Introduction

Les classes ont été introduites avec ECMAScript2015. Attention, certaines propriétés sont encore en mode expérimentale et ne sont pas implémentées dans tous les navigateurs. Vous trouverez ici un tableau de compatibilités pour les navigaeurs : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes#compatibilit%C3%A9_des_navigateurs

Définition

Les classes sont des fonctions spéciales en JS. C'est un sucre syntaxique. Elles sont une simplification de l'écriture des fonctions constructeurs.

Généralités sur la notion de classe

On utilise le mot réservé **class** suivi du nom de la classe elle-même pour la définir. La classe est un modèle à partir duquel on peut créer autant d'objets qu'on le souhaite, appelés **instances**. Pour créer une nouvelle instance, on utilise le mot-clé new suivi du nom de classe.

Lors de l'instanciation avec le mot-clé new, une méthode spéciale de la classe est appelée le **constructeur**. En JS, le constructeur porte le nom réservé constructor, et les **attributs (ou propriétés)** de la classe y sont définis. On attache les attributs à l'instance de classe à l'aide du mot clé this :

```
class Rectangle{

    constructor(w, h){
        // attributs de classe
        this._w = w;
        this._h = h;

        this.name = "Rectangle";
    }

    // ...
}

// création de deux instances
const unRectangle = new Rectangle(30, 50);
const unAutreRectangle = new Rectangle(20, 35);
```

Contrairement à d'autres langages comme PHP, JS n'implémente pas de visibilité private (...pas avant ES2019 en tout cas !), tous les attributs et toutes les méthodes sont accessibles dans le script courant. On peut toutefois faire "comme si" une propriété était privée, ne la manipuler directement que depuis l'intérieur de la classe, et définir des accesseurs **setter** et **getter** pour y accéder en écriture et en lecture. Il faudra cependant faire attention au nom de ces attributs dans la classe, une technique classique consiste à préfixer leurs noms par un underscore.

On définira les propriétés "privées" en les préfixant d'un underscore. Elles seront appelées, une fois les setter et getter définis, dans le script courant sans l'underscore et seront traitées comme des variables attachés à l'objet.

```
class Model{
    // constructeur
    constructor(tableName){
        this._table = tableName;
    }

    get table (){
        return this._table;
    }

    set table(tableName){
        this._table = tableName;
    }
}

const m = new Model("posts");

m.table ; // accéder à la valeur de l'attribut, l'accesseur get table est invoqué

m.table = "POSTS";  // modifier l'attribut, l'accesseur set table est invoqué
```

Les **méthodes** de classe sont des simples méthodes nommées; voyez l'exemple de la méthode **dim** ci-dessous dans la classe Rectangle :

```
class Rectangle{
    // constructeur
    constructor(w, h){
        this._w = w;
        this._h = h;
    }

    // EXEMPLE DE METHODE DE CLASSE
    dim(){
        return `Width : ${this._w} Height : ${this._h}`;
    }

    // getter
    get area(){
        return this._w * this._h;
    }

    // setter
    set w(w){
        this._w = w;
    }

     set h(h){
        this._h = h;
    }

}

let r1 = new Rectangle(10, 2);
// 20
console.log(r1.area);

r1.w = 100;
r1.h = 30;

// 3000
console.log(r1.area);

console.log(r1.dim())
```

01 Exercice Parser

Créez une classe Parser qui permettra de parser une chaîne de caractères en fonction d'un motif donné. Voyez lexemple de l'utilisation de cette classe ci-dessous avant d'implémenter le code. Gardez les digits uniquement ainsi que les nombres dans la chaîne de caractères.

```
const phrase = '8790: bonjour le monde:8987:7777:Hello World:    9007';

const p = new Parser(':');
p.parse(phrase);
console.log(p.str);
//8790 8987 7777 9007
```

Héritage ou sous classe

Vous pouvez spécialiser une classe en héritant d'une classe parente plus générale. Rappelez-vous du principe de l'héritage en objet : une classe fille est une sorte de par rapport à la classe mère. Par exemple, un Lion est une sorte d'Animal. Dans ce cas la classe Lion est la classe fille de la classe Animal. C'est une spécialisation de la classe Animal.

Pour définir une classe étendue vous devez utiliser le mot clé extends.

Le mot clé super permet de faire passer des valeurs au constructeur de la classe mère. Attention, si vous êtes dans une classe dérivée (fille) et que vous définissez un constructeur de classe, vous êtes obligé d'utiliser super pour accéder au constructeur de la classe mère. Si toutefois vous ne définissez pas de constructeur dans votre classe dérivée, le constructeur de la classe mère est automatiquement appelé. Notez enfin que si vous définissez des attributs de classe dans le constructeur de votre classe dérivée, vous devez le faire après la méthode super; JS bloquera la compilation si ce principe de syntaxe n'est pas respecté.

```
class Animal {
    constructor(name) {
        this._name = name;
    }

    set name (name){
        this._name = name;
    }

    speak(){
        return `Name : ${this._name}`;
    }
}

class Lion extends Animal {
  constructor(name) {
    // écrivez super avant la définition des nouvelles propriétés de classe
    super(name);
    this.force = 100;
  }
}

let lion = new Lion("Shere")
lion.name = "Shere Khan";

console.log(lion.speak())
```

02 Exercice Square Rectangle

Créez une classe Square et Rectangle. Laquelle des deux classes hérite de l'autre ? Répondez à la question avant de les implémenter ?

En utilisant le principe de l'héritage créez la classe Square avec un constructeur. Cette class n'aura pas d'autre méthode. Implémentez dans la classe Rectangle les méthodes suivantes : area, dim. Créez les setter et getter permettant de mettre à jour les attributs de la classe.

```
let square = new Square( /* CODE TODO */ ) ;  // à vous de compléter cette classe. On aimerait avoir un carré héritant de Rectangle.
```

Attribut statique

Vous pouvez définir des attributs statiques dans une classe JS. Dans ce cas cet attribut dépendra de la classe elle-même et non de son instance.

```
class Lion extends Animal {

    static AGE_MIN = 1;

    constructor(name) {
        super(name);
    }

    get age() {
        return AGE_MIN;
    }

}
```

Méthodes privées (expérimentales)

Vous pouvez définir des attributs privés dans une classe JS de la manière suivante; ces attributs ne seront accessibles qu'à l'intérieur de la classe. Attention, les champs ou attributs privés doivent être déclarés en premier dans la déclaration des champs.

```
class Rectangle {
  #h = 0;
  #w = 0;
  constructor(h, w){
    this.#h = h;
    this.#w = w;
  }
}
```

Modifié le: vendredi 22 septembre 2023, 12:20