JavaScript orienté objet

Aous Karoui
aous.karoui@univ-lemans.fr

Concepts connus de POO

Espace de nom

Classe

Objet

Propriété (attribut/variable d'instance)

Méthode

Constructeur

Héritage

Encapsulation

Espace de nom

Permet de “regrouper” des fonctionnalités sous un même nom

Evite les conflits de nommage

Structure le code d’une application

En Java: les packages, exemple: “java.util”

En C++: les namespaces, exemple: “namespace std”

En JS: pas de mot-clef propre au espaces de nom, on utilise un objet.

Espace de nom

```
// espace de nom global
var MONAPPLICATION = MONAPPLICATION || {};

// espace de noms "fils"
MONAPPLICATION.event = {};

// On utilise un conteneur pour les événements
MONAPPLICATION.event = {
  addListener: function(el, type, fn) {
  //  le corps de la méthode
  },
  removeListener: function(el, type, fn) {
  //  le corps de la méthode
  },
  getEvent: function(e) {
  //  le corps de la méthode
  }

}

// Exemple de syntaxe pour utiliser la méthode addListener :
MONAPPLICATION.event.addListener("monÉlément", "type", callback);
```

Objet

Représente un objet du monde réel (ou pas)

Représente une instance d'un concept, exemple Pomme A et Pomme B

En Java: est une instance d'une classe, créé avec "new Classe()"

Sa structure (propriétés et méthodes) est fixe

En JS: peut ne pas avoir de classe explicite (*Object)*

Structure modifiable à souhait

```
var rectangle = {

    upperLeft : { x : 2, y : 2 },

    lowerRight : { x : 4, y : 4}
```

Constructeur

Méthode/Fonction spéciale qui permet d'instancier un objet

Sert principalement à fixer l'état initial de l'objet

Est appelé lors de l'utilisation du mot-clef "new"

En Java: est déclaré dans la classe, obligatoirement avec le même identifiant

En JS: est définie comme une fonction classique, nommée comme la "classe" représentée, par convention (commence par une majuscule)

on peut aussi utiliser *Object.create* pour instancier sans appeler le constructeur

Constructeur

```
function Personne(nom) {
  this.nom = nom;
  console.log('Nouvel objet Personne créé');
}

var personne1 = new Personne('Alice');
var personne2 = new Personne('Bob');

//on affiche le nom de personne1
console.log('personne1 est ' + personne1.nom); // personne1 est Alice
console.log('personne2 est ' + personne2.nom); // personne2 est Bob
```

Propriétés

Sont les données de l'objet: représentent son état

En Java: sont obligatoirement déclarées (visibilité, type, identifiant, valeur par défaut etc.)

En JS: ne sont pas déclarées, pas typées et sont initialisées dans le constructeur.

Pas de notion de visibilité: tout est publique

Par convention, *private* = identifiant commençant par "_"

```
function Personne(nom) {
  this.nom = nom;
}
```

Classe/Prototype

En JS: le concept de classe n’existe pas.

JS utilise des “prototypes”: un objet qui définit la structure des objets qui le “copie”

En bref: une classe sous forme d’objet

```
//un constructeur vide
var Toto = function(){};

//définition du prototype
Toto.prototype = {maProp: "coucou", maMethode: function(){console.log("toto")}};

//Instanciation
var test = new Toto();

console.log(test.maProp);
```

Méthode

En JS: les méthodes sont des fonctions classiques

Elles sont définies dans l'objet prototype

On peut les appeler en "dehors" de l'objet (avec *call* ou *apply*)

```
donnerUnNom.call(personne1); // 'Je suis Gustave'
```

```
function Personne(nom) {
  this.nom = nom;
}
Personne.prototype.afficherNom = function() {
  console.log("Je suis "+this.nom);
};
var personne1 = new Personne('Gustave');
var donnerUnNom = personne1.afficherNom;
personne1.afficherNom(); // 'Je suis Gustave'
```

Héritage

Représente la relation “est un”

Un cheval est un animal

Plus complexe à mettre en oeuvre en JS

Pas d’héritage multiple (comme en Java)

On utilise un objet de la classe mère comme prototype de la classe fille

Dans le constructeur fille, il faut appeler explicitement le constructeur de la classe mère

Héritage

```
// Le constructeur Personne
var Personne = function(nom) {
    this.nom = nom;
};
Personne.prototype.marcher = function(){
  console.log("Je marche !");
};
Personne.prototype.direBonjour = function(){
  console.log("Bonjour, je suis "+this.nom);
};
// Le constructeur Étudiant
function Etudiant(nom, sujet) {
  // On appelle le constructeur parent
  // pour profiter des propriétés définies dans la fonction
  Personne.call(this.nom);
  this.sujet = sujet;
}
// On déclare l'héritage pour bénéficier de la chaîne de prototypes
// Attention à ne pas utiliser "new Personne()". Ceci est incorrect
// on ne peut pas fournir l'argument "nom". C'est pourquoi on appelle Personne avant, dans le constructeur Étudiant.
Étudiant.prototype = Object.create(Personne.prototype);
// on corrige le constructeur qui pointe sur celui de Personne
Étudiant.prototype.constructor = Etudiant;
// on remplace la méthode direBonjour pour l'étudiant
Étudiant.prototype.direBonjour = function(){
 console.log("Bonjour, je suis un "+ this.nom + ". J'étudie " + this.sujet + ".");
};
// on ajoute la méthode aurevoir
Étudiant.prototype.aurevoir = function(){
  console.log('Au revoir');
};
```

Héritage

```
var etudiant1 = new Etudiant("Jean", "la physique appliquée");
etudiant1.direBonjour();
etudiant1.marcher();
etudiant1.aurevoir();

// on vérifie l'héritage
console.log(etudiant1 instanceof Personne); // true
console.log(etudiant1 instanceof Etudiant); // true
```

Sources / A lire

https://developer.mozilla.org/fr/docs/Web/JavaScript/Introduction_%C3%A0_JavaScript_orient%C3%A9_objet

http://javascript.info/tutorial/object-oriented-programming

https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Le_mod%C3%A8le_objet_JavaScript_en_d%C3%A9tails