TEMA 9 - Programación orientada a objetos

Herencia

La herencia es una manera de crear una clase como una versión especializada de una o más clases. La clase especializada comúnmente se llama hija, y la otra clase se le llama padre.

En JavaScript la herencia se logra mediante la asignación de una instancia de la clase primaria a

la clase secundaria, y luego se hace la especialización.

En el siguiente ejemplo definimos la clase **Estudiante** como una clase secundaria de **Persona** . Luego redefinimos el método **diHola()** y agregamos el método **diAdios()**.

```
// Definimos el constructor Persona function
Persona(primerNombre) {
    this.primerNombre = primerNombre;

    // Agregamos un par de métodos a Persona.prototype
    Persona.prototype.caminar = function() {
alert("Estoy caminando!"); };

    Persona.prototype.diHola = function(){
alert("Hola, Soy" + this.primerNombre); }; } // Definimos el
constructor Estudiante function Estudiante(primerNombre,
asignatura) {
// Llamamos al constructor padre, nos aseguramos (utilizando Function#call) que "this" se
    // ha establecido correctamente durante la llamada
    Persona.call(this, primerNombre);

    //Inicializamos las propiedades específicas de Estudiante this.asignatura =
    asignatura;

    // Reemplazar el método "diHola"
    Estudiante.prototype.diHola = function(){
   alert("Hola, Soy " + this.primerNombre + ". Estoy estudiando " + this.asignatura + ".");
```

```
    };

    // Agregamos el método "diAdios"
    Estudiante.prototype.diAdios = function() {
alert("¡ Adios !"); };

};

// Creamos el objeto Estudiante.prototype que hereda desde Persona.prototype
Estudiante.prototype = Object.create(Persona.prototype);

// Establecer la propiedad "constructor" para referenciar a Estudiante
Estudiante.prototype.constructor = Estudiante;


// Ejemplos de uso var estudiante1 = new Estudiante("Carolina", "Física Aplicada");
estudiante1.diHola(); // muestra "Hola, Soy Carolina. Estoy estudianto Física Aplicada."
estudiante1.caminar(); // muestra "Estoy caminando!" estudiante1.diAdios(); // muestra "¡
Adios !"

// Comprobamos que las instancias funcionan correctamente alert(estudiante1
instanceof Persona); // devuelve true alert(estudiante1 instanceof Estudiante);
// devuelve true
```

Encapsulación

En el ejemplo anterior, Estudiante no tiene que saber cómo se aplica el método caminar() de la clase Persona, pero, sin embargo, puede utilizar ese método.

La clase Estudiante no tiene que definir explícitamente ese método, a menos que queramos cambiarlo. Esto se denomina **la encapsulación**, por medio de la cual cada clase hereda los métodos de su elemento primario y sólo tiene que definir las cosas que desea cambiar.

Abstracción

Un mecanismo que permite modelar la parte actual del problema de trabajo. Esto se puede lograr por herencia (especialización) o por composición. JavaScript logra la especialización por herencia y por composición al permitir que las instancias de clases sean los valores de los atributos de otros objetos.

En nuestro ejemplo de la Playlist, al contener Playlist objetos tipo Song, se está realizando una abstracción.

Polimorfismo

Al igual que todos los métodos y propiedades están definidas dentro de la propiedad prototipo, las diferentes clases pueden definir métodos con el mismo nombre. Los métodos están en el ámbito de la clase en que están definidos. Esto sólo es verdadero cuando las dos clases no tienen una relación primario-secundario (cuando uno no hereda del otro en una cadena de herencia).

Glosario:

Clase: Define las características del Objeto.

Objeto: Una instancia de una Clase.

Propiedad: Una característica del Objeto, como el color.

Método: Una capacidad del Objeto, como caminar.

Constructor: Es un método llamado en el momento de la creación de instancias de una clase.

Herencia: Una Clase puede heredar características de otra Clase.

Encapsulamiento: Una Clase sólo define las características del Objeto, un Método sólo define cómo se ejecuta el Método.

Abstracción: La conjunción de herencia compleja, métodos y propiedades que un objeto debe ser capaz de simular en un modelo de la realidad.

Polimorfismo: Diferentes Clases podrían definir el mismo método o propiedad.

Sobrecarga: una clase hija implementa un método que tiene el mismo nombre en la clase padre, cambiando el comportamiento.