Interfaces en TypeScript

1 La interface es una estructura que define el contrato en su aplicación. Define la sintaxis que seguirán las clases. Las clases que se derivan de una interface deben seguir la estructura proporcionada por su interface.

El compilador de TypeScript no convierte la interface a JavaScript. Utiliza una interface para la verificación de tipos. Esto también se conoce como "tipificación pato" o "subtipificación estructural".

Una interface se define con la palabra clave interface y puede incluir propiedades y declaraciones de métodos usando una función o una función de flecha .

En el ejemplo anterior, la IEmployee incluye dos propiedades empCodey empName. También incluye una declaración de método getSalaray utilizando una función de flecha que incluye un parámetro numérico y un tipo de retorno numérico. El getManagerName se declara mediante una función normal. Esto significa que cualquier objeto de tipo IEmployee debe definir dos propiedades y dos métodos.

```
interface IEmployee {
  empCode: number;
  empName: string;
  getSalary: (n:number) => number; // arrow function
  getManagerName(y:number): string;
}
```

La interface en TypeScript se puede utilizar para definir un tipo y también para implementarlo en la clase.

La siguiente interfaz IEmployeedefine un tipo de variable.

```
interface KeyPair {
  key: number;
  value: string;
}

let kv1: KeyPair = { key:1, value:"Steve" }; // OK

let kv2: KeyPair = { key:1, val:"Steve" }; // Compiler Error: 'val' doesn't exist in type
'KeyPair'

let kv3: KeyPair = { key:1, value:100 }; // Compiler Error:
```

El compilador de TypeScript mostrará un error si hay algún cambio en el nombre de las propiedades o si el tipo de datos es diferente

La interface TypeScript también se utiliza para definir un tipo de función. Esto asegura la firma de la función.

```typescript
interface KeyValueProcessor{
    (key: number, value: string): void;
};

function addKeyValue(key:number, value:string):void {
    console.log('addKeyValue: key = ' + key + ', value = ' + value)
}

function updateKeyValue(key: number, value:string):void {
    console.log('updateKeyValue: key = '+ key + ', value = ' + value)
}

let kvp: KeyValueProcessor = addKeyValue;
kvp(1, 'Bill'); //Output: addKeyValue: key = 1, value = Bill

kvp = updateKeyValue;
kvp(2, 'Steve'); //Output: updateKeyValue: key = 2, value = Steve
```

Una interface también puede definir el tipo de matriz donde puede definir el tipo de índice y los valores.

```typescript
interface NumList {
  [index:number]:number
}

let numArr: NumList = [1, 2, 3];
numArr[0];
numArr[1];

interface IStringList {
  [index:string]:string
}

let strArr : IStringList = {};
strArr["TS"] = "TypeScript";
strArr["JS"] = "JavaScript";
```

En el ejemplo anterior, la interface NumLis tdefine un tipo de matriz con índice como número y valor como tipo de número. De la misma manera, IStringList define una matriz de cadenas con índice como cadena y valor como cadena.

```typescript
interface IEmployee {
    empCode: number;
    empName: string;
    empDept?:string;
}

let empObj1:IEmployee = {    // OK
    empCode:1,
    empName:"Steve"
}

let empObj2:IEmployee = {    // OK
    empCode:1,
    empName:"Bill",
    empDept:"IT"
}
```

A veces, podemos declarar una interface con propiedades excesivas, pero no esperamos que todos los objetos definan todas las propiedades de la interface dadas. Podemos tener propiedades opcionales, marcadas con un "?". En tales casos, los objetos de la interface pueden definir o no estas propiedades.

En el ejemplo anterior, empDeptestá marcado con ?, por lo que los objetos de IEmployee pueden incluir o no esta propiedad.

TypeScript proporciona una forma de marcar una propiedad como de solo lectura. Esto significa que una vez que a una propiedad se le asigna un valor, ¡no se puede cambiar!

```typescript
interface Citizen {
    name: string;
    readonly SSN: number;
}

let personObj: Citizen  = { SSN: 110555444, name: 'James Bond' }

personObj.name = 'Steve Smith'; // OK
personObj.SSN = '333666888'; // Compiler Error
```

Las interfaces pueden ampliar una o más interfaces. Esto hace que las interfaces de escritura sean flexibles y reutilizables.

```typescript
interface IPerson {
    name: string;
    gender: string;
}

interface IEmployee extends IPerson {
    empCode: number;
}

let empObj:IEmployee = {
    empCode:1,
    name:"Bill",
    gender:"Male"
}
```

```typescript
interface IEmployee {
    empCode: number;
    name: string;
    getSalary:(empCode: number) => number;
}

class Employee implements IEmployee {
    empCode: number;
    name: string;

    constructor(code: number, name: string) {
        this.empCode = code;
        this.name = name;
    }

    getSalary(empCode:number):number {
        return 20000;
    }
}

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

Implementar una interface:

Al igual que en lenguajes como Java y C#, las interfaces en TypeScript se pueden implementar con una clase. La clase que implementa la interface debe ajustarse estrictamente a la estructura de la interface.

Por supuesto, la clase implementadora puede definir propiedades y métodos adicionales, pero al menos debe definir todos los miembros de una interface.

Ejercicio

1. Crea una interface Vehicle con propiedades comunes a distintos vehículos como model, year, color, etc. Luego crea interfaces Car y Motorcycle que hereden de Vehicle y tengan propiedades específicas, implemente en una clase.
2. Crea una interface User y otra interfaz Admin que herede de User. Crea una función para imprimir datos de usuario que acepte tanto User como Admin.
3. Crea una interface Product con name, price, etc. Crea una interface Inventory que contenga un array de Product y funciones para agregar y buscar productos.
4. Crea una interface BaseObject con una propiedad id. Luego crea interfaces User, Product y Order que hereden de BaseObject. Crea una función genérica que pueda imprimir los datos.
5. Crea una interface Database con funciones como connect, find, update, etc. Luego crea una clase MySQLDatabase e SQLiteDatabase que implementen esa interface con distintas funcionalidades.