cedia | TMO Training Management Office

info@cedia.org.ec
cedia.edu.ec

CURSO DE JAVASCRIPT

Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia

Ing. María Fernanda Granda J., PhD

19-11-2025

"Conectando ideas, transformando sociedades"

Módulo 3: Programación Orientada a Objetos

- Prototipos de Objetos en JavaScript
- Prototipos de Java Script
- Programación Orientada a Objetos
- ¿Qué es la POO?
- Bases de la POO
- Creación de clases
- Encapsulamiento
- Herencia

Prototipos de objeto de Javascript

Javascript es un lenguaje basado en objetos basados en prototipos. Hay algunas características

- Cada objeto JavaScript tiene un prototipo. El prototipo también es un objeto.
- Todos los objetos Javascript heredan propiedades de sus prototipos.
- Define y crea un conjunto de objetos con funciones constructoras.
- Puede adicionar o eliminar propiedades dinámicamente a objetos individuales o el entero conjunto de objetos.
- Hereda propiedades siguiendo la cadena de prototipos.

Ejemplo de Creación de Prototipo

```
function Estudiante(pnombre, pcurso, pnota){
    this.nombre=pnombre;
    this.curso=pcurso;
    this.nota=pnota;
}
```

Usando el prototipo:

```
let estudiante1= new Estudiante ("Ana", "JavaScript", 98);
let estudiante2= new Estudiante ("Pedro","HTML", 97);
```

Prototipos de JavaScript

- El prototipo es un objeto especial que sirve como modelo del que otros objetos pueden heredar propiedades y métodos.
- Es una especie de plantilla o “objeto padre”.
- Todo objeto en JavaScript tiene un enlace interno a un prototipo, llamado [[Prototype]].
- A través del prototipo se implementa la herencia prototípica.

Prototipos de JavaScript

- Todos los objetos Javascript (Date, Array, RegExp, Function, ...) heredan desde Object.prototype.
- La propiedad del prototipo permite adicionar propiedades y métodos a un objeto.
- El valor que posee el prototipo de un objeto es algunas veces llamado el enlace interno del prototipo. Este ha sido llamado siempre _proto_
- Declarando una propiedad del prototipo:

```
objeto.prototype.nombre=valor;
o
objeto._proto_.nombre=valor;
```

Ejemplo de Propiedades de Prototipos

```
function empleado(nombre,departamento){
    this.nombre=nombre;
    this.departamento=departamento;
}

empleado.prototype.salario=20000;

let xy=new empleado("Máximo", "TI");
console.log(xy.salario);
```

Ejemplo de Propiedades de Prototipos

```
function Persona(nombre) {
  this.nombre = nombre;
}

Persona.prototype.saludar = function() {
  console.log("Hola, soy " + this.nombre);
};

const p1 = new Persona("Carlos");
p1.saludar();  // Hereda el método desde el prototipo
```

- Persona.prototype es el prototipo.
- p1 es un objeto que hereda el método saludar.

Ejemplo de Propiedades de Prototipos

- El objeto contiene datos y métodos propios.
- El prototipo es un objeto auxiliar del que otros objetos heredan comportamiento.
- Es la base del modelo de herencia de JavaScript.

Concepto	Qué es	Para qué sirve
Objeto	Instancia real con datos	Usar y manipular valores
Prototipo	Plantilla de la cual otros objetos heredan	Compartir comportamiento sin duplicar código

Programación Orientada a Objetos

- Es un **estilo** que define cómo se **estructuran y escriben los programas**.
- Cada paradigma establece **reglas** sobre cómo los desarrolladores pueden crear, **organizar y manipular** el código para resolver problemas.
- Ejemplos: **programación imperativa, orientada a objetos, funcional y lógica,** cada una con **sus propias características y principios**.
- Comprender los paradigmas es crucial para **seleccionar el enfoque más adecuado** al resolver problemas y diseñar sistemas de software eficientes y mantenibles.

¿Qué es la Programación Orientada a Objetos?

- Es un **paradigma** que **organiza el código en objetos**, cada uno con sus propios datos y comportamientos (métodos o funcionaes).
- Promueven conceptos como **abstracción**, **encapsulamiento**, **herencia** y **polimorfismo** para facilitar el desarrollo de software modular, reutilizable y fácil de mantener.
- En POO, los objetos **interactúan** entre sí mediante **mensajes**, lo que permite una interacción más cercana de los conceptos del mundo real y una mejor gestión de la complejidad de los programas.

Bases de la POO

- **Clases**: plantillas para crear objetos. Definen propiedades (atributos) y comportamiento (métodos). Encapsulan datos y funcionalidades que permiten la reutilización de código y una programación modular y la abstracción.
- **Atributos**: atributos que describen el estado del objeto
- **Métodos**: serie de acciones o comportamientos asociados al objeto.
- **Objetos**: instancias específicas de una clase que **encapsulan datos** y **comportamientos** relacionados.

Bases de la POO

- **Abstracción**: permite representar entidades del mundo real como objetos con características y comportamientos relevantes para el problema que se está resolviendo. Esto simplifica la complejidad del sistema.
- **Herencia**: permite que una clase hija (subclase) herede atributos y métodos de la clase padre (superclase). Promueve la reutilización del código y la organización jerárquica de las clases.

Bases de la POO

- **Encapsulamiento**: los detalles internos de un objeto deben estar ocultos fuera de su definición y solo deben ser accesibles a través de una interfaz claramente definida. Esto promueve la seguridad y la integridad de los datos al prevenir accesos no autorizados y facilita el mantenimiento del código.
- **Polimorfismo**: es la capacidad de objetos de diferentes clases de responder al mismo mensaje de manera diferente. Ejemplo, un mismo método o mensaje puede producir diferentes resultados según el tipo de objeto que reciba. Esto permite escribir código más genérico y reutilizable.

Versiones detrás de Escenas

- Antes de ES6, no se tenía clases formales. Se usaban funciones constructoras y prototipos para simular la POO.
- Con ES6 se introdujo la sintaxis class, aunque internamente sigue usando prototipos.

Creación de Clases

- La forma más sencilla

```
//Clase es una plantilla
class Persona {
  // constructor es el método especial para instanciar objetos de la clase
    constructor(nombre, edad, profesion){
        this.nombre=nombre;
        this.edad=edad;
        this.profesion=profesion;
    }
}
// De esta forma estamos instanciando objetos basados en la plantilla (clase)
const carolina=new Persona("Carolina", 35, "Ingeniera");
const ricardo=new Persona("Ricardo", 45, "Médico");

//GET:Obteniendo los datos del Objeto
console.log("Edad de Carolina: "+carolina.edad);
console.log(ricardo.profesion);

//SET: cambiar información del objeto
carolina.edad=36;
ricardo.profesion="Cirujano";
```

Encapsulación

En ES2022 se agregó la posibilidad de hacer privadas las características utilizando # al comenzar el nombre.

Vamos a :

- cambiar las características a privadas y
- crear métodos get (obtener) y set (establecer) para cada característica de la clase.

```
class Persona {
    #nombre
    #edad
    #profesion
    constructor(nombre, edad, profesion){
        this.#nombre=nombre;
        this.#edad=edad;
        this.#profesion=profesion;
    }
    //GETTERS
    obtenerEdad(){
        return this.#edad;
    }
    obtenerProfesion(){
        return this.#profesion;
    }
    //SETTERS
    establecerEdad(edad){
        this.#edad=edad;
    }
    establecerProfesion(profesion){
        this.#profesion=profesion;
    }
}
```

Encapsulación

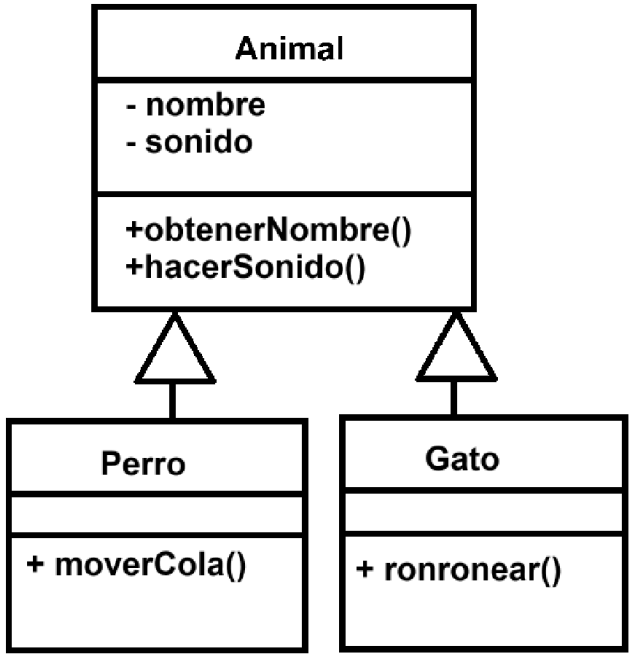
- Ejemplo de utilizar los datos encapsulados. Se tiene que acceder a través de los métodos get y set

```
// creando las instancias de los objetos
const carolina=new Persona("Carolina", 35, "Ingeniera");
const ricardo=new Persona("Ricardo", 45, "Médico");

//GET:Obteniendo los datos del Objeto
console.log("Edad de Carolina: "+carolina.obtenerEdad());
console.log("Profesión Ricardo :"+ricardo.obtenerProfesion());

//SET: cambiar información del objeto
carolina.establecerEdad(36);
ricardo.establecerProfesion("Cirujano");
```

Herencia

Ejemplo del uso de herencia.

Crear una jerarquía de clases




```
class Animal {
    // características de la clase padre
    #nombre
    #sonido
    // método constructor
    constructor (nombre, sonido){
        this.#nombre=nombre;
        this.#sonido=sonido;
    }
    // métodos que son comunes para todas subclases
    obtenerNombre(){
        return this.#nombre;
    }
    hacerSonido(){
        console.log(`${this.#nombre} hace: ${this.#sonido}`);
    }
}
```

Herencia

```
// hereda de la clase Animal
class Perro extends Animal{
    constructor(nombre){
        super(nombre, "Guau");  //ejecuta el método de la clase padre
    }
    // este método es particular para la clase Perro
    moverCola(){
        console.log(`${this.obtenerNombre()} está moviendo la cola feliz`);
    }
}
```

Herencia

```
// hereda de la clase Animal
class Gato extends Animal {
    constructor(nombre){
        super(nombre, "Miau"); //ejecuta el método de la clase padre
    }
    // este método es particular para la clase Gato
    ronronear(){
        console.log(`${this.obtenerNombre()} está ronroneando contento`);
    }
}
```

Herencia

```
// el uso de la herencia
const perro=new Perro("Manchita");
// hacer sonido se puede usar en ambos ya que lo heredan de animal
perro.hacerSonido();
perro.moverCola();
console.log("------------------");
const gato=new Gato("Pelusa");
gato.hacerSonido();
gato.ronronear();
```

Practicar la herencia

- Crear un formulario web donde el usuario ingrese datos de una Persona o un Estudiante.
- Se usen clases en JavaScript con herencia (extends) y se muestra el resultado en una tabla.
- Se validen campos obligatorios:
  - Nombre no vacío y mínimo 2 caracteres.
  - Edad entre 1 y 120.
  - Matrícula obligatoria solo si es estudiante.
  - Matrícula alfanumérica (3–10 caracteres).
- Se muestran errores en rojo debajo del formulario.
- Se evite agregar la fila si hay errores.

cedia | TMO Training Management Office

GRACIAS

cedia.edu.ec
info@cedia.org.ec
(+593) 7-407-9300