Prog. Orientada a Objetos

Carrera Programador full-stack

Polimorfismo + Repaso

Agenda

- Repaso de Herencia
- Redefinición de métodos
- Ejemplo con Clase Televisor
- Repaso de la Semana
  - Herencia
  - Composición
  - Diagramas de Clase
  - Polimorfismo
- Recomendaciones Generales
- Ejercicios

Repaso de Herencia

- Cuando tenemos dos clases con variables y métodos similares
  - Las cosas en común ponerlas en una superclase
  - Las dos clases originales extenderán la superclase
- Sirve para evitar que dupliquemos el código
  - Por lo tanto → para escribir menos
  - Código más claro y prolijo
- Variables *protected* → privadas salvo para subclases
- El constructor de la subclase *debe* incluir una llamada al constructor de la superclase → *super*
  - En caso de no incluirla → *tsc* se queja

Redefinición de Métodos

- Cuando una clase hereda de otra, nos traemos los métodos y variables de dicha clase
  - Pero también se pueden redefinir los métodos de la clase padre
- Suponer la clase Auto, y la clase AutoDeportivo que hereda de Auto
  - Ambos pueden acelerar
  - Sin embargo, aceleran *diferente*
- Otro ejemplo, un teléfono de hace diez años con cámara, y un teléfono nuevo con cámara
  - Ambos pueden sacar fotos
  - Sin embargo, uno saca mejores fotos

Redefinición - Clase Auto

```
class Auto {
    private marca: string;
    private modelo: string;
    protected velocidadActual: number;

    public constructor(marca: string, modelo: string) {
        this.marca = marca;
        this.modelo = modelo;
        this.velocidadActual = 0;
    }

    public acelerar(): void {
        this.velocidadActual += 10;
    }
}
```

```
class AutoDeportivo extends Auto {
    public constructor(marca: string, modelo: string) {
        super(marca, modelo);
    }

    public acelerar(): void {
        this.velocidadActual += 50;
    }
}
```

```
let primerAuto: Auto = new Auto('Ford', 'Fiesta');
let superAuto: Auto = new AutoDeportivo('Ford', 'Mustang');

primerAuto.acelerar();
superAuto.acelerar();

console.log(primerAuto);
console.log(superAuto);
```

```
C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios\poo>node auto.js
Auto { marca: 'Ford', modelo: 'Fiesta', velocidadActual: 10 }
AutoDeportivo { marca: 'Ford', modelo: 'Mustang', velocidadActual: 50 }
```

Los dos son un Auto, pero reaccionan diferente cuando los aceleramos

Redefinición de Métodos

- Cuando invocamos a un método, se arranca de *abajo hacia arriba* → arranca en los hijos, para terminar en el padre
- En el ejemplo anterior, cuando invocamos al método “acelerar”, si la clase AutoDeportivo no definiese nada → se llama al “acelerar” de la clase padre
- La idea central es entender que ambas clases son un auto, pero reaccionan diferente al acelerarlos
  - Este concepto se llama *Polimorfismo*
  - La idea concreta es mucho más sencilla que la sensación que transmite el nombre del concepto

Prog. Orientada a Objetos

Carrera Programador full-stack

Repaso de la Semana

Repaso de la Semana - Herencia

- Cuando tenemos dos clases con variables y métodos similares
  - Las cosas en común ponerlas en una superclase
  - Las dos clases originales extenderán la superclase
- Sirve para evitar que dupliquemos el código
  - Por lo tanto → para escribir menos
  - Código más claro y prolijo
- Variables *protected* → privadas salvo para subclases
- El constructor de la subclase *debe* incluir una llamada al constructor de la superclase → *super*
  - En caso de no incluirla → *tsc* se queja

Repaso - Composición

- Es cuando una clase más compleja está compuesta por clases más sencillas
  - En otras palabras, cuando una variable interna de una clase no es un tipo básico, sino otra clase
- En caso de elegir entre herencia y composición
  - Pensar en escribir la menor cantidad de código posible
  - Pero también evitar exponer funcionalidad no pretendida → no abusar de la herencia
- Ejemplo: un auto está compuesto por ruedas, motor, puertas, etc.

Repaso - Diagramas de Clase

- Es una de las formas de entender un sistema sin tener que ver el código
  - Sobre todo cuando tenemos miles de líneas de código
- Plantear el diagrama de antemano es una buena práctica para poder organizar mejor el código
  - Aunque a veces tenemos que ir y venir para acomodar las cosas
- Una solución bien planteada es mucho más fácil de implementar
  - Cuando tenemos un sistema real, si no hacemos un planteo de antemano, la organización del equipo desaparece y el desarrollo se torna un caos

Repaso - Polimorfismo

- Suponer dos clases: AutoCarreras y AutoCiudad
  - Ambos son autos → heredan de Auto
  - Ambos pueden acelerar
  - Sin embargo → aceleran *diferente*
- La clase padre Auto definen el método “acelerar”
  - Pero las clases hijas lo *redefinen*
  - Concretamente, definen en el código un método con el mismo nombre que en la clase padre

Recomendaciones

- Evitar *siempre* duplicar el código
- Tener cuidado cuando se usa herencia, *no forzarla*
  - Un auto y un avión tienen patente, pero no tiene nada que ver una cosa con la otra
- Herencia hace que de una clase padre, se le transfiera a la clase hija todo lo que sea *public* y *protected* → a veces puede que traigamos cosas que no necesitamos
- Tomarse el tiempo necesario para plantear correctamente la solución
  - Cuanto más tiempo le dediquen al diagrama, más problemas se van a evitar en el código

Prog. Orientada a Objetos

Carrera Programador full-stack

Ejercicios

Ejercicios - En Clase

Ejercicio 1

- Iniciar proyecto NPM y Git
- Plantear diagrama de clases para clases Auto y AutoCarreras redefiniendo métodos
- Implementar el código
- Subir a GitHub y avisar por Slack

Ejercicio 2

- Idem ejercicio anterior, pero proponiendo un ejercicio que emplee una clase padre y dos clases hijas → usando *polimorfismo*
- Subir a GitHub y avisar por Slack

Ejercicios - Fuera de Clase

- Iniciar proyecto NPM y Git
- Plantear un diagrama de clase con los siguientes requisitos e implementar
  - Herencia
  - Composición
  - Variables protected
  - Polimorfismo
  - Métodos privados
- Definir tarea NPM para compilar y correr los archivos necesarios
- Subir proyecto a GitHub
- Mandar por Slack el link al proyecto 🤙