

# Curso CFP

## CFP Programador full-stack

*Encapsulamiento + Composición*

# Agenda

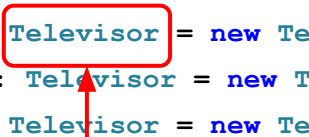
- Repaso
  - Noción de Clase e Instancia
  - Abstracción
- Encapsulamiento
- Constructores de Clase
- Parámetros Opcionales
- Demostración en Clase
- Las clases como tipos
- Composición de Clases
- La Clase Televisor
  - Planteo
  - Implementación
  - Recomendaciones
- Ejercicios

# Elementos Básicos de POO

## Repaso

- Un objeto se lo puede modelar tal como en la vida real
  - A partir de cómo se lo utiliza → a partir de las funciones que se llaman en el código
  - Tiene un estado → variables internas de la clase
- Un objeto se crea a partir de una clase
  - Por ejemplo en el código se pueden crear varios televisores de la clase Televisor

```
let volumenInicial: number = 10;  
let canalInicial: number = 24;  
  
let primerTelevisor: Televisor = new Televisor(volumenInicial, canalInicial);  
let segundoTelevisor: Televisor = new Televisor(5, 20);  
let tercerTelevisor: Televisor = new Televisor(15, 30);
```

A red rectangular box highlights the word 'Televisor' in the first line of object creation. A red arrow points from this box down to the word 'Televisor' in the second line of object creation. Another red arrow points from the box down to the word 'Televisor' in the third line of object creation.

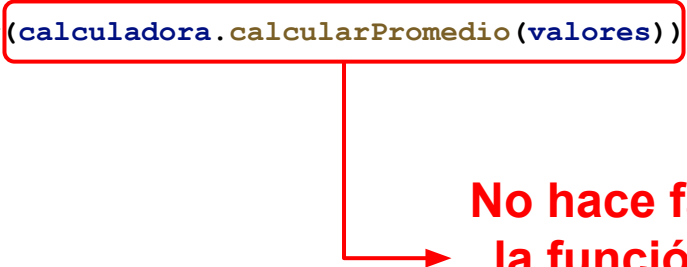
**Notar que la clase Televisor se usa como un tipo**

# Repaso - Abstracción

- Desde afuera se interactúa con el objeto a través de las funciones (métodos) que provee
- No hace falta saber desde afuera cómo está implementada la función → *Abstracción*
  - Alcanza con saber que una determinada clase provee una determinada funcionalidad y listo

```
let valores: number[] = [3, 4, 5, 6, 7];
```

```
let calculadora: Calculadora = new Calculadora();  
console.log(calculadora.calcularPromedio(valores));
```



**No hace falta saber cómo está implementada la función → solo alcanza con saber que la función retorna el promedio de los valores**

# Encapsulamiento (1)

- Lo que interesa conocer de una clase son las funciones que provee
- Desde el punto de vista del usuario de una clase, las variables internas no le interesan → por lo tanto no debería poder acceder/modificar dichas variables
- La forma correcta es que se modifique a través de una función
  - Para poder controlar la forma en que se modifica una variable interna
  - De otra manera podría poner por ejemplo el canal del televisor en -1 → INCORRECTO!

# Encapsulamiento (2)

- La forma correcta es que se modifique a través de una función
  - Para poder controlar la forma en que se modifica una variable interna
  - De otra manera podría poner por ejemplo el canal del televisor en -1 → INCORRECTO!

```
let volumenInicial: number = 10;  
let canalInicial: number = 24;
```

```
let primerTelevisor: Televisor = new Televisor(volumenInicial, canalInicial);
```

```
primerTelevisor.canal = 30;
```

**FORMA INCORRECTA**

```
primerTelevisor.elegirCanal(30);
```

**FORMA CORRECTA**

# Encapsulamiento (3)

- Para poder controlar a qué cosas se puede acceder y a qué cosas no, existen dos palabras especiales
  - `public` → cualquiera puede acceder
  - `private` → solamente dentro de la clase
- Cuando no se especifica ninguna de las dos, automáticamente es “`public`”
  - Siempre se recomienda especificar alguna de las dos, para que el código sea más legible

# Encapsulamiento (4)

```
class Televisor {  
    private estaPrendido: boolean  
    private volumenActual: number  
    private canalActual: number  
  
    constructor(volumenInicial: number, canalInicial: number) {  
        this.volumenActual = volumenInicial;  
        this.canalActual = canalInicial;  
    }  
  
    public prenderApagar(): void {  
        if (this.estaPrendido)  
            this.estaPrendido = false  
        else  
            this.estaPrendido = true  
    }  
  
    public subirVolumen(): void {  
        this.volumenActual = this.volumenActual + 1  
    }  
  
    ....  
}
```

**Se le dicen variables internas porque solamente se acceden internamente**

**Los métodos son públicos porque queremos que se accedan desde afuera**



# Constructor de Clase

- Es un método especial que se invoca al crear una instancia de una determinada clase
- Se usa para crear diferentes versiones de un objeto de una determinada clase

```
class Auto {  
    private marca: string;  
    private modelo: string;  
  
    constructor(marca: string, modelo: string) {  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
  
    public acelerar(): void {  
        this.velocidadActual += 10;  
    }  
}  
  
let primerAuto: Auto = new Auto('Ford', 'Fiesta');  
let segundoAuto: Auto = new Auto('Renault', 'Clio');  
let tercerAuto: Auto = new Auto('Peugeot', '307');
```

# Constructor con Parámetros Opcionales

- Puede ocurrir que cuando se quiera instanciar una clase, no tengamos todos los valores necesarios
- TypeScript permite que se pueda hacer una llamada pero sin todos los parámetros

```
class Auto {  
    private marca: string;  
    private modelo: string;  
    private año: number;
```

```
    constructor(marca: string, modelo: string, año?: number) {
```

El caracter '?' indica parámetro opcional

```
        this.marca = marca;  
        this.modelo = modelo;
```

```
        if (año == undefined)  
            this.año = -1;  
        else  
            this.año = año;
```

```
    }
```

```
    ....
```

```
}
```

```
let primerAuto: Auto = new Auto('Ford', 'Fiesta', 2004);  
let segundoAuto: Auto = new Auto('Renault', 'Clio');  
let tercerAuto: Auto = new Auto('Peugeot', '307');
```

# Parámetros Opcionales

- El concepto de parámetros opcionales no es exclusivo de los constructores
- También podemos aplicarlo a funciones y/o métodos

```
function imprimirMensaje(mensaje?: string): void {  
    if (mensaje == undefined)  
        console.log('Imprimiendo mensaje por default');  
    else  
        console.log(mensaje);  
}  
  
imprimirMensaje();  
imprimirMensaje('Hola como andas');
```

# Encapsulamiento - Demo en Clase

- Implementación de clase utilizando modificadores de acceso → public/private
- Forzar errores
- Ver un poco de parámetros opcionales


# Clases empleadas como Tipos

- Notar que cuando instanciamos una clase, escribimos el nombre de la clase como si fuese un tipo
- Esto otorga muchas posibilidades

```
class Auto {  
  private marca: string;  
  private modelo: string;  
  private año: number;  
  
  constructor(marca: string, modelo: string, año?: number) {  
    this.marca = marca;  
    this.modelo = modelo;  
  
    if (año == undefined)  
      this.año = -1;  
    else  
      this.año = año;  
  }  
}
```

```
let primerAuto: Auto = new Auto('Ford', 'Fiesta', 2004);  
let segundoAuto: Auto = new Auto('Renault', 'Clio');  
let tercerAuto: Auto = new Auto('Peugeot', '307');  
  
let arregloAutos: Auto[] = [primerAuto, segundoAuto, tercerAuto];
```

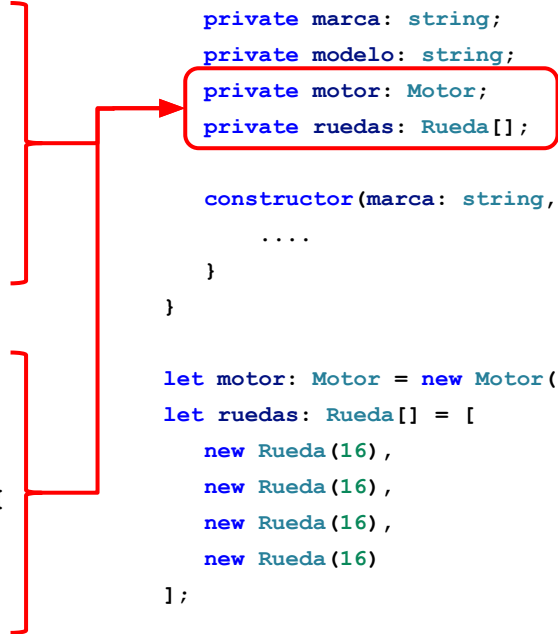
**Ahora podemos trabajar en las tareas que normalmente se realizan: insertar, buscar, eliminar, etc.**



# Composición de Clases

- Es muy común usar clases más simples para armar clases más complejas
- Por ejemplo un auto puede estar compuesto por un motor y ruedas, entre otras cosas

```
class Motor {  
    private tipo: string;  
  
    constructor(tipo: string) {  
        this.tipo = tipo;  
    }  
}  
  
class Rueda {  
    private tamaño: number;  
  
    constructor(tamaño: number) {  
        this.tamaño = tamaño;  
    }  
}  
  
class Auto {  
    private marca: string;  
    private modelo: string;  
    private motor: Motor;  
    private ruedas: Rueda[];  
  
    constructor(marca: string, modelo: string, motor: Motor, ruedas: Rueda[]) {  
        ....  
    }  
  
    let motor: Motor = new Motor('Nafta');  
    let ruedas: Rueda[] = [  
        new Rueda(16),  
        new Rueda(16),  
        new Rueda(16),  
        new Rueda(16)  
    ];  
  
    let primerAuto: Auto = new Auto('Fiat', 'Palio', motor, ruedas);
```

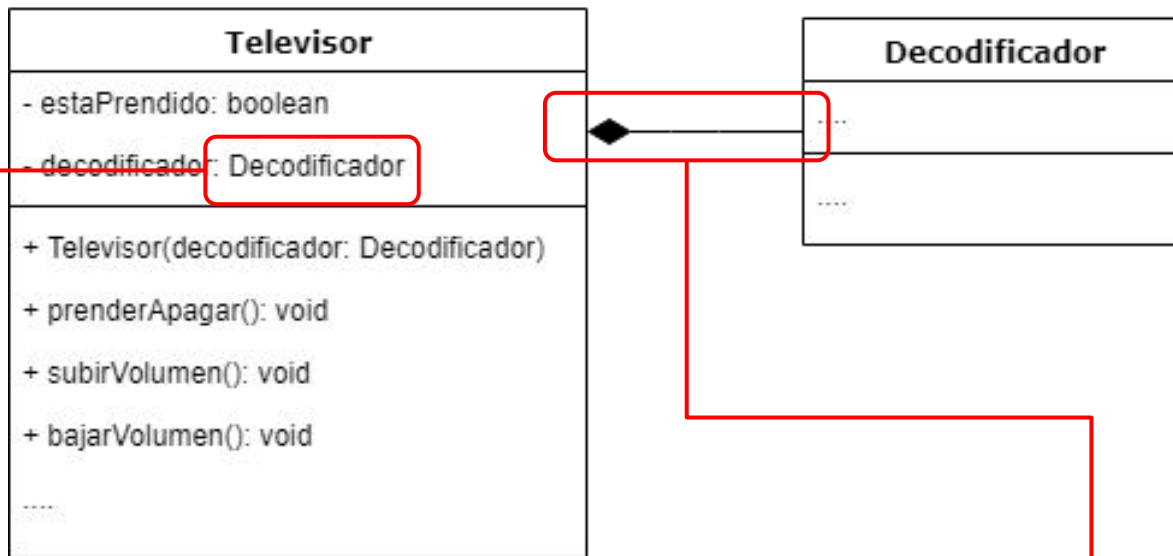


# La Clase Televisor - Planteo

```
class Televisor {  
    variables_internas  
        estaPrendido: boolean  
        volumenActual: number  
        canalActual: number  
        decodificador: Decodificador  
    métodos  
        prenderApagar()  
        subirVolumen()  
        bajarVolumen()  
        subirCanal()  
        bajarCanal()  
        cambiarCanal(canal)  
        verCanalActual()  
        verVolumenActual()  
}
```

- Es muy importante hacer un primer planteo de lo que vamos a hacer antes de pasar al código
- Si bien es una buena práctica hacer un planteo, no hay una forma única de hacerlo → cada uno lo hace de la manera que mejor le sirva

# La Clase Televisor - Diagrama



La clase Televisor está compuesta por la clase Decodificador

La flecha indica composición



# La Clase Televisor - Implementación

```
class Televisor {  
    private estaPrendido: boolean;  
    private decodificador: Decodificador;  
  
    constructor(marca: string, decodificador: Decodificador) {  
        this.decodificador = decodificador;  
    }  
  
    public prenderApagar(): void {  
        if (this.estaPrendido)  
            this.estaPrendido = false;  
        else  
            this.estaPrendido = true;  
    }  
  
    public subirVolumen(): void {  
        if (this.estaPrendido)  
            this.decodificador.subirVolumen();  
    }  
  
    public bajarVolumen(): void {  
        if (this.estaPrendido)  
            this.decodificador.subirVolumen();  
    }  
  
    public subirCanal(): void {  
        if (this.estaPrendido)  
            this.decodificador.subirCanal();  
    }  
  
    public bajarCanal(): void {  
        if (this.estaPrendido)  
            this.decodificador.bajarCanal();  
    }  
  
    public cambiarCanal(canal: number): void {  
        if (this.estaPrendido)  
            this.decodificador.cambiarCanal(canal);  
    }  
  
    public verCanalActual(): number {  
        return this.decodificador.verCanalActual();  
    }  
  
    public verVolumenActual(): number {  
        return this.decodificador.verVolumenActual();  
    }  
}  
  
let decodificador: Decodificador = new Decodificador();  
  
let primerTelevisor: Televisor = new Televisor(decodificador);  
primerTelevisor.cambiarCanal(15);
```

# Recomendaciones

- Los nombres de las clases arrancan en mayúsculas
- Siempre hacer un planteo de lo que debería hacer una clase → después pasarlo a código
- Pueden existir las funciones privadas: sirven para hacer cálculos auxiliares que no sean necesarios que se muestren para el usuario de la clase
- Los métodos (funciones) que una clase tenga, deben tener relación entre si
  - Ejemplo: la clase Televisor no puede tener un método que se encargue de calcular el promedio de un arreglo de números
- Práctica, mucha práctica

# CFP

# Programador

# full-stack

*Ejercicios*

# Ejercicios - En Clase

Para todos los ejercicios, crear proyecto NPM, subir a GitHub y avisar por Slack

## Ejercicio 1

- Agregar los conceptos vistos hoy al ejercicio 1 y 2 de la clase anterior

## Ejercicio 2

- Armar la clase Matriz (similar al de la clase anterior) pero aplicando conceptos de abstracción y encapsulamiento

## Ejercicio 3

- Implementar la clase Televisor y Decodificador

# Ejercicios - Fuera de Clase

- Crear proyecto NPM y subir a Github
- Implementar la clase LectorArchivos → partir del código facilitado en la clase anterior
- Implementar la clase RegistroAutomotor: similar al ejercicio de la clase pasada, pero incorporando los conceptos nuevos, y la clase LectorArchivos
- Mandar por Slack el link al repositorio de GitHub

*Aclaración:* no hay una sola forma de tener bien los ejercicios → lo que importa es saber justificar bien las decisiones que se tomen