56 - TypeScript: interfaces

Una interface declara una serie de métodos y propiedades que deben ser implementados luego por una o más clases.

Las interfaces vienen a suplir la imposibilidad de herencia múltiple.

Por ejemplo podemos tener dos clases que representen un avión y un helicóptero. Luego plantear una interface con un método llamado volar. Las dos clases pueden implementar dicha interface y codificar el método volar (los algoritmos seguramente sean distintos pero el comportamiento de volar es común tanto a un avión como un helicóptero)

La sintaxis en TypeScript para declarar una interface es:

```
interface [nombre de la interface] {
    [declaración de propiedades]
    [declaración de métodos]
}
```

Problema

Definir una interface llamada Punto que declare un método llamado imprimir. Luego declarar dos clases que la implementen.

```
interface Punto {
  imprimir(): void;
}

class PuntoPlano implements Punto {
  constructor(private x: number, private y: number) { }

  imprimir() {
    console.log(`Punto en el plano: (${this.x},${this.y})`);
  }
}

class PuntoEspacio implements Punto {
  constructor(private x: number, private y: number, private z: number
) { }

  imprimir() {
    console.log(`Punto en el espacio: (${this.x},${this.y},${this.z})
`);
  }
}



let puntoPlano1: PuntoPlano;
puntoPlano1 = new PuntoPlano(10, 4);
puntoPlano1.imprimir();

let puntoEspacio1: PuntoEspacio;
puntoEspacio1 = new PuntoEspacio(20, 50, 60);
puntoEspacio1.imprimir();
```

Para declarar una interface en TypeScript utilizamos la palabra clave interface y seguidamente su nombre. Luego entre llaves indicamos todas las cabeceras de métodos y propiedades. En nuestro ejemplo declaramos la interface Punto e indicamos que quien la implemente debe definir un método llamado imprimir sin parámetros y que no retorna nada:

```
interface Punto {
  imprimir(): void;
}
```

Por otro lado declaramos dos clases llamados PuntoPlano con dos propiedades y PuntoEspacio con tres propiedades, además indicamos que dichas clases implementarán la interface Punto:

```
class PuntoPlano implements Punto {
  constructor(private x: number, private y: number) { }

  imprimir() {
    console.log(`Punto en el plano: (${this.x},${this.y})`);
  }
}

class PuntoEspacio implements Punto {
  constructor(private x: number, private y: number, private z: number) { }

  imprimir() {
    console.log(`Punto en el espacio: (${this.x},${this.y},${this.z})`);
  }
}
```

La sintaxis para indicar que una clase implementa una interface requiere disponer la palabra clave implements y en forma seguida el o los nombres de interfaces a implementar. Si una clase hereda de otra también puede implementar una o más interfaces.

El método imprimir en cada clase se implementa en forma distinta, en uno se imprimen 3 propiedades y en la otra se imprimen 2 propiedades.

Luego definimos un objeto de la clase PuntoPlano y otro de tipo PuntoEspacio:

```
let puntoPlano1: PuntoPlano;
puntoPlano1 = new PuntoPlano(10, 4);
puntoPlano1.imprimir();

let puntoEspacio1: PuntoEspacio;
puntoEspacio1 = new  PuntoEspacio(20, 50, 60);
puntoEspacio1.imprimir();
```

Si una clase indica que implementa una interfaz y luego no se la codifica, se genera un error en tiempo de compilación informándonos de tal situación (inclusive el editor Visual Studio Code detecta dicho error antes de compilar):

```
prueba.ts(5,7): error TS2420: Class 'PuntoPlano' incorrectly implements interface 'Punto'.
  Property 'imprimir' is missing in type 'PuntoPlano'.
prueba.ts(20,13): error TS2339: Property 'imprimir' does not exist on type 'PuntoPlano'.
```

Este error se produce si codificamos la clase sin implementar el método imprimir:

```
class PuntoPlano implements Punto{
  constructor(private x:number, private y:number) {}
}
```

Problema

Se tiene la siguiente interface:

```
interface Figura {
  superficie: number;
  perimetro: number;
  calcularSuperficie(): number;
  calcularPerimetro(): number;
}
```

Declar dos clases que representen un Cuadrado y un Rectángulo. Implementar la interface Figura en ambas clases.

```
interface Figura {
  superficie: number;
  perimetro: number;
  calcularSuperficie(): number;
  calcularPerimetro(): number;
```

```
}


class Cuadrado implements Figura {
  superficie: number;
  perimetro: number;
  constructor(private lado:number) {
    this.superficie = this.calcularSuperficie();
    this.perimetro = this.calcularPerimetro();
  }

  calcularSuperficie(): number {
    return this.lado * this.lado;
  }

  calcularPerimetro(): number {
    return this.lado * 4;
  }
}


class Rectangulo implements Figura {
  superficie: number;
  perimetro: number;
  constructor(private ladoMayor:number, private ladoMenor:number) {
    this.superficie = this.calcularSuperficie();
    this.perimetro = this.calcularPerimetro();
  }

  calcularSuperficie(): number {
    return this.ladoMayor * this.ladoMenor;
  }

  calcularPerimetro(): number {
    return (this.ladoMayor * 2) + (this.ladoMenor * 2);
  }
}


let cuadrado1: Cuadrado;
cuadrado1 = new Cuadrado(10);
console.log(`Perimetro del cuadrado : ${cuadrado1.calcularPerimetro()
}`);
console.log(`Superficie del cuadrado : ${cuadrado1.calcularSuperficie
()}`);
let rectangulo1: Rectangulo;
rectangulo1 = new Rectangulo(10, 5);
```

```
rectangulo1 = new Rectangulo(10, 5);
console.log(`Perimetro del rectangulo : ${rectangulo1.calcularPerimetro()}`);
console.log(`Superficie del rectangulo: ${rectangulo1.calcularSuperficie()}`);
```

En este problema la interface Figura tiene dos métodos que deben ser implementados por las clases y dos propiedades que también deben definirlos:

```
interface Figura {
  superficie: number;
  perimetro: number;
  calcularSuperficie(): number;
  calcularPerimetro(): number;
}
```

La clase Cuadrado indica que implementa la interface Figura, esto hace necesario que se implementen los métodos calcularSuperficie y calcularPerimetro, y las dos propiedades:

```
class Cuadrado implements Figura {
  superficie: number;
  perimetro: number;
  constructor(private lado:number) {
    this.superficie = this.calcularSuperficie();
    this.perimetro = this.calcularPerimetro();
  }

  calcularSuperficie(): number {
    return this.lado * this.lado;
  }

  calcularPerimetro(): number {
    return this.lado * 4;
  }
}
```

La clase Cuadrado tiene una propiedad llamada lado que la recibe el constructor.

De forma similar la clase Rectangulo implementa la interface Figura:

```
class Rectangulo implements Figura {
  superficie: number;
  perimetro: number;
  constructor(private ladoMayor:number, private ladoMenor:number) {
    this.superficie = this.calcularSuperficie();
    this.perimetro = this.calcularPerimetro();
  }

  calcularSuperficie(): number {
    return this.ladoMayor * this.ladoMenor;
  }

  calcularPerimetro(): number {
    return (this.ladoMayor * 2) + (this.ladoMenor * 2);
  }
}
```

Finalmente definimos un objeto de la clase Cuadrado y otro de la clase Rectangulo, luego llamamos a los métodos calcularPerimetro y calcularSuperficie para cada objeto:

```
let cuadrado1: Cuadrado;
cuadrado1 = new Cuadrado(10);
console.log(`Perimetro del cuadrado : ${cuadrado1.calcularPerimetro()}`);
console.log(`Superficie del cuadrado : ${cuadrado1.calcularSuperficie()}`);
let rectangulo1: Rectangulo;
rectangulo1 = new Rectangulo(10, 5);
console.log(`Perimetro del rectangulo : ${rectangulo1.calcularPerimetro()}`);
console.log(`Superficie del cuadrado : ${rectangulo1.calcularSuperficie()}`);
```

Las interfaces exige que una clase siga las especificaciones de la misma y se implementen algoritmos más robustos. En nuestro ejemplo tanto la clase Rectangulo como Cuadrado tienen una forma similar de trabajar gracias a que implementan la interfaz Figura.

Parámetros de tipo interface.

Un método o función puede recibir como parámetro una interface. Luego le podemos pasar objetos de distintas clases que implementan dicha interface:

```
interface Figura {
  superficie: number;
  perimetro: number;
  calcularSuperficie(): number;
  calcularPerimetro(): number;
}


class Cuadrado implements Figura {
  superficie: number;
  perimetro: number;
  constructor(private lado:number) {
    this.superficie = this.calcularSuperficie();
    this.perimetro = this.calcularPerimetro();
  }

  calcularSuperficie(): number {
    return this.lado * this.lado;
  }

  calcularPerimetro(): number {
    return this.lado * 4;
  }
}


class Rectangulo implements Figura {
  superficie: number;
  perimetro: number;
  constructor(private ladoMayor:number, private ladoMenor:number) {
    this.superficie = this.calcularSuperficie();
    this.perimetro = this.calcularPerimetro();
  }
```

```
  calcularSuperficie(): number {
    return this.ladoMayor * this.ladoMenor;
  }

  calcularPerimetro(): number {
    return (this.ladoMayor * 2) + (this.ladoMenor * 2);
  }
}

function imprimir(fig: Figura) {
  console.log(`Perimetro: ${fig.calcularPerimetro()}`);
  console.log(`Superficie: ${fig.calcularSuperficie()}`);
}

let cuadrado1: Cuadrado;
cuadrado1 = new Cuadrado(10);
console.log('Datos del cuadrado');
imprimir(cuadrado1);
let rectangulo1: Rectangulo;
rectangulo1 = new Rectangulo(10, 5);
console.log('Datos del rectángulo');
imprimir(rectangulo1);
```

La función imprimir recibe como parámetro fig que es de tipo Figura:

```
function imprimir(fig: Figura) {
  console.log(`Perimetro: ${fig.calcularPerimetro()}`);
  console.log(`Superficie: ${fig.calcularSuperficie()}`);
}
```

Podemos luego llamar a la función imprimir pasando tanto objetos de la clase Cuadrado como Rectangulo:

```
imprimir(cuadrado1);

imprimir(rectangulo1);
```

Es importante notar que solo podemos acceder a los métodos y propiedades definidos en la interfaz y no a propiedades y métodos propios de cada clase.

Creación de objetos a partir de una interface.

TypeScript permite crear objetos a partir de una interfaz. La sintaxis para dicha creación es:

```
interface Punto {
  x: number;
  y: number;
}

let punto1: Punto;
punto1 = {x:10, y:20};
console.log(punto1);
```

No podemos utilizar el operador new para la creación del objeto.

Podemos definir la variable e inmediatamente iniciarla:

```
let punto1: Punto = {x:10, y:20};
```

Propiedades opcionales.

Una interface puede definir propiedades opcionales que luego la clase que la implementa puede o no definirlas. Se utiliza la misma sintaxis de los parámetros opcionales, es decir se le agrega el caracter '?' al final del nombre de la propiedad.

```
interface Punto {
  x: number;
  y: number;
  z?: number;
}

let puntoPlano: Punto = {x:10, y:20};
console.log(puntoPlano);
let puntoEspacio: Punto = {x:10, y:20, z:70};
console.log(puntoEspacio);
```

Como vemos el objeto 'puntoPlano' solo implementa las propiedades 'x' e 'y'.

Se produce un error en tiempo de compilación si no implementamos todas las propiedades obligatorias, por ejemplo:

```
let puntoPlano: Punto = {x:10};
```

Esta línea genera un error ya que solo se define la propiedad 'x' y falta definir la propiedad 'y'.

Herencia de interfaces.

TypeScript permite que una interface herede de otra:

```
interface Punto {
  x: number;
  y: number;
}

interface Punto3D extends Punto {
  z: number;
}

let punto1: Punto = {x:10, y:20};
let punto2: Punto3D = {x:23, y:13, z:12};
console.log(punto1);
console.log(punto2);
```

Retornar