

TypeScript incorpora muchas características de la programación orientada a objetos disponibles en lenguajes como Java y C#.

Sintaxis básica de una clase:

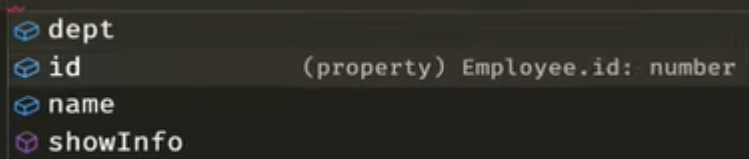
```
class Employee {
  // Atributos
  id: number;
  name: string;
  dept: string;

  constructor(id: number, name: string, dept: string) {
    this.id = id;
    this.name = name;
    this.dept = dept;
    this.showInfo();
  }

  // Métodos
  showInfo(): void {
    console.log(`${this.name}`);
  }
}
```

Ejemplo creación de una **instancia** de la clase:

```
const emp = new Employee(1, 'Dominicode', 'Sales');
emp.
```



observa que la instancia **emp** tiene acceso a las tres propiedades y al método. Recuerda que la visibilidad de las propiedades y métodos la podemos controlar...

Control de acceso

Los miembros de una clase pueden ser *public*, *private* o *protected*. Si no especificamos nada como en el ejemplo anterior, el control de acceso es *public*. A continuación puedes ver el efecto de declarar los miembros de la clase como *private*:

```
class Employee{
    // atributos
    private id:number;
    private name: string;
    private dept: string;

    constructor(id:number, name:string, dept:string){
        this.id=id;
        this.name=name;
        this.dept=dept;
    }

    //métodos
    private showInfo():void{console.log`${this.name} ${this.dept}`}
}

let emp = new Employee(1, "Perico", "ventas")
emp.
```

Observa que ahora la instancia **emp** no tiene acceso a ninguna de las propiedades ni al método definido (son **private** y el editor no las ofrece...).

Automatic properties en TypeScript (constructores abreviados)

Las propiedades automáticas de TypeScript permiten refactorizar el código anterior y escribirlo como se muestra a continuación:

```
class Employee{

    constructor(private id:number, private name:string, private dept:string){

    }

    //Métodos
    private showInfo():void{console.log`${this.name} ${this.dept}`}
}

let emp = new Employee(1, "Perico", "ventas");
```

El resultado es el mismo: cuando se declara cada una de las propiedades en el constructor typescript hace automáticamente la asignación quedando el código más limpio.

Extender una clase en TypeScript

Las clases pueden “extender” una clase base. Una clase derivada tiene todas las propiedades y métodos de su clase base y también define miembros adicionales.

Ejemplo:

```
class Person{
  constructor(){
  }
  greet(): void{
    console.log("Hello!!!");
  }
}

class Employee extends Person{

  constructor(private id:number, private name:string, private dept:string){
    super();
  }

  //Métodos
  private showInfo():void{console.log` ${this.name} ${this.dept}`}
}

let emp = new Employee(1, "Perico", "ventas");
emp.
```

Ahora la instancia **emp** tiene acceso al método **greet()** - método público de la clase **Person** (recuerda que por defecto es público). *Person* no tiene propiedades definidas y las propiedades definidas para la clase *Employee* son privadas, por eso lo único que nos ofrece el editor es el método *greet()*.

```
let emp = new Employee(1, "Perico", "ventas");
emp.
  greet (method) Person.greet(): void
```

Ahora que estamos extendiendo una clase, ha llegado el momento de ver el efecto de una propiedad **protected...**

Supongamos las siguientes definiciones:

```
class Person{
    public zona="Caribe";
    protected city="Santo Domingo";
    private country="R.D";
    constructor(){
    }
    saludar(): string{
        return "Hello!!!" + this.city + this.country +this.zona;
    }
}

class Employee extends Person{
    constructor(private id:number, private name:string, private dept:string){
        super();
    }

    //Métodos
    private showInfo():string{
        return `info del empleado: ${this.name} ${this.dept} ${this.`
    }
}
```

- city
- dept
- id
- name
- saludar
- showInfo
- zona

Lógicamente **dentro** de la clase **Person** puedo acceder a las tres propiedades (*observa que el método saludar() hace referencia a las tres.*

Pero ¿qué ocurre en la clase **Employee** que extiende la clase **Person**?

- Tengo acceso a **zona** sin problema (*public*)
- A la clase **city** (*protected*) también porque hemos extendido. Con el control de acceso *protected* estamos dando acceso a la clase que hereda (*a las clases que utilicen la clase padre*)
- A **country** (*private*) **no** tengo acceso, porque es una propiedad privada y su scope está limitado a la clase **Person** (*recuerda: cuando extendemos, los miembros **private** no quedan expuestos*)

Conclusión: la subclase Employee puede acceder a los miembros de la clase padre Person si los mismos se definieron en forma **public** o **protected**.

TS-Tarea3-clases.ts

Revisa la documentación oficial referente a [clases](#). Trabaja los conceptos, que seguro ya conoces a través de otros lenguajes de programación, y prepara una demo **completa** que ponga en práctica los conceptos repasados:

- crea una primera clase y una segunda que la extienda
- muestra por pantalla la información que permita verificar la herencia en función del control de acceso establecido a los miembros
- realiza tantos ejemplos como precisas para verificar el control de acceso a los miembros