

PRÁCTICA 3 Proyecto Clases Herencias

- Ciclo formativo: 2º ASIR
- Módulo: GBDLI
- Fecha de entrega de la práctica: 21/09/2021
- Nombre y apellidos: Daniel Muñoz Núñez

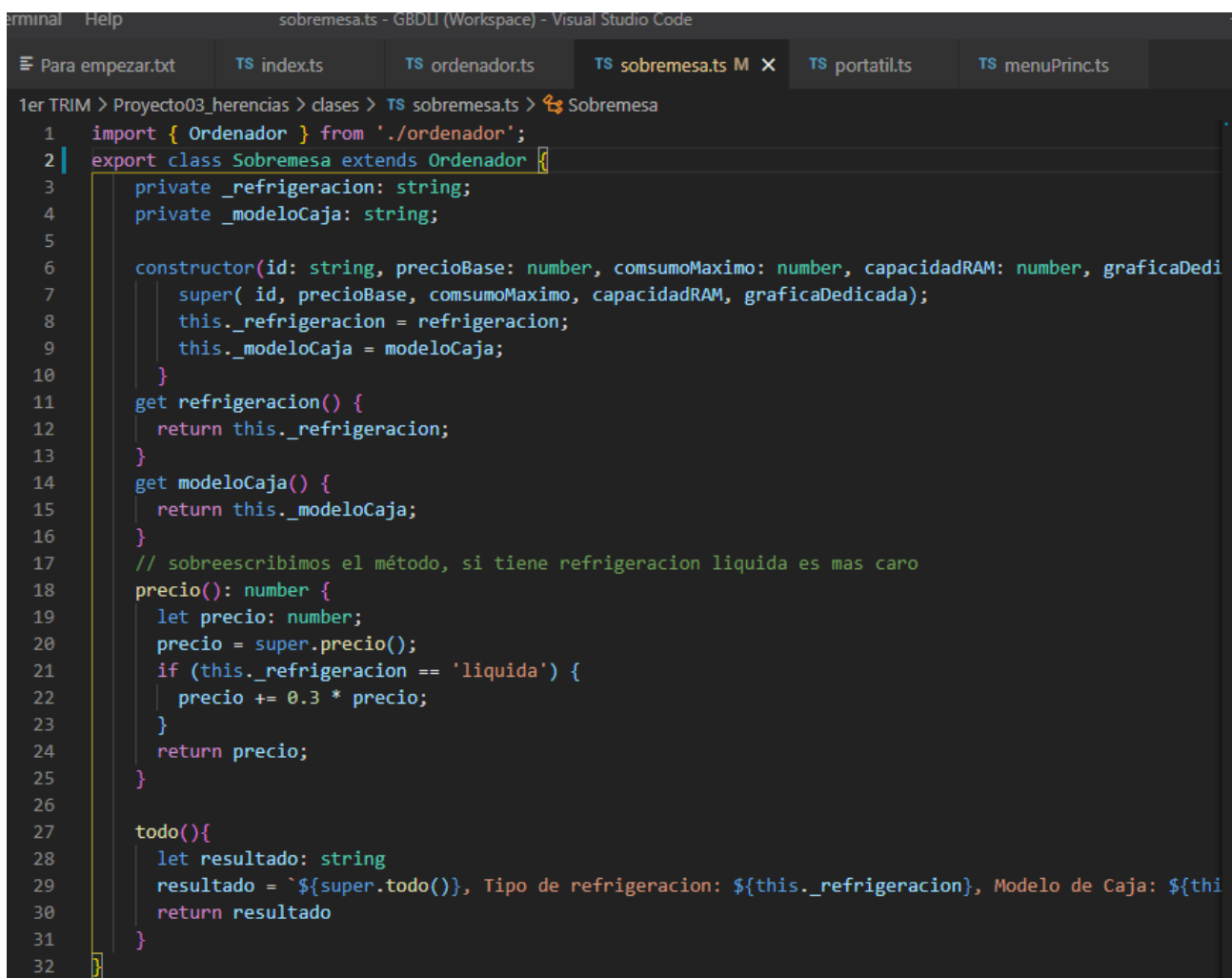
Índice de contenido

1) Encapsulación.....	3
2) Herencias.....	5
3) Sobreescritura de otros metodos.....	6
4) Uso del Super.....	7

1) Encapsulación.

Una de las características de Typescript es la posibilidad de crear módulos, los cuales no son más que una forma de encapsular código en su propio ámbito. Nos permiten agrupar nuestro código en diferentes ficheros, permitiéndonos exportarlos y utilizarlos donde los necesitemos. Esto nos facilita la tarea de crear software más ordenado y por ende más escalable y mantenible.

Por ejemplo en nuestro código tenemos la clase `Sobremesa` a la que le añadimos la palabra **export** al inicio, con esto conseguimos que se exporte y para utilizarla en otro fichero solo necesitamos importarla al inicio con **import { Sobremesa } from './clases/sobremesa'**;



```
1  import { Ordenador } from './ordenador';
2  export class Sobremesa extends Ordenador {
3      private _refrigeracion: string;
4      private _modeloCaja: string;
5
6      constructor(id: string, precioBase: number, consumoMaximo: number, capacidadRAM: number, graficaDedi
7          super( id, precioBase, consumoMaximo, capacidadRAM, graficaDedicada);
8          this._refrigeracion = refrigeracion;
9          this._modeloCaja = modeloCaja;
10     }
11     get refrigeracion() {
12         return this._refrigeracion;
13     }
14     get modeloCaja() {
15         return this._modeloCaja;
16     }
17     // sobreescribimos el método, si tiene refrigeracion liquida es mas caro
18     precio(): number {
19         let precio: number;
20         precio = super.precio();
21         if (this._refrigeracion == 'liquida') {
22             precio += 0.3 * precio;
23         }
24         return precio;
25     }
26
27     todo(){
28         let resultado: string
29         resultado = `${super.todo()}, Tipo de refrigeracion: ${this._refrigeracion}, Modelo de Caja: ${thi
30         return resultado
31     }
32 }
```

```
File Edit Selection View Go Run Terminal Help index.ts - GBDLI (Workspace) - Visual Studio Code

EXPLORER
GBDLI (WORKSPACE)
  dist
  doc
    teoria.pdf
  node_modules
  src
    clases

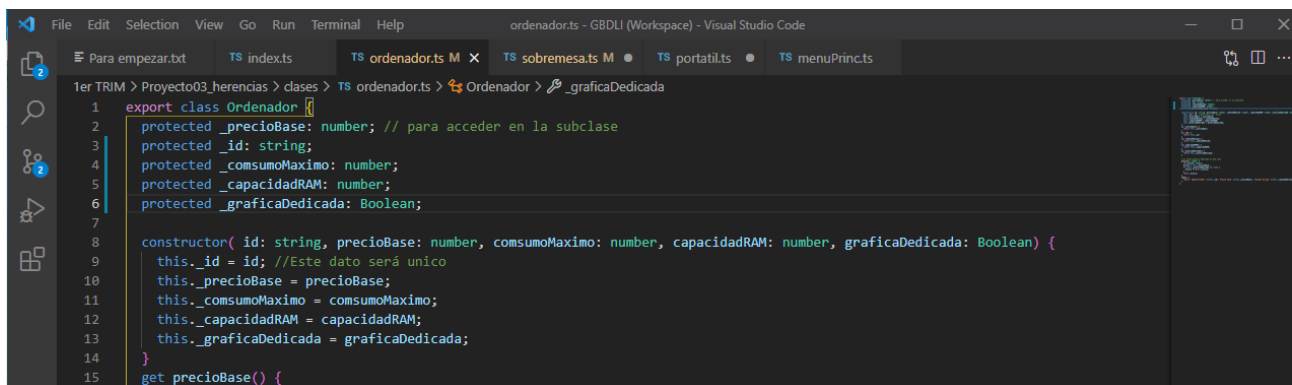
1er TRIM > Proyecto03_herencias > TS index.ts > ...
1 import { Ordenador } from './clases/ordenador';
2 import { Sobremesa } from './clases/sobremesa';
3 import { menuPrinc, tipos } from './view/menuPrinc';
4 import { leerTeclado } from './view/entradaTeclado';
5 import { Portatil } from './clases/portatil';
6
```

2) Herencias.

La herencia es otra característica fundamental de la programación orientada a objetos y TypeScript lo implementa.

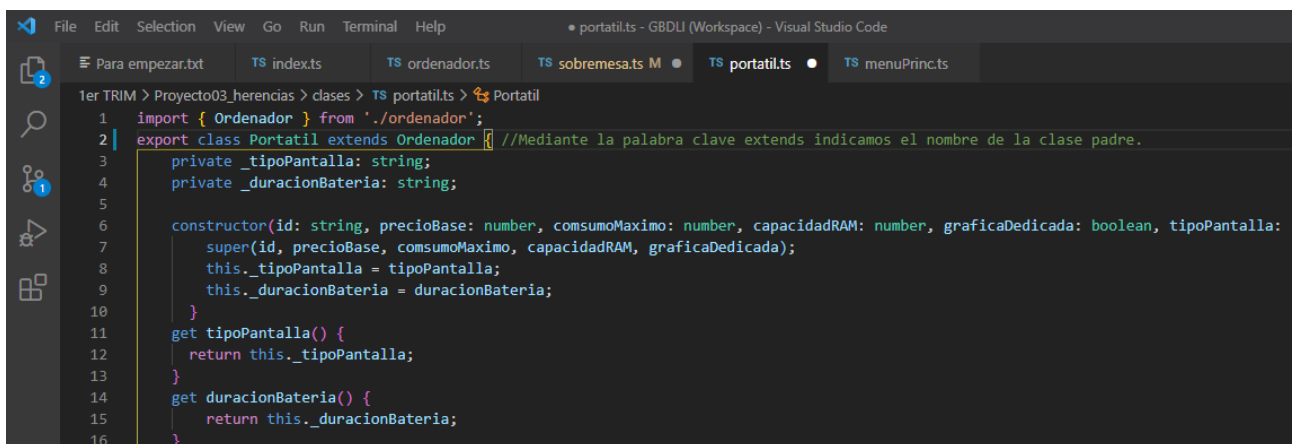
La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios

Un ejemplo sería este:



```
1  export class Ordenador {
2      protected _precioBase: number; // para acceder en la subclase
3      protected _id: string;
4      protected _consumoMaximo: number;
5      protected _capacidadRAM: number;
6      protected _graficaDedicada: Boolean;
7
8      constructor( id: string, precioBase: number, consumoMaximo: number, capacidadRAM: number, graficaDedicada: Boolean) {
9          this._id = id; //Este dato será unico
10         this._precioBase = precioBase;
11         this._consumoMaximo = consumoMaximo;
12         this._capacidadRAM = capacidadRAM;
13         this._graficaDedicada = graficaDedicada;
14     }
15     get precioBase() {
```

La subclase portatil puede acceder a las propiedades de la clase padre ordenador si los mismos se definieron en forma public o protected:



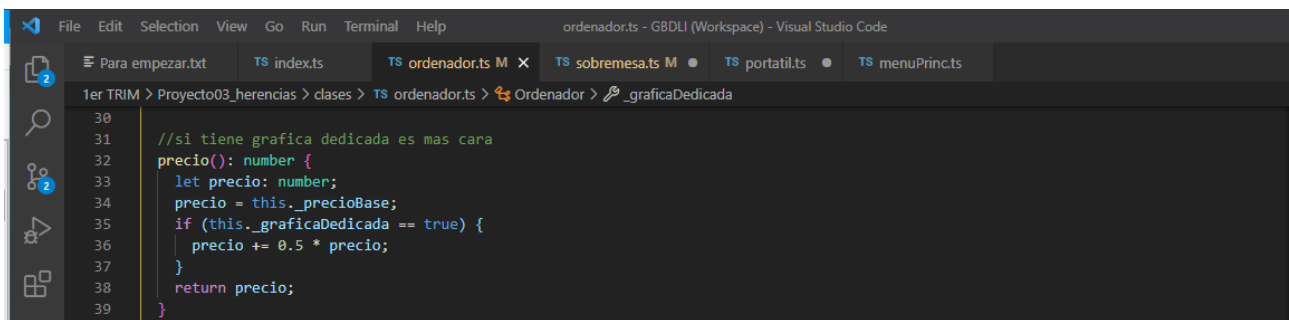
```
1  import { Ordenador } from './ordenador';
2  export class Portatil extends Ordenador { //Mediante la palabra clave extends indicamos el nombre de la clase padre.
3      private _tipoPantalla: string;
4      private _duracionBateria: string;
5
6      constructor(id: string, precioBase: number, consumoMaximo: number, capacidadRAM: number, graficaDedicada: boolean, tipoPantalla:
7          super(id, precioBase, consumoMaximo, capacidadRAM, graficaDedicada);
8          this._tipoPantalla = tipoPantalla;
9          this._duracionBateria = duracionBateria;
10     }
11     get tipoPantalla() {
12         return this._tipoPantalla;
13     }
14     get duracionBateria() {
15         return this._duracionBateria;
16     }
17 }
```

3) Sobreescritura de otros metodos.

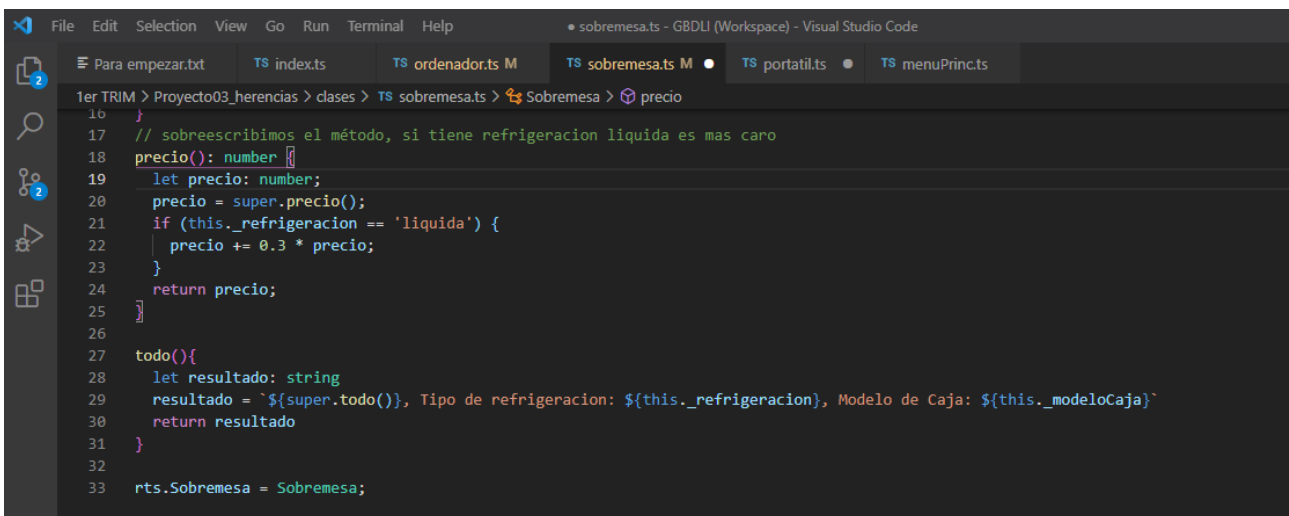
Una subclase hereda todos los métodos de su superclase que son accesibles a dicha subclase a menos que la subclase sobreescriba los métodos.

Una subclase sobreescribe un método de su superclase cuando define un método con las mismas características (nombre, número y tipo de argumentos) que el método de la superclase.

En este caso tenemos el metodo precio que aumenta un 50% el precio del ordenador si este tiene gráfica dedicada.



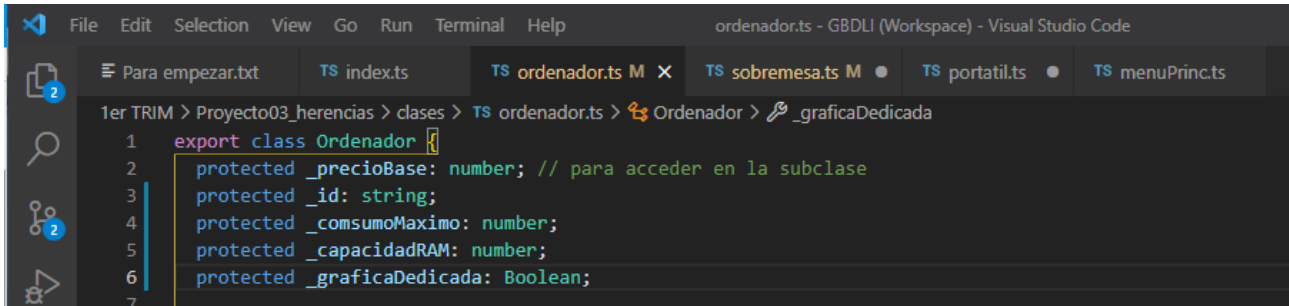
```
ordenador.ts - GBDLI (Workspace) - Visual Studio Code
1er TRIM > Proyecto03_herencias > clases > TS ordenador.ts > Ordenador > _graficaDedicada
30
31 //si tiene grafica dedicada es mas cara
32 precio(): number {
33     let precio: number;
34     precio = this._precioBase;
35     if (this._graficaDedicada == true) {
36         precio += 0.5 * precio;
37     }
38     return precio;
39 }
```



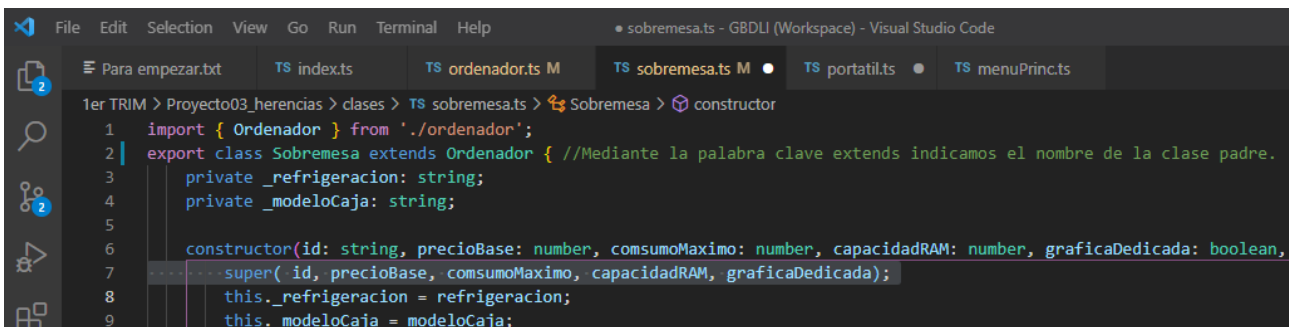
```
sobremesa.ts - GBDLI (Workspace) - Visual Studio Code
1er TRIM > Proyecto03_herencias > clases > TS sobremesa.ts > Sobremesa > precio
16 }
17 // sobreescribimos el método, si tiene refrigeracion liquida es mas caro
18 precio(): number {
19     let precio: number;
20     precio = super.precio();
21     if (this._refrigeracion == 'liquida') {
22         precio += 0.3 * precio;
23     }
24     return precio;
25 }
26
27 todo(){
28     let resultado: string
29     resultado = `${super.todo()}, Tipo de refrigeracion: ${this._refrigeracion}, Modelo de Caja: ${this._modeloCaja}`
30     return resultado
31 }
32
33 rts.Sobremesa = Sobremesa;
```

4) Uso del Super

Como en nuestro proyecto tenemos una clase padre y 2 subclases, si queremos añadir atributos nuevos a alguna subclase debemos incluir al inicio la palabra super para que haga referencia a los atributos de la clase padre.



```
1 export class Ordenador {
2   protected _precioBase: number; // para acceder en la subclase
3   protected _id: string;
4   protected _consumoMaximo: number;
5   protected _capacidadRAM: number;
6   protected _graficaDedicada: Boolean;
7 }
```



```
1 import { Ordenador } from './ordenador';
2 export class Sobremesa extends Ordenador { //Mediante la palabra clave extends indicamos el nombre de la clase padre.
3   private _refrigeracion: string;
4   private _modeloCaja: string;
5
6   constructor(id: string, precioBase: number, consumoMaximo: number, capacidadRAM: number, graficaDedicada: boolean,
7     ...super(id, precioBase, consumoMaximo, capacidadRAM, graficaDedicada);
8     this._refrigeracion = refrigeracion;
9     this._modeloCaja = modeloCaja;
```