Curso CFP

CFP Programador full-stack

Palabra reservada this + Repaso

Agenda

- Palabra reservada this
- Ejemplos del uso de this
- Repaso de la semana
  - Noción de Clase e Instancia
  - Encapsulamiento
  - Constructores
  - Composición Básica de Clases
- Recomendaciones
- Ejercicios

Palabra Reservada *this*

- Hasta el momento se vio que se usaba *solamente* para diferenciar una variable cualquiera, de una interna
- Puede usarse también para llamar métodos
- Es una variable que almacena una instancia
  - Guarda variables
  - Guarda métodos
- Es la forma que usa TypeScript (y otros lenguajes) para acceder al código de la instancia que está *dentro* de la clase
  - El código que está por fuera de la clase, se llama con la variable que concretamente definimos

Ejemplos de Uso

```
public a(): string {
    return this.marca;
}

public b(): string {
    console.log(this);

    return this.a();
}
```

Hace referencia a una variable interna de la clase

Hace referencia a un método de la clase

```
let decodificador: Decodificador = new Decodificador();

let primerTelevisor: Televisor = new Televisor("Samsung", decodificador);

console.log(primerTelevisor.b());
```

```
PS C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios\poo> tsc .\televisor.ts; node ./televisor.js
Televisor {
  marca: 'Samsung',
  decodificador: Decodificador { canalActual: 0, volumenActual: 10 } }
Samsung
```

Imprimir 'this' sería imprimir el estado de la instancia

El retorno del método 'b' es el retorno del método 'a'

Curso CFP

CFP Programador full-stack

Repaso de la Semana

Repaso - Clase e Instancia

- Una clase modela una entidad que agrupa variables y métodos
  - Puede pensarse como una plantilla
  - Una instancia es un objeto que sale de esa plantilla

```
let decodificador_a: Decodificador = new Decodificador();
decodificador_a.prenderApagar();

let decodificador_b: Decodificador = new Decodificador();
decodificador_b.prenderApagar();
```

Se tienen dos instancias de la misma clase (Decodificador)

Repaso - Abstracción

- Al momento de implementar una clase, se tiene que tener en cuenta la forma en que queremos que sea utilizada
- La idea siempre es proveer una determinada funcionalidad que no requiera conocer la forma en que está implementada
  - En caso de tener una clase que haga operaciones con matrices, no debería ser necesario saber cómo se hace cada operación
  - En caso de tener una calculadora, hay funciones que no necesitamos saber internamente cómo se hacen
- El concepto a seguir es ahorrar tiempo usando cosas que ya están hechas, en vez de reinventar la rueda

Repaso - Encapsulamiento

- Relacionado con la abstracción
- Los métodos de las clases pueden guardar estados en variables internas de la clase, por lo que se puede evitar que desde afuera se usen a través del modificador *'private'*
  - No hay que mostrar cosas que no son necesarias que se sepan
  - La idea es que el usuario de la clase, esté el menor tiempo posible para entender lo que hace
  - Si una persona quiere hacer una multiplicación de matrices, no le interesa saber cómo está hecha → solo quiere llamar al método que le haga el trabajo

Repaso - Constructores

- Al momento de instanciar una clase, se llama a un método especial que se llama *constructor*
- Se encarga de inicializar el estado interno de una instancia
- También puede utilizarse para asignar valores por defecto

```
public constructor(marca: string, decodificador?: Decodificador)
{
        this.marca = marca;
        this.decodificador = decodificador;
}

public constructor() {
        this.canalActual = 0;
        this.volumenActual = 10;
}
```

Constructor usando una configuración por parámetro

Constructor usando una configuración por defecto

Repaso - Composición

- Un conjunto de clases sencillas pueden formar una clase más compleja
- La posibilidad de que una clase se use como tipo, nos otorga la posibilidad de (por ejemplo) usar una clase como tipo de una variable interna de otra clase

```
class Punto {
    private x: number;
    private y: number;

    public constructor(x: number, y: number) {
        this.x = x;
        this.y = y;
    }
}
```

```
class Recta {
    private punto_a: Punto;
    private punto_b: Punto;

    public constructor(pA: Punto, pB: Punto) {
        this.punto_a = pA;
        this.punto_b = pB;
    }
}
```

Una clase compleja (Recta) está compuesta por una clase más sencilla (Punto)

Recomendaciones Generales

- Priorizar *siempre* la legibilidad del código
  - Usar nombres descriptivos
- Si una clase tiene funcionalidades que no tienen nada que ver una con otra → *separar clases*
- Hacer *siempre* un planteo de lo que se va a implementar
- Pensar *defensivamente*: chequear siempre los parámetros que llegan
- Usar un archivo por clase → 'nombreclase.ts'
- Evitar en la medida de lo posible el *código duplicado*
  - Métodos con código repetido, usar un método privado, y que ambas métodos lo invoquen

Curso CFP

CFP Programador full-stack

Ejercicios

Ejercicios - En Clase

Usar los conceptos y recomendaciones vistas durante esta semana y la anterior

- Armar una base de datos de libros
  - Hacer el planteo de las clases necesarias
  - Implementar la clase Libro
  - Implementar la clase GestorLibros → debe soportar insertar/consultar/modificar/eliminar libros (la entrada de información por teclado)
  - Luego incorporar en donde se crea necesario un mecanismo para leer libros desde un archivo de texto
  - Subir las cosas a GitHub y avisar por Slack

Ejercicios - Fuera de Clase

Incorporar los conceptos y recomendaciones vistas durante esta semana y la anterior

- Librerías NPM
  - Iniciar un proyecto NPM
  - Elegir una librería en https://www.npmjs.com/ e incorporarla en el proyecto
  - Modelar una clase con composiciones, que utilice dicha librería
  - Definir tarea NPM para compilar y correr los archivos necesarios
  - Subir proyecto a GitHub y avisar por Slack