Programación Orientada a Objetos en JavaScript

Tema 5 — POO y Gestión de Objetos

1. Qué es la Programación Orientada a Objetos (POO)

La **POO** es un paradigma de programación que organiza el código en torno a **objetos**, que combinan datos (propiedades) y comportamiento (métodos).

JavaScript **no está basado en clases** como Java o C++, sino en **prototipos**. Esto significa que los objetos pueden servir de plantilla para crear otros objetos.

> Según Mozilla: *JavaScript es un lenguaje basado en prototipos, donde cada objeto puede heredar directamente de otro.*

Aunque hoy en día se usan **clases (desde ES6)**, internamente JavaScript sigue funcionando con prototipos.

2. Gestión de Objetos en JavaScript

Trabajar con objetos implica saber:

- Cómo crear y acceder a **propiedades y métodos**.
- Cómo **construir objetos genéricos** (como `Object`).
- Cómo **recorrer, copiar o eliminar** propiedades.
- Cómo **heredar y especializar** comportamientos entre clases.

3. Propiedades y Métodos

Acceso a propiedades

```
objeto.propiedad
objeto["propiedad"]
```

Ejemplo:

```
let vector = [4, 2, 7, 9];
console.log(vector.length);     // 4
console.log(vector["length"]);  // 4
```

Acceso a métodos

```
console.log("Hola");
console["log"]("Hola");
```

4. Operador `instanceof`

Permite comprobar si un objeto pertenece a un tipo determinado:

```
let vector = [4, 2, 7];
console.log(vector instanceof Array);  // true

let conjunto = new Set();
console.log(conjunto instanceof Map);  // false
```

5. El objeto genérico `Object`

`Object` es la base de todos los objetos en JavaScript.

Podemos crear objetos **al vuelo**:

```
let notas = new Object();
notas.valores = [7.5, 3.2, 9.6];
notas.cantidad = notas.valores.length;
notas.verMedia = function() {
  let suma = notas.valores.reduce((a, b) => a + b, 0);
  return suma / notas.cantidad;
};

console.log(notas.verMedia());  // 6.76
```

O usar notación **JSON**:

```
let viaje = {
  origen: "Granada",
  destino: "El Cairo",
  dias: 8,
  precio: 750,
  mostrar: function() {
    console.log(`${this.origen} / ${this.destino}`);
    console.log(`durante ${this.dias} días: EUR${this.precio}`);
  }
};

viaje.mostrar();
```

6. El objeto `this`

`this` hace referencia al **objeto actual** que ejecuta el método.

```
let viaje = {
  origen: "Granada",
  destino: "El Cairo",
  dias: 8,
  precio: 750,
  mostrar: function() {
    console.log(`${this.origen} / ${this.destino}`);
    console.log(`durante ${this.dias} días: EUR${this.precio}`);
  }
};

viaje.mostrar();
```

Si se asigna el objeto a otra variable:

```
let oferta = viaje;
viaje = null;
oferta.mostrar(); // Error: Cannot read properties of null
```

7. Constructores

Los constructores sirven para **crear objetos nuevos**.

```
class Viaje {
  constructor(origen, destino, dias, precio) {
    this.origen = origen;
    this.destino = destino;
    this.dias = dias;
    this.precio = precio;
  }

  mostrar() {
    console.log(`${this.origen} / ${this.destino}`);
    console.log(`durante ${this.dias} días: EUR${this.precio}`);
  }
}

let miViaje = new Viaje("Barcelona", "Ibiza", 2, 112);
miViaje.mostrar();
```

> Las clases de JavaScript introducidas en ECMAScript 2015 son una **mejora sintáctica** sobre los prototipos, no un nuevo modelo.

8. Herencia y Polimorfismo

La **herencia** permite que una clase hija extienda otra. El **polimorfismo** permite que distintos objetos usen el mismo método con comportamientos diferentes.

```
class Miembro {
  constructor(nombre, alta, estado) {
    this.nombre = nombre;
    this.alta = alta;
    this.estado = estado;
  }

  cobrar() {
    console.log(`El Miembro ${this.nombre} ha cobrado`);
  }
}

class Profesor extends Miembro {
  constructor(nombre, alta, estado, alumnos) {
    super(nombre, alta, estado);
    this.alumnos = alumnos;
  }

  cobrar() {
    console.log(`El Profesor ${this.nombre} ha cobrado`);
  }
}

class Alumno extends Miembro {
  constructor(nombre, alta, estado, asignaturas) {
    super(nombre, alta, estado);
    this.asignaturas = asignaturas;
  }

  cobrar() {
    console.log(`El Alumno ${this.nombre} ha recibido su beca`);
  }
}

let prof = new Profesor("Ana Ruiz", "01/10/2022", "vigente", 30);
let alum = new Alumno("Carlos Pérez", "02/11/2022", "vigente", 8);

prof.cobrar();
alum.cobrar();
```

9. Recorridos en Objetos

Para recorrer propiedades:

```
for (let propiedad in miViaje) {
  console.log(propiedad);
}
```

Evitar propiedades heredadas:

```
for (let propiedad of Object.getOwnPropertyNames(miViaje)) {
  console.log(propiedad);
}
```

10. Eliminación de Propiedades

```
delete miViaje.precio;
```

🔹 Actividades Prácticas

Actividad 1: Primeros Objetos

Crea un objeto `usuario` que permita autenticar una persona.

```
const usuario = {
  nombre: "Antonio García",
  nombreUsuario: "agar007",
  contraseña: "agar007pass",
  login: function(nombreUsuario, contraseña) {
    if (nombreUsuario === this.nombreUsuario && contraseña === this.contraseña) {
      console.log("Sesión iniciada con éxito");
    } else {
      console.log("Credenciales no válidas");
    }
  }
};

usuario.login("agar007", "agar007pass");
```

Actividad 2: Constructores

Crea una clase que modele un **teléfono móvil** con las propiedades:

- CPU, RAM, Almacenamiento, Ancho, Alto, numCamaras

Y un método `toString()`.

```
class Telefono {
  constructor(cpu, ram, almacenamiento, ancho, alto, numCamaras) {
    this.cpu = cpu;
    this.ram = ram;
    this.almacenamiento = almacenamiento;
    this.ancho = ancho;
    this.alto = alto;
    this.numCamaras = numCamaras;
  }

  toString() {
    return `${this.cpu} | ${this.ram}GB RAM | ${this.almacenamiento}GB |
    ${this.numCamaras} cámaras`;
  }
}

let movil1 = new Telefono("Snapdragon 8 Gen2", 8, 128, 7.5, 15, 3);
let movil2 = new Telefono("Dimensity 9000", 12, 256, 7.2, 15.5, 4);

console.log(movil1.toString());
console.log(movil2.toString());
```

Actividad 3: Recorridos con Herencia

Usa un bucle `for...in` para mostrar todas las propiedades de los objetos creados en la actividad de herencia. Luego prueba `Object.getOwnPropertyNames()` y comenta las diferencias.