/* Herencia */

Ejercicio guiado: Herencia con personal administrativo

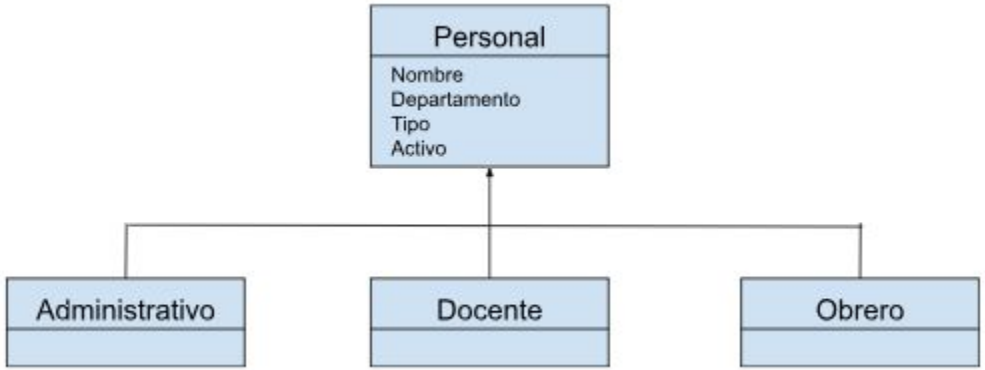
{desafío} latam_

Ejercicio guiado

Herencia con personal administrativo

Una institución de educación solicita que se cree un sistema para centralizar la información del personal que trabaja en ella, como es el caso del personal administrativo, docente y obrero, partiendo de un Diagrama de Clases UML. Se solicita:

Crear las clases, inicializar los constructores para las propiedades donde corresponda y comprobar cómo automáticamente se hereda un constructor sin la necesidad de indicarlo en una clase hija. Finalmente, instanciar y mostrar por consola los valores instanciados en las clases hijas.



Fuente: Desafío Latam

Ejercicio guiado

Herencia con personal administrativo

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 primeramente debes crear la clase padre, la cual llevará el nombre de “Personal”, y dentro de su constructor, las propiedades mencionadas anteriormente “nombre, departamento, tipo y activo”, como se muestra a continuación:

```
class Personal{
   constructor(nombre, departamento, tipo, activo) {
       this.nombre = nombre;
       this.departamento = departamento;
       this.tipo = tipo;
       this.activo = activo;
   }
}
```

Ejercicio guiado

Herencia con personal administrativo

Paso 2: Crear las clases hijas mediante la instrucción “extends” después de la declaración de la clase, para así heredar todos los atributos de la clase padre. Seguidamente, aplicaremos el constructor con la palabra reservada super para llamar y acceder a las propiedades de la clase padre. En este caso, solo aplicaremos el constructor directamente a las clases hijas Administrativo y Docente, mientras que a Obrero, no aplicaremos ningún constructor, vamos a ahora como quedaría el código:

```
class Administrativo extends Personal {
   constructor(nombre, departamento, tipo,
activo){
      super(nombre, departamento, tipo,
activo);
   }
}
class Docente extends Personal {
   constructor(nombre, departamento, tipo,
activo){
      super(nombre, departamento, tipo,
activo);
   }
}
class Obrero extends Personal {
}
```

{desafío} latam_

Ejercicio guiado

Herencia con personal administrativo

Paso 3: Se puede observar en el código anterior, como las clases Administrativo y Docente se extiende de la clase Personal, lo cual permitirá a las clases hijas utilizar las propiedades que ya se tienen declaradas en el constructor la clase padre. Ahora bien, también se puede observar como la clase Obrero no tiene ni un tipo de constructor, es decir, se encuentra completamente en blanco. Pero, gracias a la herencia, esta clase también heredará y podrá utilizar las propiedades o atributos de la clase padre cuando creamos la instancia. Como se muestra a continuación:

```
let admin1 = new Administrativo('Jocelyn','Contenido', 'Fijo', true);
let docente1 = new Docente('Juan','FrontEnd', 'Contratado', true);
let obrero1 = new Obrero('Manuel','Electricidad','Fijo',true);
console.log(admin1);
console.log(docente1);
console.log(obrero1);
```

Ejercicio guiado

Herencia con personal administrativo

Paso 4: Ahora, el resultado lo podemos obtener al ejecutar el archivo “script.js” mediante la terminal con Node, con el comando: `node script.js`, resultando:

```
Administrativo {
  nombre: 'Jocelyn',
  departamento: 'Contenido',
  tipo: 'Fijo',
  activo: true
}
Docente {
  nombre: 'Juan',
  departamento: 'FrontEnd',
  tipo: 'Contratado',
  activo: true
}
Obrero {
  nombre: 'Manuel',
  departamento: 'Electricidad',
  tipo: 'Fijo',
  activo: true
}
```

{desafío} latam_

Ejercicio guiado

Herencia con personal administrativo

Revisando el resultado obtenido anteriormente, se puede apreciar, cómo cada uno de los objetos pertenecientes a las clases hijas, cuenta con las propiedades heredadas de la clase padre, sin tener que repetir cada inicialización de cada propiedad, igualmente como la clase Obrero, a pesar de que no cuenta con un constructor, automáticamente hereda todas las propiedades de la clase padre por ser una extensión.

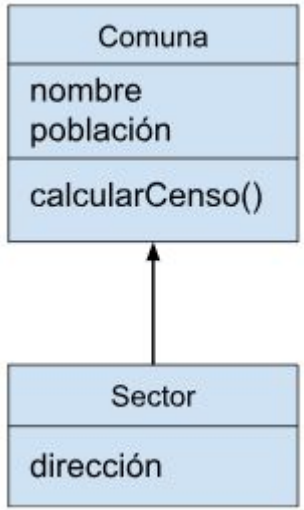
En conclusión, la Herencia nos permite crear una subclase o clase hija que posea automáticamente las propiedades y métodos de la superclase o clase padre. De esta forma podemos reutilizar código y agilizar el proceso de programación. Igualmente, recuerda que un constructor o un método para que puedan llamar a miembros de la clase padre debe utilizar la palabra reservada “super”, que es una referencia a la clase superior. Por otra parte, en el caso de que la clase hija no disponga de un constructor, siempre se incluye un constructor por defecto, que llama al constructor de la clase padre con los parámetros que se hayan pasado al realizar la instancia con new.

Ejercicio guiado: Herencia con comunas y sectores

Ejercicio guiado

Herencia con comunas y sectores

El ejercicio posee dos clases, una clase padre denominada “Comuna”, la cual tiene como atributo “nombre y población”, además de un método denominado “calcularCenso”, que simplemente concatena y mostrará por consola el mensaje “Calculando el censo del sector...”. Por otra parte, existirá la clase “Sector”, la cual será la clase hija y podrá usar los atributos y el método que posee la clase padre. Igualmente, esta clase hija tendrá un propio atributo denominado “dirección”. En la imagen, se muestra el diagrama de clases del ejercicio.



Fuente: Desafío Latam

{desafío}
latam_

Ejercicio guiado

Herencia con comunas y sectores

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 dentro del archivo se debe crear la clase padre mediante la función constructora, la cual llevará el nombre de “`Comuna`”, recibiendo como parámetros las propiedades mencionadas anteriormente “nombre y población”. Igualmente, debemos crear el método “calcularCenso” fuera de la función constructora, mediante el prototype lo asignamos al objeto Comuna, como se muestra a continuación:

```
function Comuna(nombre,poblacion){
  this.nombre = nombre;
  this.poblacion  = poblacion;
};

Comuna.prototype.calcularCenso = function() {
  console.log('Calculando el censo del sector...');
};
```

{desafío} latam_

Ejercicio guiado

Herencia con comunas y sectores

Paso 2: Crear la función constructora para la clase `Sector`, pero como esta clase será hija, hereda de una clase padre "`Comuna`" los atributos y métodos que ella posee, se debe hacer el llamado a las propiedades de la clase padre, para ello se debe implementar el método `call()`, asimismo inicializamos la propiedad interna del objeto "Sector" denominada "direccion".

```
function Sector(nombre,poblacion,direccion) {
     Comuna.call(this, nombre,poblacion);
     this.direccion = direccion;
};
```

Ejercicio guiado

Herencia con comunas y sectores

Paso 3: El objeto hijo "`Sector`", realiza el llamado a las propiedades del objeto padre, por lo que hace falta indicarle a ambos objetos que uno es extensión de otro, esto lo lograremos mediante la instrucción "`Object.create`", asignando al prototipo del objeto hijo "`Sector`" como nuevo objeto el prototipo del objeto padre. Además, anularemos el constructor para que apunte a la función `Sector`; de lo contrario, usaría la propiedad del constructor en `Comuna`.

```
Sector.prototype = Object.create(Comuna.prototype);
Sector.prototype.constructor = Sector;
```

Ejercicio guiado

Herencia con comunas y sectores

Paso 4: Finalmente, nos queda por instanciar el objeto hijo, pasando los valores para las propiedades. Llamando al método que posee el objeto padre denominado "`calcularCenso`".

```
// instancia del objeto
var centro = new Sector('Santiago', 3000000, 'Rebeca Matte 18');
// llamada a método y propiedades
console.log(centro.nombre);
console.log(centro.poblacion);
console.log(centro.direccion);
centro.calcularCenso();
```

Ejercicio guiado

Herencia con comunas y sectores

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

```
Santiago
3000000
Rebeca Matte 18
Calculando el censo del sector...
```

Continuando con el ejercicio anterior:

Continuando con el ejercicio anterior

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 dentro del archivo se debe crear la clase padre “`Comuna`”, recibiendo como parámetros en el constructor las propiedades mencionadas anteriormente “nombre, población”. Así mismo, debemos crear el método denominado “calcularCenso” dentro de la clase padre, para que muestre como string la concatenación de como se muestra a continuación:

```
class Comuna{
   constructor(nombre, poblacion){
     this.nombre = nombre;
      this.poblacion  = poblacion;
   }
   calcularCenso(){
     console.log('Calculando el censo del sector...');
   }
}
```

{desafío} latam_

Continuando con el ejercicio anterior

Paso 2: Crear la clase hija con el nombre de `Sector`, la cual será una extensión de la clase creada anteriormente denominada Comuna, recibiendo en su constructor los valores necesarios para asignar a los atributos, como lo son el nombre y población, además de agregar el atributo propio de la clase hija “direccion”. 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 Sector extends Comuna{
   constructor(nombre, poblacion, direccion) {
     super(nombre, poblacion);
     this.direccion = direccion;
   }
}
```

Continuando con el ejercicio anterior

Paso 3: Finalmente, nos queda por instanciar la clase hija, pasar los valores de nombre, población y dirección. Luego, se hace el llamado al método perteneciente a la clase padre desde el objeto instanciado por la clase hija.

```
var centro = new Sector('Santiago', 3000000, 'Rebeca Matte 18');
console.log(centro.nombre);
console.log(centro.poblacion);
console.log(centro.direccion);
centro.calcularCenso();
```

Continuando con el ejercicio anterior

Paso 4: Ahora, el resultado lo podemos obtener al ejecutar el archivo “script.js” mediante la terminal con Node, con el comando: `node script.js` resultando:

```
Santiago
3000000
Rebeca Matte 18
Calculando el censo del sector...
```

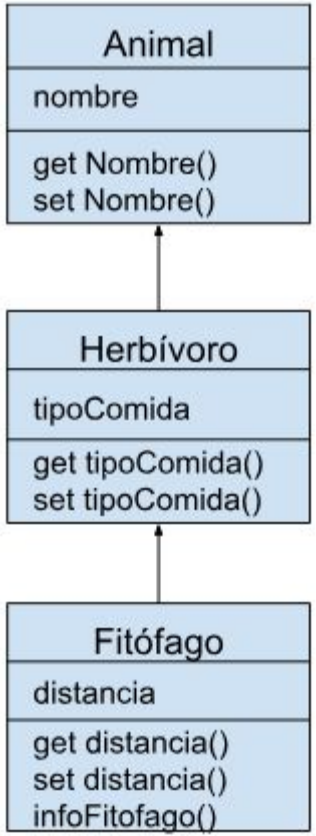
Ejercicio guiado: Múltiples niveles de herencia con getters y setters

{desafío} latam_

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Crear una clase para cada nivel en conjunto con sus atributos y métodos, siendo la clase padre o superior “Animal”, mientras que las otras clases van a ir heredando de la clase padre los atributos y métodos disponibles. Como se muestra en el siguiente diagrama:



Fuente: Desafío Latam

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Llevaremos este diagrama a código, que es bastante intuitivo, ahora que hemos trabajado un poco con herencia. Igualmente, es importante destacar que la clase Fitófago tendrá un método particular, el cual, debe retornar el siguiente mensaje: "`` `${this.nombre} come ${this.tipoComida}, además se mueve ${this._distancia} km diarios` ``"

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 1: Crear la clase Animal, la cual tendrá el método **“get nombre”**, para retornar el nombre y **“set nombre”**, para modificar el nombre.

```
class Animal {
    constructor(nombre) {
        this._nombre = nombre;
    }
    get nombre() {
        return this._nombre;
    }
    set nombre(nombre_nuevo) {
        this._nombre = nombre_nuevo;
    }
}
```

{desafío} latam_

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 2: Definir la clase Herbívoro, la cual se extiende de Animal y define el método **“get tipoComida”**, quien retorna el tipo de comida que tendrá un Animal, así como el método **“set tipoComida”** que podrá modificar el tipo de comida del animal.

```
class Herbivoro extends Animal {
    constructor(nombre, tipoComida) {
        super(nombre);
        this._tipoComida = tipoComida;
    }
    get tipoComida() {
        return this._tipoComida;
    }
    set tipoComida(comidaNueva) {
        this._tipoComida = comidaNueva;
    }
}
```

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 3: Definir la clase Fitófago, la cual define **“get distance”**, permitiendo obtener la distancia que recorre el animal diariamente, al igual que el método **“set distancia”** que permitirá modificar el atributo de distancia. Entonces nos quedará de esta forma.

```
class Fitofago extends Herbivoro {
   constructor(nombre, tipoComida, distancia) {
       super(nombre, tipoComida);
       this._distancia = distancia;
   }
   get distancia() {
       return this._distancia;
   }
   set distancia(distanciaNueva) {
       this._distancia = distanciaNueva;
   }
   infoFitofago() {
       return `${this.nombre} come
${this.tipoComida}, además se mueve
${this._distancia} km diarios`;
   }
}
```

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 4: Acceder a las distintas partes de nuestro código. Si generamos una instancia de Fitofago, con el nombre de conejo, siendo este un animal herbívoro del tipo fitófago, deberíamos ser capaces de obtener tanto el nombre como el tipo de comida, dado que hereda de ambas clases.

```
const conejo = new Fitofago('Pepito','Zanahorias 🥕', '2 metros');
```

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 5: Una vez definida la instancia, podremos acceder a todas las partes del código, por ejemplo, vamos a mostrar el nombre, tipoComida, distancia e infoFitofago.

```
console.log(conejo.nombre);
console.log(conejo.tipoComida);
console.log(conejo.distancia);
console.log(conejo.infoFitofago());
```

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 6: Al ejecutar en la terminal el código anterior con ayuda de Node, nos entregará como resultado, el nombre, lo que come y la distancia, además de una línea con todo concatenado. Es importante que entendamos que el nombre lo obtenemos a partir de la clase Animal, luego lo que come, que está definido en la clase Herbívoro y finalmente la distancia, que está definida en la clase Fitofago, además el resultado de llamar a infoFitofago() que está definido también en Fitofago.

```
Pepito
Zanahorias 🥕
2 metros
Pepito come Zanahorias 🥕 además salta 2 metros
```

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 7: Perfecto, todo funciona bien, veamos otro ejemplo, para este definiremos otra instancia, le llamaremos “conejo2”, pero en este caso será instancia de Herbívoro. Quedando de esta forma:

```
const conejo2 = new Herbivoro('Roger', 'Lechuga 🌿');
```

Paso 8: Ahora ejecutaremos las mismas instrucciones que antes, pero modificando la instancia, agregando también un nuevo nombre mediante el método set, quedando así:

```
console.log(conejo2.nombre);
console.log(conejo2.tipoComida);
console.log(conejo2.distancia);
conejo2.nombre = "PomPom";
console.log(conejo2.nombre);
conejo2.tipoComida = "Maní 🥜"
console.log(conejo2.tipoComida);
```

Ejercicio guiado

Múltiples niveles de herencia con getters y setters

Paso 9: Al ejecutar en la terminal el código anterior con ayuda de Node, nos entregará como resultado, el nombre, dado que es parte de la clase superior Animal, luego nos entrega lo que come, ya que el método es propio de la instancia de clase. Posteriormente, nos retorna "**undefined**" esto se debe a que Herbívoro no es instancia de Fitofago, por consiguiente, al tratar de acceder a algún elemento de una clase de la cual no es instancia, no sabe a qué debe acceder. Igualmente, podremos observar cómo accedemos a los métodos set, modificamos el nombre y el tipo de comida en las clases Animal y Herbívoro respectivamente.

```
Roger
Lechuga 🥬
undefined
PomPom
Maní 🥜
```

/* Propiedades computadas (get y set) */

Ejercicio guiado: Integrando objetos, herencia y HTML

{desafío} latam_

Ejercicio guiado

Integrando objetos, herencia y HTML

Crear un programa en ES6 implementando clases y herencia de clases, el cual recibirá la información desde un formulario para registrar el nombre y la raza de un perro cuando se haga clic sobre el botón de registrar.

Por lo que se deben crear dos clases, una clase padre con el nombre de Animal y otra clase hija con el nombre de Perro.

```
<h1>Registra tu perrito</h1>
<label>Nombre</label>
<input  id="nombre" type="text"/>
<label> Raza</label>
<input id="raza" type="text"/>
<button
id="registrar">Registrar</button>
<div id="data"></div>
<script src="script.js"></script>
```

Ejercicio guiado

Integrando objetos, herencia y HTML

Por lo tanto, para realizar este ejercicio se debe seguir los siguientes pasos:

Paso 1: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

Paso 2: En el index.html debes escribir la estructura básica de un documento HTML incluyendo el formulario indicado en el enunciado del ejercicio, como se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
content="width=device-width,
initial-scale=1.0">
   <title>ES6 - Clases</title>
</head>
<body>
   <h1>Registra tu perrito</h1>
   <label>Nombre</label>
   <input  id="nombre" type="text"/>
   <label> Raza</label>
   <input id="raza" type="text"/>
   <button
id="registrar">Registrar</button>
   <div id="data"></div>
   <script src="script.js"></script>
</body>
</html>
```

{desafío} latam_

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 3: En el archivo script.js definir una clase llamada Animal, con un constructor que recibe el parámetro “nombre” y un método get para retornar el nombre. Entonces nuestro código queda de esta forma:

```
class Animal {
    constructor(nombre) {
        this._nombre = nombre;
    }
    get nombre() {
        return this._nombre;
    }
}
```

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 4: Crear una clase llamada Perro, la cual hereda de Animal, esta tendrá como atributo “raza” en el constructor, al igual que la palabra clave “super” para utilizar ese dato de la clase padre. Luego, se crearán dos métodos, uno con get para obtener la raza y otro con set para modificar la raza. Entonces nuestro código queda de esta forma:

```
class Perro extends Animal {
    constructor(nombre, raza) {
        super(nombre);
        this._raza = raza;
    }
    get raza() {
        return this._raza;
    }
    set raza(raza) {
        this._raza = raza;
    }
};
```

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 5: Captar los valores del HTML mediante DOM, es decir, primeramente los dos botones mediante los “id” de cada uno de ellos, agregando un escucha a cada botón para poder activar funciones individuales que ejecuten procedimientos separados.

```
let registrar = document.getElementById('registrar');
registrar.addEventListener('click',observando);
```

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 6: Definidos los escuchas y las funciones que se deben ejecutar, trabajamos ahora con ellas. La primera función denominada “registrando”, será utilizada para registrar los elementos ingresados por el usuario, es decir, el nombre y la raza. Al igual que instanciar el objeto pasando como argumento al constructor el nombre y a través del setter, el valor correspondiente al atributo “raza”.

```
function registrando() {
      let nombre = document.getElementById("nombre").value;
      let raza = document.getElementById("raza").value;
      var perrito = new Perro(nombre);
      return perrito;
}
```

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 7: En la segunda función denominada “observando”. Se creará una variable para llamar la primera función creada y así tener acceso a la clase y sus métodos. Con esto se puede obtener el valor del nombre y la raza cargados anteriormente al objeto. Luego, se creará un nuevo elemento, en este caso podría ser un “ ” para poder agregar el mensaje donde se muestre el nombre del perro y la raza. Finalmente, se limpiarán los campos (<inputs>) para dejarlos disponibles para otros datos.

```
function observando() {
      const perroData = registrando();
      const nombre = perroData.nombre;
      const raza = perroData.raza;
      const data = document.getElementById('data');
      const p = document.createElement('p');
      p.innerHTML = `🐶 Nombre: ${nombre} - Raza: ${raza}`;
      data.appendChild(p);
      document.getElementById('nombre').value = '';
      document.getElementById('raza').value = '';
}
```

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 8: Al ejecutar el index.html en el navegador web, haciendo doble clic sobre el archivo, veremos lo siguiente:

Fuente: Desafío Latam

Ejercicio guiado

Integrando objetos, herencia y HTML

Paso 9: Ingresemos dos valores, por ejemplo: “Taty y Pastor Alemán”, luego pulsamos el botón de “**Registrar**”:

Fuente: Desafío Latam