UD04 Classes i objectes. Notació JSON

Definició d'objecte

- Tècnicament, un objecte JavaScript és un array associatiu format per les propietats i els mètodes de l'objecte i es pot crear amb l'operador **new i Object()** (veurem JSON com a alternativa)
- La forma més directa de definir les propietats i mètodes és la **notació de punts** dels array associatius. També es pot utilitzar la notació objecte["propietat"] ("propietat" pot ser també una expressió que torne una cadena de text)
- Per a eliminar una propietat s'ha d'utilitzar la paraula **delete**

Exemple :

```
var elObjeto = new Object();
elObjeto.id = "10";
elObjeto.nombre = "Objeto de prueba";

elObjeto.propiedad2 = "propiedad2";

elObjeto.muestraId = function() {
  alert("El ID del objeto es " + this.id ); }

delete elObjeto.propiedad2
```

La paraula clau **this**

- L'ús de la paraula clau **this**, utilitzada en mètodes d'objectes, sempre apunta a l'objecte que invoca el mètode.
- S'utilitza en la definició de mètodes quan es vol fer referència a l'objecte que el crida de manera genèrica, sense haver de preocupar-se del nom de la instància.

- També podem assignar als mètodes d'un objecte funcions definides amb anterioritat :

```
var elObjeto = new Object();
function obtieneId() {
     return this.id;
}
elObjeto.obtieneId = obtieneId;
```

Objectes que contenen altres objectes.

- També podem fer que un objecte continga un altre objecte :

Exemple :

```
var Aplicacion = new Object();
Aplicacion.Modulos = new Array();
Aplicacion.Modulos[0] = new Object();
Aplicacion.Modulos[0].titulo = "Lector RSS";


var inicial = new Object();
inicial.estado = 1;
inicial.publico = 0;
inicial.nombre = "Modulo_RSS";
inicial.datos = new Object();


Aplicacion.Modulos[0].objetoInicial = inicial;
```

La notació **JSON**

- JSON (JavaScript Object Notation) és un format senzill per a l'intercanvi d'informació.
- El format JSON permet representar estructures de dades (arrays) i objectes (arrays associatius) en forma de text.
- En els últims anys, JSON s'ha convertit en una alternativa al format XML, ja que és més curt, més fàcil de llegir i escriure.
- La notació JSON permet definir els arrays associatius d'una forma molt més concisa.
- Ja que els objectes són arrays associatius, es poden definir mitjançant la notació JSON

La notació **JSON**

- La notació JSON es compon de tres parts :
  - Els continguts de l'array associatiu es tanquen entre claus ({ ... }), l'array numèric entre claudàtors ([…]).
  - Cada element es compon d'una clau i un valor separats per dos punts (:). La clau sempre entre cometes dobles. El valor depén del seu tipus.
  - Els elements de l'array se separen per mitjà d'una coma (,)
  - Podem incloure un array associatiu o un array numèric dins del valor d'un element

```
var titulosModulos = {
    "Lector RSS": "rss",
    "Gestor de email":   "email",
    "Agenda": "agenda"
    };
```

Definir objectes mitjançant la notació **JSON**

- La forma habitual per a definir els objectes en JavaScript es basa en el següent model creat amb la notació JSON :

```
var objeto = {
  "propiedad1": valor_simple_1,
  "propiedad2": valor_simple_2,
  "propiedad3": [array1_valor1, array1_valor2],
  "propiedad4": { "propiedad anidada": valor },
  "metodo1": nombre_funcion_externa,
  "metodo2": function() { ... },
  "metodo3": function() { ... },
  "metodo4": function() { ... }
};
```

L'especificació completa de JSON es pot consultar en RFC 4627 (https://www.ietf.org/rfc/rfc4627.txt) i el seu tipus MIME oficial és application/json

Més informació : http://www.w3schools.com/js/js_json.asp

Definició de **Classe**

- La forma habitual de treball consistix a definir classes a partir de les quals es creguen els objectes.
- JavaScript no permet crear classes semblants a les de llenguatges com Java o C++.
- De fet, la paraula class només està reservada per al seu ús en futures versions de JavaScript
- És possible utilitzar en JavaScript uns elements semblants a les classes i que s'anomenen **pseudoclasses**.
- Els conceptes que s'utilitzen per a simular les classes són les **funcions constructores** i el **prototype** dels objectes.

Funcions constructores.

- JavaScript emula el funcionament dels constructors per mitjà de l'ús de funcions.
- Les funcions són objectes en Javascript i es poden instanciar amb l'operador **new** :
  - El següent exemple crea una funció anomenada Factura que s'utilitza per a crear objectes que representen una factura.

```
function Factura(idFactura,idCliente)
{
  this.idFactura = idFactura;
  this.idCliente = idCliente;
}
```

  - És possible crear un objecte de tipus Factura i simular el funcionament d'un constructor.

```
var laFactura = new Factura(3, 7);
```

Funcions constructores.

- Les funcions constructores no sols poden establir les propietats de l'objecte, sinó que **també poden definir els seus mètodes.**

```
function Factura(idFactura, idCliente) {
  this.idFactura = idFactura;
  this.idCliente = idCliente;

  this.muestraCliente = function() {
    alert(this.idCliente);
  }

  this.muestraId = function() {
    alert(this.idFactura);
  }
}
```

```
var laFactura = new Factura(3, 7);

laFactura.muestraCliente();

var otraFactura =

otraFactura.muestraId();
                    new Factura(5,4);
```

- El més habitual, no obstant això, és definir els mètodes fora de la funció constructora (exemple pàgina 12)
- Una vegada definida la pseudoclasse per mitjà de la funció constructora, ja és possible crear objectes d'eixe tipus amb l'operador **new**.

Funcions constructores. Prototype.

- **Inconvenient :**
- Amb esta tècnica, cada vegada que s'instància un objecte, es definixen tantes noves funcions com a mètodes incloga la funció constructora.
- La penalització en el rendiment i el consum excessiu de recursos d'esta tècnica pot suposar un inconvenient en les aplicacions professionals realitzades amb JavaScript.
- **Solució :**
- Tots els objectes de JavaScript inclouen una propietat anomenada **prototype** que és una referència interna a un altre objecte
- Qualsevol propietat o mètode que continga l'objecte **prototype**, **només està present** de forma automàtica en l'objecte original i la resta d'objectes fan referència a aquest. .

Funcions constructors. Prototype.

Per a aquells elements comuns per a tots els objectes, normalment mètodes i constants, s'utilitza la propietat **prototype** de l'objecte

- D'aquesta manera, una vegada creada la funció constructora que s'utilitza per a crear l'objecte on s'afegiran els elements els valors dels quals siguen diferents entre objectes, es poden crear amb **prototype** els elements comuns, normalment fora de la funció constructora.

- Exemple :

```
function Car(sColor, iDoors, iMpg) {
    this.color = sColor;
    this.doors = iDoors;
    this.mpg = iMpg;
    this.drivers = new Array("Mike", "Sue");
}

Car.prototype.showColor = function () {
     alert(this.color);
};
```

```
var oCar1 =
    new Car("red", 4, 23);
var oCar2 =
    new Car("blue", 3, 25);
oCar1.drivers.push("Matt");
alert(oCar1.drivers);
alert(oCar2.drivers);

oCar1.showColor();
oCar2.showColor();
```

Cada tipus d'objecte diferent hereda d'un objecte **prototype** diferent

Classes en ES2015

- Amb la versió de Javascript ES6 (ECMAScript 2015) podem utilitzar **class** per a la definició d'objectes.

- La paraula reservada **class** és un tipus de funció, però en lloc d'usar la paraula clau **function**, s'utilitza la paraula **class** i les propietats són assignades usant el mètode **constructor()** .

- Exemple :

```
class Car {
    constructor(sColor, iDoors, iMpg) {
        this.color = sColor;
        this.doors = iDoors;
        this.mpg = iMpg;
        this.drivers = new Array("Mike", "Sue");
    }
    showColor = function () { alert(this.color); }
};

oCar1 = new Car("red", 4, 23);
```

Els mètodes creats dins de la classe, però fora del constructor internament s'afigen al **prototype** de la funció constructora

Classes en ES2015

- **Herència** : En ES2015, una classe pot heretar d'una altra utilitzant la paraula reservada **extends**.
  - Heretarà totes les propietats i mètodes de la classe pare. Per descomptat, podrem sobreescriure'ls en la classe filla, encara que continuarem podent cridar als mètodes de la classe pare utilitzant la paraula reservada **super**.
    - Si es crea un constructor en la classe filla, hem de cridar el constructor pare.

- **Mètodes estàtics** : En ES2015 podem declarar mètodes estàtics amb la paraula **static**, però no propietats estàtiques.
  - Estos mètodes es criden directament utilitzant el nom de la classe i no tenen accés a l'objecte this (no hi ha objecte instanciat) .

El mètode call() i el mètode apply().

- Ens permeten executar una funció com si fóra un mètode d'un altre objecte.
- L'única diferència entre els dos mètodes és la forma en què es passen els arguments a la funció, en apply() els paràmetres es passen com un array :

```
function miFuncion(x) {
  return this.numero + x;
}
 var elObjeto = new Object();
 elObjeto.numero = 5;
 var resultado = miFuncion.call(elObjeto, 4);
 // var resultado = miFuncion.apply(elObjeto, [4]);
 alert(resultado);
```

Propietats i mètodes privats

- A partir d'ES2022, es pot definir camps privats per a una classe usant el signe # prefixat als identificadors.
- Es pot accedir a camps privats des de dins de la declaració de classe

```
class Circle {
     var #radius;
     constructor(value) {
        // You can access private field from constructor
        this.#radius = value;
    }
    get area() {
        return Math.PI * Math.pow(this.#radius, 2);}
}
const circle = new Circle(10);
console.log(circle.#privateField);   // Syntax error
```

Propietats i mètodes privats

- Igual que els camps públics, els camps privats s'agreguen en el moment de la construcció en una classe base o en el punt on s'invoca **super()** en una subclasse.
- Però intentar accedir al camp privat des d'una subclasse donarà com resultat SyntaxError.
- Els camps privats només són accessibles en la classe adjunta (classe on estan definits) .
- Javascript no admet la paraula clau **private**, però es pot assumir per ara que la paraula clau **private** en les classes podria ser compatible en alguns navegadors

```
class Cylinder extends Circle {
    var #height;
    var cRadius;
    constructor(radius, height) {
        super(radius);
        this.#height = height;
        // cannot access the #radius of the Circle class here
        this.cRadius = this.#radius // Syntax Error
    }
}
```