INTRODUCTION À L'ALGORITHMIQUE

OBJETS

2015-01

NETCAT / @netcat

CONCEPT

- Basés sur les tableaux associatifs
- exemples d'objets usuels : String, Array, Date, ...
- en JS tout est objet ! (fonction = objet = ...)

UTILISATION DES OBJETS

- propriétés, méthodes et notation pointée

```
"abc".length --> 3 // length est une "propriété"
"abc".toUpper() --> "ABC" // toUpper est une "méthode"
"abc".toUpper().length --> 3
```

UN OBJET USUEL : STRING

UN OBJET USUEL : DATE

UN OBJET USUEL : MATH

STRUCTURES / OBJETS

DÉFINITION AVEC { *ATTR* : *VALEUR* ... }

- Sorte de *table de hashage*
- Permet de regrouper des propriétés / méthodes
- Permet la notation pointée

```
michel = {
    nom: "Michel",
    age: 29
};

console.log(michel.nom + " " + michel.age);
```

LES MÉTHODES : DES FONCTIONS COMME VALEURS

On peut aussi avoir une méthode comme attribut:

```
michel = {
    nom: "Michel",
    age: 29,
    incrementerAge: function(inc) { this.age += inc; },
    estVieux: function() {
        return (this.age > 30);
    }
};

michel.incrementerAge(4);
console.log(michel.estVieux());
```

INSTANCES ET OBJET *THIS*

- Lorsqu'on exécute une fonction avec `new`, `this` sera le nouvel objet créé
- Quand il est déclaré dans la méthode d'un objet, il représente cet objet
- Hors du contexte d'un objet, `this` sera alors l'objet `window`
- `this` est dynamique en Javascript: il est identifié lors de l'exécution d'une fonction
- L'objet this peut être fourni explicitement en paramètre

PROTOTYPE

PARTAGE DE STRUCTURE ENTRE INSTANCES

On veut maintenant représenter Paul, 46 ans

- Il partage les même deux méthodes, ainsi que les deux attributs de Michel (avec des valeurs différentes)
- On pourra créer Paul en utilisant Michel comme prototype
- Il héritera alors de tous les attributs de Michel
- On pourra changer la valeur de certains attributs, qui seront alors locaux à l'objet qui représente Paul

PROTOTYPE - EXEMPLE

```
// On crée l'objet vide
paul = { };

// Paul héritera des attributs de Michel
paul.__proto__ = michel;

// On redéfinit les attributs locaux de Paul
paul.nom = "Paul";
paul.age = 46;
console.log(paul.nom + " " + paul.age); // Paul 46

// Si l'objet ne possède pas d'attribut local, on
// le cherche dans son prototype
console.log(paul.estVieux()) // true
delete paul.nom
console.log(paul.nom + " " + paul.age); // Michel 46
```

CLASSES

On aimerait bien avoir une manière plus pratique de définir des "classes"

Pour y arriver, il faut faire appel à des caractéristiques spéciales des fonctions:

- Toute fonction f possède un attribut spécial prototype, qu'on peut faire pointer sur n'importe quel objet
- Appelons p cet objet
- L'exécution de l'opération new f() créera un objet qui se verra automatiquement attribuer comme prototype l'objet p
- Ce type de fonction est appelé constructeur et il est d'usage d'utiliser un nom commençant par une majuscule

CLASSES - SUITE

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

Personne.prototype = {
ajusterAge: function(inc) { this.age += inc; },
            estVieux: function() { return (this.age > 30); }
};

michel = new Personne("Michel",29);
paul = new Personne("Paul",46);
console.log(paul.nom + " " + paul.age); // Paul 46
console.log(paul.estVieux()) // true
console.log(michel.nom + " " + michel.age); // Michel 29
console.log(michel.estVieux()) // false
```

CONSTRUCTEUR

- une simple fonction chargée d’initialiser un objet quand elle est invoquée par l'opérateur new.
- fonction dont le nom commence par une majuscule et utilisant this pour référencer l'objet lui-même
- les arguments de la fonction servent à initialiser les propriétés de l’objet

```
function Personne (nom, age) {
     this.nom = nom;
     this.age = age;
```

DESTRUCTEUR

- En javascript - impossible de créer une fonction destructeur pour supprimer un objet
- L'opérateur delete permet de supprimer des propriétés, des variables (sauf celles déclarées avec l'instruction var), des éléments de tableau ...
- Le mot clé null peut également être utilisé pour supprimer une référence.

```
obj = new Object();
delete obj;
// Supprime obj
obj = null;
// Vide obj

var obj = new Object();
delete obj;
// Ne supprime pas obj
obj = null;
// Vide obj
```

EN VRAC

- portée d*s variables ? (privée / publique)
- lister les propriétés (for ... in / hasOwnProperty)

HÉRITAGE

THE MOST MODERN, BEST PATTERN

```
B.prototype = Object.create(A.prototype);
```

This uses the Object.create function to set B.prototype to a new object, whose internal [[Prototype]] is A.prototype. This is basically exactly what you want: it will make B instances delegate to A.prototype when appropriate. (cf http://es5.github.io/#x15.2.3.5)

THE ORIGINAL PATTERN

```
B.prototype = new A();
```

MIXINS

https://javascriptweblog.wordpress.com/2011/05/31/a-fresh-look-at-javascript-mixins/

RÉFÉRENCES

- Classe et héritage en JS
- Javascript - Michel Gagnon - Ecole polytechnique de Montréal