, pues ahora lo importamos en **todoTerreno.ts**

```
export class TodoTerreno extends Automovil
```

Y como se ve arriba, la clase **todoTerreno** extiende de la superclase **Automovil** vía herencia.

```
private _traccion: string;
```

De nuevo se hace un encapsulamiento vía “private”, por ello se crea un método para acceder:

```
get traccion() {
  return this._traccion
}
```

Y a continuación se da un caso de sobreescritura (overwriting) con **precio()** gracias a super

```
precio(): number {
  let precio: number;
  precio = super.precio();
  if (this._traccion == '4x4') {
    precio += 0.1 * precio;
  }
  return precio
}
```

Y lo mismo ocurre con **todo()**

```
todo(){
  let resultado: string
  resultado = `${super.todo()}, tracción: ${this._traccion}`
  return resultado
}
```

Y por último el archivo **index.ts** (que ejecuta las opciones del menú, según datos del teclado), invocando el código contenido en las distintas clases.

Entre los ejemplos más importantes, cabe destacar:

```
src > TS index.ts > ...
1  import { menuPral } from './view/menuPral'
2  import { leerTeclado } from './view/entradaTeclado'
3  import { Automovil } from '../classes/automovil';
4  import { TodoTerreno } from '../classes/todoTerreno';
5
```

Gracias a estas invocaciones, importamos de todas las fuentes que necesitamos.

```
const main = async () => {
  let n: number
  let n1: number
  let f1: string
  let auto: Automovil;
  let todoTerreno: TodoTerreno;
  let seleccionado: string | any;
```