3.0 Programación orientada a objetos JavaScript

En JavaScript, el enfoque de programación se basa en prototipos en lugar de clases como en otros lenguajes orientados a objetos. En lugar de crear objetos directamente a partir de clases, en JavaScript se utilizan prototipos como plantillas para crear objetos.

```
 // Definimos un prototipo de objeto
let animal = {
    tipo: 'desconocido',
    sonido: 'hace ruido',
    hacerSonido: function() {
      console.log(this.sonido);
    }
  };

  // Creamos un nuevo objeto utilizando el prototipo 'animal'
  let perro = Object.create(animal);
  perro.tipo = 'perro';
  perro.sonido = 'ladra';

  // Accedemos a las propiedades y métodos del objeto 'perro'
  console.log(perro.tipo); // Output: perro
  perro.hacerSonido(); // Output: ladra
  // Modificamos el valor de una propiedad existente en el prototipo
  perro.sonido = 'hace ruido';
  perro.hacerSonido(); // Output: hace ruido
```

En este ejemplo, **animal** es un objeto prototipo que tiene propiedades **tipo** y **sonido**, junto con un método **hacerSonido** que imprime el sonido del animal. Después, creamos un nuevo objeto **perro** usando **Object.create(animal)**. Esto establece **animal** como prototipo de **perro**.

perro hereda las propiedades y métodos de **animal**, pero puede sobrescribirlos si es necesario. En este caso, **perro** sobrescribe las propiedades **tipo** y **sonido** para representar un perro que ladra.

Esto ilustra cómo JavaScript usa prototipos para la herencia en lugar de las clases tradicionales. Los objetos en JavaScript pueden heredar propiedades y comportamientos de otros objetos (prototipos) y modificarlos según sea necesario.

Ejemplo de Herencia con Clases:

A partir de ECMAScript 6 (también conocido como ES6 o ES2015), JavaScript introdujo la sintaxis de clase que se asemeja más a la programación orientada a objetos tradicional. Las clases en JavaScript ofrecen una sintaxis más clara y familiar para aquellos que provienen de otros lenguajes orientados a objetos, como Java o C++.

```
class Animal {
    constructor(tipo, sonido) {
      this.tipo = tipo;
      this.sonido = sonido;
    }

    hacerSonido() {
      console.log(this.sonido);
    }
  }

  class Perro extends Animal {
    constructor() {
      super('perro', 'ladra');
    }
  }

  class Gato extends Animal {
    constructor() {
      super('gato', 'maulla');
    }

    hacerSonido() {
      console.log('El gato dice: ' + this.sonido);
    }
  }

  let miPerro = new Perro();
  let miGato = new Gato();

  miPerro.hacerSonido(); // Output: ladra
  miGato.hacerSonido(); // Output: El gato dice: maulla
```

Ejemplo de Herencia con Prototipos:

```
  let animal = {
    tipo: 'desconocido',
    sonido: 'hace ruido',
    hacerSonido: function() {
      console.log(this.sonido);
    }
  };
```

```
let perro = Object.create(animal);
perro.tipo = 'perro';
perro.sonido = 'ladra';

let gato = Object.create(animal);
gato.tipo = 'gato';
gato.sonido = 'maulla';

gato.hacerSonido = function() {
  console.log('El gato dice: ' + this.sonido);
};

perro.hacerSonido(); // Output: ladra
gato.hacerSonido(); // Output: El gato dice: maulla
```

Polimorfismo

El polimorfismo se refiere a la capacidad de objetos diferentes de responder al mismo mensaje o método de manera distinta. En los ejemplos anteriores, podemos agregar polimorfismo sobrescribiendo métodos en los objetos heredados para que se comporten de manera diferente.

Ejemplo de Polimorfismo con Clases:

```
class Animal {
    constructor(tipo, sonido) {
      this.tipo = tipo;
      this.sonido = sonido;
    }

    hacerSonido() {
      console.log(this.sonido);
    }
  }

  class Perro extends Animal {
    constructor() {
      super('perro', 'ladra');
    }
```

```
  hacerSonido() {
    console.log('El perro dice: ' + this.sonido);
  }
}

class Gato extends Animal {
  constructor() {
    super('gato', 'maulla');
  }

  hacerSonido() {
    console.log('El gato dice: ' + this.sonido);
  }
}

let miPerro = new Perro();
let miGato = new Gato();

miPerro.hacerSonido(); // Output: El perro dice: ladra
miGato.hacerSonido(); // Output: El gato dice: maulla
```

En este ejemplo, las clases **Perro** y **Gato** sobrescriben el método **hacerSonido()** de la clase base **Animal**. Cada uno tiene su propia implementación del método **hacerSonido()**, lo que permite que objetos diferentes respondan al mismo mensaje (**hacerSonido**) de manera distinta.

Ejemplo de Polimorfismo con Prototipos:

```
let animal = {
  tipo: 'desconocido',
  sonido: 'hace ruido',
  hacerSonido: function() {
    console.log(this.sonido);
  }
};

let perro = Object.create(animal);
perro.tipo = 'perro';
perro.sonido = 'ladra';

let gato = Object.create(animal);
gato.tipo = 'gato';
```

```
gato.sonido = 'maulla';

perro.hacerSonido = function() {
  console.log('El perro dice: ' + this.sonido);
};

gato.hacerSonido = function() {
  console.log('El gato dice: ' + this.sonido);
};

perro.hacerSonido(); // Output: El perro dice: ladra
gato.hacerSonido(); // Output: El gato dice: maulla
```