ES6 : les classes

Jusqu'à ES5 JavaScript ignore les classes et l'héritage classique. Avec ES6 on va disposer désormais de quelque chose qui se rapproche vraiment de ce qui existe dans les autres langages, même s'il s'agit juste d'une autre syntaxe pour un système objet qui reste fondamentalement le même et on peut parler ici de simulation parce que les prototypes restent toujours aux commandes !

Les classes

ES6 introduit le mot-clé **class** pour créer une classe. Voici un exemple :

```
class User {
    constructor(name) {
        this.name = name;
    }
    salut() {
        alert('Coucou ' + this.name + ' !');
    }
}
```

Si vous pratiquez des langages comme **C#**, **Java**, ou même **PHP**, vous ne devez pas être dépaysé avec la syntaxe :

- on déclare la classe User avec **class**,
- on définit une propriété **name**,
- on a ensuite un constructeur (**constructor**) avec un paramètre qui permet d'assigner la propriété,
- pour finir on a une méthode (**salut**) qui permet de saluer l'utilisateur.

Pour utiliser cette classe et ainsi créer et utiliser un objet c'est aussi une syntaxe classique :

```
let user = new User('Marcel');
user.salut();  // Coucou Marcel !
console.log(user instanceof User);  // true
console.log(user instanceof Object);  // true
```

```
console.log(typeof User);  // function
```

Avec ES5 on aurait écrit ceci :

```
function User(name) {
  this.name = name;
}
User.prototype.salut = function() {
  console.log('Coucou ' + this.name + ' !');
};
var user = new User('Marcel');
user.salut();
console.log(user instanceof User); // true
console.log(user instanceof Object); // true
console.log(typeof User); // "function"
```

Pour JavaScript ça semble être exactement la même chose avec juste une syntaxe différente.

Mais il y a plusieurs différences :

- le code dans une classe est automatiquement en mode **strict**,
- les méthodes ne sont pas énumérables,
- vous obtenez une erreur si vous n'utilisez pas **new** pour la classe, ou si vous utilisez **new** pour une méthode,
- surcharger le nom de la classe avec le nom d'une méthode renvoie aussi une erreur.

Les méthodes statiques

On peut créer des méthodes statiques avec le mot clé **static** :

```
class Nombre {
  ajoute(nombre1, nombre2) {
    return nombre1 + nombre2;
  }
  static soustrait(nombre1, nombre2) {
    return nombre1 - nombre2;
  }
}
let nombres = new Nombre(6, 3);
console.log(nombres.ajoute(6, 3));  // 9
```

```
console.log(Nombre.soustrait(10, 4));  // 6
```

On voit ici la différence entre la méthode **ajoute** qui est classique et fonctionne sur une instance et la méthode **soustrait** qui est statique qui fonctionne à partir de la classe elle-même.

Les accesseurs

On a vu ci-dessus qu'on peut créer une propriété directement dans la classe mais on peut aussi utiliser des accesseurs :

```
class User {
    constructor(nom, prenom) {
        this.nom = nom;
        this.prenom = prenom;
    }
    get nomComplet() {
        return this.prenom + ' ' + this.nom;
    }
    set nomComplet(value) {
        let decompose = value.split(' ');
        this.prenom = decompose[0];
        this.nom = decompose[1];
    }
}
let user = new User('Durand','Pierre');
console.log(user.nomComplet);  // Pierre Durand
user.nomComplet  = 'Marc Assain';
console.log(user.nomComplet);  // Marc Assain
```

L'héritage

La syntaxe est complétée par le mot clé **extends** pour l'héritage :

```
class User {
  constructor(name) {
    this.name = name;
  }
  salut() {
    console.log('Coucou ' + this.name + ' !');
  }
```

```
}
class Redactor extends User {
  constructor(name, category) {
    super(name);
    this.category = category;
  }
  salut() {
     console.log('Coucou ' + this.name + ' de la catégorie ' +
this.category + ' !');
  }
}
let redactor = new Redactor('Marcel', 'Jeunesse');
redactor.salut();  // Coucou Marcel de la catégorie Jeunesse !
```

Ici la classe **Redactor** hérite de la classe **User**.

Le mot clé **super** sert à appeler une méthode définie dans la classe parente, ici on appelle le constructeur de la classe parente pour assigner une valeur à la propriété **name**.

On se rend compte également que la méthode **salut** de la classe **Redactor** surcharge la méthode de même nom de la classe parente.

En résumé

- ES6 ajoute le mot clé class pour simuler le fonctionnement classique des classes mais les prorotypes restent toujours aux commandes.
- Le mot clé **extends** permet de créer de l'héritage de classe.
- On peut créer des méthodes statiques dans une classe avec le mot clé **static**.