APRENDERAPROGRAMAR.COM

PROTOTYPE JAVASCRIPT. EJEMPLOS DE PROTOTIPOS Y HERENCIA. CÓMO USARLOS. SINTAXIS. EFICIENCIA. (CU01147E)

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

Fecha revisión: 2029

Resumen: Entrega nº47 del Tutorial básico "JavaScript desde cero".

Autor: César Krall

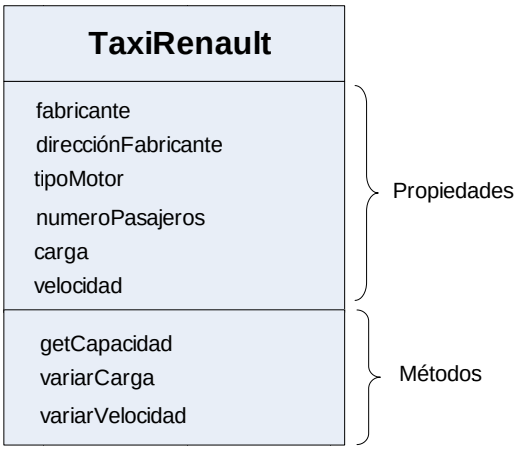
HERENCIA Y PROTOTYPE

Hasta ahora hemos visto como definir objetos con propiedades y métodos y cómo instanciarlos. Trataremos ahora de ver cómo podemos hacer que muchos objetos de un mismo tipo compartan (hereden) propiedades y métodos de modo que se optimice el código.

Cada vez que instanciamos un objeto con new se crea una instancia que "porta" espacio de memoria donde están las propiedades y los métodos.

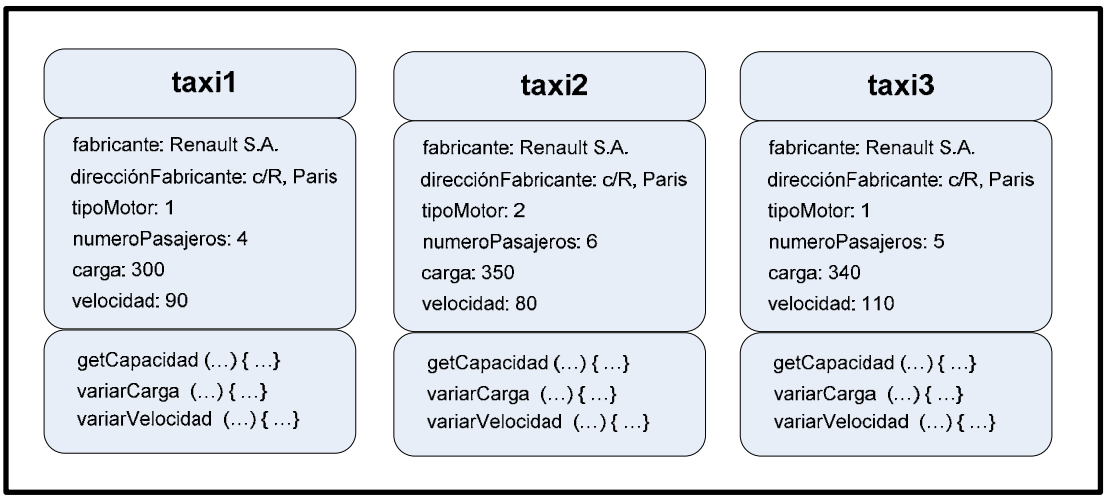
Supongamos que hemos definido un objeto TaxiRenault cuyas propiedades son fabricante, direccionFabricante, tipoMotor, numeroPasajeros, carga, y velocidad, y cuyos métodos son getCapacidad (devuelve la capacidad del depósito de combustible), variarCarga y variarVelocidad (suponemos que la carga y velocidad aumentan si se pasa un valor positivo a los métodos responsables, o que disminuyen si se pasa un valor negativo). Suponemos que propiedades y métodos se han definido usando la palabra clave this, lo cual significa que pertenecen al objeto.

Este esquema resume cómo se ha concebido el patrón o clase para los objetos de tipo TaxiRenault:



Renault podría haber sido el valor de la propiedad marca de un objeto Taxi (en vez de usar como objeto TaxiRenault), pero ten en cuenta que esto es un ejemplo para estudiar la herencia, no pretendemos que uses esto como código real.

Supongamos ahora que creamos tres objetos de tipo TaxiRenault mediante la sentencia new, a los que denominamos taxi1, taxi2 y taxi3. Un esquema para representarlos podría ser el siguiente:



Aquí vemos que cada objeto cuenta con sus propiedades y con sus métodos. Cada propiedad y cada método ocupa espacio de memoria y genera "una carga" a la hora de trabajar con ellos. También observamos que algunas propiedades siempre tienen el mismo valor en todos los objetos. En este ejemplo la propiedad fabricante siempres es Renault S.A., y la dirección del fabricante siempre es c/R, París. Otras propiedades (aunque casualmente puedan tener el mismo valor en varios objetos) son las que realmente caracterizan al objeto (tipoMotor, numeroPasajeros, carga, velocidad), es decir, son propiedades inherentes a cada objeto individual y específico, no compartidas por todos los objetos.

En cuanto a los métodos, todos los métodos hacen lo mismo: reciben (o no) unos datos, y realizan un proceso. En todos los objetos los métodos son iguales pero están repetidos.

Escribe y prueba el siguiente código, que refleja al esquema anterior.

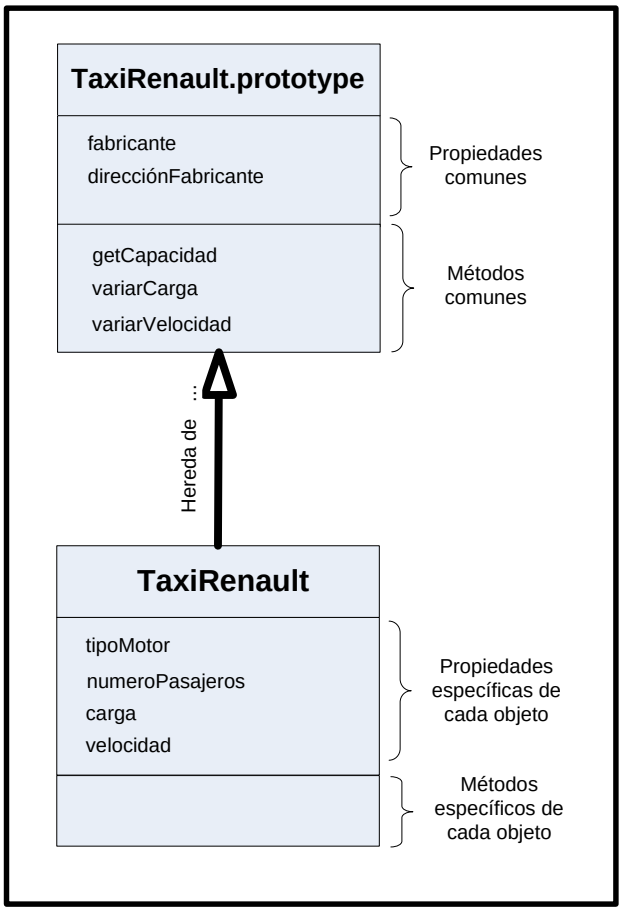
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Ejemplo aprenderaprogramar.com</title><meta charset="utf-8">
<script type="text/javascript">
function TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.fabricante = 'Renault, S.A.'; this.direccionFabricante = 'c/R, Paris';
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga;      this.velocidad = velocidad;
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
this.variarCarga = function (variacion) { this.carga = this.carga + variacion; }
this.variarVelocidad = function (variacion) { this.velocidad = this.velocidad + variacion; }
}
function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('La velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10);
alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</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>
```

Si los objetos TaxiRenault fueran objetos muy complejos y generáramos cientos o miles de objetos de esta manera nos encontraríamos con que podemos saturar la memoria disponible y ralentizar la ejecución de la página web, o incluso generar un bloqueo.

HERENCIA BASADA EN PROTOTIPOS

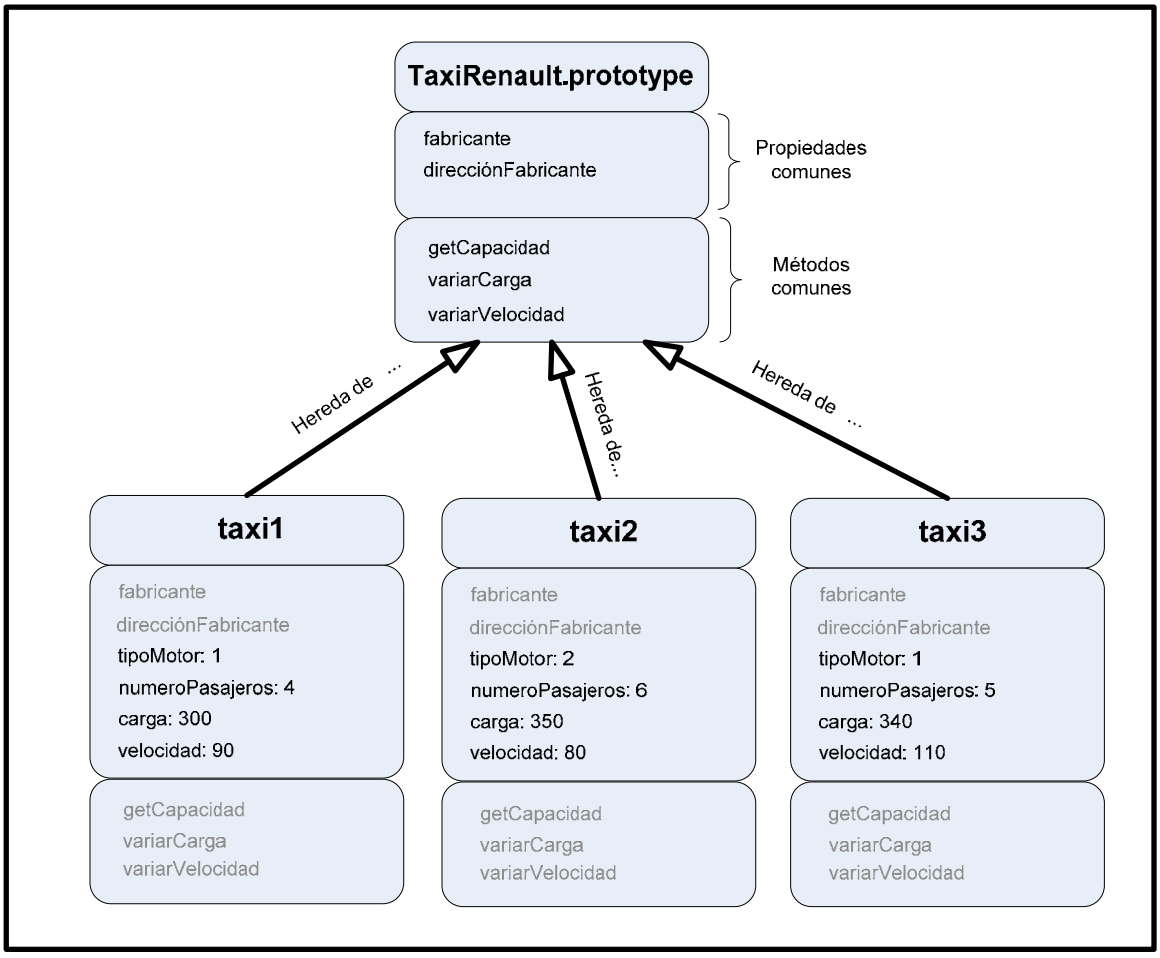
La idea de la herencia basada en prototipos JavaScript es definir un objeto (el objeto "padre" o prototipo) donde se aloja toda la información común que comparten todos los objetos de ese tipo (los objetos "hijos"). De esta manera se evita que cada objeto repita las propiedades y métodos comunes, lo cual ahorra memoria y agiliza la ejecución.

Las propiedades y métodos comunes se alojan en el prototipo u objeto padre, lo cual podemos representar con un esquema como este:



La ventaja de este esquema es que las propiedades y métodos comunes sólo existen una vez. Cuando un objeto invoca una propiedad, por ejemplo taxi1.fabricante, se comprueba si dicha propiedad está definida como propiedad específica del objeto, y si no es así, se busca en su prototipo de modo que si la propiedad existe en el prototipo, se devuelve esa propiedad de la misma manera que si fuera una propiedad del objeto. Si no se encontrara en el prototipo, se buscaría en el prototipo del prototipo (el padre del padre...) y así sucesivamente hasta encontrarla o no: a esto se le denomina cadena de prototipos o prototype chain y hablaremos sobre esto más adelante.

Al crear varios objetos TaxiRenault el esquema sería como este:



Lo que este esquema representa es que los objetos taxi1, taxi2 y taxi3 disponen de los atributos fabricante y direcciónFabricante, así como de los métodos getCapacidad, variarCarga y variarVelocidad. Pero estos atributos y métodos ya no están repetidos en cada uno de los objetos (no son propiedades y métodos de instancia) sino que están alojados en el prototipo, evitando así la repetición de información y sobrecarga de memoria.

Ten en cuenta que el uso de prototipos y herencia no es algo que haya que utilizar "para todo": sólo debemos pensar en ello cuando sea necesario. ¿Cuándo es necesario? Esta pregunta debe ser respondida para cada problema particular, pero en general piensa que no será necesario pensar en prototipos y herencia cuando sólo se vayan a crear unos pocos objetos. En cambio, en aplicaciones como juegos donde se usan intensivamente objetos (por ejemplo para representar meteoritos cayendo desde el espacio) será útil el uso de prototipos.

Si has trabajado previamente con otro tipo de lenguajes orientados a objetos (por ejemplo Java), posiblemente estés tratando de buscar las similitudes entre Java y JavaScript. Podríamos hacerlo, pero posiblemente esto nos generaría un lío de conceptos. Intenta olvidarte de las clases y mecanismos de herencia propios de otros lenguajes y cambia el chip: piensa en JavaScript.

A diferencia de otros lenguajes (como Java) donde existen clases y objetos, en JavaScript deberíamos decir que existen prototipos y objetos, aunque en muchos casos se usa la palabra clase por analogía.

DEFINIR PROTOTIPOS

En JavaScript el prototipo de un objeto está definido de forma predeterminada como una "propiedad oculta" de todo objeto, a la que podemos acceder con la sintaxis: nombreObjeto.prototype. El prototipo es a su vez un objeto (recuerda que una propiedad puede ser tanto un tipo primitivo como un objeto) y por tanto a dicho objeto se le pueden añadir propiedades y métodos.

Dado que JavaScript ofrece numerosas alternativas para trabajar con objetos, propiedades y métodos, existen numerosas alternativas para definir el prototipo de un objeto.

Alternativa 1: definir propiedades y métodos con la notación de punto:

```
function nombreObjeto (par1, par2, ..., parN) {
    this.propiedad1 = valorPropiedad1;
    this.propiedad2 = valorPropiedad2;
    ...
    }
nombreObjeto.prototype.propiedadComún1 = valorPropieadadComún1;
nombreObjeto.prototype.propiedadComún2 = valorPropieadadComún2;
...
nombreObjeto.prototype.métodoComún1 = function (par1, par2, ...) { ... }
nombreObjeto.prototype.métodoComún2 = function (par1, par2, ...) { ... }
...
```

Escribe este código y comprueba los resultados:

```
<!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 TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros; this.carga = carga;
this.velocidad = velocidad;
}
TaxiRenault.prototype.fabricante = 'Renault, S.A.';
TaxiRenault.prototype.direccionFabricante = 'c/R, Paris';
TaxiRenault.prototype.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
TaxiRenault.prototype.variarCarga = function (variacion) { this.carga = this.carga + variacion; }
TaxiRenault.prototype.variarVelocidad = function (variacion) { this.velocidad = this.velocidad + variacion; }
function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10); alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</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>
```

Alternativa 2: definir un objeto y convertir una instancia de dicho objeto en prototipo:

```
function nombreObjeto (par1, par2, ..., parN) {
this.propiedad1 = valorPropiedad1; this.propiedad2 = valorPropiedad2; ...
}
function objetoDestinadoASerPrototipo (par1, par2, ...) { ... }
nombreObjeto.prototype = new objetoDestinadoASerPrototipo();
```

Escribe este código y comprueba los resultados:

```
<!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 TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}
function prototipoTaxiRenault () {
this.fabricante = 'Renault, S.A.'; this.direccionFabricante = 'c/R, Paris';
this.getCapacidad = function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} }
this.variarCarga = function (variacion) { this.carga = this.carga + variacion; }
this.variarVelocidad = function (variacion) { this.velocidad = this.velocidad + variacion; }
}

TaxiRenault.prototype = new prototipoTaxiRenault();

function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10); alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</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>
```

Las propiedades y métodos de un prototipo a veces se dice que son propiedades y métodos de clase o estáticas tratando de buscar analogías con otros lenguajes.

Alternativa 3: definir un objeto y asignarle contenido a su propiedad prototype creando un objeto único con la sintaxis de literal (propiedades enumeradas con dos puntos y separadas por comas):

```
function nombreObjeto (par1, par2, ..., parN) {
this.propiedad1 = valorPropiedad1; this.propiedad2 = valorPropiedad2; ...
}
nombreObjeto.prototype = {
propiedadComún1: valorPropiedadComún1;
propiedadComún2: valorPropiedadComún2;
...
métodoComún1: function (par1, par2, ..., parN) { ... },
métodoComún2: function (par1, par2, ..., parN) { ... },
...
}
```

Escribe este código y comprueba los resultados:

```
<!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 TaxiRenault (tipoMotor, numeroPasajeros, carga, velocidad) {
this.tipoMotor = tipoMotor; this.numeroPasajeros = numeroPasajeros;
this.carga = carga; this.velocidad = velocidad;
}

TaxiRenault.prototype = {
fabricante: 'Renault, S.A.',
direccionFabricante: 'c/R, Paris',
getCapacidad: function () { if (tipoMotor == 'Diesel') { return 40;} else {return 35;} },
variarCarga: function (variacion) { this.carga = this.carga + variacion; },
variarVelocidad: function (variacion) { this.velocidad = this.velocidad + variacion; }
}

function ejemploObjetos() {
var taxi1 = new TaxiRenault(1, 4, 300, 90); var taxi2 = new TaxiRenault(2, 6, 350, 80);
var taxi3 = new TaxiRenault(1, 5, 340, 110);
alert ('El fabricante del taxi 2 es ' + taxi2.fabricante + ' y la velocidad del taxi 2 es ' + taxi2.velocidad);
taxi2.variarVelocidad(-10);
alert ('El taxi 2 ha reducido su velocidad y ahora es ' + taxi2.velocidad);
}
</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>
```

Podríamos especificar varias formas más de definir el prototipo de un objeto, pero con lo visto hasta ahora podemos seguir avanzando.

EJERCICIO

Define un tipo de objeto Cometa cuyas propiedades de instancia (específicas de cada objeto) sean diametro, temperatura y nombre. La temperatura será un valor numérico que suponemos está en grados centígrados. Como propiedad común a todos los objetos de tipo cometa define definicionSegunDiccionario (que debe contener la definición de cometa según el diccionario) y como métodos comunes obtenerRadio (que debe devolver el radio) y obtenerTemperaturaFarenheit (que debe devolver el valor de temperatura expresado en grados Farenheit). Crea tres objetos de tipo cometa y comprueba que puedes acceder tanto a las propiedades específicas como a las propiedades comunes y métodos comunes desde cada objeto.

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

Próxima entrega: CU01148E

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