POO: Herencia

Introducción

- Supongamos que queremos modelar un programa para gestionar una universidad, que trabaje con Personas, las cuales pueden ser **Estudiantes** o **Profesores**.
- Lo que se nos puede ocurrir, es trabajar con el paradigma de POO, y crear dos clases: Estudiante y Profesor.
- De ambas clases, queremos trabajar con el nombre, apellido, fecha de nacimiento, sexo, y orientación.
- De los estudiantes, el nombre de carrera, y el total de créditos acumulados
- De los profes, el numero de cursos dictados y las materias

Introducción

- Podemos ver que estos objetos (**Estudiante** y **Profesor**), comparten cosas en común.
- Lo que podríamos hacer es crear una clase llamada **Persona** que contenga estos datos
- Y crear dos clases llamadas Estudiante y profesor, que tengan sus propios atributos, y que además “**hereden**” los atributos de la clase Persona.

Concepto de Herencia

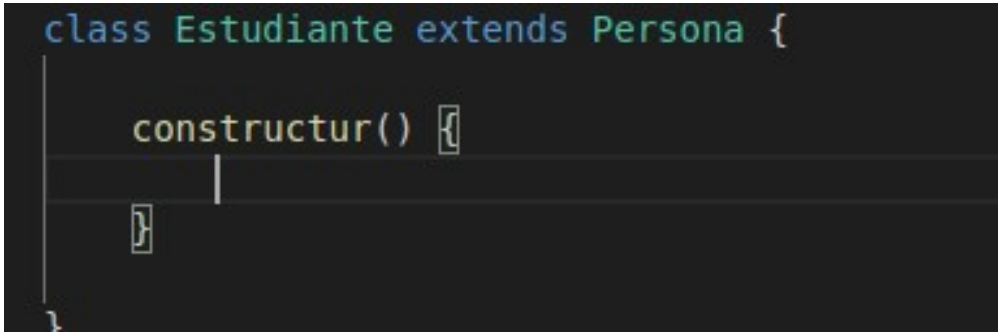
- La herencia es una “técnica” que podemos usar en la lógica de nuestra aplicación orientada a objetos, la cual nos permite crear clases, basadas en una clase ya existente, para poder reutilizar código y abstraer la información.

- Las clases que se construyen a partir de una clase, se les llaman clases hijas, y la clase de la cual heredamos, se le llama clase padre.
- En nuestro ejemplo, Persona sería la clase padre, mientras que Profesor y Estudiante serían las clases hijas, que heredan atributos de la clase Persona.

Herencia

- Una clase sólo puede heredar de otra clase, es decir, no puede haber herencia hacia dos o mas clases.

- Para aplicar herencia en JavaScript, utilizaremos la palabra reservada **extends**



Accediendo al constructor de la clase padre

- Recordemos que Estudiante hereda de Persona
- A la hora de programar el constructor de Estudiante, vamos a querer crear un objeto con todos los datos del estudiante (incluyendo las propiedades heredadas de Persona)
- Para esto, debemos llamar al constructor de la clase padre, usando la palabra reservada **super,** y pasarle como parámetros, las propiedades que heredamos.

super()

- La clase padre, Persona:

```
class Persona {

    constructor(nombre, apellido, fechaNacimiento, sexo, añoIngreso) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.fechaNacimiento = fechaNacimiento;
        this.sexo = sexo;
        this.añoIngreso = añoIngreso;
    }
```

- La clase Estudiante, aplicando herencia y llamando al constructor de la clase Persona:

```
class Estudiante extends Persona {

    constructor(nombre, apellido, fechaNacimiento, sexo, añoIngreso, nombreCarrera, creditos) {
        super(nombre, apellido, fechaNacimiento, sexo, añoIngreso);
        this.nombreCarrera = nombreCarrera;
        this.creditos = creditos;
    }
```

Así quedarían nuestras clases:
La clase Profesor:

```
class Profesor extends Persona {

    constructor(nombre, apellido, fechaNacimiento, sexo, añoIngreso, cursosDictados, materias) {
        super(nombre, apellido, fechaNacimiento, sexo, añoIngreso);
        this.cursosDictados = cursosDictados;
        this.materias = materias;
    }

}
```

Así quedarían nuestras clases:
La clase Estudiante:

```
class Estudiante extends Persona {

    constructor(nombre, apellido, fechaNacimiento, sexo, añoIngreso, nombreCarrera, creditos) {
        super(nombre, apellido, fechaNacimiento, sexo, añoIngreso);
        this.nombreCarrera = nombreCarrera;
        this.creditos = creditos;
    }

    matricularse() {
        return null;
    }

}
```

Así quedarían nuestras clases:
La clase Persona:

```
class Persona {

    constructor(nombre, apellido, fechaNacimiento, sexo, añoIngreso) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.fechaNacimiento = fechaNacimiento;
        this.sexo = sexo;
        this.añoIngreso = añoIngreso;
    }

}
```

Jerarquía a la hora de importar los archivos en nuestro html

- Primero vamos a importar la clase padre.
- Luego, vamos a importar las clases hijas (no importa su orden)
- Y luego, vamos a importar la lógica de nuestro programa.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Estudiantes</title>
</head>
<body>
    <script src="Persona.js"></script>
    <script src="Estudiante.js"></script>
    <script src="Profesor.js"></script>
    <script src="App.js"></script>
</body>
</html>
```

Consigna

- 1) Modificar la clase Persona: agregarle un método que nos retorne un string con las propiedades, para poder mostrarlo en consola o en la página html.

- 2) Crear un objeto del tipo Estudiantes

- 3) Llamar al método creado anteriormente y mostrar los datos del estudiante en la página.

Parte 1)

- Le llamaremos toString() a la función

```
toString() {
    return `<b>Nombre:</b> ${this.nombre} ${this.apellido} <br>
        <b>Sexo:</b> ${this.sexo} <br>
        <b>Año de ingreso:</b> ${this.añoIngreso}`;
}
```

- Backticks (o plantillas de cadenas)
- Nos sirven para insertar o incrustar expresiones dentro de una cadena.

Backticks

- `Texto de la cadena ${una expresión}`
- Ejemplo:
- const nombre = “Juan”
- console.log(`El nombre es ${nombre}`);

- Nos imprime: El nombre es Juan

- Podemos reemplazar la concatenación de varios strings con el operador +, por el uso de backticks.

Etiquetas y
 de HTML

```
toString() {
    return `<b>Nombre:</b> ${this.nombre} ${this.apellido} <br>
        <b>Sexo:</b> ${this.sexo} <br>
        <b>Año de ingreso:</b> ${this.añoIngreso}`;
}
```

La etiqueta **** se utiliza para poner un texto en negrita: Texto

La etiqueta **
** se utiliza para dar un salto de línea. (No es necesario cerrarla)

Parte 2)

- En la lógica de nuestro programa (App.js), vamos a crear una instancia de la clase Estudiantes.

App.js > ...

```
1
2   const e = new Estudiante("Juan", "Perez", "2001-01-01",
3       "Hombre", 2017, "Ing. Computación", 105);
4
5   document.write(e.toString())
6
7
```

Parte 3) Mostrar los datos en el documento HTML

- Para esto, utilizaremos un objeto llamado **document**
- De este objeto, llamaremos a la propiedad write, la cual escribe html en nuestro documento.
- Mas adelante en el curso trabajaremos mas en profundidad con este objeto y todos sus métodos.

document.write(“Texto a escribir en formato html”)

Parte 3) Mostrar los datos en el documento HTML

- Como el método toString nos devuelve una cadena con etiquetas html, lo que haremos será mostrar en el html esta cadena que obtenemos, usando document.write()

  ```
  const e = new Estudiante(params...);
  ```

- ```
  document.write(e.toString());
  ```

- Así se verá nuestro documento html:

Nombre: Juan Perez
Sexo: Hombre
Año de ingreso: 2017
  ```

Bibliografía

- https://ifgeekthen.everis.com/es/herencia-en-programacion-orientada-objetos
- https://www.netmentor.es/entrada/herencia-poo