Boilerplate

Los ejemplos que vamos a codificar los puedes probar en la sandbox de TypeScript, si no sabes cómo funciona, échale un vistazo al módulo de setup y si te quedan dudas contacta con tu mentor.

Ojo con el this

Una de las principales pegas del manejo de clases en Javascript es el uso de *this*.

En este caso, vamos a simular que hacemos una llamada a un servidor, utilizando set timeout, para obtener el precio de un producto. Cuando nos llegue el precio, queremos aplicarle un descuento y mostrarlo por consola.

```
class PreciosAPI {
  descuento: number;
  constructor() {
    this.descuento = 0.8;
  }

  cargaPrecioDeServidor() {
    setTimeout(function () {
      const precio = 2; // <- simulando precio desde el servidor

      console.log(precio * this.descuento);
    });
  }
}

const preciosAPI = new PreciosAPI();
preciosAPI.cargaPrecioDeServidor();
```

Nos da un error y nos dice qué *'this' implicitly has type 'any' because it does not have a type annotation*. ¿Qué ha pasado? ¿Por qué no sabe qué es *this*?

El problema es que *this* no es lo que esperamos que sea. En este caso, *this* no es la instancia de la clase, sino que es el contexto de ejecución de la función que estamos pasando a *setTimeout*. En este caso, *this* es el objeto global de Javascript, que en el navegador es *window* y en NodeJS es *global*.

¿Cómo podemos arreglar esto? Podemos usar varias aproximaciones:

Existe en JavaScript el método, `bind` es una función incorporada en JavaScript que se utiliza para establecer explícitamente el valor del contexto (`this`) para una función. Es decir, `bind` permite enlazar una función a un objeto específico como su contexto, lo que garantiza que dentro de la función, `this` se referirá al objeto especificado.

Y vamos a utilizar `// @ts-ignore` para evitar que TypeScript muestre un error por el uso de `this.descuento` dentro de la función anónima dentro de `setTimeout`. Como mencionamos anteriormente,

el uso de `this` dentro de esa función anónima cambia el contexto y TypeScript no puede determinar a qué objeto `this` se refiere en este caso.

```
class PreciosAPI {
  constructor() {
    this.descuento = 0.80;
}

cargaPrecioDeServidor() {
  setTimeout(function() {
    const precio = 2;

    // @ts-ignore
    console.log(precio * this.descuento);
+   }.bind(this))
-   })
  }
}
const preciosAPI = new PreciosAPI();
preciosAPI.cargaPrecioDeServidor();
```

Al agregar `// @ts-ignore`, le estás diciendo al compilador de TypeScript que ignore el error que normalmente mostraría en esa línea en particular, permitiendo que el código compile sin problemas. Sin embargo, es importante tener en cuenta que el uso excesivo de `// @ts-ignore` puede ocultar problemas reales y, en general, es recomendable tratar de resolver los errores adecuadamente en lugar de simplemente ignorarlos.

Otra forma de solucionarlo es definir la función como una fat arrow o función flecha (aquí toma el `this` en tiempo de interpretación, no de ejecución), con este cambio, *this* es la instancia de la clase, ya no tenemos el error y podemos acceder a la propiedad *descuento*.

```
class PreciosAPI {
  descuento: number;
  constructor() {
    this.descuento = 0.80;
  }

  cargaPrecioDeServidor() {
-   setTimeout(function() {
+   setTimeout(() => {
      const precio = 2;
-     // @ts-ignore
      console.log(precio * this.descuento);
-   }.bind(this)
+     })
  }
}
```

```
const preciosAPI = new PreciosAPI();
preciosAPI.cargaPrecioDeServidor();
```

Por simple curiosidad y para que os suene cuando lo veáis, a veces veremos código legacy que asigna el valor de `this` dentro de una variable por fuera de la función interna para poder usarla.

```
class PreciosAPI {
  descuento: number;
  constructor() {
    this.descuento = 0.8;
  }

  cargaPrecioDeServidor() {
+    const self = this;

-   setTimeout(() => {
+   setTimeout(function () {
      const precio = 2;

-     console.log(precio * this.descuento);
+     console.log(precio * self.descuento);
    });
  }
}

const preciosAPI = new PreciosAPI();
preciosAPI.cargaPrecioDeServidor();
```