Objetivos

- qué es una interface en TypeScript
- crear una interface
- usar nuestra interface en una variable
- usar la interface en una función
- propiedades y métodos opcionales en una interface
- extender interfaces en TypeScript
- implementar interfaces en TypeScript

Previo: Interfaces vs Clases en TypeScript

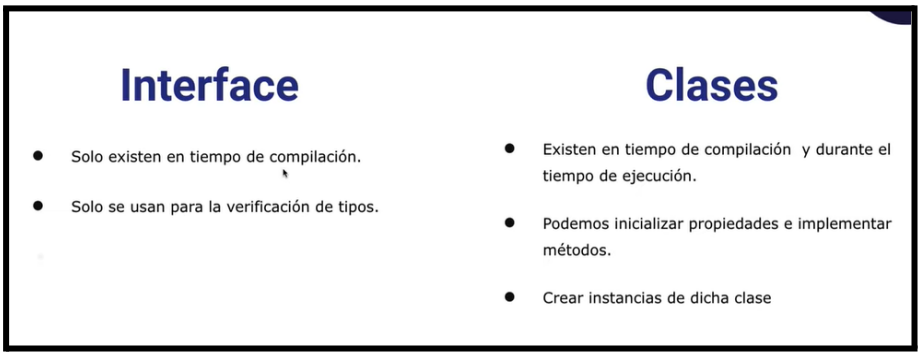
Las **clases** y las **interfaces** son estructuras poderosas que facilitan no solo la programación orientada a objetos sino también la comprobación de tipos en TypeScript.

Una clase es un plano a partir del cual podemos crear objetos que comparten la misma configuración: propiedades y métodos.

Una interfaz es un grupo de propiedades y métodos relacionados que describen un objeto, pero no proporciona implementación ni inicialización para ellos.

La principal diferencia entre estos dos conceptos en TypeScript reside en que las interfaces solo existen en tiempo de desarrollo, mientras que las clases existen en tiempo de compilación y ejecución, y con ellas podemos inicializar propiedades, implementar métodos y crear instancias de la clase.

Cuando nuestro código es transpilado a javascript las interfaces no existen, es decir las interfaces no se convierten a JavaScript. Su impacto en tiempo de ejecución es cero. Solamente ***las usamos para verificación de tipos***.



Lo recomendable es que empecemos a trabajar con una interface y si vemos que necesitamos por ejemplo inicializar una propiedad, entonces pasemos a trabajar con una clase.

¿Qué es una interface?
Básicamente una interfaz es un “contrato de código" que define la forma de los datos con los que vamos a trabajar, podemos decir que es un *contrato sintáctico* que debe cumplir una entidad.

Ejemplo de definición de una interface con 4 propiedades y un método *(recuerda que tanto las propiedades como los métodos pueden ser requeridos u opcionales (?))*: la propiedad ***coAuthor*** es opcional, el método ***isLoan()*** espera un number y no devuelve nada *(es un **void**; recuerda que esta es la manera de tipar algo que no devuelve nada).*

```ts
interface Book {
  id: number;
  title: string;
  author: string;
  coAuthor?: string;
  isLoan: (id: number) => void
}
```

```ts
interface Book {
  id: number;
  title: string;
  author: string;
  coAuthor?: string;
  isLoan: (id: number) => void
}
```

El siguiente ejemplo muestra ***book*** definido como un objeto vacío (no sujeto a la interfaz ***Book***)

```ts
interface Book {
  id: number;
  title: string;
  author: string;
}

const book = {};
```

Si al objeto ***book*** le asignamos el tipo ***Book*** el editor "se queja" y nos avisa del error porque no estamos cumpliendo el contrato de la interfaz:

```ts
interface Book {
  id:
  titl
  auth
}
Type '{}' is missing the following properties from type 'Book': id, title, author ts(2739)
const book: Book
View Problem  Quick Fix... (⌘.)
const book: Book = {};
```

El siguiente ejemplo muestra:

- la definición de ***book*** sujeta al contrato de la interfaz ***Book***
- la definición de ***books*** que espera un array de libros (Book) y
- la de definición de una función ***getBook()*** sin tipar que devuelve un libro (observa que el objeto que devuelve la función no tiene el atributo author y no hay error…)

```ts
src > interfaces.ts > getBook
interface Book {
  id: number;
  title: string;
  author: string;
}

const book: Book = {
  id: 1,
  title: 'My title',
  author: 'Dominicode'
};

const books: Book[] = [];

function getBook() {
  return { id: 1, title: 'My title' };
}
```

Si tipamos la función ***getBook()*** el objeto a retornar debe definirse acorde a la interfaz ***Book***:

```ts
interface Book {
  id: number;
  title: string;
  author: string;
}

const book: Book = {
  id: 1,
  title: 'My title',
  author: 'Dominicode'
};

const books: Book[] = [];

function getBook(): Book {
  return { id: 1, title: 'My title', author: 'Bezael' };
}
```

Extendiendo interfaces en TypeScript

Podemos definir en nuestro código tantas interfaces como precisemos. Por ejemplo:

```ts
// Interfaces

interface Person {
  id: number;
  name: string;
}

interface Employee {
  id: number;
  name: string;
  dept: string;
}

interface Customer {
  id: number;
  name: string;
  country: string;
}
```

observa que se repiten dos propiedades en las tres interfaces definidas

Refactorizando el código anterior…

src > interface_extends_implements.ts > Person

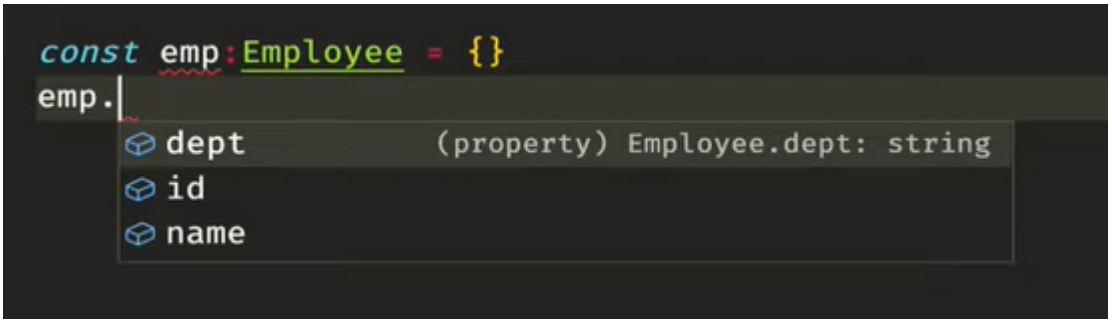
```ts
// Interfaces

interface Person {
  id: number;
  name: string;
}

interface Employee extends Person{
  dept: string;
}

interface Customer extends Person{
  country: string;
}
```

se ha reutilizado la interfaz base y se heredan las propiedades (la siguiente imagen muestra cómo el editor las ofrece)



Recuerda: serán muchas las ocasiones y escenarios en las que te encontrarás con que puedes reutilizar una base y extenderla.

Implementando interfaces en TypeScript

Cuando trabajamos con interfaces, también podemos hacer una implementación en una clase. Nos referimos a:

La propiedad name y el primero de los métodos son requeridos. El segundo método de la interface es opcional. ***Implements*** es útil cuando queremos "obligar" a una persona que está implementando nuestra interfaz a que tenga, por ejemplo, ciertos métodos.

Tarea - Ayudándote de los ejemplos y de la documentación oficial de TypeScript relativa a las interfaces:

- define una interface *(determina tú sus propiedades y métodos)*
- haz una demo del uso de la interface en una variable (crea un objeto a partir de la interfaz)
- ídem del uso de la interface en una función
- extiende la interface
- define una clase que implemente la interfaz