APRENDERAPROGRAMAR.COM

HERENCIA JAVASCRIPT: EJEMPLO CON CÓDIGO BÁSICO. JERARQUÍA DE CLASES EN CADENA DE PROTOTIPOS. (CU01149E)

Sección: Cursos
Categoría: Tutorial básico del programador web: JavaScript desde cero

Fecha revisión: 2029

Resumen: Entrega nº49 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

HERENCIA Y OBJETO PROTOTYPE

JavaScript carece de clases como existen en otros lenguajes, por tanto no existe la herencia como existe en otros lenguajes. Coloquialmente nos referimos en ocasiones a "clases" entre comillas, pero hay que recordar que en JavaScript prácticamente todo con lo que se trabaja son objetos (instancias) y por tanto no es posible diferenciar entre clases e instancias.

No se puede crear por tanto una jerarquía de clases como se hace en otros lenguajes, aunque se puede crear una simulación basada en prototipos.

Todo objeto tiene una propiedad oculta denominada prototype que es un objeto de referencia. Este objeto de referencia actúa como si fuera la "clase padre" donde el objeto consulta atributos y métodos en caso de que no los encuentre definidos como atributos y métodos propios de sí mismo.

Ejemplo:

```
function Taxi (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

Taxi.prototype = { ruedas: 4, saludar: function() {
alert('Hola soy un taxi de ' + this.ruedas + ' ruedas y ' +this.numeroPasajeros +' pasajeros');} }

function ejemploObjetos() {
var taxi1 = new Taxi(1, 6, 300, 90);
taxi1.saludar();
}
```

En este ejemplo, se define la "clase" Taxi y se dota de contenido al objeto prototype asignándole propiedades y métodos. Se crea un objeto taxi1 y se invoca su método saludar(). En primer lugar este método se busca dentro de los métodos de "la clase" y al no encontrarse se busca en el objeto prototype, donde sí se encuentra y se devuelve. Al ejecutar el método saludar() se encuentra una invocación a this.ruedas. Esta propiedad se busca primero en la "clase" y al no encontrarse se busca en el prototipo. Lo mismo ocurre con numeroPasajeros, aunque en este caso sí se encuentra.

El objeto prototype existe como objeto vacío desde que se crea una instancia. ¿Por qué? Porque JavaScript lo crea en segundo plano, como si incluyéramos dentro del código de la clase la sentencia: prototype = {};

Por ello podemos acceder al objeto y asignarle propiedades y funciones y por ello este código de ejemplo es equivalente al anterior:

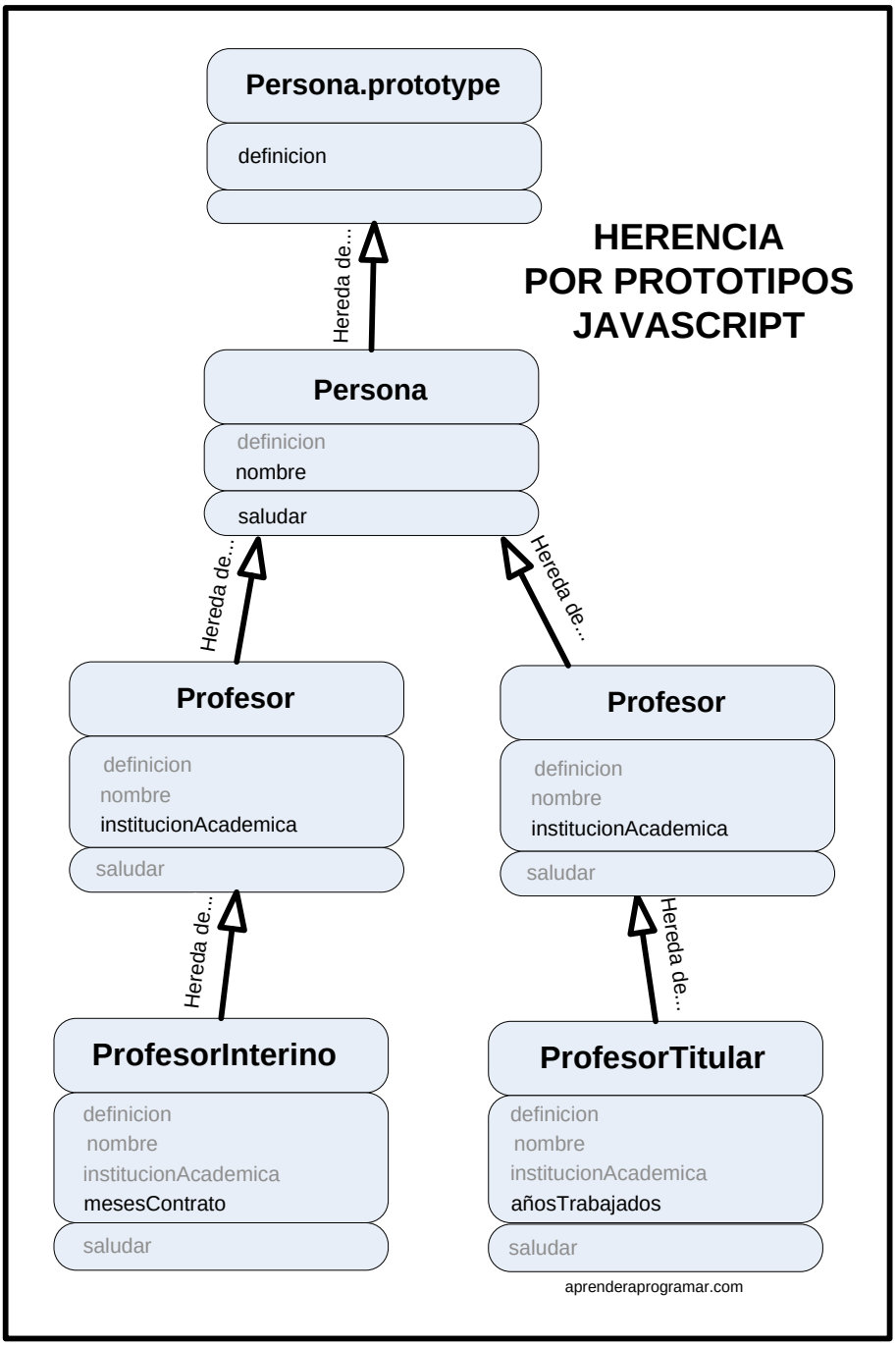
```
function Taxi (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

Taxi.prototype.ruedas = 4;
Taxi.prototype.saludar = function() {
alert('Hola soy un taxi de ' + this.ruedas + ' ruedas y ' +this.numeroPasajeros +' pasajeros');}

function ejemploObjetos() {
var taxi1 = new Taxi(1, 6, 300, 90);
taxi1.saludar();
}
```

Podemos hacer una simulación de "jerarquía de clases" basándonos en los objetos prototipo. Para ello hemos de crear prototipos en cadena. Veamos un código de ejemplo básico:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function Persona () { this.nombre = 'Nombre desconocido';
this.saludar = function () {alert('Hola, soy ' + this.nombre);}
}
Persona.prototype.definicion = 'Ser humano';
function Profesor () { this.institucionAcademica = 'Institución desconocida'; }
Profesor.prototype = new Persona();
function ProfesorInterino() { this.mesesContrato = 0;}
ProfesorInterino.prototype = new Profesor();
function ProfesorTitular() { this.añosTrabajados = 0;}
ProfesorTitular.prototype = new Profesor();
function ejemploObjetos() {
var unProfesorTitular = new ProfesorTitular();
alert ('El nombre de unProfesorTitular es ' + unProfesorTitular.nombre);
unProfesorTitular.saludar();
unProfesorTitular.nombre = 'Juan';
unProfesorTitular.institucionAcademica = 'Universidad de Chapingo';
unProfesorTitular.añosTrabajados = 14;
var msg = 'El profesor titular creado tiene nombre ' + unProfesorTitular.nombre + ', trabaja en '
msg = msg + unProfesorTitular.institucionAcademica + ', tiene ' + unProfesorTitular.añosTrabajados + ' años trabajados';
msg = msg + ' y definición de ' + unProfesorTitular.definicion;
alert(msg);
unProfesorTitular.saludar();
}
</script></head>
<body><div id="cabecera"><h2>Cursos aprenderaprogramar.com</h2><h3>Ejemplos JavaScript</h3></div>
<div style="color:blue;" id ="pulsador" onclick="ejemploObjetos()"> Probar </div>
</body></html>
```

El código anterior se corresponde con este esquema de herencia, donde podemos observar que hay objetos Profesor duplicados ¿Por qué? Lo explicamos a continuación.

Nota: en gris señalamos propiedades o métodos accesibles a través del prototipo. Dichas propiedades o métodos no se encuentran en los objetos en sí, pero son accesibles.



El resultado de ejecución esperado para el código es:

El nombre de unProfesorTitular es Nombre desconocido
Hola, soy Nombre desconocido
El profesor titular creado tiene nombre Juan, trabaja en Universidad de Chapingo, tiene 14 años trabajados y definición de Ser humano
Hola, soy Juan

Analicemos el código paso a paso:

Se define profesorTitular de modo que toda instancia de profesorTitular se inicializará con la propiedad añosTrabajados = 0

Se define que el objeto prototipo de ProfesorTitular es un objeto Profesor.

Se define ProfesorInterino de modo que toda instancia de profesorInterino se inicializará con la propiedad mesesContrato = 0;

Se define que el objeto prototipo de ProfesorInterino es un objeto Profesor.

Se define Profesor de modo que toda instancia de Profesor se inicializará con la propiedad institucionAcademica = 'Institución desconocida';

Se define que el prototipo de Profesor es un objeto Persona.

Por último se define Persona de modo que toda instancia de Persona se inicializará con la propiedad nombre = 'Nombre desconocido'; y se define que el objeto prototipo de Persona (de tipo Object) tiene la propiedad << definicion = 'Ser humano'; >>

En el código de ejecución comenzamos por crear un objeto de tipo profesorTitular al que denominamos unProfesorTitular. Al invocar la propiedad nombre y el método saludar del objeto profesorTitular el intérprete JavaScript busca estas propiedades y métodos en el propio objeto, pero no los encuentra, ya que la única propiedad en el objeto es añosTrabajados. Al no encontrarlas, pasa a buscar las propiedades y métodos en el objeto prototipo (que es un objeto Profesor), pero en este objeto tampoco las encuentra. Finalmente, busca en el prototipo del objeto Profesor que es un objeto Persona. Como todo objeto Persona tiene inicialmente la propiedad nombre = 'Nombre desconocido' y dispone del método saludar, son éstos los que se devuelven. El resultado es que se muestra por pantalla <<El nombre de unProfesorTitular es Nombre desconocido. Hola, soy Nombre desconocido >>

Seguidamente se establecen valores para las propiedades nombre, institucionAcademica y añosTrabajados de unProfesorTitular y se muestra por pantalla un mensaje donde además de estas propiedades se muestra cómo unProfesorTitular dispone de la propiedad <<definicion>> que se obtiene recorriendo la cadena de prototipos.

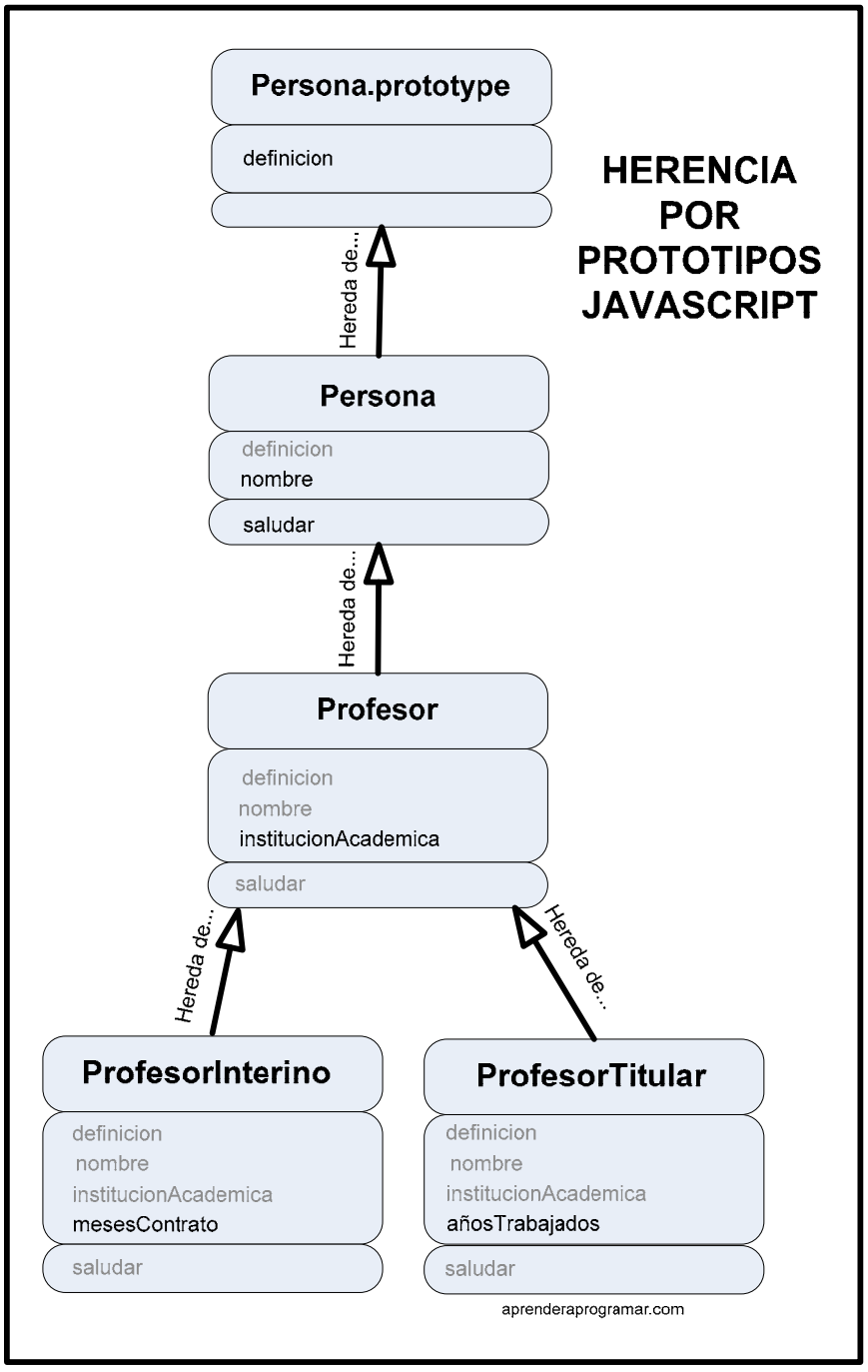
PENSAR EN OBJETOS Y NO EN CLASES

Muchas personas (normalmente los programadores que provienen de Java) pueden pensar que sería más correcto un esquema donde ProfesorInterino y ProfesorTitular dependan de un único Profesor.

Esto podría simularse haciendo que ProfesorInterino y ProfesorTitular compartan un mismo prototipo y esto podría lograrse con un código como este:

```
function Profesor () { this.institucionAcademica = 'Institución desconocida'; }
Profesor.prototype = new Persona();

function ProfesorInterino() { this.mesesContrato = 0;}
ProfesorInterino.prototype = ProfesorTitular.prototype;
```

El esquema sería este:



Esta concepción tiene un inconveniente importante. Dado que las propiedades y métodos comunes ("de clase") en JavaScript se establecen a través del prototipo, usar un mismo objeto como prototipo para los profesores interinos y los profesores titulares nos impediría diferenciar propiedades comunes exclusivas de los profesores interinos y propiedades comunes exclusivas de los profesores titulares. Si el prototipo es un mismo objeto, las propiedades comunes lo serían tanto para profesores interinos como para profesores titulares.

A modo de conclusión: cuando trabajes con JavaScript no pienses en clases, piensa en objetos.

CONSTRUCTORES CON HERENCIA

En el código de ejemplo visto anteriormente trabajamos con objetos que no reciben parámetros para su constructor, sino que inicializan sus propiedades a unos valores de defecto. ¿Cómo implementar herencia de modo que se pueda inicializar un objeto con unos valores concretos? Lo veremos en la próxima entrega.

EJERCICIO

Crea un esquema de herencia en JavaScript que refleje estos requisitos:

a) Hay tres tipos de hortalizas: zanahoria, lechuga y tomate. La zanahoria tiene como propiedad su valor calórico que es de 45 cal, mientras que la lechuga tiene 31 cal y el tomate 39 cal.

b) Toda hortaliza tiene como propiedad específica tipoHortaliza y su valor inicial debe ser “indefinido”.

c) Todas las hortalizas tienen una propiedad común: su componente principal: <<Agua>>

c) Una hortaliza es un tipo de planta cultivada. Una planta cultivada tiene como propiedad específica nombreCientifico y su valor inicial debe ser “desconocido”.

d) Una planta cultivada es un tipo de vegetal. Una propiedad de los vegetales es la movilidad y su valor es común para todos los vegetales: <<Ser vivo sin movilidad>>

Escribir el código correspondiente. Haciendo uso de la herencia por prototipos, crear un objeto de tipo tomate al que se denomine tomate1 y hacer que se muestre por pantalla <<tomate1 tiene la propiedad movilidad: Ser vivo sin movilidad>>.

Para comprobar si tus respuestas y código son correctos puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU01150E

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=78&Itemid=206