CLASSES AVEC NODE.JS

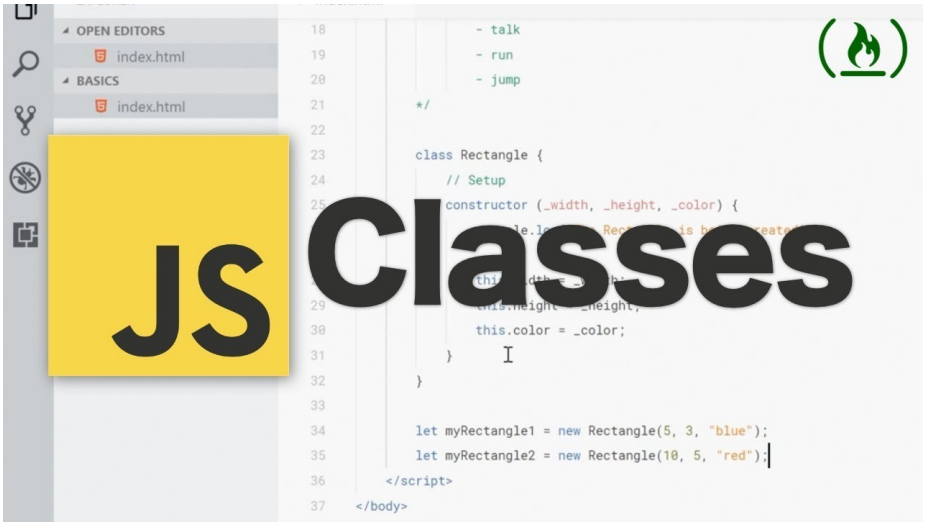


TABLE OF CONTENTS

1 Introduction

1.1 Généralités

JavaScript est un langage objet très puissant, qui n'a évolué pendant des années : auparavant, il n'était utilisé que côté client (navigateur) et satisfaisait les développeurs de page web.

Depuis que Google a créé NodeJS (JavaScript depuis les serveurs), l'ECMA a fait beaucoup évolué JavaScript et depuis ES6 (ou ES2015) a introduit la notion de classe tel que nous le connaissons dans les autres langages.

1.2 Objectifs

Ce support permet donc de comprendre comment JavaScript gérait les classes avant ES6, comment fonctionne la notion de prototype et enfin, comment utiliser les nouvelles classes dans un projet Node.JS.

Il doit donc également vous permettre de convertir de vieux codes dans la nouvelle norme.

2 Classes sous ES5

2.1 Classe de base

L'idée est d'utiliser un objet JavaScript et de l'instancier bêtement :

```
// définition de la classe
var Chevalier = function(nom) {
      var arme = "Excalibur" // équivalent variable privée
      this.nom = nom
      this.getArme = function() {
            return arme // accès local possible
      }
}

// Utilisation de la classe
var Arthur = new Chevalier("Arthur")
console.log(Arthur.getArme())     // affiche 'Excalibur'
console.log(Arthur.arme)          // undefined
console.log(Arthur.nom)
```

Cependant, ce modèle n'est pas satisfaisant, car chaque fois que l'on instancie la classe, on instancie les variables mais aussi les méthodes : sur une grosse application professionnelle, ce n'est pas acceptable.

Si une méthode est lourde et longue (comprendre, prend beaucoup de place en mémoire), chaque instanciation prend beaucoup plus de place qu'elle ne devrait.

2.2 PROTOTYPE

Pour lutter contre le problème d'instanciation des méthodes, une solution consiste à utiliser les prototypes.

Cela s'applique sur le constructeur. Lorsque le langage tente d'accéder à une propriété inexistante de l'instance, il tente de valider s'il existe un prototype de cette propriété. Un prototype contient les attributs manquants d'une instance.

En fait, **le prototype est le plan de la classe que le constructeur doit créer**.

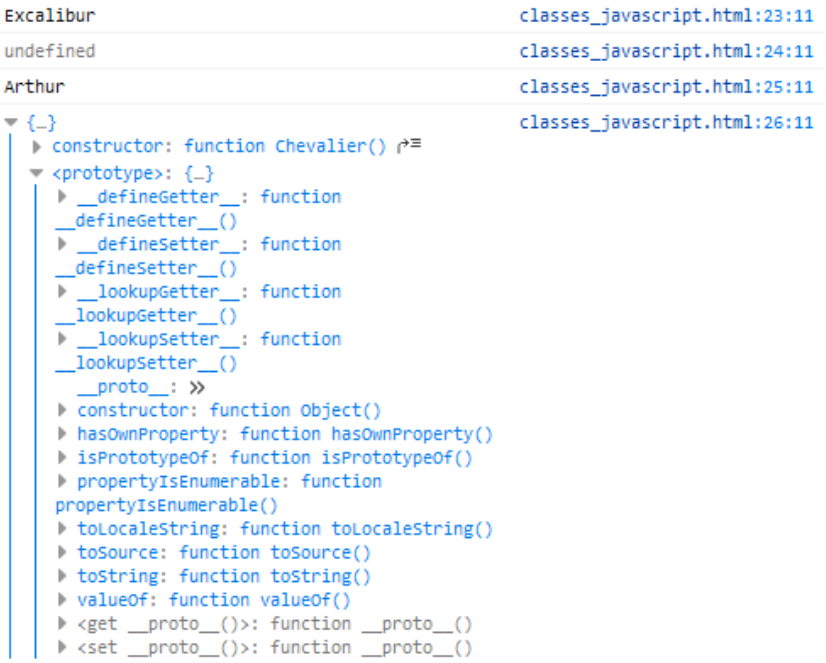
En effet, il est possible d'instancier une classe sans appeler le constructeur (particularité JavaScript) en utilisant Object.create().

```
var Lancelot = Object.create(Chevalier, {"nom" : {value : "Lancelot"}})
console.log(Lancelot.nom)
```

Pour mieux comprendre le concept, ajoutons la ligne suivante au code précédent :

```
console.log(Chevalier.prototype)
```

Le résultat est le suivant :



Le mot-clé prototype permet à Javascript d'ajouter des propriétés ou des fonctions à une classe existante (de manière dynamique).

```
Chevalier.prototype.titre = "Duc";
Chevalier.prototype.Parer = function() { console.log("Pare les coups"); }
var Bayard = new Chevalier('Bayard')
console.log(Bayard.titre)
```

2.3 UTILISATION DES PROTOTYPES

Ainsi, le prototype permet de n'avoir qu'une instance de la fonction.

```
// définition de la classe
var Chevalier = function(nom) {
      var arme = "Excalibur" // équivalent variable privée
      this.nom = nom
      this.level = 1
      this.getArme = function() {
            return arme // accès local possible
      }
}
Chevalier.prototype.getAttaque = function() {
      console.log(this.level)
      return Math.random()*this.level
}
Chevalier.prototype.setLevel = function(level) {
      this.level = level
}

// Utilisation de la classe
var Arthur = new Chevalier("Arthur")
console.log(Arthur.getArme())     // affiche 'Excalibur'
console.log(Arthur.arme)          // undefined
console.log(Arthur.nom)
Arthur.setLevel(12)
console.log(Arthur.getAttaque())
console.log(Arthur.prototype)

// Définition partielle de Lancelot (sans constructeur)
var Lancelot = Object.create(Chevalier, {"nom" : {value : "Lancelot"}})
console.log(Lancelot.nom)         // fonctionne
Lancelot.setLevel(12)             // il manque des propriétés
console.log(Lancelot.getAttaque())
```

prototype

getter

Définition de la classe

3 LES CLASSES ES6

Depuis ES6 ou ES2015, JavaScript s'est enrichi d'un véritable système de classe.

Il s'agit d'un sucre syntaxique qui rapproche JavaScript des véritables langages orientés objets : on trouve désormais la déclaration de constructeur (mot-clé constructor), les fonctions (qui ne sont pas instanciées avec les objets) et une syntaxe cohérente.

Avant (ES5)

```
function User(firstname, lastname) {
  if (!(this instanceof User)) {
    throw new TypeError("Class
constructors cannot be invoked without
'new'");
  }
  this.firstname = firstname;
  this.lastname = lastname;
}

User.prototype.sayName = function() {
  return this.firstname + " " +
this.lastname;
};


// instanciation ----------------
var user = new User("John", "Doe");

// appel de la méthode sayName()
console.log(user.sayName()); // John
Doe
```

Après (ES6)

```
class User {
  // méthode constructor
  constructor(firstname, lastname) {



    this.firstname = firstname;
    this.lastname = lastname;
  }

  sayName() {
    return `${this.firstname} $
{this.lastname}`;
  }
}

// instanciation -----------------
const user = new User("John", "Doe");

// appel de la méthode sayName()
console.log(user.sayName()); // John
Doe
```

3.1 DÉCLARATION ET USAGE D'UNE CLASSE

Voici comment utiliser une classe.

```
class User {
  // constructor
  constructor(firstname, lastname, type) {
    this.firstname = firstname
    this.lastname = lastname
    this.type = type
  }

  // méthode
  sayName() {
   return `${this.firstname}  ${this.lastname}`
  }

  // getter
  get role() {
    return this.type
  }

  // setter
  set role(value) {
    return this.type = value
  }
}

// le `new` est obligatoire pour appeler une classe
const user = new User("John", "Doe", "Contributor")

console.log(user.sayName()) // John Doe
console.log(user.role) // Contributor
user.role = "Owner"
console.log(user.role) // Owner
```

3.2 MÉTHODES ET ATTRIBUTS PRIVÉS

Comme dans de nombreux langage, ECMAScript en version 12 propose l'usage d'attributs et méthodes privés dans une classe. L'attribut ou la méthode doit être précédé du symbole #.

À noter : ES12 permet l'usage d'attributs et de méthodes privées depuis peu de temps, ainsi, l'implémentation complète de la norme est récente : il faut donc redoubler de prudence avec un code utilisant cette possibilité, qui ne serait pas compatible avec d'anciens navigateurs (http://kangax.github.io/compat-table/es6/)

Le code d'une classe avec un attribut et une méthode privée (il s'agit surtout d'une convention comprise de tous, car dans la réalité, JavaScript n'implémente pas réellement la notion de portée privée).

```
class User {
  type
  #type = 42     // type sera inaccessible en accès direct (objet.type)
  getType() {
    return this.#type
  }
  // la fonction setType n'est pas accessible en dehors de la classe
  #setType(type) {
    this.#type = type
  }
}
let Bob = new User()
console.log(Bob.#type)  // génère une erreur
```

3.3 HÉRITAGE DE CLASSE

Comme pour les attributs et méthodes privés, l'héritage est récent (2022), il faut donc éviter ce type de code si le programme doit fonctionner sur d'anciennes versions de navigateurs (version < 100).

héritage

```
class Contributor extends User {
  constructor(firstname, lastname, numberCommit) {
    // le mot clé super est utilisé comme super contructeur. Il permet d'appeler
    // et d'avoir accès aux méthodes du parent
    super(firstname, lastname);
    this.numberCommit = numberCommit;
  }

  sayNameWithCommit() {
    // on peut appeler une méthode de la classe parente avec `super.method`
    return super.sayName() + " " + this.sayNumberCommit();
  }

  sayNumberCommit() {
    return this.numberCommit;
  }
}

// instanciation
const contributor = new Contributor("Jane", "Smith", 10);

// appel de la méthode sayName()
console.log(contributor.sayName());
console.log(contributor.sayNumberCommit());
```

4 UTILISER LES CLASSES AVEC NODE.JS

Ce chapitre aborde la différence entre les classes métiers qui permettent de structurer un code, et les collections d'objets.

Ensuite il présente l'utilisation des classes dans un modèle MVC.

4.1 DIFFÉRENCE DES CLASSES

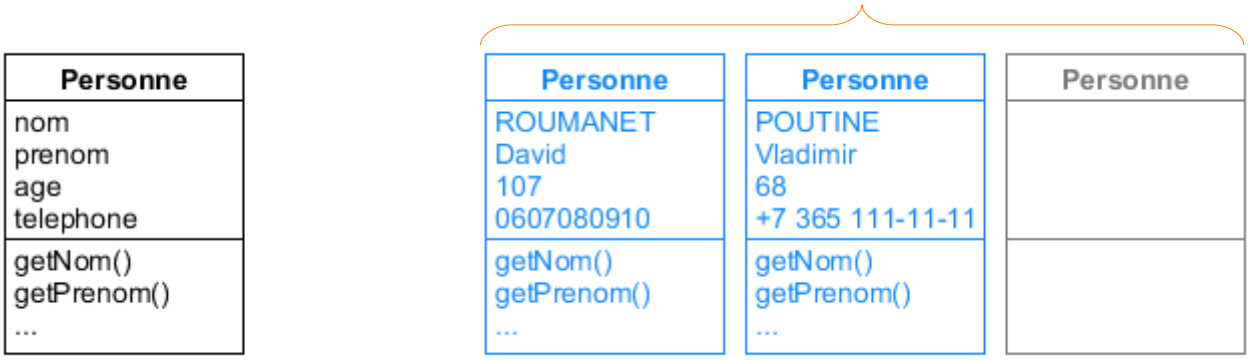
En POO, il faut différencier les classes et les objets, ce qui peut surprendre puisqu'on crée une classe pour instancier des objets.

4.1.1 Classe objet

La collection d'objet, est utilisée dans le traitement des informations. Le chargement des données complexes (par opposition aux données simples comme une variable texte, un entier, etc.) à chaque traitement peut être long. Une gestion des données en mémoire est bien plus rapide.

La collection d'objets permet donc ce genre de traitement, tout en respectant le format original des données.

Par exemple, il est possible de modifier facilement le téléphone de toutes les personnes en ajoutant un entête (par exemple, + international) en fonction de la nationalité.



4.1.2 Classe métier – classe application

Les **classes métiers** concernent les traitements propres aux objets traités pour la réalisation du travail tel que le pratique le client. Les classes métiers manipulent les factures, les produits, les clients…

Les **classes applicatives** portent sur les traitements utiles pour les développeurs pour faire fonctionner l'application. La gestion de l'interface graphique, les accès aux bases de données, etc.

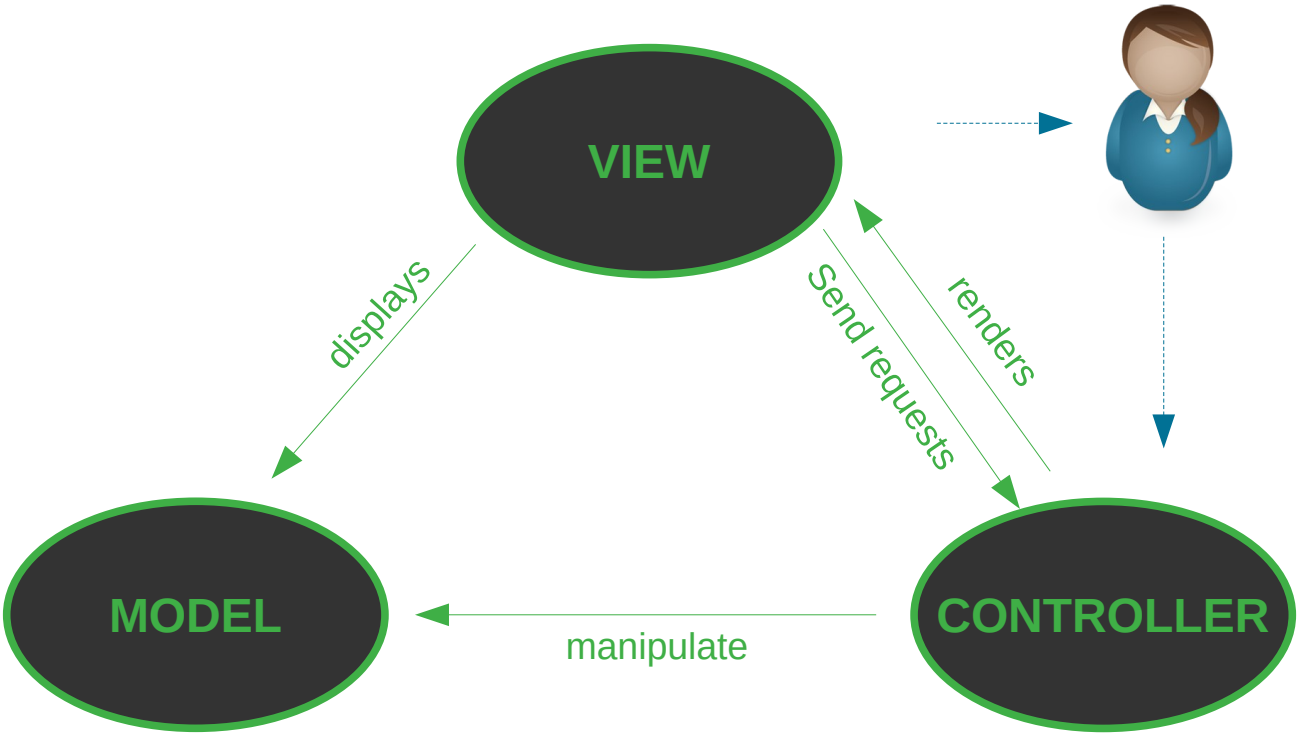
Un développeur peut intégrer tous les éléments dans un seul fichier mais en cas de modification sur la base de données, il devra rechercher tous les accès dans l'ensemble du code.

Il peut aussi créer plusieurs fichiers séparés contenant différentes fonctions mais sur une grosse application, il risque d'y avoir des collisions avec des fonctions dans d'autres fichiers qui porteraient le même nom.

L'utilisation des classes métiers et applicatives, et de séparer les codes : les fonctions peuvent avoir le même nom dans deux classes séparées, car le compilateur (ou l'interpréteur) ne peut appliquer la méthode que sur le type d'objet concerné.

En général on associe donc le modèle de travail MVC aux classes métiers et applicatives :

- MODEL : contient les classes liées aux données (base de données, requêtes, etc.)
- VIEW : contient les classes d'affichage
- CONTROLLER : contient les classes d'interactions avec l'utilisateur



Dans l'exploration, nous verrons comment passer d'un exemple simple (l'application Node.JS livre d'Or) a un code respectant les classes et le MVC.

5 Structure d'un projet Node.JS

Pour un exemple ou une petite application, il n'est pas nécessaire de créer une grosse structure. Cependant, pour un projet réel, il est préférable de placer les parties de codes dans des fichiers séparés.

5.1 Exemple de structure

Voici une structure proposée par RipTutorial (https://riptutorial.com/fr/node-js/example/30554/une-simple-application-nodejs-avec-mvc-et-api)

```
D:.
|    .gitignore
|    app.js
|    package-lock.json
|    package.json
|    Readme.md
|
|---bin
|---node_modules
|---public
|    |---images
|    |        logo.png
|    |
|    |---styles
|             style.css
|
|---routes
|        index.js
|
|---views
         dashboard.ejs
```

5.1.1 public

Le répertoire public contiendra toute la partie statique de l'application : les images, les styles CSS.

5.1.2 Routes

Le répertoire peut ne contenir qu'un seul fichier de route (dans le cas d'une application légère) ou bien plusieurs fichiers de sous-routes (selon la complexité de la chose).

5.1.3 Views

Dans le répertoire 'views', les pages HTML ou bien le framework de votre choix (Angular.Js, react.JS, Vue.JS ou Express.JS). Dans ce cours, nous utiliserons Express.JS.

6 SOURCES

O'Reilly

https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch09s07.html

Blog

https://blog.xebia.fr/2013/06/10/javascript-retour-aux-bases-constructeur-prototype-et-heritage/

MSDN Mozilla

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

Classes ES6, ES6

https://putaindecode.io/articles/es6-es2015-les-classes/

Vidéo Node.js en classes ES6

https://www.youtube.com/watch?v=b4GPyuqBsuA