

Prog. Orientada a Objetos

Carrera Programador full-stack

Intro a Programación Orientada a Objetos

Agenda

- Nueva forma de pensar los problemas → objetos
- Un objeto en la vida real
 - Funcionalidad
 - Estado
- Noción de abstracción
- Ejemplos de objetos reales
- Representación de Objetos
 - En papel
 - En máquina
- La clase Televisor
- Constructores de Clase
- Demostración en Clase

Nueva forma de pensar los problemas

- Hasta el momento se encararon los problemas directamente escribiendo el código
- Normalmente se plantea una *estrategia* para resolver un problema
- Tener una estrategia antes de resolver un problema hace más fácil y clara la programación
- Una de las estrategias posibles es la *Programación Orientada a Objetos*

Un Objeto en la Vida Real (1)

- Tomar como ejemplo un *televisor*
- Un televisor tiene varias funcionalidades
 - Mostrar los programas
 - Cambiar canales
 - Cambiar volumen
- Un televisor también tiene un estado
 - Canal actual
 - Volumen actual
 - Encendido/apagado



Un Objeto en la Vida Real (2)

- Un objeto puede ser definido conociendo:
 - Funcionalidades
 - Estado
- Es decir, que nos interesa saber cuál es el estado de un objeto, y qué cosas hace
- Por lo tanto podemos hacer dos asociaciones
 - Funcionalidades → Funciones
 - Estado → Variables

Otros Ejemplos de Objetos

- Un auto por ejemplo también tiene sus funciones y su estado
 - Funciones → Acelerar, frenar, prender, apagar, etc.
 - Estado → Velocidad actual, prendido/apagado, etc.
- Teléfono
 - Funciones → Sacar fotos, llamar, mensajear, etc.
 - Estado → Prendido/apagado, nivel de batería, etc.
- Impresora
 - Funciones → Imprimir, prender/apagar, etc.
 - Estado → Prendido/apagado, nivel de tinta, etc.

Noción de Abstracción

- Notar que en todos los casos, se hizo énfasis en que nos interesa el *estado* y las *funciones*
- Notar también, que en *ningún* caso nos interesa saber cómo funciona internamente el objeto (visto desde afuera)
 - No nos interesa saber cómo hace la tele internamente para cambiar de canal
 - No nos interesa saber cómo funciona el acelerador de un auto
 - Tampoco nos interesa saber cómo es el mecanismo de la cámara de un teléfono
- En todos los casos nos interesa la función concreta, sin pensar en cómo está implementada

Ejemplos de ustedes

Dar un ejemplo que se les ocurra en este momento.

Representación de Objetos - Planteo

Televisor

funciones

prenderApagar

subirVolumen

bajarVolumen

subirCanal

bajarCanal

estado

estaPrendido

volumenActual

canalActual

- Antes de escribir el código, lo mejor es plantear cuál sería el estado (variables) y las funciones de lo que quiero modelar

Representación de Objetos - Pseudocódigo

Televisor

estado

estaPrendido: boolean

volumenActual: number

canalActual: number

funciones

prenderApagar: void {

if (estaPrendido)

estaPrendido = false

else

estaPrendido = true

}

subirVolumen: void {

volumenActual = volumenActual + 1

}

bajarVolumen: void {

volumenActual = volumenActual - 1

}

subirCanal: void {

canalActual = canalActual + 1

}

bajarCanal: void {

canalActual = canalActual - 1

}

- A partir del planteo general, es más fácil ir planteando las funciones y el estado

Repaso de Conceptos

- Un objeto se modela de la forma que se asemeja a la vida real
 - Tiene estado y funcionalidades
- Cuando usamos un objeto, lo que nos interesa son las funciones que provee
 - Por lo tanto desde afuera no necesitamos conocer cómo un objeto funciona internamente

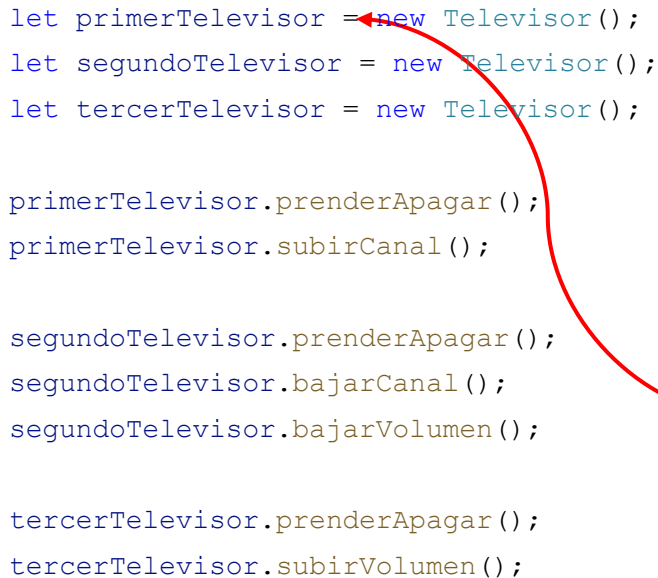
Definiciones

- Objeto:
- Instancia:
- Clase:
- Abstraccion:

Concepto de Clase y Objeto

OBJETO

CLASE



```
let primerTelevisor = new Televisor();
let segundoTelevisor = new Televisor();
let tercerTelevisor = new Televisor();

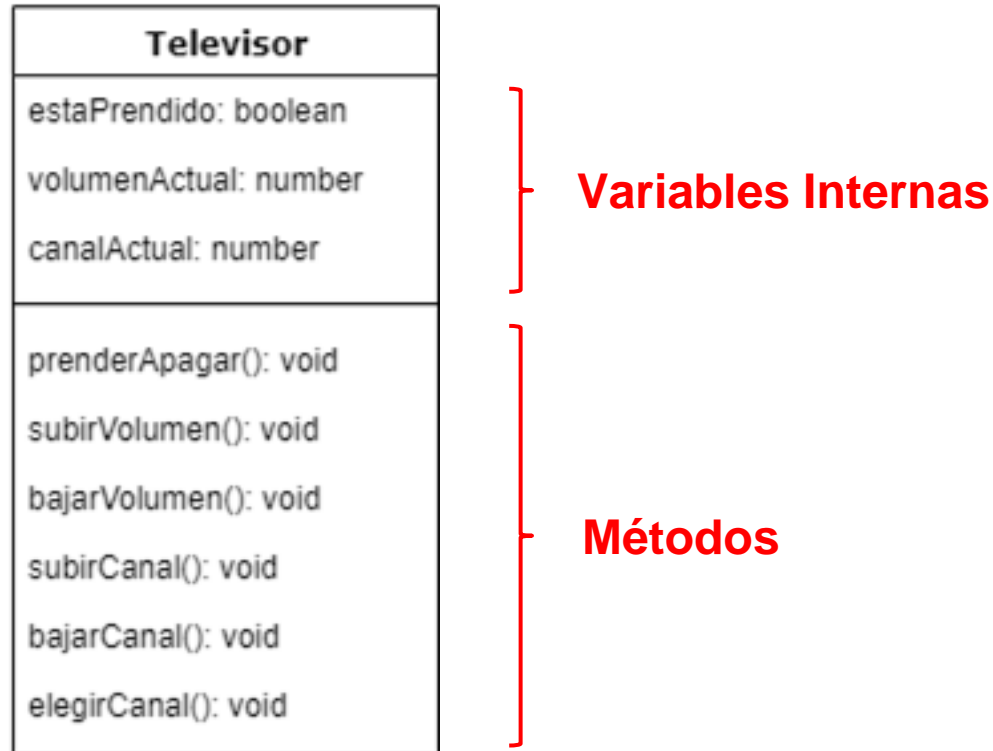
primerTelevisor.prenderApagar();
primerTelevisor.subirCanal();

segundoTelevisor.prenderApagar();
segundoTelevisor.bajarCanal();
segundoTelevisor.bajarVolumen();

tercerTelevisor.prenderApagar();
tercerTelevisor.subirVolumen();
```

- Cuando diseñamos el televisor, en realidad lo que hacemos es implementar la clase Televisor
- primerTelevisor se dice que es un objeto de la clase Televisor
- Podemos tener la cantidad de televisores que querramos

La Clase Televisor - Representación



La Clase Televisor - Pseudocódigo

```
clase Televisor
  variables_internas
    estaPrendido: boolean
    volumenActual: number
    canalActual: number
  funciones
    prenderApagar(): void {
      if (estaPrendido)
        estaPrendido = false
      else
        estaPrendido = true
    }

    subirVolumen(): void {
      volumenActual = volumenActual + 1
    }

    bajarVolumen(): void {
      volumenActual = volumenActual - 1
    }

    subirCanal(): void {
      canalActual = canalActual + 1
    }

    bajarCanal(): void {
      canalActual = canalActual - 1
    }
```

- Siempre la clase va a llevar un nombre
- Van a tener variables internas de la clase
 - Desde afuera no se pueden modificar directamente: o sea que afuera de la clase no se puede hacer → estaPrendido = true
- Van a tener funciones a través de las cuales se interactúa con el televisor

La Clase Televisor - TypeScript

```
class Televisor {  
    estaPrendido: boolean  
    volumenActual: number  
    canal: number  
  
    prenderApagar(): void {  
        if (estaPrendido)  
            estaPrendido = false  
        else  
            estaPrendido = true  
    }  
  
    subirVolumen(): void {  
        volumenActual = volumenActual + 1  
    }  
  
    bajarVolumen(): void {  
        volumenActual = volumenActual - 1  
    }  
  
    subirCanal(): void {  
        canal = canal + 1  
    }  
  
    bajarCanal(): void {  
        canal = canal - 1  
    }  
  
    elegirCanal(canal: number): void {  
        canal = canal  
    }  
}
```

- Este código no está del todo listo → cuando lo quieran compilar con “tsc” arroja una serie de errores
- Ver la última función elegirCanal

La Clase Televisor - TypeScript (1)

Televisor

funciones

prenderApagar
subirVolumen
bajarVolumen
subirCanal
bajarCanal

estado

estaPrendido
volumenActual
canalActual

Televisor

estado

estaPrendido: boolean
volumenActual: number
canalActual: number

funciones

```
prenderApagar: void {
    if (estaPrendido)
        estaPrendido = false
    else
        estaPrendido = true
}
```

```
subirVolumen: void {
    volumenActual = volumenActual + 1
}
```

```
bajarVolumen: void {
    volumenActual = volumenActual - 1
}
```

```
subirCanal: void {
    canalActual = canalActual + 1
}
```

```
bajarCanal: void {
    canalActual = canalActual - 1
}
```

class Televisor

variables_internas

estaPrendido: boolean
volumenActual: number
canalActual: number

funciones

```
prenderApagar(): void {
    if (estaPrendido)
        estaPrendido = false
    else
        estaPrendido = true
}
```

```
subirVolumen(): void {
    volumenActual = volumenActual + 1
}
```

```
bajarVolumen(): void {
    volumenActual = volumenActual - 1
}
```

```
subirCanal(): void {
    canalActual = canalActual + 1
}
```

```
bajarCanal(): void {
    canalActual = canalActual - 1
}
```

La Clase Televisor - TypeScript (2)

```
class Televisor
```

```
  variables_internas
```

```
    estaPrendido: boolean
```

```
    volumenActual: number
```

```
    canalActual: number
```

```
  funciones
```

```
    prenderApagar(): void {
```

```
      if (estaPrendido)
```

```
        estaPrendido = false
```

```
      else
```

```
        estaPrendido = true
```

```
    }
```

```
    subirVolumen(): void {
```

```
      volumenActual = volumenActual + 1
```

```
    }
```

```
    bajarVolumen(): void {
```

```
      volumenActual = volumenActual - 1
```

```
    }
```

```
    subirCanal(): void {
```

```
      canalActual = canalActual + 1
```

```
    }
```

```
    bajarCanal(): void {
```

```
      canalActual = canalActual - 1
```

```
    }
```

```
class Televisor {
```

```
  estaPrendido: boolean
```

```
  volumenActual: number
```

```
  canal: number
```

```
  prenderApagar(): void {
```

```
    if (estaPrendido)
```

```
      estaPrendido = false
```

```
    else
```

```
      estaPrendido = true
```

```
  }
```

```
  subirVolumen(): void {
```

```
    volumenActual = volumenActual + 1
```

```
  }
```

```
  bajarVolumen(): void {
```

```
    volumenActual = volumenActual - 1
```

```
  }
```

```
  subirCanal(): void {
```

```
    canal = canal + 1
```

```
  }
```

```
  bajarCanal(): void {
```

```
    canal = canal - 1
```

```
  }
```

```
  elegirCanal(canal: number): void {
```

```
    canal = canal
```

```
  }
```

```
class Televisor {
```

```
  private estaPrendido: boolean
```

```
  private volumenActual: number
```

```
  private canalActual: number
```

```
  prenderApagar(): void {
```

```
    if (this.estaPrendido)
```

```
      this.estaPrendido = false
```

```
    else
```

```
      this.estaPrendido = true
```

```
  }
```

```
  subirVolumen(): void {
```

```
    this.volumenActual = this.volumenActual + 1
```

```
  }
```

```
  bajarVolumen(): void {
```

```
    this.volumenActual = this.volumenActual - 1
```

```
  }
```

```
  subirCanal(): void {
```

```
    this.canalActual = this.canalActual + 1
```

```
  }
```

```
  bajarCanal(): void {
```

```
    this.canalActual = this.canalActual - 1
```

```
  }
```

```
  elegirCanal(canal: number): void {
```

```
    this.canalActual = canal;
```

```
  }
```

Constructor de Clase (1)

- Cuando se crea un objeto de la clase Televisor, se puede ver que la llamada es parecida a una función
- El constructor de una clase es una función especial que permite crear un objeto a partir de los parámetros que se le pase

```
let primerTelevisor = new Televisor();  
primerTelevisor.prenderApagar();  
primerTelevisor.subirCanal();
```

Constructor de Clase (2)

```
class Televisor {  
    estaPrendido: boolean  
    volumenActual: number  
    canalActual: number  
  
    constructor(volumenInicial: number, canalInicial: number) {  
        this.volumenActual = volumenInicial;  
        this.canalActual = canalInicial;  
    }  
    .....  
}
```

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

```
let miTelevisor = new Televisor(volumenInicial, canalInicial);
```

```
class Televisor {  
    estaPrendido: boolean  
    volumenActual: number  
    canalActual: number  
  
    .....  
}
```

```
let miTelevisor = new Televisor();
```

EN CASO DE NO EXISTIR
CONSTRUCTOR, SE CREA
AUTOMÁTICAMENTE UNO
SIN PARÁMETROS

This

this es una palabra clave que se refiere al objeto actual en el contexto en el que está siendo ejecutado un método. En otras palabras, **this** representa la instancia de la clase a la que pertenece el método que está siendo invocado.

```
class Persona {  
  constructor(nombre, edad) {  
    this.nombre = nombre;  
    this.edad = edad;  
  }  
  
  saludar() {  
    console.log(`Hola, mi nombre es ${this.nombre} y  
tengo ${this.edad} años.`);  
  }  
}
```

```
const persona1 = new Persona('Ana', 30);  
persona1.saludar();
```

Prog. Orientada a Objetos

Carrera Programador full-stack

Ejercicios

Ejercicios - En Clase

En un mismo proyecto NPM

Ejercicio 1

- Plantear la clase que describiste.
- Implementar en TypeScript

Ejercicio 2

- Plantear una clase Auto de la forma en que se vió en la clase → especificando variables internas y métodos
- Implementar en TypeScript