ECMAScript 6

Découverte

BDX I/O

Bordeaux Developer eXperience

Philippe Charrière | **@k33g_org**

Directeur Technique

SQLI Lyon
ENTERPRISE

plutôt typé front end
mais 💙 donner son avis sur le back

Objectifs & fonctionnement

Développer un framework **MVC** (MV***)

“Skeleton”

avec des fonctionnalité d’**ES6**
en mode exercices
1 exercice = **théorie** + **specs** + **pratique**

Plan

Feuille de route | au moins 2 exos

model & collection
“synchronisation”
view-model
router
démo Polymer
démo Node.js

Copiez les fichiers sur votre poste

ECMAScript **6** ?

ES6

ES6 = la future version de javascript

ECMAScript c’est le nom de la version standardisée

ES6 > fin 2014 > publication mi-2015

ES6 aujourd’hui c’est possible

ES6 aujourd'hui c'est possible

> Projet **Traceur**

https://github.com/google/traceur-compiler

```
sudo npm install traceur -g
```

```
traceur --out out/Greeter.js --script Greeter.js
```

ES6 aujourd'hui c'est possible

```
<html>
  <head>
    <script src="bin/traceur-runtime.js"></script>
    <script src="out/Greeter.js"></script>
  </head>
  <body>
  </body>
</html>
```

ES6 aujourd'hui c'est possible

> Projet **Traceur**

> **6to5**

https://github.com/sebmck/6to5

> **es6-transpiler**

https://github.com/termi/es6-transpiler

> …

Qu’est-ce que cela nous apporte ?

Partie 1
les classes & les modules
& d’autres petites choses…

PS: dans chaque répertoire vous avez le texte des slides

class

```
class Dog {
  /* mot-clé constructor + valeurs par défaut */
  constructor (name="cookie") {
    /* propriétés définies dans le constructeur */
    this.name = name;
  }
  wouaf () { /* pas de mot-clé function */
    console.log(this.name + ": wouaf! wouaf!");
  }
}

let wolf = new Dog();
wolf.wouaf()
```

extends

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

class Dog extends Animal {
  constructor (name="cookie") {
    /* on appelle le constructeur de la classe mère */
    super(name)
  }
  wouaf () {
    console.log(this.name + ": wouaf! wouaf!");
  }
}
```

export - import

```
/* Animal.js */

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

```
/* Dog.js */

import Animal from './Animal';
/* pas d'extension .js */

class Dog extends Animal {
  constructor (name="cookie") {
    super(name)
  }
  wouaf () {
    console.log(this.name + ": wouaf! wouaf!");
  }
}
export default Dog;
```

```
/* main.js */

import Dog from './Dog'
let wolf = new Dog();
wolf.wouaf()
```

=>

```
/* Avant */
var sayHello = function(name) { return "hello " + name; }

/* Après */
var sayHello = (name) => "hello " + name
// ou var sayHello = (name) => { return "hello " + name; }

sayHello("Bob Morane")
```

=> !“newable”, !arguments

REST parameters

```
var sayHello = (...people) =>
    people.forEach(
      (somebody) => console.log("Hello", somebody)
    );

sayHello("Bob Morane", "John Doe", "Jane Doe");
```

=> & lexical this binding

```
/* Avant */

function Animal(friends) {
  this.friends = friends;
  this.hello = function(friend) {
    console.log("hello " + friend);
  }
  this.helloAll = function() {
    this.friends.forEach(function(friend) {
      this.hello(friend); /* error */
    });
  }
}

var wolf = new Animal(["rox", "rookie"]);
wolf.helloAll();
```

=> & lexical this binding

```
/* Avant */

function Animal(friends) {
  this.friends = friends;
  this.hello = function(friend) {
    console.log("hello " + friend);
  }
  this.helloAll = function() {
    this.friends.forEach(function(friend) {
      this.hello(friend);
    }.bind(this)); /* ou var that = this */
  }
}

var wolf = new Animal(["rox", "rookie"]);
wolf.helloAll();
```

=> & lexical this binding

```
/* Après */

class Animal {
  constructor (friends=[]) {
    this.friends = friends;
  }

  hello(friend) { console.log("hello " + friend); }

  helloAll() {
    this.friends.forEach((friend) => this.hello(friend));
  }
}
```

La valeur de this est déterminée par l'endroit où se trouve la "Arrow function"

let versus var

```
let bob = {
  firstName:"Bob", lastName:"Morane"
}

let bob = { foo:"foo" }
/* Duplicate declaration, bob */
```

ES5 > Getter & Setter

```
let bob = {}

Object.defineProperty(bob, "Name", {
  get: function (){
    console.log("get value:", this.name) /* ! nous avons Name et name */
    return this.name;
  },
  set: function (value) {
    console.log("set value to:", value)
    this.name=value;
  }
});

bob.Name = "BOB MORANE";
console.log(bob.Name);
```

“Transpilation” … “à la volée”

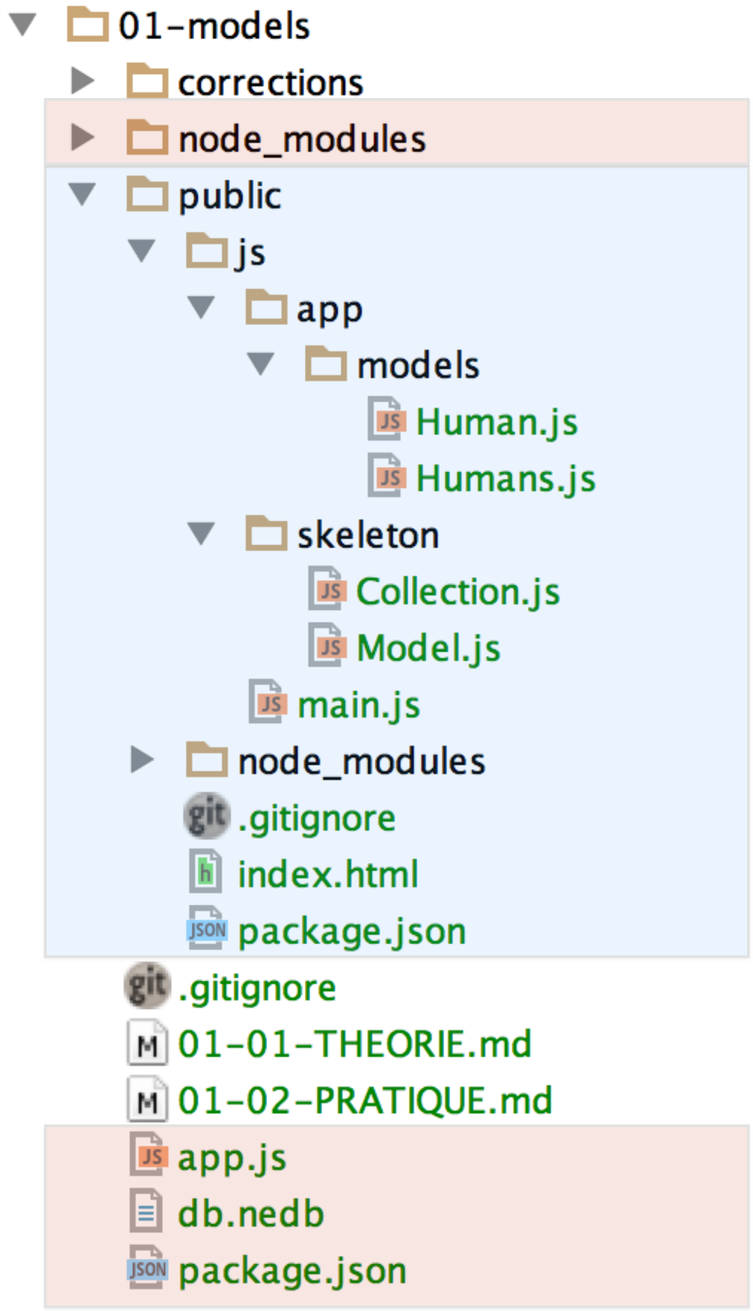
```
<script src="node_modules/traceur/bin/traceur.js"></script>

<script>
  traceur.options.experimental = true;
</script>

<script>
  System.import('js/main').catch(function (e) {console.error(e);});
</script>
```

Partie 1
EXERCICE
modèle & collection

PS: les “spécifications” sont dans les fichiers à compléter



❸ *côté client*

```
{
  "name": "es6",
  "devDependencies": {
    "qunitjs": "^1.15.0"
  },
  "dependencies": {
    "traceur": "0.0.66"
  }
}
```

❷ *app.js*

```
app.get("/about", function(req, res) {
  res.send({message:"Hello World!"});
});

// Liste de tous les humans
app.get("/humans", function(req, res) {
  db.find({}, function (err, docs) {
    res.send(docs);
  });

});

// …
```

❶ *côté serveur*

```
{
  "name": "es6",
  "dependencies": {
    "body-parser": "1.0.2",
    "express": "4.1.x",
    "nedb": "0.10.5"
  }
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>
<!-- ... -->
</head>

<body style="padding: 20px">

<h1>ECMAScript 6 | Skeleton MVC</h1>

<div id="qunit"></div>
<div id="qunit-fixture"></div>

<script src="node_modules/qunitjs/qunit/qunit.js"></script>
<script src="node_modules/traceur/bin/traceur.js"></script>

<script>
  traceur.options.experimental = true;
</script>

<script>
  System.import('js/main').catch(function (e) {console.error(e);});
</script>

</body>
</html>
```

main.js

```
import Human from 'js/app/models/Human';
import Humans from 'js/app/models/Humans';

QUnit.test( "john is John Doe", ( assert ) => {

  let john = new Human();
  assert.ok( john.firstName == "John", "john.firstName is John");
  assert.ok( john.lastName == "Doe", "john.lastName is Doe");

  john.lastName = "DOE";
  assert.ok( john.fields.lastName == john.lastName && john.lastName == "DOE",
"john.lastName is DOE");

});

QUnit.test( "bob is Bob Morane", ( assert ) => {

  let bob = new Human({firstName:"Bob", lastName:"Morane"});

  assert.ok(bob.firstName == "Bob", "bob.firstName is Bob");
  assert.ok(bob.lastName == "Morane", "bob.lastName is Morane");
  assert.ok(bob.toString() == '{"firstName":"Bob","lastName":"Morane"}', 'bob.toString()
returns {"firstName":"Bob","lastName":"Morane"}');

});
```

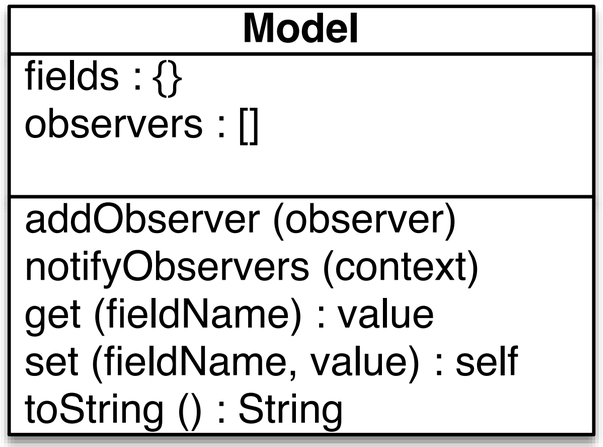
énoncé

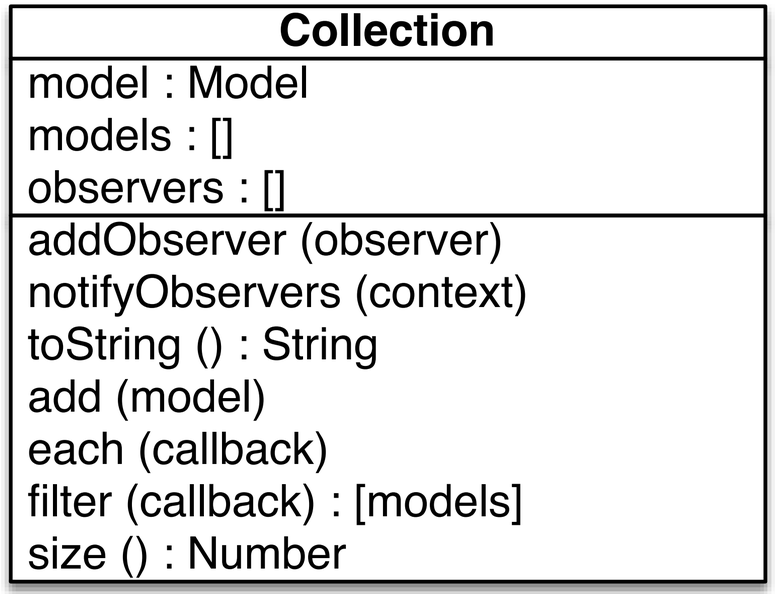
Créer nos 1ers modèles & collections :

- `public/js/skeleton/Model.js`
- `public/js/skeleton/Collection.js`
- `public/js/app/models/Human.js`
- `public/js/app/models/Humans.js`

Remarque: les spécifications sont décrites dans les fichiers

```
cd 01-models
node app.js // http://localhost:3000
```


Model
fields : {}
observers : []
addObserver (observer)
notifyObservers (context)
get (fieldName) : value
set (fieldName, value) : self
toString () : String


Collection
model : Model
models : []
observers : []
addObserver (observer)
notifyObservers (context)
toString () : String
add (model)
each (callback)
filter (callback) : [models]
size () : Number

c’est parti !

Partie 2

Promises & Strings

Promises

```
let doSomeThing = new Promise((resolve, reject) => {

  // faites quelque chose (d'asynchrone par ex.)

  let allisfine = true; // essayez avec false

  if (allisfine) {
    resolve("Hello World!");
  }
  else {
    reject(Error("Ouch"));
  }
});

doSomeThing
  .then((data) => { console.log(data); })
  .catch((err) => { console.log(err); });
```

http://www.html5rocks.com/en/tutorials/es6/promises

Template strings (``backtick)

```
let firstName = "Bob", lastName = “Morane";

console.log(`Hello I'm ${firstName} ${lastName}`);

// Hello I'm Bob Morane
```

Multiline strings

```
let firstName = "Bob", lastName = "Morane";

console.log(`
  Hello I'm
  ${firstName}
  ${lastName}
`
);

/*
  Hello I'm
  Bob
  Morane
*/
```

Partie 2
EXERCICE
appels au serveur

PS: les “spécifications” sont dans les fichiers à compléter

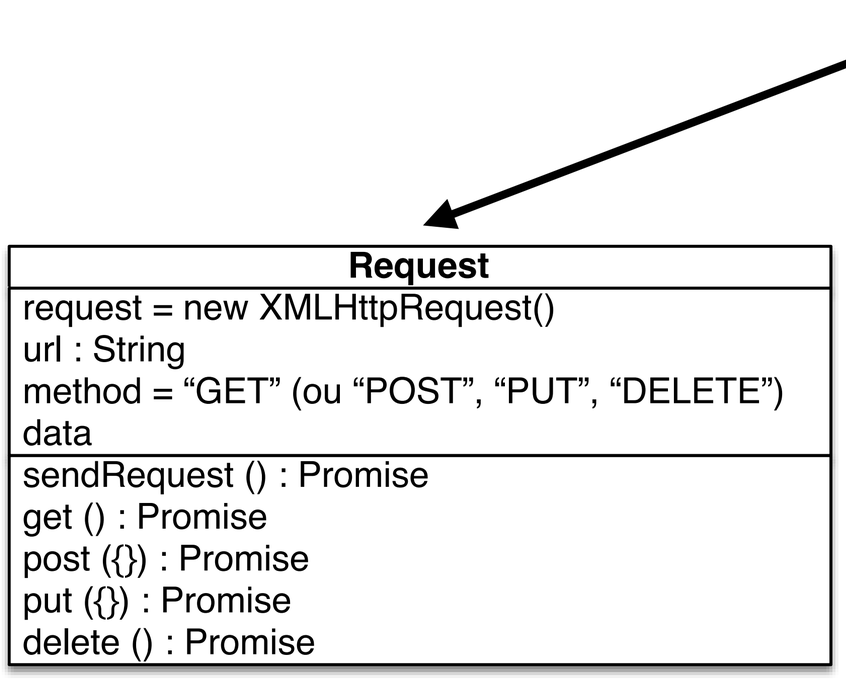
énoncé

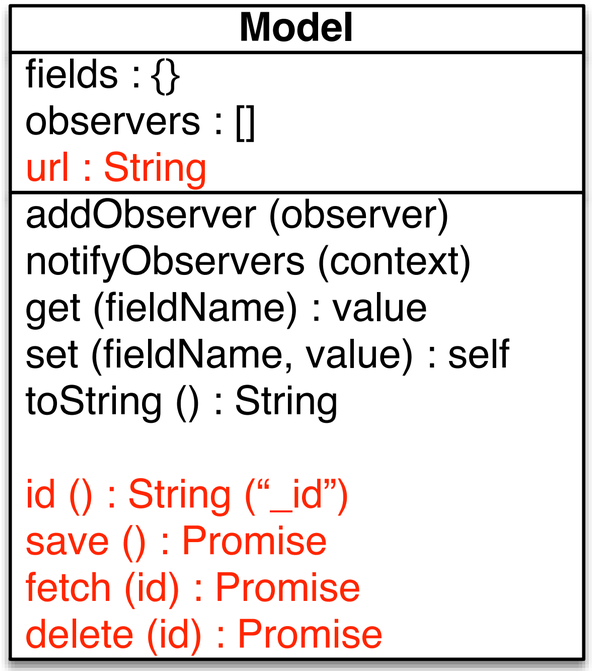
Donner la possibilité aux modèles et aux collections de “discuter” avec le serveur :

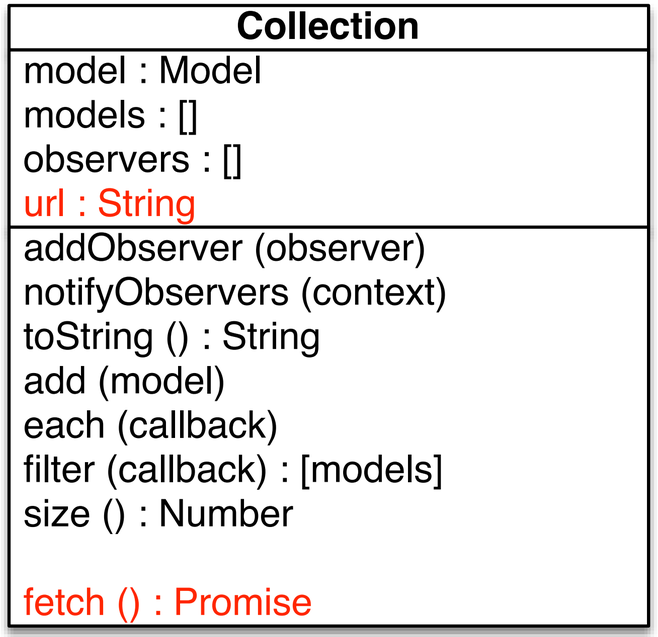
- compléter Request.js (requêtes ajax)
- compléter Model.js (utilisera Request)
- compléter Collection.js (utilisera Request)
- Modifier Human.js (déjà fait, à relire)
- Modifier Humans.js (déjà fait, à relire)

Remarque: les spécifications sont décrites dans les fichiers

```
Stoppez exercice 1
cd ..
cd 02-models-sync
node app.js // http://localhost:3000
```








```
// save - create
this.notifyObservers({
  event: "create", model: this
});

// save - update
this.notifyObservers({
  event: "update", model: this
});

// fetch(id)
this.notifyObservers({
  event: "fetch", model: this
});

// delete(id)
this.notifyObservers({
  event: "delete", model: this
});
```

```
// fetch
this.notifyObservers({
  event: "fetch",
  models:models
});
```

c’est parti !

Partie 3
Mixin & Array.from

Object.assign

```
let tonyStark = {
  firstName:"Tony", lastName:"Stark"
};

let armorAbilities = {
  fly:() => console.log("I'm flying")
};

Object.assign(tonyStark, armorAbilities);

tonyStark.fly(); // I'm flying
```

Array.from

```
/* Avant */
var items = [].slice.apply(document.querySelectorAll("h1"));

// var items = Array.prototype.slice.apply(document.querySelectorAll("h1"));

items.forEach(function(item) { item.innerHTML = "Hello"; });

/* Maintenant */

Array.from(document.querySelectorAll(“h1"))
  .forEach((item) => item.innerHTML = "Hello")
```

Partie 3
EXERCICE
afficher les données

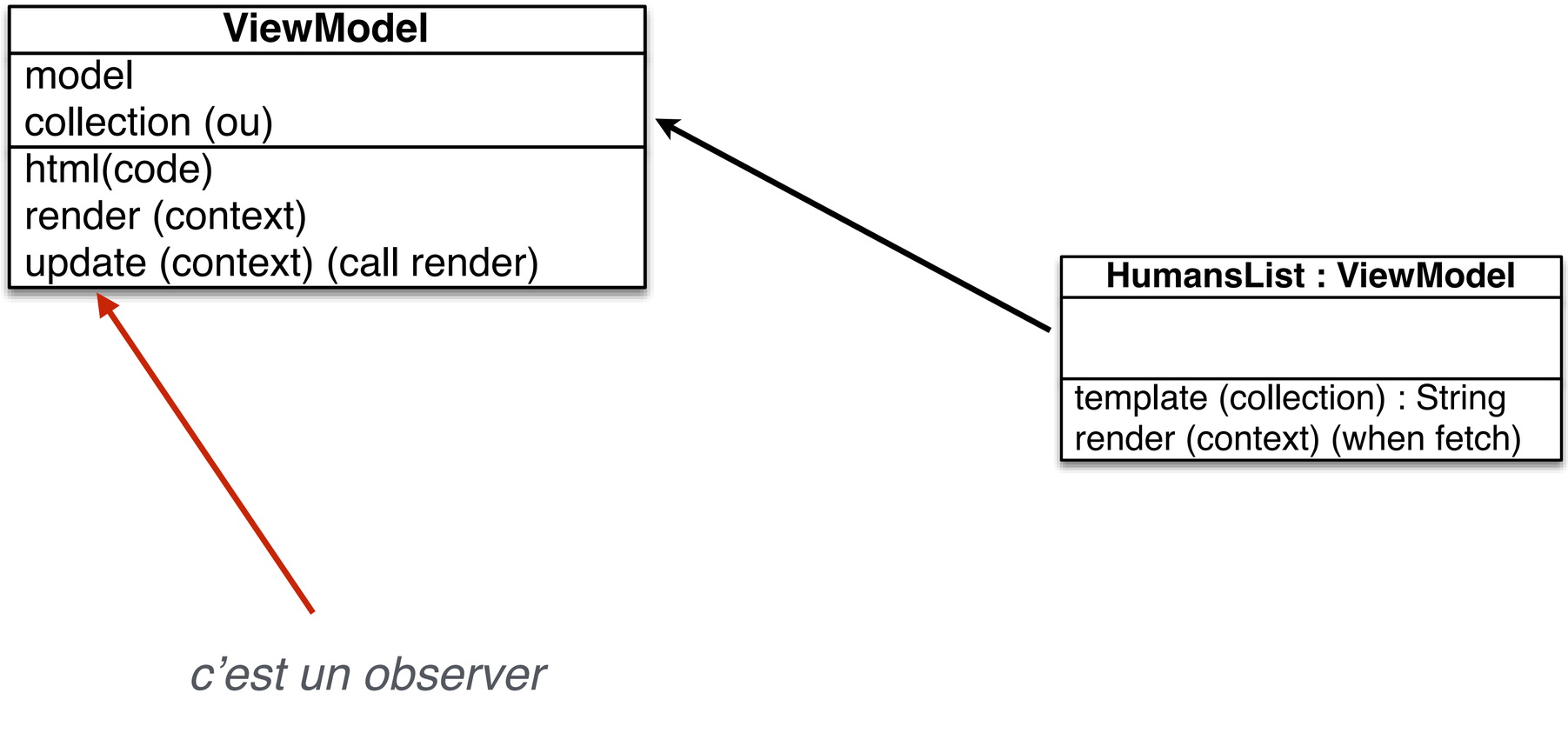
PS: les “spécifications” sont dans les fichiers à compléter

énoncé

- créer un mini jQuery (mini mini) que l'on utilisera de cette façon: `$q(selector)` : complétez js/skeleton/selector.js
- créer une classe `ViewModel` : complétez js/skeleton/ViewModel.js
- créer une classe `HumansList` héritant de `ViewModel` déstinée à afficher une liste de modèles `Humans`, complétez js/app/viewModels/HumansList.js

Remarque: les spécifications sont décrites dans les fichiers

```
Stoppez exercice 2
cd ..
cd 03-views-models
node app.js // http://localhost:3000
```


ViewModel
model
collection (ou)
html(code)
render (context)
update (context) (call render)
HumansList : ViewModel
template (collection) : String
render (context) (when fetch)
c’est un observer

c’est parti !

Partie 4
Maps

Map

```
let map = new Map();

map.set("one",{firstName:"John", lastName:"Doe"});
map.set("two",{firstName:"Jane", lastName:"Doe"});

console.log(map.has("one"));  // true
console.log(map.get("one"));  // Object {firstName: "John", lastName: "Doe"}
console.log(map.size);        // 2
```

Parcourir une Map

```
for (let key of map.keys()) {
  console.log("Key: %s", key);
}
/* Key: one, Key: two */

for (let value of map.values()) {
  console.log("Value: %s %s", value.firstName, value.lastName);
}
/* Value: John Doe, Value: Jane Doe */

for (let item of map) {
  console.log("Key: %s, Value: %s", item[0], item[1].firstName, item[1].lastName);
}
/* Key: one, Value: John Doe, Key: two, Value: Jane Doe */
```

… et aussi

```
let myOtherMap = new Map([
  ["one",{firstName:"John", lastName:"Doe"}],
  ["two",{firstName:"Jane", lastName:"Doe"}],
  ["three",{firstName:"Bob", lastName:"Morane"}]
]);

myOtherMap.delete("three")

myOtherMap.forEach((item)=>{
  console.log(item)
})
/*
 Object {firstName: "John", lastName: "Doe"}
 Object {firstName: "Jane", lastName: "Doe"}
*/
```

Le saviez-vous ?

Vos classes peuvent hériter des types javascript

Partie 4
EXERCICE
Router (rudimentaire)

PS: les “spécifications” sont dans les fichiers à compléter

énoncé

Créer un routeur qui "écoutera" le navigateur :

- click sur un lien (met à jour window.location.hash)
- saisie d'url (met à jour window.location.hash)
- bouton back (met à jour window.location.hash)

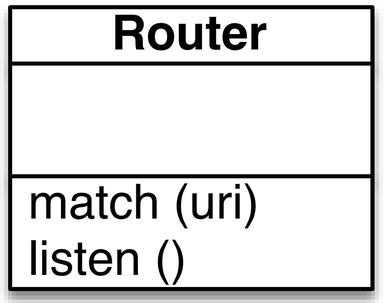
Ce routeur contiendra des routes (couple {url, traitement}),
si l'url d'une route = window.location.hash,
alors le traitement correspondant est déclenché

Remarque: les spécifications sont décrites dans les fichiers

```
Stoppez l'exercice 3
cd ..
cd 04-router
node app.js // http://localhost:3000
```

https://www.google.fr/#/humans/1234

```
window.location.hash = “#/humans/1234”
```




```
let router = new Router();

router.set("humans", (args) => {
  // …
});
```

c’est parti !

Démo
Utilisation avec Polymer

Démo
Utilisation avec Node.js

Faut-il utiliser ES6 aujourd’hui ? & pourquoi ?

Angular 2 avec Traceur

Ember (next) : les modules avec ES6 Module Transpiler

Backbone est déjà ES6 compliant : https://github.com/addyosmani/todomvc-backbone-es6/blob/gh-pages/js/todo-app.js

Qui suivre ?

Addy Osmani (@addyosmani)

https://github.com/addyosmani/es6-tools

Axel Rauschmayer (@rauschma)

http://www.2ality.com/search/label/esnext

Nicholas Zakas (@slicknet)

https://github.com/nzakas/understandinges6

Merci à vous + ?