Herencia

Herencia

{desafío} latam_

Reutilizar código en JavaScript aplicando el concepto de herencia, polimorfismo y closures para dar solución a una problemática.

Reconocer los elementos fundamentales del Document Object Model y los mecanismos para la manipulación de elementos en un documento html.

{desafío} latam_

- Unidad 1: ES6+ y POO
- Unidad 2: Herencia
- Unidad 2: Callbacks y APIs

Te encuentras aquí

¿Qué aprenderás en esta sesión?

- *Implementa herencias a partir de prototipos mediante la sintaxis de ES5 y ES6.*
- *Implementa la cadena de prototipos para lograr distintos niveles de herencia.*
- *Construye propiedades get y set en ES6 para acceder o modificar los atributos de una clase.*

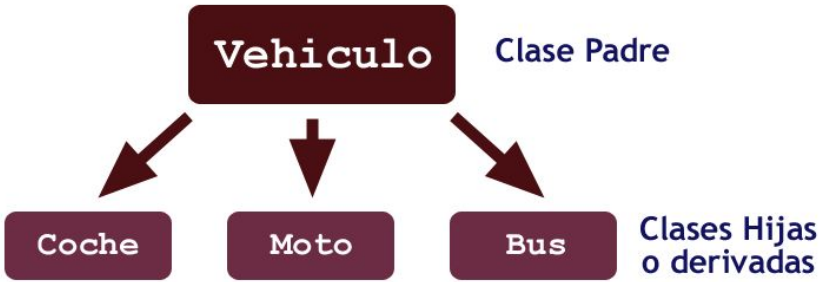
Comenta con tus palabras:

¿Qué sabes de las herencias?

/* Herencia */

Herencia

Para ejemplificar esto, veamos el siguiente diagrama en donde se puede observar como existe una clase superior o padre denominada “vehículo”, y de ella dependen otras clases hijas que pueden heredar los atributos y métodos de la clase padre.



Fuente: desarrolloweb

Ejercicio guiado:
Herencia con vehículos

{desafío}
latam_

Ejercicio guiado

Herencia con vehículos

Crear una clase padre con el nombre de “Vehículo” en conjunto con los atributos “marca”, “color”, “matrícula”, además de las tres clases hijas “coche”, “moto” y “bus”, heredando los atributos de la clase vehículo. Finalmente, instanciar cada una de las clases hijas y mostrar una de las propiedades para cada una de ellas.

Paso 1: Lo primero que debemos hacer, es crear una carpeta en nuestro lugar de trabajo y dentro de esta carpeta se debe crear un archivo de JavaScript con el nombre de “script.js”, luego, dentro del archivo debes crear la clase padre, la cual llevará el nombre de “Vehículo”, y dentro de su constructor, las propiedades mencionadas anteriormente “marca, color y matrícula”, como se muestra a continuación:

```
class Vehiculo{
   constructor(marca, color, matricula) {
       this.marca = marca;
       this.color = color;
       this.matricula = matricula;
   }
}
```

Ejercicio guiado

Herencia con vehículos

Paso 2: En el código anterior, se puede observar como el constructor de la clase alberga tres propiedades que serán comunes igualmente para las clases que se derivan de ella, es decir, para las clases hijas denominadas: coche, moto y bus. Por ende, en ES6 simplemente tenemos que utilizar la palabra reservada “extends” después de la declaración de la clase e indicar el nombre de la clase de la cual queremos que herede todos los métodos y propiedades. Como se muestra a continuación en el siguiente trozo de código:

```
class Vehiculo{
   constructor(marca, color, matricula) {
       this.marca = marca;
       this.color = color;
       this.matricula = matricula;
   }
}

class Coche extends Vehiculo{ //la clase
coche es hija de vehículo
}
class Moto extends Vehiculo{ //la clase
moto es hija de vehículo
}
class Bus extends Vehiculo{ //la clase
bus es hija de vehículo
}
```

Ejercicio guiado

Herencia con vehículos

Paso 3: Se puede observar en el código anterior como la clase Coche se extiende de la clase Vehículo, lo cual, permitirá a la clase Coche utilizar las propiedades que ya tiene declarada en el constructor la clase Vehiculo. Ahora bien, para lograr esto se debe implementar una nueva instrucción en el constructor de la clase hija “Coche” mediante la palabra reservada “super”, pasando como argumento los valores de las propiedades que deseamos inicializar de la clase padre, por lo que no es necesario inicializar las propiedades en la clase hija. Ahora, agreguemos el constructor a las clases hijas “marca, color y matrícula” como se muestra a continuación:

```
class Vehiculo{
   constructor(marca, color, matricula) {
       this.marca = marca;
       this.color = color;
       this.matricula = matricula;
   }
}

class Coche extends Vehiculo{
   constructor(marca,color,matricula){
      super(marca,color,matricula);
   }
}

class Moto extends Vehiculo{
   constructor(marca,color,matricula){
      super(marca,color,matricula);
   }
}

class Bus extends Vehiculo{
   constructor(marca,color,matricula){
      super(marca,color,matricula);
   }
}
```

Ejercicio guiado

Herencia con vehículos

Paso 4: En el código anterior, se puede observar como las clases de Coche, Moto y Bus son extensiones de una clase padre o principal, en este caso la clase Vehiculo, heredando las propiedades que posee esta clase mediante la palabra reservada “super” en el constructor de cada clase hija. Es importante destacar que los argumentos que se envían mediante la instrucción super, deben ser de las mismas características con que fueron inicializados en la clase padre.

```
let coche1 = new Coche('Toyota','Negro', '123ABC');
 let moto1 = new Moto('Honda','Rojo', '456CDF');
 let bus1 = new Bus('Fuso','Blanco', '678EDC');

 console.log(coche1);
 console.log(moto1);
 console.log(bus1);
 console.log(coche1.marca);
 console.log(moto1.color);
 console.log(bus1.matricula);
```

Ejercicio guiado

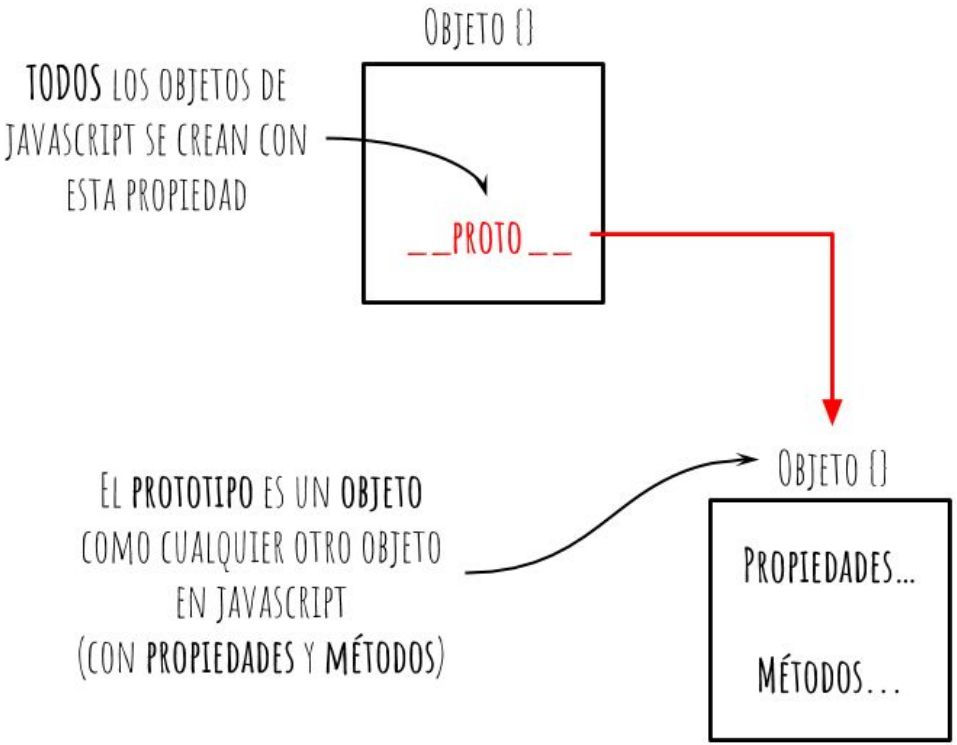
Herencia con vehículos

Paso 5: Si observas detalladamente en el código anterior, nunca se hizo una instancia para la clase padre con el nombre de “Vehículo”, esto se debe a que realmente no hace falta, ya que las clases hijas heredarán mediante la palabra reservada `extends` todas esas características de su clase padre. Ahora, el resultado lo podemos obtener al ejecutar el archivo “script.js” mediante la terminal con Node, con el comando: `node script.js`

```
Coche { marca: 'Toyota', color: 'Negro', matricula: '123ABC' }
Moto { marca: 'Honda', color: 'Rojo', matricula: '456CDF' }
Bus { marca: 'Fuso', color: 'Blanco', matricula: '678EDC' }
Toyota
Rojo
678EDC
```

/* Herencia de prototipos */

Herencia de prototipos



Fuente: gustavodohara

Herencia de prototipos

Este principio básico de reusabilidad de código es lo que llamamos herencia basada en clases. Sin embargo, JavaScript no posee el concepto de clases sino de prototipos. Implementar un sistema de este tipo en JavaScript es lo que conocemos como delegación basada en herencia o herencia prototípica. Tras la idea de la herencia, independientemente del sistema (clases o prototipos), consiste en declarar un objeto (o clase) cuyos métodos se heredan por referencia en otros objetos.

Veamos el siguiente ejemplo, donde creamos un objeto con una propiedad y un método, luego crearemos un nuevo objeto, al cual le asignaremos como objeto padre el primer objeto creado, esto se logrará mediante el método Object.create(), quien crea un objeto nuevo utilizando un objeto existente como el prototipo del nuevo objeto creado, luego mostraremos por consola ambos objetos para ver el prototipo de cada uno.

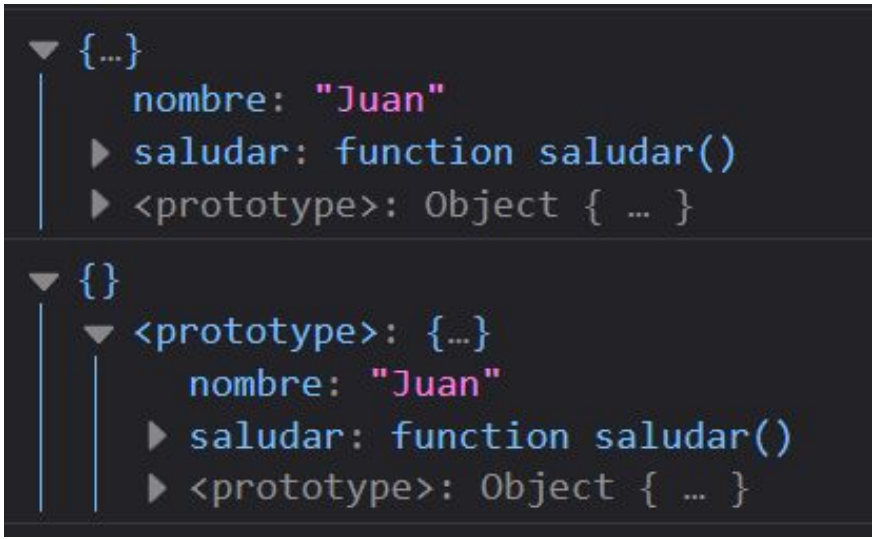
Herencia de prototipos

```
var persona_uno = {
  nombre: "Juan",
  saludar: function(){
      console.log("Hola, soy "+this.nombre);
  }
}

console.log(persona_uno);
var persona_dos = Object.create(persona_uno);
console.log(persona_dos);
```

Herencia de prototipos

Al ejecutar el código anterior en la consola del navegador web, el resultado será:



Ejercicio guiado: Herencia prototípica

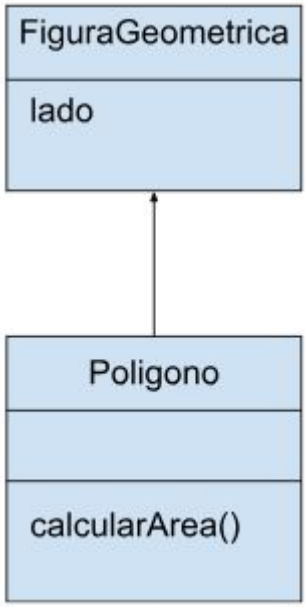
{desafío} latam_

Ejercicio guiado

Herencia prototípica

Realizar el siguiente ejercicio de clases mediante herencia prototípica con nomenclatura de ES5, para ver en acción los prototipos, primeramente con un ejercicio sin métodos en la clase padre. Luego, convertir el código anterior a uno actualizado implementado ES6.

Crear una clase padre o principal, mediante una función constructora con el nombre de “FiguraGeometrica”, la cual recibe como parámetros los valores de un lado de la figura. Además, se solicita crear una clase denominada “Polígono”, pero que herede las propiedades de la clase padre. Además, la clase Polígono debe poseer un método propio que permita calcular el área de la figura geométrica con las propiedades inicializadas en la clase padre.



Fuente: Desafío Latam

Ejercicio guiado

Herencia prototípica

Paso 1: Crear una carpeta en nuestro lugar de trabajo y dentro de esta carpeta se debe crear un archivo de JavaScript con el nombre de “script.js”, luego, dentro del archivo debes crear la clase padre mediante la función constructora, la cual llevará el nombre de “FiguraGeometrica”, recibiendo como parámetros las propiedades mencionadas anteriormente “lado”, como se muestra a continuación:

```
function FiguraGeometrica(lado){
  this.lado = lado;
}
```

Ejercicio guiado

Herencia prototípica

Paso 2: Crear la función constructora para la clase de Polígono, pero como esta clase será hija, es decir, hereda de una clase padre “FiguraGeometrica” los atributos y métodos que ella posea, se debe hacer el llamado a las propiedades de la clase padre, para ello se debe implementar el método call(), el cual llama a una función con un valor “this” asignado, además de los argumentos provistos de forma individual, en este caso, los argumentos serán el lado que recibe el Polígono, y se pasara dos veces para completar los valores que necesita la función constructora de nuestra clase padre, para luego ser inicializados y asignados a las propiedades respectivas.

```
function Poligono(lado){
   FiguraGeometrica.call(this, lado);
}
```

Ejercicio guiado

Herencia prototípica

Paso 3: Ya nuestra clase hija “Polígono”, realiza el llamado a las propiedades de la clase padre, por lo que hace falta indicarle a ambas clases que una es extensión de la otra, esto lo lograremos mediante la instrucción “Object.create”, asignando al prototipo de la clase hija “Polígono” como nuevo objeto el prototipo de la clase padre:

```
function Poligono(lado){
  FiguraGeometrica.call(this, lado);
}
Poligono.prototype = Object.create(FiguraGeometrica.prototype);
```

Paso 4: El último paso para poder completar la clase hija con lo indicado en el ejercicio, sería crear un método para el cálculo del área del Polígono, el cual se realizará como se ha trabajado hasta el momento, agregandolo al prototipo de la clase, como se muestra a continuación:

```
Poligono.prototype.calcularArea = function(){
      return this.lado * this.lado;
}
```

Ejercicio guiado

Herencia prototípica

Paso 5: Finalmente, nos queda por instanciar la clase hija, pasar los valores del lado, llamar al método para calcular el área del Polígono y mostrarlo mediante un `console.log()`. Aprovecharemos la instancia que crearemos para el Polígono para mostrarla y observar el contenido de la misma.

```
var cuadrado = new Poligono(3);
console.log(cuadrado);
console.log(cuadrado.calcularArea());
```

Paso 6: Ahora, el resultado lo podemos obtener al ejecutar el archivo “script.js” mediante la terminal con Node, con el comando: `node script.js`, resultando:

```
FiguraGeometrica { lado: 3 }
9
```

Continuando con el ejercicio anterior: Transformando ES5 as ES6

{desafío} latam_

Continuando con el ejercicio anterior

Transformando ES5 as ES6

Continuando con el ejercicio anterior, vamos a transformar ese código escrito en ES5, en un código más actual, implementando las clases de ES6 aplicando herencia. Por lo tanto, debemos seguir los siguientes pasos:

Paso 1: Crear una carpeta en nuestro lugar de trabajo y dentro de esta carpeta se debe crear un archivo JavaScript con el nombre de “script.js”, luego dentro del archivo primeramente debes crear la clase padre “FiguraGeometrica”, recibiendo como parámetros en el constructor las propiedades mencionadas anteriormente “lado”, como se muestra a continuación:

```
class FiguraGeometrica {
   constructor(lado){
     this.lado = lado;
   }
}
```

Continuando con el ejercicio anterior

Transformando ES5 as ES6

Paso 2: Crear la clase hija con el nombre de Polígono, la cual, será una extensión de la clase creada anteriormente denominada FiguraGeometrica, recibiendo en su constructor los valores necesarios para asignar a los atributos, como lo son el alto y el ancho. Luego, mediante la palabra reservada super(), hacemos el llamado al constructor con los atributos de la clase padre y enviamos los valores recibidos en el constructor de la clase hija. Esto es lo que se hizo utilizando el método call() en ES5.

```
class Poligono extends FiguraGeometrica {
   constructor(lado){
     super(lado);
   }
}
```

Continuando con el ejercicio anterior

Transformando ES5 as ES6

Paso 3: Para finalizar con la clase hija, solo hace falta crear el método que nos permita calcular el área del cuadrado. Este método retorna la multiplicación del atributo lado por lado, lo implementaremos mediante la nomenclatura de ES6:

```
class Poligono extends FiguraGeometrica {
   constructor(lado){
     super(lado);
   }
   calcularArea(){
     return this.lado * this.lado;
   }
}
```

Continuando con el ejercicio anterior

Transformando ES5 as ES6

Paso 4: Finalmente, nos queda por instanciar la clase hija, pasar los valores de alto y ancho, llamar al método para calcular el área del Polígono y mostrarlo mediante un console.log(). Aprovecharemos la instancia que crearemos para el cuadrado para mostrarla y observar el contenido de la misma.

```
var cuadrado = new Poligono(3);
console.log(cuadrado);
console.log(cuadrado.calcularArea());
```

Continuando con el ejercicio anterior

Transformando ES5 as ES6

Paso 5: Ahora, el resultado lo podemos obtener al ejecutar el archivo “script.js” mediante la terminal con Node, con el comando: node script.js, resultando:

```
FiguraGeometrica { lado: 3 }
9
```

Continuando con el ejercicio anterior

Transformando ES5 as ES6

En el resultado anterior, se puede observar como la clase hija “Polígono” instanciada en el objeto con nombre “cuadrado”, muestra las propiedades como si fueran propias o creadas en esa clase, ya que al utilizar las palabras reservadas de ES6 “extends” y “super”, automáticamente JavaScript mediante los prototipos apunta directamente al constructor de la clase hija. Esto se puede lograr también en ES5 utilizando la instrucción: Poligono.prototype.constructor = Poligono;.

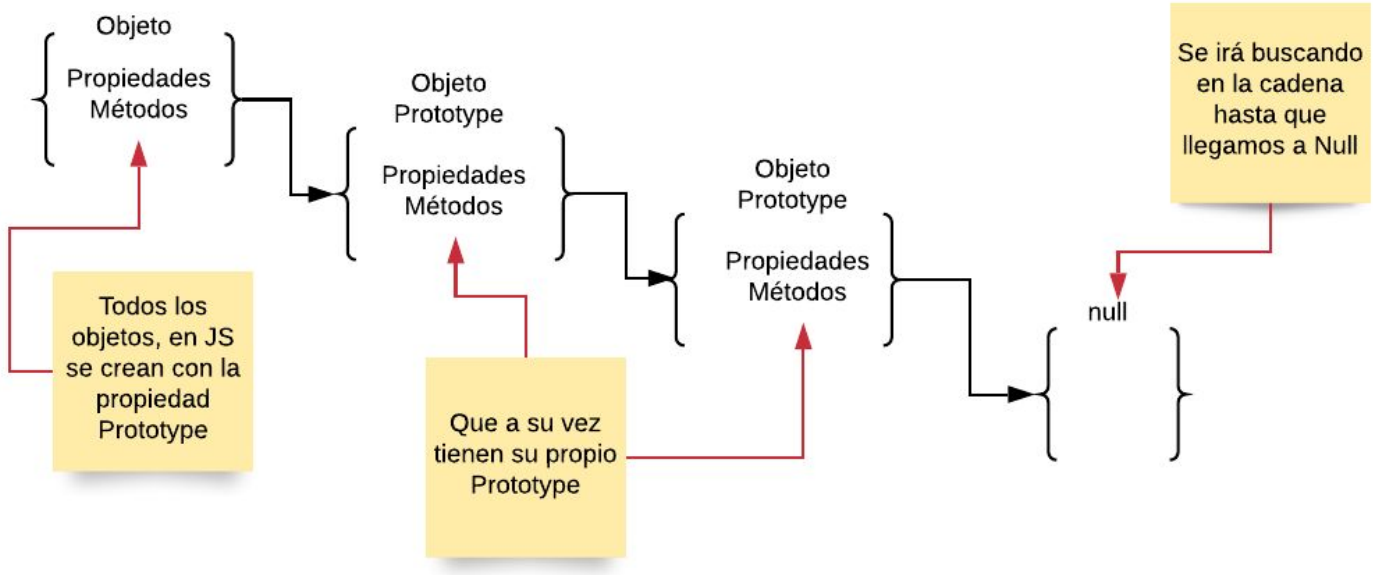
En conclusión, la Herencia en JavaScript funciona mediante prototipos, se puede tener acceso a las propiedades de un objeto desde otro objeto mientras se encuentren relacionados entre si, lo cual facilita compartir las propiedades y métodos de un objeto con otro. Pero hasta el momento solo logramos trabajar con la herencia de prototipos para heredar propiedades, veamos ahora en el siguiente tema mediante la cadena de prototipos como una clase padre hereda sus métodos a otras clases.

/* Qué es una cadena de prototipos */

Qué es una cadena de prototipos

Primeramente, revisemos el concepto de prototipos, los cuales son un mecanismo por el cual un objeto hereda propiedades y métodos de un padre, entonces en JavaScript la herencia funciona por prototipos, tal cual como lo estudiamos en el tema anterior. Ahora bien, en una Cadena de prototipos cada objeto tiene un prototipo, y este prototipo puede tener otro prototipo, así sucesivamente hasta encontrar a un prototipo que no tiene prototipo, llegando a tener un valor nulo “null”. Por definición, null no tiene un prototipo y actúa como el enlace final de esta cadena de prototipos. Tal cual como se muestra en la imagen a continuación:

Qué es una cadena de prototipos



Fuente: Desafío Latam

Qué es una cadena de prototipos

Revisemos ahora un ejemplo, creando un objeto principal con una propiedad y un método, luego crearemos otro objeto (hijo) con el método Object.create(), utilizando el objeto padre como el prototipo del nuevo objeto creado (hijo), luego crearemos otro nuevo objeto siguiendo el mismo proceso para crear el objeto hijo, pero esta vez será un nieto y recibirá el prototipo del hijo.

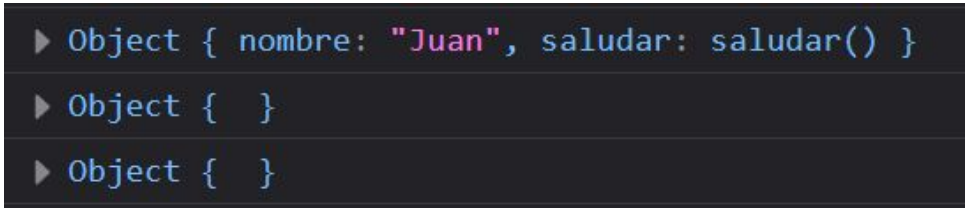
```
var padre = {
  nombre: "Juan",
  saludar: function(){
     console.log("Hola, soy "+this.nombre);
  }
}
console.log(padre);

var hijo = Object.create(padre);
console.log(hijo);

var nieto = Object.create(hijo);
console.log(nieto);
```

Qué es una cadena de prototipos

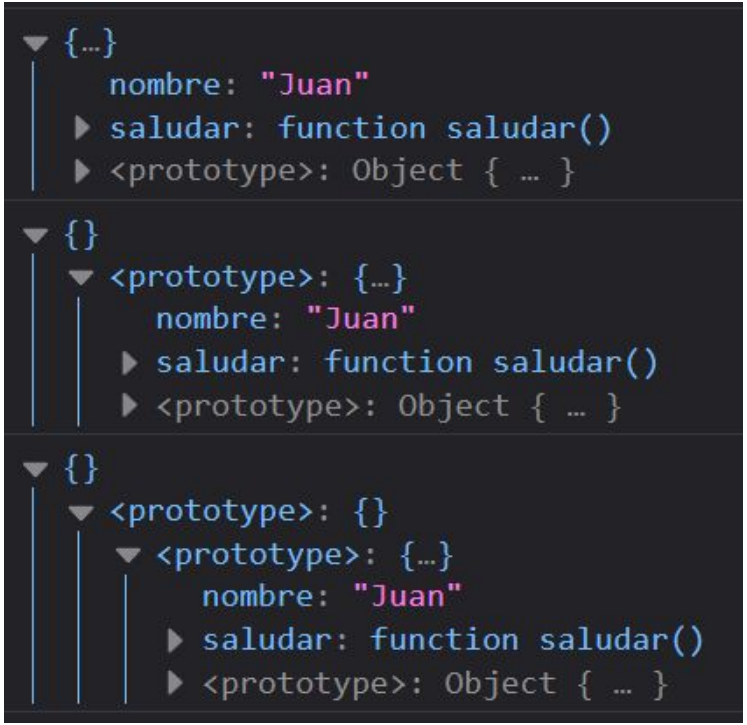
Ahora, al ejecutar el código anterior en la consola del navegador web, el primer resultado sería:



Fuente: Desafío Latam

Qué es una cadena de prototipos

Por lo que podemos observar que el primer objeto perteneciente al “padre”, muestra directamente los valores que posee, mientras que los otros dos objetos (hijo y nieto) aparecen sin ningún tipo de valor porque no se les indicó ningún tipo de propiedad. Pero, al hacer un clic sobre los dos objetos que se encuentran vacíos en la consola del navegador web, el resultado sería:



Fuente: Desafío Latam

/* Múltiples niveles de herencia */

Múltiples niveles de herencia

También te podrías preguntar si es posible hacer herencia multinivel, es decir, tener clases padres, hijos, nietos y así sucesivamente, y la respuesta es sí. Recuerda que los objetos van dejando disponibles sus prototipos a medida que le vamos indicando a cada objeto que tendrá acceso a otro.

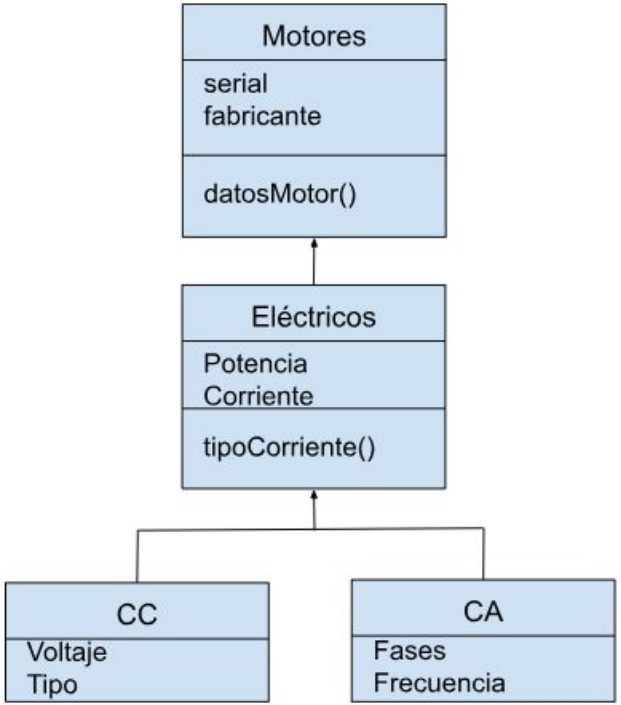
Ejercicio guiado:
Múltiples niveles de herencia

{desafío}
latam_

Ejercicio guiado

Múltiples niveles de herencia

Realizar un programa para un taller de reparación de motores, donde necesitan clasificar los motores en eléctricos, de explosión o a vapor. La empresa solicita crear el programa solo para los motores del tipo eléctrico, los cuales se clasifican dependiendo del tipo de corriente eléctrica que requieran para su funcionamiento, siendo esta Corriente Continua (CC) o Corriente Alterna (AC). Como se muestra en el diagrama de clases a continuación:



Fuente: Desafío Latam

{desafío} latam_

Ejercicio guiado

Múltiples niveles de herencia

Paso 1: Crear una carpeta en nuestro lugar de trabajo y dentro de esta carpeta se debe crear un archivo con el nombre de “script.js”, luego crearemos la clase Motores, con su constructor y método “datosMotor”, para retornar los datos del motor como serial y fabricante.

```
class Motores {
     constructor(serial,fabricante){
        this.serial = serial;
        this.fabricante = fabricante;
     }
     datosMotor(){
        return `El número de serial es:
${this.serial} y el fabricante es la
empresa: ${this.fabricante}`;
     }
}
```

{desafío} latam_

Ejercicio guiado

Múltiples niveles de herencia

Paso 2: Definiremos la clase Eléctricos, la cual extiende de Motores y define el método **"tipoCorriente"**, que retorna el tipo de corriente del motor eléctrico.

```
class Electricos extends Motores {
     constructor(serial, fabricante,
potencia, corriente){
        super (serial, fabricante);
        this.potencia = potencia;
        this.corriente = corriente;
     }
     tipoCorriente(){
        return `El tipo de corriente es:
${this.corriente}`;
     }
}
```

Ejercicio guiado

Múltiples niveles de herencia

Paso 3: Finalmente definiremos las clases `CC` y `CA`, las cuales serán extensiones de la clase Eléctricos, por lo tanto, se puede decir que son nietas de la clase principal denominada Motor. Esto mediante la herencia permitirá acceder desde las clases más inferiores a los métodos de las clases superiores y ejecutarlos para obtener un resultado. Igualmente, es importante destacar que al constructor se le deben ir indicando como parámetros todos los atributos de las clases superiores, así como los propios de cada clase.

```
class CC extends Electricos {
      constructor(serial, fabricante, potencia,
corriente, voltaje, tipo){
         super (serial, fabricante, potencia,
corriente);
         this.voltaje = voltaje;
         this.tipo = tipo;
      }
}

class CA extends Electricos {
      constructor(serial, fabricante, potencia,
corriente, fases, frecuencia){
         super (serial, fabricante, potencia,
corriente);
         this.fases = fases;
         this.frecuencia = frecuencia;
      }
}
```

Ejercicio guiado

Múltiples niveles de herencia

Paso 4: Acceder a las distintas partes de nuestro código. Si generamos una instancia para cada tipo de motor, deberíamos ser capaces de acceder y ejecutar los métodos pertenecientes a las clases superiores.

```
let motorCC = new CC('133DGH', 'GE', '2000W', 'CC', '220CC', 'Shunt');
let motorCA = new CA('7566DGD', 'ABB', '2HP', 'CA', 'Monofasico', '50Hz');
console.log(motorCC.datosMotor());
console.log(motorCC.tipoCorriente());
console.log(motorCA.datosMotor());
console.log(motorCA.tipoCorriente());
```

Ejercicio guiado

Múltiples niveles de herencia

Paso 5: Una vez definida las instancias, podremos ejecutar el código directamente desde la terminal con ayuda del comando `node script.js`. Obteniendo como resultado:

```
El número de serial es: 133DGH y el fabricante es la empresa: GE
El tipo de corriente es: CC
El número de serial es: 7566DGD y el fabricante es la empresa: ABB
El tipo de corriente es: CA
```

/* Propiedades computadas (get y set) */

Propiedades computadas (get y set)

Al referirnos a propiedades computadas, simplemente nos estamos refiriendo al uso de getters y setters dentro de una clase, es decir, serán los métodos que nos permitirán obtener un atributo o modificarlo. En este caso, los métodos “getters” son la forma de definir propiedades computadas de lectura en una clase. Mientras que los métodos “setters” serán las propiedades computadas que permitirán modificar los atributos de una clase. También es importante recordar que el uso de la nomenclatura del guion bajo “_” para los atributos dentro del constructor de la clase al momento de inicializarlos, lo cual permitirá diferenciarlos de los métodos get y set que apliquemos a esos atributos.

¿Qué tema de la clase de hoy te pareció más interesante?

Resumiendo

- Los prototipos y la herencia son un mecanismo por el cual un objeto hereda propiedades y métodos de un padre, entonces en JavaScript la herencia funciona por prototipos.
- Los métodos “getters” son la forma de definir propiedades computadas de lectura en una clase. Mientras que los métodos “setters” serán las propiedades computadas que permitirán modificar los atributos de una clase.

Próxima sesión...

- *Desafío opcional - Lista de animales*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales