APRENDERAPROGRAMAR.COM

CONSTRUCTORES JAVASCRIPT. CREAR OBJETOS VACÍOS. AÑADIR PROPIEDADES Y MÉTODOS. OBJETOS ÚNICOS O SINGLETON. EJEMPLOS (CU01145E)

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

Fecha revisión: 2029

Resumen: Entrega nº45 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

CONSTRUCTORES Y MÁS FORMAS DE CREAR OBJETOS

JavaScript tiene ciertas diferencias respecto a otros lenguajes de programación orientados a objetos. En JavaScript no existe el concepto de clase ni el concepto de constructor propiamente dicho, aunque por analogía con otros lenguajes muchas veces usamos estos términos.

CONSTRUCTORES EN "CLASES" JAVASCRIPT

En programación orientada a objetos se denomina constructor al código que se ejecuta cuando se crea un objeto de un tipo determinado (a cada objeto creado se le denomina instancia). En otros lenguajes el constructor se delimita dentro de etiquetas separándolo del resto del código de la clase, pero en JavaScript esto no es así. Cuando se instancia un objeto, funciona como constructor todo el código declarado dentro de la función, es decir, todo código dentro de la función que pueda ser ejecutado será ejecutado (no se ejecutarán los métodos de la función, ya que será necesario invocarlos para que se ejecuten).

Escribe este código, guárdalo con extensión html y comprueba lo que ocurre.

```
<!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 Taxi (tipoMotor) {
this.tipoMotor = tipoMotor;
alert('Se ha creado un objeto taxi con tipo motor: ' + this.tipoMotor);
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
}

function ejemploObjetos() {
var coche1 = new Taxi('Diesel');
var coche2 = new Taxi('Gasolina');
alert ('El coche 1 tiene capacidad ' + coche1.getCapacidad() + ' litros\n');
alert ('El coche 2 tiene capacidad ' + coche2.getCapacidad() + ' litros\n');
}
</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>
```

Al pulsar sobre el texto "Probar" comprobamos que se muestran los mensajes:

Se ha creado un objeto taxi con tipo motor: Diesel. Se ha creado un objeto taxi con tipo motor: Gasolina El coche 1 tiene capacidad 40 litros. El coche 2 tiene capacidad 35 litros

Como obervamos, cada vez que se instancia (crea un objeto) de la clase, se ejecuta el código asociado.

En este ejemplo tenemos el código this.tipoMotor = tipoMotor; donde debemos recordar que this.tipoMotor alude a la propiedad de la clase, mientras que tipoMotor es el parámetro recibido. Dado que ambos tienen el mismo nombre, la forma de diferenciar cuándo nos referimos a la propiedad y cuándo al parámetro es usar this cuando queramos referirnos a la clase. En general, toda propiedad y todo método dentro de la clase irán antecedidos de la palabra clave this para indicar que nos referimos a propiedades y métodos y no a variables cualquiera.

MÁS FORMAS DE CREAR OBJETOS

Hemos visto que podemos declarar un tipo y luego crear objetos usando la invocación basada en new nombreDelObjeto. Existen más alternativas para crear objetos:

Creación de un objeto vacío

Para crear un objeto vacío podemos usar esta sintaxis.

```
var objetoCreado = { };
```

Esta sintaxis equivale a escribir var objetoCreado = new Object ();

El objeto vacío creado con esta sintaxis pertenece al tipo de objeto predefinido de JavaScript al cual pertenecen todos los demás objetos existentes: Object.

Además existe un objeto Object, objeto predefinido de JavaScript, que nos facilita métodos para manipular objetos. No confundir el tipo de dato Object con el objeto predefinido Object. Prueba este ejemplo donde se usa el objeto Object para definir una propiedad de un objeto creado inicialmente vacío:

```
var pintura1 = {};
Object.defineProperty(pintura1, 'autor', {value: 'Vincent Van Gogh', writable:true, enumerable:true, configurable:true});
alert ('La propiedad autor del objeto pintura1 es: ' + pintura1.autor);
pintura1.autor = 'Michelangelo';
alert ('La propiedad autor del objeto pintura1 es: ' + pintura1.autor);
pintura1.deletrear = function () {
var letras = []; var msg = '';
for(var i=0; i<pintura1.autor.length;i++){msg = msg+pintura1.autor[i]+'-'; }
alert (msg);
}
pintura1.deletrear();
```

Con este código en primer lugar creamos un objeto vacío pintura1. Luego le añadimos una propiedad valiéndonos del método defineProperty del objeto predefinido Object. En este método pasamos como parámetros el objeto del que definimos la propiedad (pintura1), el nombre de la propiedad (autor), el valor de la propiedad (Vincent Van Gogh) y sus características writable, enumerable y configurable (no vamos ahora a hablar sobre estas características). Posteriormente, accedemos a la propiedad y la modificamos para que pase a valer 'Michelangelo'. Finalmente definimos una función para el objeto creado. A continuación resumimos la sintaxis utilizada. Adición de una propiedad a un objeto:

```
Object.defineProperty(nombreObjeto, 'nombrePropiedad', {value: 'valorAsignado', writable:true, enumerable:true, configurable:true});
```

Adición de una función a un objeto:

```
nombreObjeto.nombreFuncionDefinimos = function () { ... código ... }
```

Definición y creación simultánea de un objeto

Para definir y crear un objeto en un solo paso usamos esta sintaxis:

```
var nombreObjetoCreado = {
    propiedad1: valorPropiedad1,
    propiedad2: valorPropiedad2,
    propiedadN: valorPropiedadN,
    ...
    método1: function () { ... código ... }
    método2: function (par1, par2, ..., parN) { ... código ... }
    métodoN: function () { ... código ... }
}
```

Comprueba los resultados de ejecutar este código:

```
var pintura2 = {
   autor: "Vincent Van Gogh",
   añoCreacion: 1871,
   titulo: 'Los rosales corintios',
   getInfo: function () {    return this.titulo + ':' + this.autor + ', ' + this.añoCreacion +'.';    }
}
alert(pintura2.getInfo());
```

Con esta sintaxis no se pueden crear objetos adicionales del tipo definido. El único objeto existente es el que hemos definido y creado simultáneamente. Algunos autores denominan a esta construcción un Singleton (objeto único) por analogía con el patrón Singleton usado en otros lenguajes orientados a objetos para crear objetos únicos.

EJERCICIO

Una de las utilidades de crear objetos vacíos es evitar conflictos de nombres. Supón que creas funciones como:

```
function crearEntrada() {
   // hacer algo
}

function crearSalida() {
   // hacer algo
}
```

El problema que se presenta es que en otro momento se pueda definir otra función con el mismo nombre que alguna de las ya definidas, creando un conflicto de nombres.

Crea un objeto vacío denominado GestionDeUsuarios y añádele dos métodos: un método preguntarNombre y un método despedir. Al invocar GestionDeUsuarios.preguntarNombre(user) se debe crear un objeto de tipo usuario con id de usuario user y almacenar su nombre e id de usuario. Al invocar el método GetionDeUsuarios.despedir(user) se debe mostrar un mensaje de despedida "Hasta luego nombreDeUsuario" donde nombreDeUsuario será el nombre correspondiente.

Responde la siguiente pregunta: ¿si se crea una función despedir entrará en conflicto con el método definido?

Nota: una de las librerías más populares construidas para JavaScript es jQuery. jQuery dispone de un objeto global al que se denomina $, de modo que se accede a las funciones de jQuery escribiendo algo como $.nombreFuncion(...). Esto es, con la diferencia de escala, algo similar a la creación del objeto GestionDeUsuarios visto en el ejemplo. $ evita que jQuery entre en conflictos de nombres.

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

Próxima entrega: CU01146E

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