CLASES Y OBJETOS

INTRODUCCIÓN

En el paradigma de programación orientada a objetos (POO, o OOP en inglés), un objeto es la unidad individual que en tiempo de ejecución realiza las tareas de un programa. Estos objetos interactúan unos con otros, en contraposición a la visión tradicional en la cual un programa es una colección de subrutinas (funciones o procedimientos), o simplemente una lista de instrucciones para el computador. Cada objeto es capaz de recibir mensajes, procesar datos y enviar mensajes a otros objetos de manera similar a un servicio.

En el mundo de la programación orientada a objetos (POO), un objeto es el resultado de la instanciación de una clase. Una clase es el anteproyecto que ofrece la funcionalidad en ella definida, pero ésta queda implementada sólo al crear una instancia de la clase, en la forma de un objeto.

SINTAXIS

La sintaxis básica para la declaración de clases es la siguiente:

```
var MiClase = new Class  ({
       //Cuerpo de la clase
});
```

Aqui tenemos nuestra primera clase llamada MiClase. El cuerpo de una clase comúnmente va encerrado entre llaves como las funciones, pero con mootools delimitaremos el inicio y fin de la clase con ({ y });

```
    //Clases
    var diccionario = new Class ({
         queidioma: function() {alert("Te voy a decir mi idioma");}
    });
```

Ahora tenemos una clase diccionario con un método queidioma. Los métodos se declaran de manera **nombre:function()**, donde function es la función que se ejecutara cada vez que llamemos a nuestro método nombre. Crearemos una instancia de la clase diccionario para ver nuestro ejemplo en acción:

```
    //Usando clases dentro de onload.
    //El código debe tener algo como <div id='i0'>Que idioma</div>
    var i0 = $('i0');
    i0.onclick= function() {
        var dic = new diccionario();
        dic.queidioma();
    };
```

Nuestro método "queidioma" también podría haber tenido parámetros, como el idioma:

```
    //Clases
    var diccionario1 = new Class ({
        queidioma: function(idioma) {escribe("Te via dici idioma " + idioma);}
    });
```

Pero lo mas lógico hubiera sido que el idioma sea una propiedad del diccionario. Usamos un método setIdioma que se encargara de fijar un valor en la propiedad nombre.

```
   var diccionario2 = new Class ({
     idioma: '',
     queidioma: function() {escribe("Te via dici mi idioma " + this.idioma);},
     setIdioma: function(idioma) {this.idioma = idioma}
   });
```

```
   var i2 = $('i2');
    i2.onclick= function() {
        var dic = new diccionario2();
        dic.setIdioma("ES"); //O también podríamos poner dic.idioma = "ES";
        dic.queidioma();
    };
```

Cosas nuevas a tener en cuenta:

1. Las propiedades se declaran de manera **nombre:valor**.

2. Los métodos y las propiedades siempre van separados por comas, tal como se hace con las posiciones de un array. Al finalizar una instrucción javascript de una función necesita ;

3. Para referirnos a una propiedad de la clase desde dentro de la misma lo hacemos con **this.propiedad**, tanto para lectura como para escritura. Y aunque no se vea en el ejemplo, lo mismo sucede para invocar un método, this.metodo();

Initialize

Mootools tiene tres propiedades: initialize, implement, extends. Initialize funciona como constructor de clase. En pocas palabras, es el método que se ejecuta en cuanto se crea una instancia de clase.

```
var diccionario3 = new Class ({
  idioma: '',
  initialize: function(idioma) {this.idioma = idioma},
  queidioma: function() {escribe("Te voy a decir mi idioma " + this.idioma);}
});
```

```
    var i3 = $('i3');
    i3.onclick= function() {
        var dic = new diccionario3("VA");
        dic.queidioma();
    };
```

Desaparece el método setIdioma reemplazándolo por nuestro método constructor initialize. Ahora éste es el que se encarga de fijar un valor para el idioma y los parámetros para el constructor se envían al momento de crear la instancia de nuestra clase.

Ahora podríamos crear varios diccionarios, donde cada una tendrá su idioma y sus métodos pero todo a partir de la única clase que hemos declarado:

```
var dices = new diccionario('ES');
var dicva = new diccionario('VA');
var dicen = new diccionario('EN');

dices.queidioma();
dicva.queidioma();
dicen.queidioma();
```

Extends

Otra de los tres propiedades de Mootools para clases: Extends. Este nos permite aplicar herencia de clases muy fácilmente.

```
var Padre = new Class ({
});

var Hija = new Class({Extends: Padre});
```

Ahora Hija es una copia de Padre.

```
//Clase padre
var diccionario3 = new Class ({
  idioma: '',
  p1: 'Primera palabra',
  initialize: function(idioma) {this.idioma = idioma},
  queidioma: function() {escribe("Te voy a decir mi idioma " + this.idioma);}
});

//Clase hija
var diccionario4 = new Class ({
  Extends: diccionario3,
  p1: 'Word one',  //Modificada
  p2: 'Word two'   //Ampliada
});
```

```
    var i4 = $('i4');
    i4.onclick= function() {
        var dic = new diccionario4("VA");
        dic.queidioma();
        escribe(dic.p1);
        escribe(dic.p2);
    };
```

Auto hereda todos los métodos y propiedades de la clase Vehiculo salvo la propiedad p1 que se sobrescribe.

Podemos aprovechar los métodos que sobrescribimos utilizando la función parent(), de esta manera:

```
//Clase hija
    var diccionario5 = new Class ({
         Extends: diccionario3,
         queidioma: function() {this.parent(); alert("y mas");}
    });
```

Nota: En la versión 1.11 esta propiedad era un método. Es una de las novedades en el cambio de versión.

Implement

Con extend duplicamos clases y a partir de las resultantes vamos agregando funcionalidades, con implement lo que hacemos es agregar funcionalidades/métodos a una clase en particular sin duplicarla.

Implement está como método y como propiedad. Como creo que la forma más lógica de uso es como método, así es como se explica. En la versión 1.11 solo está como método.

```
var diccionario3 = new Class ({
  idioma: '',
  p1: 'Primera palabra',
  initialize: function(idioma) {this.idioma = idioma},
  queidioma: function() {escribe("Te voy a decir mi idioma " + this.idioma);}
});

diccionario3.implement ({p3: 'Tercera palabra'});

var dir = new diccionario3();
escribe(dir.p3);
```

El/los métodos que implementamos pasan a formar parte de nuestra clase. Si implementamos un método que ya existe, lo sobrescribimos, pero en este caso no podremos reutilizarlo con parent.

Ejercicios:

1.- Crea una página html con un formulario (nombre: formulario). Que dicho formulario contenga una caja de texto y un botón enviar.

Crea una clase llamada comprobar que:

- Tenga una propiedad llamada **error1** y que tenga como valor inicial ‘El campo no puede quedar vacio’.
- Tenga una propiedad llamada **error2** y que tenga como valor inicial ‘El campo debe tener al menos 2 caracteres’.
- Tenga un método llamado **tavacio** que compruebe si el campo esta vacio o no. En caso de estar vacio devuelve un alert mostrando el valor de error1. Devuelve un return false. En caso de no estar vacio devuelve un return true y ejecuta el método slargo.

- Tenga otro método llamado **slargo** que compruebe si el campo tiene más de 2 caracteres o no (length). En caso de tener menos o igual a 2 devuelve un alert mostrando el valor de error2. Devuelve un return false. En caso de no estar vacio devuelve un return true.

Inserta ambos métodos al submit del formulario de manera que compruebe ambos casos.

2.- Completa el ejemplo anterior.
Crea un select para seleccionar idioma: a elegir entre dos valores es y va.
Extiende dos veces la clase comprobar y llama a una comprobarva y la otra comprobares. En las extensiones define las propiedades error1 y error2, de manera que valga

```
para comprobares =
        error1: 'Este campo no puede quedar vacío',
        error2: 'Este campo debe ser más largo'
para comprobarva =
        error1: 'Aquest camp no pot quedar buit',
        error2: 'Aquest camp ha de ser més llarg'
```

Elimina estas propiedades de la función comprobar.

Ahora haz que los mensajes sean en un idioma u en otro en función del idioma seleccionado.

3.- Completa el ejemplo anterior.
Crea otros dos campos en el formulario. Uno que sea un grupo de radios y el otro un campo de texto.

El grupo de radios: A elegir entre PAS, PDI y otros. Si marcamos otros, al hacer submit se debe comprobar que el campo de texto se ha rellenado.

Para ello usaremos el método implement.