

TS

TypeScript

4. Classes

1. Introducción

- Las clases son una parte fundamental de la programación orientada a objetos.
- Se utilizan clases para representar cualquier entidad que tenga algunas propiedades y funciones que puedan actuar sobre determinadas propiedades.
- TypeScript te brinda control total sobre las propiedades y funciones a las que se puede acceder dentro y fuera de su propia clase contenedora

2. Creando la primera clase

- Ejemplo muy básico de crear una clase **Persona.ts**

```
class Persona {  
    nombre: string;  
  
    constructor(elNombre: string) {  
        this.nombre = elNombre;  
    }  
  
    presentar() {  
        console.log("Hola, me llamo " + this.nombre + "!");  
    }  
}  
  
let personA = new Persona("Sally");  
  
personA.presentar();
```

2. Creando la primera clase

- El código anterior crea una clase muy simple llamada **Persona**.
- Esta clase tiene una propiedad llamada **nombre** y una función llamada **presentar**.
- La clase también tiene un **constructor**, que es básicamente una función.

2. Creando la primera clase

- Ejecutado el comando **tsc Persona.ts** obtendremos el fichero correspondiente en JS.

```
class Persona {  
    nombre: string;  
  
    constructor(elNombre: string) {  
        this.nombre = elNombre;  
    }  
  
    presentar() {  
        console.log("Hola, me llamo " + this.nombre + "!");  
    }  
}  
  
let personA = new Persona("Sally");  
  
personA.presentar();
```

3. Modificadores privados y públicos

- Todos los miembros de una clase en TypeScript son de tipo **public** por defecto.
- Podemos especificar explícitamente que una propiedad es pública o que un método es público agregando la palabra clave **public** antes de la propiedad o el método.
- A veces, podemos bloquear el acceso a una propiedad o a un método fuera de la clase que lo contiene. Esto se puede lograr haciendo que esos miembros sean privados utilizando la palabra clave **private**.

3. Modificadores privados y públicos

```
class Persona {  
  private nombre: string;
```

(property) Persona.nombre: string

La propiedad 'nombre' es privada y solo se puede acceder a ella en la clase 'Persona'. ts(2341)

[Ver el problema](#) No hay correcciones rápidas disponibles

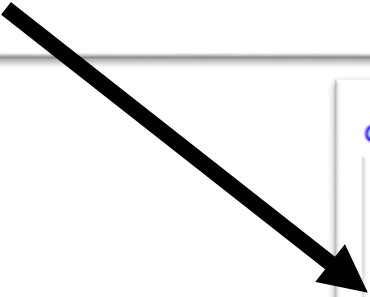
```
let pers  
personA.nombre="Yo";
```

4. Herencia en TypeScript


- La herencia te permite crear clases más complejas a partir de una clase base.
- Por ejemplo, podemos usar la clase Persona de la sección anterior como base para crear una clase Amigo que tendrá todos los miembros de Persona y agregará algunos miembros propios.

4. Herencia en TypeScript

```
class Persona {  
    private nombre: string;  
    constructor(elNombre: string) {  
        this.nombre = elNombre;  
    }  
    presentar() {  
        console.log("Hola, me llamo " + this.nombre + "!");  
    }  
}
```



```
class Amigo extends Persona {  
    anyosamigo: number;  
    constructor(nombre: string, anyosamigo: number) {  
        super(nombre);  
        this.anyosamigo = anyosamigo;  
    }  
    tiempoAmistad() {  
        console.log("Somos amigos desde hace " + this.anyosamigo + " años.");  
    }  
}
```



```
let amig0 = new Amigo("Jacobo", 6);  
amig0.presentar();  
amig0.tiempoAmistad();
```

5. Protected

- Hasta este punto, solo hemos hecho que los miembros de una clase sean `private` o `public`
- Aparece un tercer modificador **protected** que limita el acceso de un miembro solo a clases derivadas.
- También puedes usar la palabra clave **protected** con el **constructor** de una clase base. Esto evitará que alguien cree una instancia de esa clase. Sin embargo, aún podrás extender clases basadas en esta clase base.

5. Protected

```
class Persona {  
  private nombre: string;  
  protected edad: number;  
  protected constructor(elNombre: string, laEdad: number) {  
    this.nombre = elNombre;  
    this.edad = laEdad;  
  }  
  presentarse() {  
    console.log("Hola, Me llamo " + this.nombre + "!");  
  }  
}
```

```
class Amigo extends Persona {  
  anyosConocido: number;  
  constructor(name: string, age: number, yearsKnown: number) {  
    super(name, age);  
    this.anyosConocido = yearsKnown;  
  }  
  tiempoConocido() {  
    console.log("Somos amigos desde hace " + this.anyosConocido + " años.")  
  }  
  amigosDesde() {  
    let firstAge = this.edad - this.anyosConocido;  
    console.log(`Somos amigos desde que tenía ${firstAge} años.`)  
  }  
}
```

```
let friendA = new Amigo("Guillermo", 19, 8);  
friendA.amigosDesde();|
```