Herencia

Polimorfismo

{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?

- *Aplica el mecanismo de polimorfismo para la reutilización de componentes en el contexto de la Programación Orientada a Objetos.*
- *Utiliza closures para reducir el alcance de variables.*

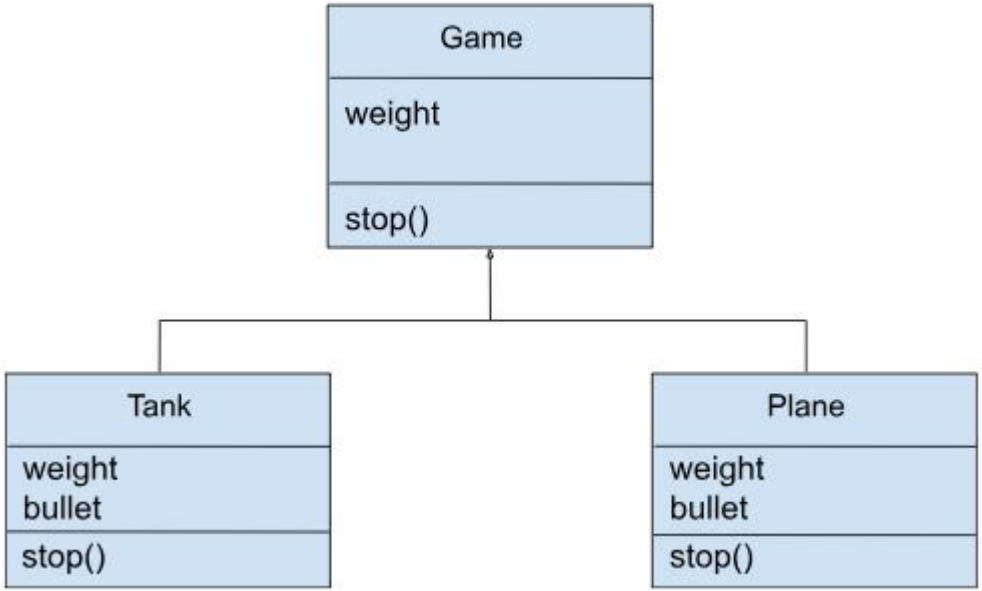
Con tus palabras: ¿Para qué imaginas que nos sirve el polimorfismo y closure?

/* Polimorfismo */

Polimorfismo

Es la posibilidad de llamar métodos en común de diferentes objetos sin alterar los resultados.

Por ejemplo, si estamos en un videojuego, el cual será nuestra superclase, donde existen dos subclases como tanques y aviones, ambos objetos pueden tener propiedades y métodos con el mismo nombre, como se muestra a continuación:



Fuente: Desafío Latam

Ejercicio guiado: Polimorfismo

{desafío} latam_

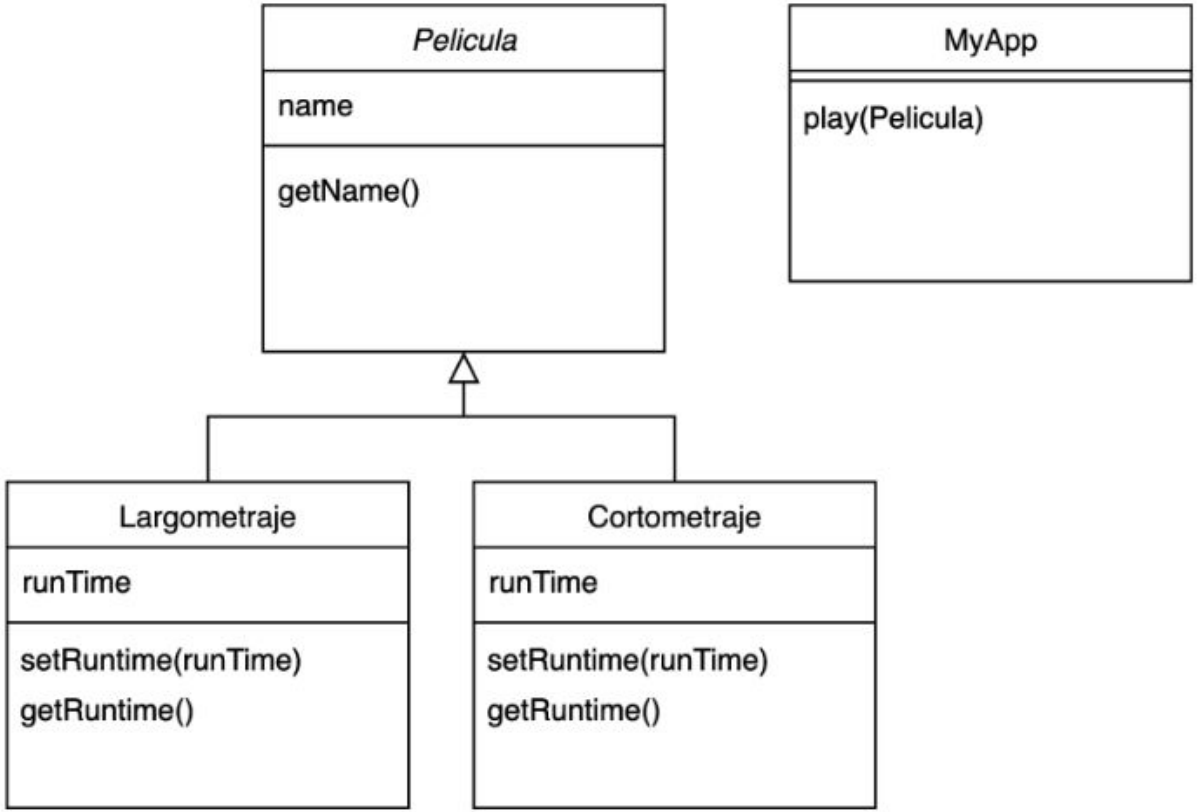
Ejercicio guiado: Polimorfismo

Implementar una aplicación que muestre películas y series bajo demanda, llamada “MyApp”. Por consiguiente, partiremos con un modelo de herencia simple, como se muestra en la imagen a continuación. En el modelo, en primer lugar tendremos una clase denominada “Película”, la cual, define el nombre de las películas, de esta clase heredarán las clases “Largometrajes” y “Cortometrajes”, cada una de estas tienen como atributo los tiempos de duración de cada una (“runtime”), siendo un método que permite obtener o modificar el valor, mediante “get” y “set”.

Además, tendremos una clase llamada “MyApp”, la cual será la clase donde se ejecuta un método “play” (reproducir la película), este método “play” recibirá como parámetro una instancia de la clase película, en otras palabras, puede recibir un Largometraje, Cortometraje o una película en sí. Por lo tanto, esta clase no tendrá constructor alguno y solo el método directo “play” que retorna una cadena de texto o string con el mensaje: la película “nombre de la película” se está reproduciendo… tiene una duración de “tiempo de la película”.

Ejercicio guiado

Polimorfismo



Fuente: Desafío Latam

Ejercicio guiado

Polimorfismo

Para dar inicio a la solución del ejemplo planteado, se deben seguir los siguientes pasos:

Paso 1: Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crear un archivo con el nombre de script.js. Luego, en este mismo archivo define la clase llamada “Película”, la cual recibe como parámetro el título “title” en su constructor y tendrá un método “get” para regresar el título de la película.

```
class Pelicula {
    constructor(name){
        this._name = name;
    }
    getName() {
        return this._name;
    }
}
```

Ejercicio guiado

Polimorfismo

Paso 2: Definir la clase “**Largometraje**”, la cual extiende de “**Película**”, es decir, hereda las características de la clase “Película” y recibe como parámetros “`name`” (el que hereda de Película), y “**runtime**” (la duración de la película). Seguidamente, se agregan los dos métodos indicados en la imagen número 2, un get para regresar el valor de la duración de la película y otro método set para modificar ese mismo valor, quedando de esta forma nuestro código:

```
class Largometraje extends Pelicula {
   constructor(name, runTime) {
       super(name);
       this._runTime = runTime;
   }
   getRuntime() {
       return this._runTime;
   }
   setRuntime(runTime) {
       this._runTime = runTime;
   }
}
```

Ejercicio guiado

Polimorfismo

Paso 3: Definir la clase “**Cortometraje**”, siendo hija de la clase “Película”, recibiendo en su constructor como parámetros **name** y **runtime**. El parámetro **title** será enviado con la palabra clave “super” a la clase padre, mientras que el **runtime** se asignará a una variable en el constructor. Posteriormente, se deben agregar los dos métodos a la clase, como lo son el método `get runtime()` para regresar el valor de la duración de la película y el método `set runtime(runtime)` para modificar ese valor.

```
class Cortometraje extends Pelicula {
    constructor(name, runTime) {
        super(name);
        this._runTime = runTime;
    }
    getRuntime() {
        return this._runTime;
    }
    setRuntime(runTime) {
        this._runTime = runTime;
    }
}
```

Ejercicio guiado

Polimorfismo

Paso 4: Definir la clase "**MyApp**", la cual define el método "**play**" recibiendo como parámetro un valor denominado "Pelicula" y retornando el mensaje: `` `la película ${Pelicula.getName()} se está reproduciendo...tiene una duración de ${Pelicula.getRuntime()}`; `` almacenado en una variable cualquiera, quedando nuestro código de esta forma.

```
class MyApp {
   play(Pelicula) {
       const result = `la película ${Pelicula.getName()} se está
reproduciendo...tiene una duración de ${Pelicula.getRuntime()}`;
       return result;
   }
}
```

{desafío} latam_

Ejercicio guiado

Polimorfismo

Paso 5: Generar las instancias de los objetos necesarios, primero definiremos la instancia de Largometraje y Cortometraje, quedando, por ejemplo, de esta forma:

```
const largometraje = new Largometraje('Sin City', '105min');
const cortometraje = new Cortometraje('Hulk vs Wolverine', '20min');
```

Ejercicio guiado

Polimorfismo

Paso 6: Definir una instancia de la clase MyApp, así como una constante llamada playing, en la cual, llamaremos a la instancia de MyApp, llamando al método play, pasando como argumento el objeto largometraje, para luego imprimir la constante playing, quedando de esta forma.

```
const myApp1 = new MyApp();
const playing = myApp1.play(largometraje);
console.log(playing);
```

Paso 7: Al ejecutar el código realizado hasta el momento en el navegador web, el resultado en la consola sería:

```
> const playing = myApp1.play(largometraje)
  console.log(playing);
  la película Sin City se está reproduciendo...tiene una duración de 105min
```

Fuente: Desafío Latam

Ejercicio guiado

Polimorfismo

Paso 8: Hacemos lo mismo que el paso anterior, pero para el objeto que instanciamos para Cortometraje, guardando en una variable el resultado del llamado y luego mostrando por consola la variable, quedando de esta forma.

```
const playing2 = myApp1.play(cortometraje);
console.log(playing2);
```

Paso 9: Al ejecutar el código realizado hasta el momento en el navegador web, el resultado en la consola sería:

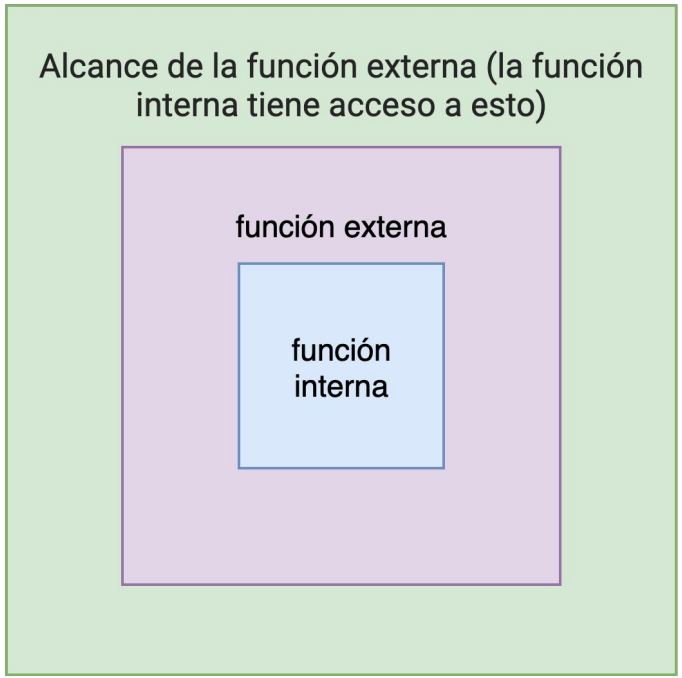
```
> const playing2 = myApp1.play(cortometraje);
  console.log(playing2)
  la película Hulk vs Wolverine se está reproduciendo...tiene una duración de
  20min
```


Fuente: Desafío Latam

/* Closures */

Closures

- Es una función en donde no existen variables, es decir, el closure no tiene variables propias.
- Es una especie diferente de objeto, ya que está compuesto por dos cosas, primero una función y luego del entorno donde se creó dicha función.
- El Scope es el alcance que puede tener una variable en el código.



Closures

Tipos de scope: global y local

1. Declarar una función denominada global y dentro de ella mostrar el valor de una variable declarada fuera de la función.

```
var a = 1;
function global() {
 console.log(a); // 1
}
```

Closures

2. Realizar el llamado a la función, posteriormente se volverá a mostrar el valor de la variable declarada al principio.

El resultado en la consola será igual en ambos casos, es decir, 1. Porque la variable está declarada de forma global y no importa si la utilizamos dentro de la función o fuera de la función.

```
global();
console.log(a); // 1
```

Ejercicio guiado: Closures

{desafío} latam_

Ejercicio guiado

Closures

- Utilizar dos funciones, una dentro de otra, para observar el comportamiento de la variable dentro y fuera de ambas funciones.

- Utilizar variables, pero dentro de la función y observar cómo reacciona el código.

Sigamos los siguientes pasos:

Paso 1: inicializa una variable con el nombre de “a” y el valor número de “1”, luego crea una función con el nombre de “`global`”.

```
let a = 1;
function global(){}A
```

{desafío} latam_

Ejercicio guiado

Closures

Paso 2: ahora dentro de la función global se llama a la variable global con el nombre de “a”, seguidamente se crea una nueva función con el nombre de “interno”, la cual, solamente mostrará la variable global “a” y se llamará dentro de la misma función global. Como se muestra en el código a continuación:

```
let a = 1;
function global() {
 console.log(a);
 function interno(){
   console.log(a);
 }
 interno();
}
```

Ejercicio guiado

Closures

Paso 3: Al ejecutar el código anterior en la consola del navegador web, mediante la instrucción global(); y luego volver a mostrar el valor de la variable local “a” con: console.log(a); encontraremos tres veces el número 1, es decir, la variable global siempre estará disponible no importa desde donde se invoque.

```
let a = 1;
function global() {
 console.log(a);
 function interno(){
   console.log(a);
 }
 interno();
}
```

Ejercicio guiado

Closures

Ahora realicemos el ejercicio utilizando variables pero dentro de la función y observemos cómo reacciona nuestro código, para ello se deben realizar los siguientes pasos:

Paso 1: Crea una función con el nombre de “local”, y dentro de ella inicializa una variable denomina “a” con el valor número de “2”.

```
function local() {
 var a = 2;
}
```

Ejercicio guiado

Closures

Paso 2: Ahora se debe mostrar dentro de la función local el valor de la variable creada bajo el nombre de “a” mediante un console.log(a). Seguidamente, fuera de la función, se debe hacer el llamado a la función local por su nombre y mostrar una vez más el valor de la variable “a” en la consola del navegador web.

```
function local() {
 var a = 2;
 console.log(a);
}
local();
console.log(a)
```

Ejercicio guiado

Closures

Paso 3: Si el código anterior es ejecutado en al consola del navegador web, el resultado cambiará para ambas salidas, es decir, para el console.log(a) dentro de la función, el resultado será 2, pero para el console.log(a) invocado fuera de la función, el resultado será `Uncaught ReferenceError: a is not defined`. Esto se debe a que la variable “a” se encuentra declarada dentro de la función, es decir, de forma local, por lo tanto no se puede acceder a ella fuera de su entorno, sin importar si es declarada con let o con var.

Ejercicio guiado

Closures

Paso 4: Posteriormente, se agrega una nueva función con el nombre de `interna` dentro de la función local y se intenta llamar a la variable creada en la función externa:

```
function local() {
 let a = 2;
 function interna () {
      console.log(a);
 }
 console.log(a);
 interna();
}
local();
```

Ejercicio guiado

Closures

Paso 5: Al ejecutar el código anterior en la consola del navegador web, el resultado sería igual para ambas salidas, es decir, el número 2. Ya que la variable que se encuentra declarada dentro de la función con el nombre de local, ahora pasará a ser una variable global, pero dentro del scope de esa función local, por esta razón la función interna puede llamar y/o utilizar la variable declarada en la función externa.

¿Puedes dar un ejemplo con los conceptos que acabamos de aprender?

Resumiendo

- El polimorfismo ss la posibilidad de llamar métodos en común de diferentes objetos sin alterar los resultados.
- Los clousures son una especie diferente de objeto, ya que está compuesto por dos cosas, primero una función y luego del entorno donde se creó dicha función.

Próxima sesión...

- *Aplica el patrón de módulo en JavaScript para hacer el código más ordenado, mantenible y reutilizable.*
- *Identifica los elementos fundamentales que componen la jerarquía del Document Object Model.*
- *Utiliza instrucciones Javascript para la manipulación de un objeto utilizando la jerarquía del DOM.*
- *Explica el rol de los eventos dentro del DOM reconociendo los más frecuentes.*
- *Utiliza instrucciones Javascript para la definición de un comportamiento simple ante un evento en el DOM determinado.*

{desafío}
latam_

Academia de
talentos digitales