Propiedades de la programación orientada a objetos en JS

Mentor: Joshua Eduardo González Ruíz

OBJETIVO

Aprender sobre las propiedades de la programación orientada a objetos en JS.

Utilizarlos para realizar procesos de forma adecuada en el código.

ÍNDICE

Abstracción

Es un proceso que permite ocultar los detalles de implementación y mostrar solo la funcionalidad a todos los usuarios. En palabras simples, podemos decir que ignora los detalles irrelevantes y muestra solo los necesarios.

- Reduce la duplicación del código
- No se puede crear una instancia de una clase abstracta

```
1   //Creating a constructor function
2   function Vehicle()
3   {
4       this.vehicleName="vehicleName";
5       throw new Error("An instance of Abstract Class cannot be created");
6   }
7
8   Vehicle.prototype.display=function()
9   {
10      return "Vehicle is: "+this.vehicleName;
11  }
12
13  //Creating a constructor function
14  function Bike(vehicleName)
15  {
16      this.vehicleName=vehicleName;
17  }
18
19  //Creating object without using the function constructor
20  Bike.prototype = Object.create(Vehicle.prototype);
21
22  var bike=new Bike("Suzuki");
23
24  console.log(bike.display());
```

```
Vehicle is: Suzuki
Hint: hit control+c
>
```

Encapsulación

Es un proceso de enlace de los datos con las funciones que actúan sobre los mismos datos y permite controlar y validar los datos.

Hay dos métodos para lograr una encapsulación en JavaScript:

- Mediante el uso de métodos "setter" para establecer los datos y métodos "getter" para recibir esos datos.
- Usando la palabra clave "let" para hacer que los miembros de datos sean privados.

Usando las siguientes propiedades, la encapsulación en JavaScript nos permite manejar un objeto:

- **Solo escritura:** en este caso, solo se utilizan los métodos setter.
- **Lectura/escritura:** en esta propiedad, el método getter se usa para leer los datos y el método setter se usa para establecer los datos.
- **Solo lectura:** solo los métodos getter son utilizados en este caso.

```
class Student {
  constructor() {
    var name;
    var marks;
  }
  getName() {
    return this.name;
  }
  setName(name) {
    this.name = name;
  }

  getMarks() {
    return this.marks;
  }
  setMarks(marks) {
    this.marks = marks;
  }

}
var stud = new Student();
stud.setName("Joshua");
stud.setMarks(95);
console.log(stud.getName() + " " + stud.getMarks());
```

```
Joshua 95
Hint: hit control+c anytime to enter REPL.
>
```

Polimorfismo

Es básicamente un concepto central del paradigma orientado a objetos que proporciona una forma para que una sola acción se pueda realizar en diferentes formas.

Proporciona la capacidad de llamar al mismo método en varios objetos de JavaScript. Dado que JavaScript es un lenguaje de tipado dinámico, los usuarios pueden pasar cualquier tipo de miembros de datos con los métodos.

```
1  class A
2    {
3      display()
4     {
5       console.log("space is invoked");
6     }
7    }
8
9  class B extends A
10   {
11   }
12
13 var b=new B();
14
15 b.display();
```

```
space is invoked
Hint: hit control+c anytime to enter REPL.
>
```

Herencia

Es básicamente un mecanismo que permite a los usuarios crear nuevas clases sobre la base de las clases ya existentes.

- Brinda flexibilidad a la clase secundaria para reutilizar los métodos y las variables de una clase principal.
- Para crear una clase secundaria sobre la base de una clase principal, se utiliza la palabra clave "extends".
    - "extends" facilita que su clase secundaria obtenga todas las propiedades y el comportamiento de su clase principal.

Puntos importantes:

- Todas las propiedades y el comportamiento del objeto incorporado, así como las clases personalizadas, se pueden adquirir utilizando la palabra clave "extends".
- El usuario también puede utilizar un enfoque basado en prototipos para lograr la herencia.
- La herencia de JavaScript mantiene una relación IS-A.
- La palabra clave "extends" se usa generalmente en expresiones de clase o declaraciones de clase.

```
1   class DateTime extends Date
2   {
3     constructor(year)
4     {
5       super(year);
6     }
7   }
8
9   var m = new DateTime("November 15,2019 20:22:10");
10
11  console.log("Year value:");
12
13  console.log(m.getFullYear());
```

```
Year value:
2019
Hint: hit control+c anytime to enter REPL.
>
```

Referencias

- Abstraction in JavaScript – Object Oriented Program. (n.d.). PHPTPOINT. Retrieved January 17, 2022, from https://www.phptpoint.com/javascript-abstraction/
- Encapsulation in JavaScript – A Object Oriented Program. (n.d.). PHPTPOINT. Retrieved January 17, 2022, from https://www.phptpoint.com/javascript-encapsulation/
- Polymorphism in JavaScript – Learn How to Implement. (n.d.). PHPTPOINT. Retrieved January 17, 2022, from https://www.phptpoint.com/javascript-polymorphism/
- Inheritance in JavaScript with Step by Step Example. (n.d.). PHPTPOINT. Retrieved January 17, 2022, from https://www.phptpoint.com/javascript-inheritance/