

Clases y herencias en TypeScript

ÍNDICE

• Introducción.....pág 3.

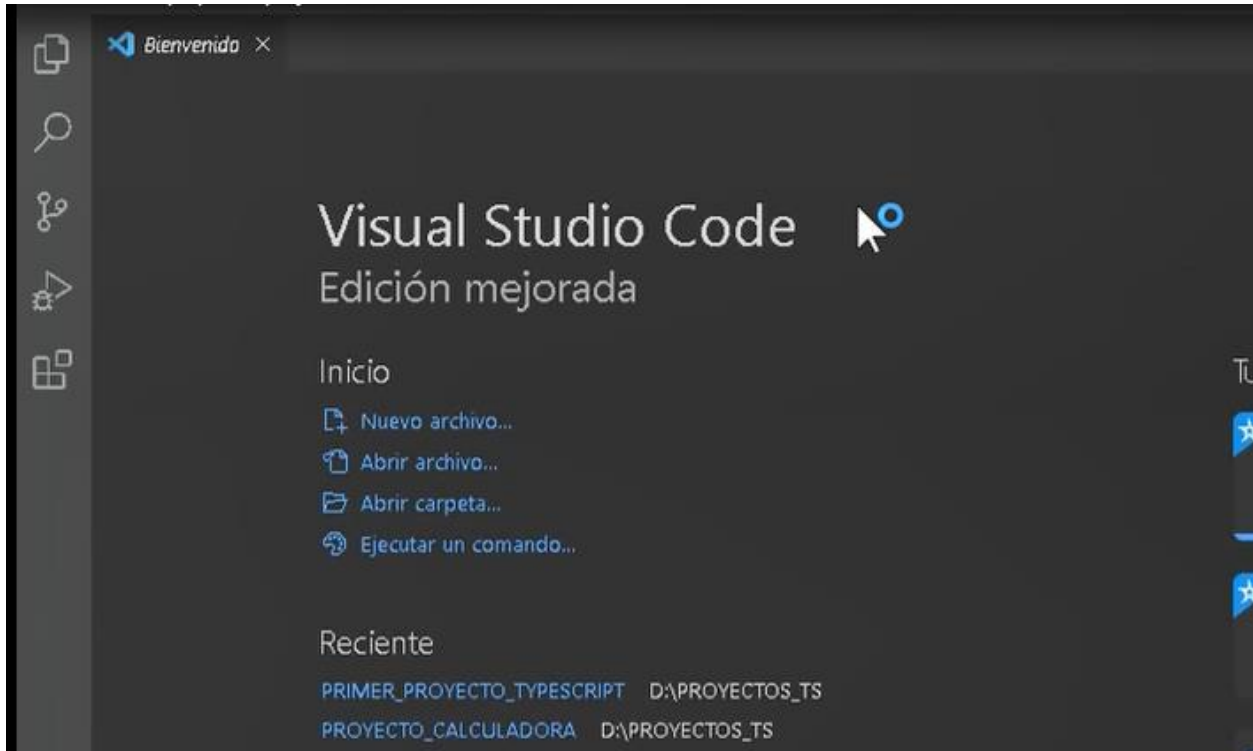
• Pasos.....pág 3

INTRODUCCIÓN

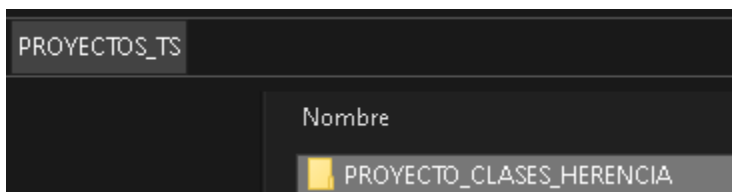
En este documento vamos a explicar, paso a paso, el montaje de un proyecto typescript en el cual utilizaremos clases, en concreto una superclase, de la cual se extenderán 2 clases más, vía herencia. Como aportación personal, también incluiremos un caso de polimorfismo y otro de sobreescritura de método (overwriting). Finalmente incluiremos un menú con distintas opciones.

PASOS

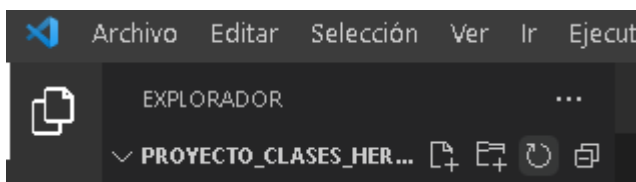
Con todo ello, procedemos a crear nuestro proyecto TypeScript. Abrimos VSC.



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  SALIDA  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\PROYECTO_CLASES_HERENCIA> npm init -y
```

Se nos crea el **package.json**



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

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

Y nos crea esto:



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

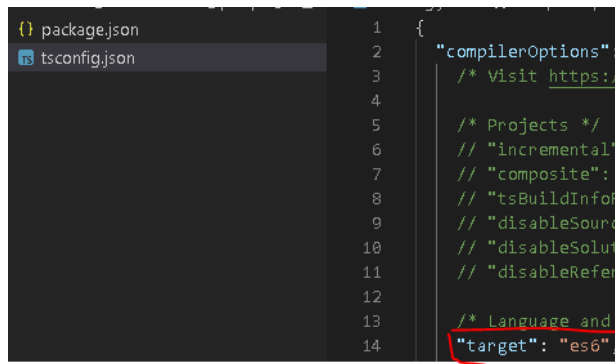
```
PS D:\PROYECTOS_TS\PROYECTO_CLASES_HERENCIA> tsc --init
```

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



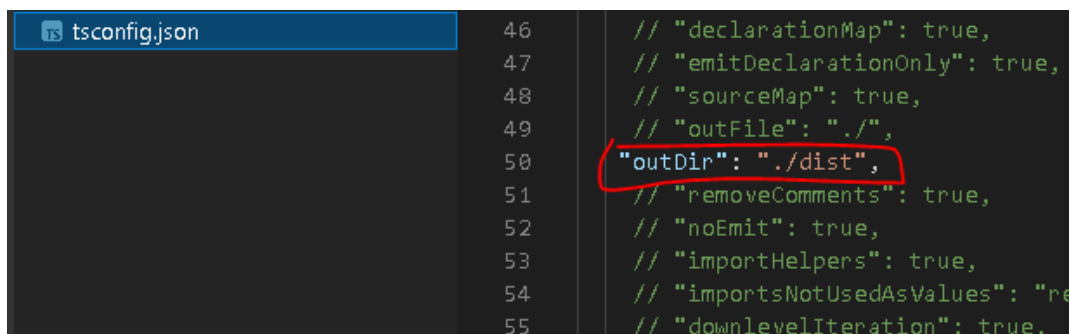
Vamos a dicho archivo para ver el contenido y hacemos los siguientes cambios:

Cambiamos "target:" a "es6"



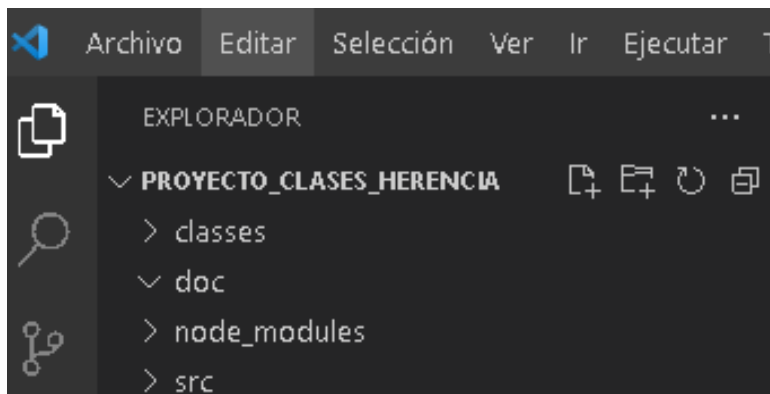
```
1 {
2   "compilerOptions": {
3     /* Visit https://
4
5     /* Projects */
6     // "incremental"
7     // "composite":
8     // "tsBuildInfoF
9     // "disableSourc
10    // "disableSolut
11    // "disableRefer
12
13    /* Language and
14    "target": "es6",
```

Cambiamos "outDir": "./dist",

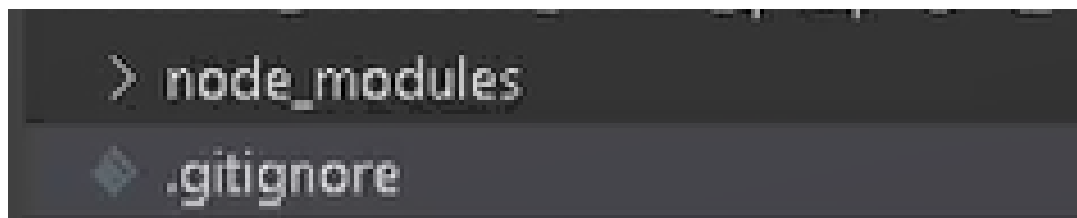


```
46 // "declarationMap": true,
47 // "emitDeclarationOnly": true,
48 // "sourceMap": true,
49 // "outFile": "./",
50 "outDir": "./dist",
51 // "removeComments": true,
52 // "noEmit": true,
53 // "importHelpers": true,
54 // "importsNotUsedAsValues": "re
55 // "downlevelIteration": true,
```

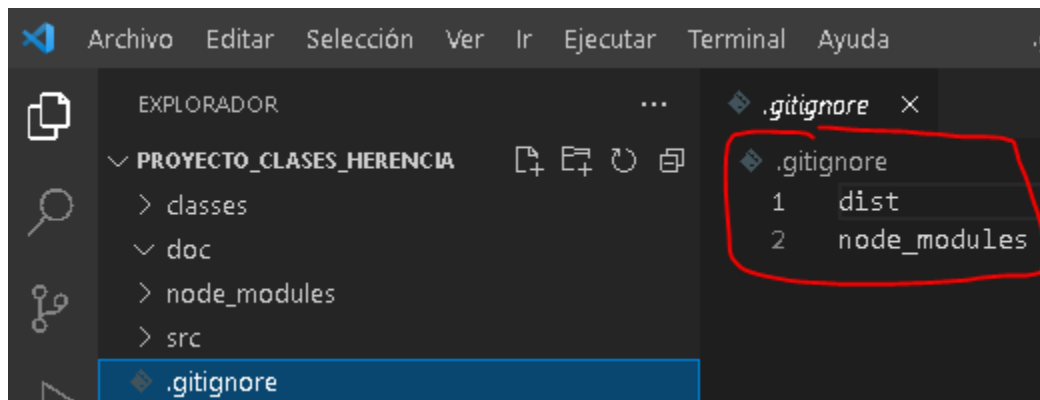
Creamos las carpetas **classes**, **doc** y **src**



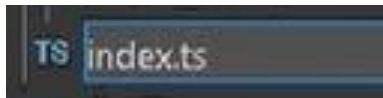
Creamos un archivo **.gitignore**



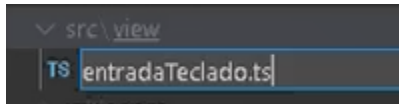
Y este es su contenido (para no subir a GitHub lo que ahí se especifica)



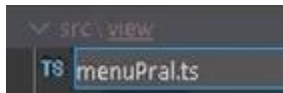
Dentro de carpeta **src** creamos el archivo **index.ts**



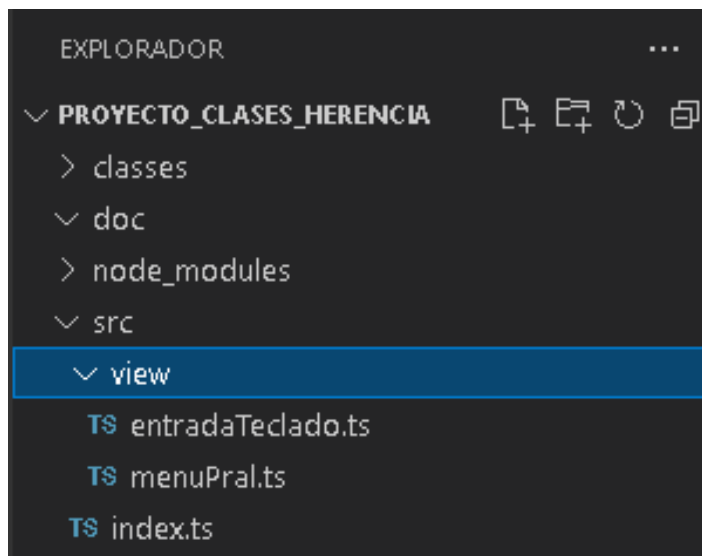
En **src** también creamos la carpeta **view** para poner dentro el archivo **entradaTeclado.ts**



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 > [x] 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 > [x] 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 nueva vivienda')
7      console.log('2.- Comparador de viviendas (PISO vs CASA vs CHALET)')
8      console.log('3.- Listar configuraciones de viviendas creadas')
9      console.log('4.- Modificar número de habitaciones')
10     console.log('5.- Ver los datos de una vivienda concreta')
11     console.log('6.- Borrar vivienda 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:

- **vivienda.ts** (que será la superclase)
- **piso.ts** (que estenderá de la superclase vía herencia)
- **chalet.ts** (que también estenderá de la superclase vía herencia)

Veamos cada uno por separado.

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

```
classes > TS vivienda.ts > 📁 Vivienda > 📄
1  export class Vivienda {
```

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

```
private _precioInicial: number;
```

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

```
get precioInicial() {
  return this._precioInicial;
}
```

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

```
constructor(precioInicial: number, cantidadHabitaciones: number) {
  this._precioInicial = precioInicial;
  this._cantidadHabitaciones = cantidadHabitaciones;
}
```

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

```
precio(): number {
  let precio: number;
  precio = this._precioInicial;
  if (this._cantidadHabitaciones > 4) {
    precio += 0.2 * precio;
  }
  return precio;
}
```

Y también **completo()**

```
completo() {
  return `Precio inicial: ${this._precioInicial}, número de habitaciones: ${this._cantidadHabitaciones}`;
}
```


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

```
TS piso.ts ×
classes > TS piso.ts > Piso
1 import { Vivienda } from './vivienda';
```

Como se ve en la imagen, antes exportamos **Vivienda**, pues ahora lo importamos en **piso.ts**

```
export class Piso extends Vivienda {
```

Y como se ve arriba, la clase **Piso** extiende de la superclase **Vivienda** vía herencia.

```
private _terrazza: string;
```

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

```
get terraza() {
  return this._terrazza;
}
```

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._terrazza == 'CERRAMIENTO') {
    precio += 0.1 * precio;
  }
  return precio;
}
```

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

```
completo(){
  let resultado: string;
  resultado = `${super.completo()}, terraza: ${this._terrazza}`;
  return resultado;
}
```

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

```
classes > TS chalets > ...
1  import { Vivienda } from './vivienda';
```

Como se ve en la imagen, antes exportamos **Vivienda**, pues ahora lo importamos en **chalet.ts**

```
export class Chalet extends Vivienda
```

Y como se ve arriba, la clase **Chalet** extiende de la superclase **Vivienda** vía herencia también.

```
private _zonaAjardinada: string;
```

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

```
get zonaAjardinada() {
  return this._zonaAjardinada
}
```

Y a continuación vemos otro caso de sobreescritura (overwriting) con **precio()** gracias a super

```
precio(): number {
  let precio: number;
  precio = super.precio();
  if (this._zonaAjardinada == 'LOSETAS') {
    precio += 0.3 * precio;
  }
  return precio
}
```

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

```
completo(){
  let resultado: string
  resultado = `${super.completo()}, zonaAjardinada: ${this._zonaAjardinada}`
  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 { Vivienda } from '../classes/vivienda';
4  import { Piso } from '../classes/piso';
5  import { Chalet } from '../classes/chalet';
6
```

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

```
const main = async () => {
  let n: number
  let n1: number
  let f1: string
  let j1: string
  let casa: Vivienda ;
  let apartamento: Piso;
  let casa_campo: Chalet;
```

Creamos **main** de forma asíncrona, y declaramos variables, como por ejemplos se ve arriba.

Usamos **switch** y **case** para ejecutar las distintas opciones del menú indicadas antes. Vemos cada una

Esta es la primera opción del menú:

```
n = await menuPral()
switch(n){
  case 1:
    console.log('Está en opción 1, PRECIO INICIAL DE UNA NUEVA VIVIENDA')
    seleccionado = await leerTeclado('Teclee 1 para CASA BÁSICA, 2 para PISO, 3 para CHALET')
    if (seleccionado == 1)
      {console.log('Ha elegido una CASA BÁSICA')
      casa = new Vivienda (75000, 2);
      console.log('Número habitaciones (NOTA: si es mayor de 5, 20% más), la cantidad base es: ${casa.cantidadHabitaciones}');
      console.log('Precio base (incluyendo las 2 habitaciones sin coste adicional): ${casa.precioInicial}');
      console.log('Precio final: ${casa.precio()}'); }
    else
      {if (seleccionado == 2)
        {console.log('Ha elegido un PISO')
        apartamento = new Piso (950000, 3, 'abierta');
        console.log('Número habitaciones (NOTA: si es mayor de 5, 20% más), la cantidad base es: ${apartamento.cantidadHabitaciones}');
        console.log('Terraza (NOTA: si es con cerramiento, 10 % más), la modalidad de terraza base es: ${apartamento.terraza}');
        console.log('Precio base PISO (incluyendo 3 habitaciones y terraza abierta sin coste adicional): ${apartamento.precioInicial}');
        console.log('Precio piso final: ${apartamento.precio()}');}
        else
        {console.log('Ha elegido un CHALET')
        casa_campo = new Chalet (2050000, 4, 'cesped');
        console.log('Número habitaciones (NOTA: si es mayor de 5, 20% más), la cantidad base es: ${casa_campo.cantidadHabitaciones}');
        console.log('Jardín (NOTA: si es con losetas, 30 % más), la modalidad de jardín base es: ${casa_campo.zonaAjardinada}');
        console.log('Precio base CHALET (incluyendo 4 habitaciones y jardín de césped sin coste adicional): ${casa_campo.precioInicial}');
        console.log('Precio piso final: ${casa_campo.precio()}');}
      }
    break
```

Esta es la segunda opción del menú: en donde usamos un **array**

Lo creamos así:

```
let viviendas: Array<Vivienda> = new Array<Vivienda>();
```

Y lo recorremos con un FOR, como se ve en las siguientes líneas de código

```
case 2:
  console.log(`Está en opción 2, COMPARADOR DE VIVIENDAS (PISO vs CASA vs CHALET)`)
  n1 = parseInt( await leerTeclado('Indicar el número de habitaciones deseadas'))
  f1 = await leerTeclado('Escriba si desea terraza ABIERTA o CERRAMIENTO (Solo en caso de PISOS)')
  j1 = await leerTeclado('Escriba si desea jardín con CÉSPED o LOSETAS (Solo en caso de CHALETS)')

  // Podemos asignar tanto Viviendas como Pisos y Chalets
  viviendas[0] = new Piso (95000, n1, f1);
  viviendas[1] = new Vivienda (75000, n1);
  viviendas[2] = new Chalet (205000, n1, j1);
  // recorremos el array
  // Esto es un caso de polimorfismo, pues la variable a puede ser de cualquiera de los tipos
  for (let a of viviendas) {
    if (a instanceof Piso == true) {
      //En caso de PISO
      console.log(`${a.completo()}`, precio final: `${a.precio()}`);}
    else{
      if (a instanceof Piso == false) {
        //En caso de CASA
        console.log(`${a.completo()}`, precio final: `${a.precio()}`);}
      else{
        //En caso de CHALET
        console.log(`${a.completo()}`, precio final: `${a.precio()}`);}}}
  }
  break
```

Ampliamos una de las zonas del código, para ver un caso de **polimorfismo**, ya que la variable “a” puede ser de cualquiera de los tipos.

```
// Esto es un caso de polimorfismo, pues la variable a puede ser de cualquiera de los tipos
for (let a of viviendas) {
  if (a instanceof Piso == true) {
    //En caso de PISO
    console.log(`${a.completo()}`, precio final: `${a.precio()}`);}
  else{
    if (a instanceof Piso == false) {
      //En caso de CASA
      console.log(`${a.completo()}`, precio final: `${a.precio()}`);}
    else{
      //En caso de CHALET
      console.log(`${a.completo()}`, precio final: `${a.precio()}`);}}}
}
```

Esta es la tercera opción del menú, donde hacemos una lista de todos los elementos del **array**

```
case 3:
  console.log(`Está en opción 3, LISTAR CONFIGURACIONES DE VIVIENDAS CREADAS`)
  console.log(`Lista de cada vivienda`)
  console.log(viviendas);

  break
```

Esta es la cuarta opción del menú, donde modificamos un elemento del **array**, como pide el proyecto

```
case 4:
  console.log(`Está en opción 4, MODIFICADOR DE NÚMERO DE HABITACIONES`)
  n1 = parseInt( await leerTeclado('Indicar el nuevo número de habitaciones'))
  console.log(`En caso de PISO`);
  viviendas[0]._cantidadHabitaciones = n1
  console.log(`${viviendas[0].completo()}, precio final: ${viviendas[0].precio()}`);
  console.log(`En caso de CASA`);
  viviendas[1]._cantidadHabitaciones = n1
  console.log(`${viviendas[1].completo()}, precio final: ${viviendas[1].precio()}`);
  console.log(`En caso de CHALET`);
  viviendas[2]._cantidadHabitaciones = n1
  console.log(`${viviendas[2].completo()}, precio final: ${viviendas[2].precio()}`);
break
```

Esta es la quinta opción del menú, donde se ven los datos de un elemento concreto del **array**.

```
case 5:
  console.log(`Está en opción 5, VER DATOS DE VIVIENDA CONCRETA`)
  let datos_piso = viviendas[0];
  let datos_casa = viviendas[1];
  let datos_chalet = viviendas[2];
  seleccionado = await leerTeclado('Teclee 1 para ver datos de PISO, 2 para CASA o 3 para CHALET')
  if (seleccionado == 1)
  {
    console.log(`Abajo están todos los datos del PISO que ha configurado`)
    console.log(`${datos_piso.completo()}, precio final: ${datos_piso.precio()}`);
  }
  else
  {
    if (seleccionado == 2)
    {
      console.log(`Abajo están todos los datos de la CASA que ha configurado`)
      console.log(`${datos_casa.completo()}, precio final: ${datos_casa.precio()}`);
    }
    else
    {
      console.log(`Abajo están todos los datos del CHALET que ha configurado`)
      console.log(`${datos_chalet.completo()}, precio final: ${datos_chalet.precio()}`);
    }
  }
break
```

Esta es la sexta opción del menú, donde borramos un elemento del **array**, como se nos pide.

```
case 6:
  console.log(`Está en opción 6, BORRAR VIVIENDA DE LA LISTA CREADA`)
  seleccionado = await leerTeclado('Teclee 1 para borrar PISO, 2 para borrar CASA o 3 para CHALET')

  if (seleccionado == 1)
  {
    console.log(`Ha elegido borrar el PISO`)
    viviendas.splice( 0, 1);
  }
  else
  {
    if (seleccionado == 2)
    {
      console.log(`Ha elegido borrar la CASA`)
      viviendas.splice( 1, 2);
    }
    else
    {
      console.log(`Ha elegido borrar el CHALET`)
      viviendas.splice( 2, 3);
    }
  }
break
```

Procedemos a demostrar ahora que todo funciona.

Ahora se puede tener dos terminales abiertos.

En uno escribimos: **tsc -w**(para que compile ante cambios, es decir, en “watchmode”)

```

PROBLEMAS  SALIDA  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\PROYECTO_CLASES_HERENCIA> tsc -w

```

Y nos sale esto:

```

PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

[20:01:36] Starting compilation in watch mode...

[20:01:37] Found 0 errors. Watching for file changes.

```

En otro terminal invocamos el ejecutable JS que está en la carpeta **dist/src** usando **node dist**

```

PROBLEMAS  SALIDA  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\PROYECTO_CLASES_HERENCIA> node dist/src

```

Y aparece el menú, tal y como esperábamos.

```

PS D:\PROYECTOS_TS\PROYECTO_CLASES_HERENCIA> node dist/src

1.- Precio base de nueva vivienda
2.- Comparador de viviendas (PISO vs CASA vs CHALET)
3.- Listar configuraciones de viviendas creadas
4.- Modificar número de habitaciones
5.- Ver los datos de una vivienda concreta
6.- Borrar vivienda del la lista creada
0.- Salir
opción :

```

Usamos la **opción 1**, y todas las subfunciones operan correctamente, eso demuestra que la superclase y las 2 subclases que se extienden de ella operan en armonía vía **herencia**.

```
opción : 1
Está en opción 1, PRECIO INICIAL DE UNA NUEVA VIVIENDA
Teclee 1 para CASA BÁSICA, 2 para PISO, 3 para CHALET: 1
Ha elegido una CASA BÁSICA
Número habitaciones (NOTA: si es mayor de 5, 20% más), la cantidad base es: 2
Precio base (incluyendo las 2 habitaciones sin coste adicional): 75000
Precio final: 75000
```

```
opción : 1
Está en opción 1, PRECIO INICIAL DE UNA NUEVA VIVIENDA
Teclee 1 para CASA BÁSICA, 2 para PISO, 3 para CHALET: 2
Ha elegido un PISO
Número habitaciones (NOTA: si es mayor de 5, 20% más), la cantidad base es: 3
Terraza (NOTA: si es con cerramiento, 10 % más), la modalidad de terraza base es: abierta
Precio base PISO (incluyendo 3 habitaciones y terraza abierta sin coste adicional): 950000
Precio piso final: 950000
```

```
opción : 1
Está en opción 1, PRECIO INICIAL DE UNA NUEVA VIVIENDA
Teclee 1 para CASA BÁSICA, 2 para PISO, 3 para CHALET: 3
Ha elegido un CHALET
Número habitaciones (NOTA: si es mayor de 5, 20% más), la cantidad base es: 4
Jardín (NOTA: si es con losetas, 30 % más), la modalidad de jardín base es: cesped
Precio base CHALET (incluyendo 4 habitaciones y jardín de césped sin coste adicional): 2050000
Precio piso final: 2050000
```

Usamos la **opción 2**, y funciona, demostrando que los cálculos de incremento en los pisos (por el tipo de terraza) y el los chalets, (por el tipo de zona ajardinada) operan correctamente, y que el array está siendo recorrido perfectamente por el FOR, dándose un caso de **polimorfismo** ya que solo estamos usando una única variable "a"

```
opción : 2
Está en opción 2, COMPARADOR DE VIVIENDAS (PISO vs CASA vs CHALET)
Indicar el número de habitaciones deseadas: 4
Escriba si desea terraza ABIERTA o CERRAMIENTO (Solo en caso de PISOS): CERRAMIENTO
Escriba si desea jardín con CÉSPED o LOSETAS (Solo en caso de CHALETs): LOSETAS
Precio inicial: 95000, número de habitaciones: 4, terraza: CERRAMIENTO, precio final: 104500
Precio inicial: 75000, número de habitaciones: 4, precio final: 75000
Precio inicial: 205000, número de habitaciones: 4, zonaAjardinada: LOSETAS, precio final: 266500
```

Usamos la **opción 3**, y funciona, ya que lista el contenido del array, como se nos pide.

```
opción : 3
Está en opción 3, LISTAR CONFIGURACIONES DE VIVIENDAS CREADAS
Lista de cada vivienda
[
  Piso {
    _precioInicial: 95000,
    _cantidadHabitaciones: 4,
    _terrazza: 'CERRAMIENTO'
  },
  Vivienda { _precioInicial: 75000, _cantidadHabitaciones: 4 },
  Chalet {
    _precioInicial: 205000,
    _cantidadHabitaciones: 4,
    _zonaAjardinada: 'LOSETAS'
  }
]
```

Usamos la **opción 4**, y funciona, demostrando que los cálculos por el cambio de número de habitaciones (al pasar de 5) funcionan.

```
opción : 4
Está en opción 4, MODIFICADOR DE NÚMERO DE HABITACIONES
Indicar el nuevo número de habitaciones: 7
En caso de PISO
Precio inicial: 95000, número de habitaciones: 7, terraza: CERRAMIENTO, precio final: 125400
En caso de CASA
Precio inicial: 75000, número de habitaciones: 7, precio final: 90000
En caso de CHALET
Precio inicial: 205000, número de habitaciones: 7, zonaAjardinada: LOSETAS, precio final: 319800
```

Además en esta imagen de abajo se comprueba que efectivamente se ha modificado con éxito un elemento del array, pasando de 4 a 7 habitaciones, y el resto ha permanecido con los valores correspondientes, no se han perdido.

```
Lista de cada vivienda
[
  Piso {
    _precioInicial: 95000,
    _cantidadHabitaciones: 7,
    _terrazza: 'CERRAMIENTO'
  },
  Vivienda { _precioInicial: 75000, _cantidadHabitaciones: 7 },
  Chalet {
    _precioInicial: 205000,
    _cantidadHabitaciones: 7,
    _zonaAjardinada: 'LOSETAS'
  }
]
```

Usamos **opción 5**, y funciona, demostrando que podemos ver un objeto en concreto del array

```
opción : 5
Está en opción 5, VER DATOS DE VIVIENDA CONCRETA
Teclee 1 para ver datos de PISO, 2 para CASA o 3 para CHALET: 3
Abajo están todos los datos del CHALET que ha configurado
Precio inicial: 205000, número de habitaciones: 7, zonaAjardinada: LOSETAS, precio final: 319800
```

Usamos **opción 6**, y funciona

```
opción : 6
Está en opción 6, BORRAR VIVIENDA DE LA LISTA CREADA
Teclee 1 para borrar PISO, 2 para borrar CASA o 3 para CHALET: 3
Ha elegido borrar el CHALET
```

Y como se ve abajo, el chalet ha desaparecido de la lista del array

```
Lista de cada vivienda
[
  Piso {
    _precioInicial: 95000,
    _cantidadHabitaciones: 7,
    _terrazza: 'CERRAMIENTO'
  },
  Vivienda { _precioInicial: 75000, _cantidadHabitaciones: 7 }
]
```

Por tanto, nuestro proyecto ha concluido con éxito.