Transacciones y API REST

Clases, constructores, accesadores y mutadores

{desafío} latam_

Implementar operaciones transaccionales en una base de datos para mantener la consistencia de los datos utilizando el entorno Node.js.

Implementar la capa de acceso a datos utilizando un ORM con entidades no relacionadas para realizar operaciones CRUD.

Utilizar asociaciones uno a uno, uno a muchos y muchos a muchos en la definición de las relaciones entre las entidades de un modelo que resuelven un problema.

{desafío} latam_

- Unidad 1:
  Implementación y gestión de una base de datos
- Unidad 2:
  Transacciones y API REST
- Unidad 3:
  Trabajo práctico

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Reconoce las características del lenguaje Javascript ES6 para la definición de clases y objetos.*

{desafío}
latam_

¿Podrías explicar brevemente qué son las clases en programación y cuál es su propósito principal?

{desafío} latam_

/* Qué es una Clase */

Qué es una Clase

Estructura

Son plantillas en las que nos basamos para crear objetos y para crearlos debemos conocer los siguientes conceptos:

- Palabras reservadas
- Modificador de acceso
- class
- Nombre Clase
- Atributo de instancia
- Atributo local
- Operaciones o Métodos

Qué es una Clase

Estructura

- Palabras reservadas: Existen palabras únicas del lenguaje que no se pueden utilizar como variables, nombre de métodos o de clases. Estas son las que actualmente se reserva Javascript:

  break , case , catch , continue , default , delete , do , else , finally , for , function , if , in , instanceof , new , return , switch , this , throw , try , typeof , var , void , while , with .

- Modificador de acceso: Determina la accesibilidad que tendrán otras clases sobre la clase. Normalmente, las clases utilizan el término public.

- class: Palabra reservada del lenguaje que determina la definición de una clase.

- Nombre Clase: Identificador que representa el nombre de la clase.

Qué es una Clase

Estructura

- Atributo de instancia: Cualquier método u operación en la clase puede utilizarlos.
- Atributo local: Se pueden crear dentro de los métodos y serán visibles desde dentro y no fuera de la clase.
- Operaciones o Métodos: Son todas las operaciones especiales de una clase que no se declaran como atributos de la clase. Estos métodos nos permiten que la clase realice acciones propias y existen de 1 a N métodos dentro de una clase. Estos métodos deben ser coherentes con la clase.

Ejemplo de una clase llamada Persona:

/* Definición de una clase por medio de una declaración */

Definición de una clase por medio de una declaración

En JavaScript, la programación orientada a objetos se basa en prototipos en lugar de clases tradicionales. Sin embargo, a partir de la versión de ECMAScript 6 (ES6), se introdujeron las clases para proporcionar una sintaxis más familiar para la programación orientada a objetos. En la siguiente diapositiva tienes un ejemplo de cómo se vería una definición de clase en JavaScript utilizando la sintaxis de clases de ES6:

Definición de una clase por medio de una declaración

```
class MiClase {
  // Constructor de la clase
  constructor() {
    // Atributos de la clase
    this.atributo1 = "Hola";
    this.atributo2 = 42;
  }

  // Métodos de la clase
  miMetodo() {
    console.log("Este es un método de la clase");
  }
}

// Crear un objeto de la clase
const objeto = new MiClase();

// Acceder a los atributos y métodos del objeto
console.log(objeto.atributo1);  // Imprime: Hola
console.log(objeto.atributo2);  // Imprime: 42
objeto.miMetodo();               // Imprime: Este es un método de la
clase
```

/* Definición de una clase por medio de una expresión */

Definición de una clase por medio de una expresión

En JavaScript, también es posible definir una clase utilizando una expresión. A diferencia de la declaración de clase que acabamos de ver, donde usamos la palabra clave class, la expresión de clase se asigna a una variable. En la siguiente slide tienes un ejemplo:

Definición de una clase por medio de una expresión

```
// Expresión de clase
const MiClase = class {
  // Constructor de la clase
  constructor() {
    // Atributos de la clase
    this.atributo1 = "Hola";
    this.atributo2 = 42;
  }

  // Métodos de la clase
  miMetodo() {
    console.log("Este es un método de la clase");
  }
};

// Crear un objeto de la clase
const objeto = new MiClase();

// Acceder a los atributos y métodos del objeto
console.log(objeto.atributo1);  // Imprime: Hola
console.log(objeto.atributo2);  // Imprime: 42
objeto.miMetodo();              // Imprime: Este es un método de la clase
```

Considerando los ejemplos proporcionados para la declaración y expresión de clases en JavaScript, ¿puedes identificar las similitudes y diferencias clave entre ambas formas de definir clases? Además, ¿por qué podrías elegir usar una expresión de clase en lugar de una declaración de clase, o viceversa, en diferentes situaciones de programación?

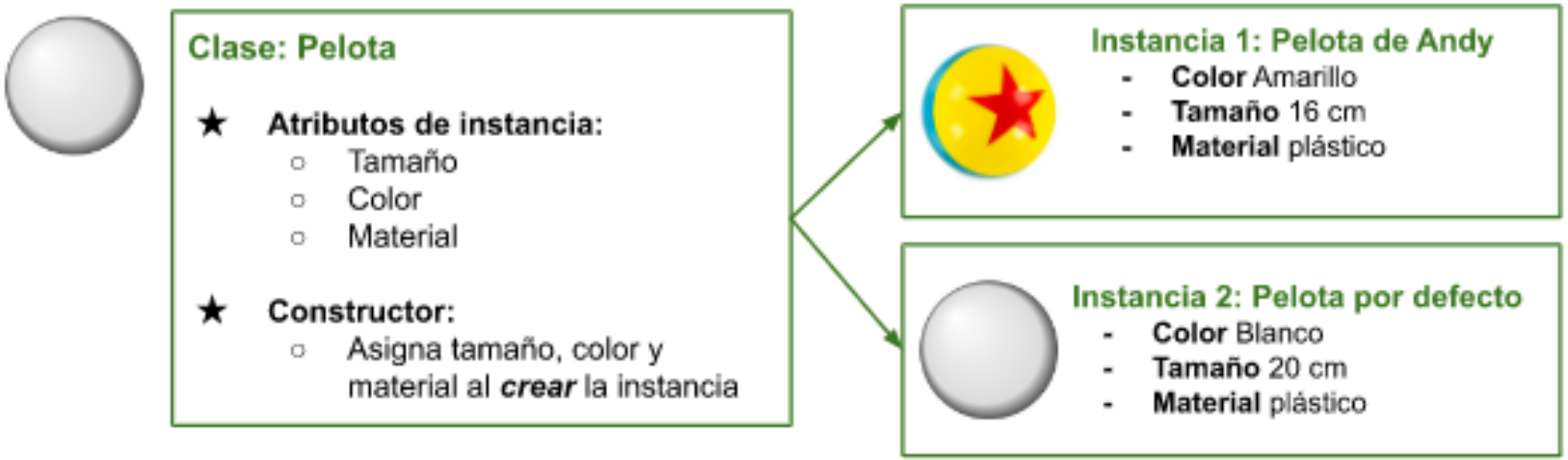
/* Qué es un método constructor y para qué sirve */

Qué es un método constructor y para qué sirve

- Es un método que **se ejecuta automáticamente** al momento de crear una **instancia** de la clase, sin necesidad de ser llamado explícitamente.
- Su función es dar valores a los atributos de la instancia recién creada. Los valores que se dan a los atributos pueden ser simplemente un valor por defecto (definido en el constructor) o valores que se deben especificar explícitamente al momento de crear la instancia.
- Dependiendo del problema a resolver, existen dos opciones a utilizar:
  - valores por defecto
  - valores entregados explícitamente
  - además, se puede utilizar una combinación de ambas, es decir, valores por defecto que pueden ser sobreescritos por valores entregados.

Qué es un método constructor y para qué sirve

Ejemplo



/* Accesadores y mutadores */

Accesadores y mutadores

Accesadores (getters)

- Son métodos que permiten acceder al valor de un atributo en una instancia.
- Permiten acceder a información de los atributos expresada de forma diferente.

Mutadores (setters)

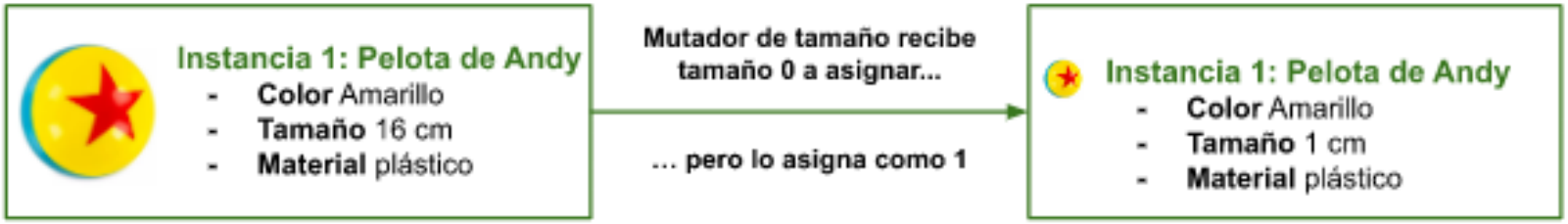
- Son métodos que permiten modificar el valor de un atributo en una instancia.
- Permiten aplicar reglas al momento de asignar un valor a un atributo.

Ambos se definen explícitamente dentro de la clase, y la forma de hacerlo dependerá del lenguaje de programación utilizado.

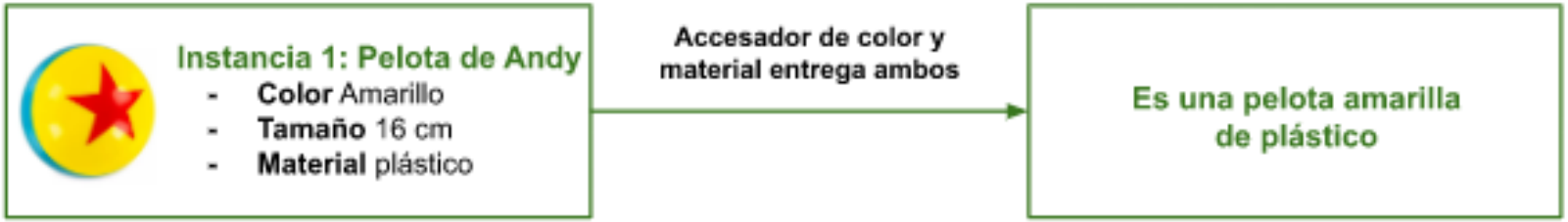
La principal razón para forzar el acceso o modificación de un atributo mediante un método, es impedir que el valor sea visto o modificado directamente desde la instancia de una clase (mediante sintaxis de punto `.`)

Accesadores y mutadores

Mutador de tamaño:



Accesador de color y material:



/* Métodos personalizados */

Métodos personalizados

El término "métodos personalizados" generalmente se refiere a funciones definidas por el usuario que son asignadas como propiedades a objetos. Los métodos son simplemente funciones que se convierten en propiedades de un objeto. Aquí hay un ejemplo sencillo:

```
// Objeto
const miObjeto = {
  propiedad: "Hola",

  // Método personalizado
  miMetodo: function() {
    console.log("Este es un método personalizado");
  }
};

// Llamar al método personalizado
miObjeto.miMetodo(); // Imprime: Este es un método personalizado
```

{desafío} latam_

Métodos personalizados

En este ejemplo, miMetodo es un método personalizado porque es una función que está asociada a un objeto específico (miObjeto). Los métodos personalizados permiten encapsular lógica relacionada con el objeto y proporcionan una forma de interactuar con los datos del objeto.

Con la introducción de ECMAScript 6 (ES6), también puedes definir métodos de manera más concisa utilizando la sintaxis de funciones flecha. El ejemplo anterior podría reescribirse de la siguiente manera:

```
const miObjeto = {
  propiedad: "Hola",

  // Método personalizado con función flecha
  miMetodo: () => {
    console.log("Este es un método personalizado");
  }
};

miObjeto.miMetodo(); // Imprime: Este es un método personalizado
```

/* Instanciación de una clase */

Instanciación de una clase

Es una forma de representar una clase "dándole vida" en forma de objeto. Se dice que para “instanciar” una clase, debemos ocupar el operador y la palabra reservada new. Con esta palabra podemos crear una nueva instancia de la clase.

```
Cliente cliente = new Cliente();
```

Cuando se realiza una instancia con el operador new generamos copias de nuestras clases para realizar su ejecución y tratamientos de nuevos valores, pero sin afectar el objeto principal.

¿Qué ventajas tiene acceder o modificar valores de atributos mediante accesadores y mutadores?

Próxima sesión...

- *Identifica las características, ventajas y desventajas de utilizar un ORM para la implementación de la capa de acceso a datos en un aplicativo Node.js.*
- *Define modelos que representan una entidad especificando propiedades, tipos de datos y otras opciones acorde a la librería Sequelize.*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales