

Objets ES5 et ES6 (classes, héritage)

Philippe Genoud
Philippe.Genoud@univ-grenoble-alpes.fr



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Objets et ES5/ES6

- La création d'objets avec une fonction constructeur et l'opérateur `new` est historique, elle masque le mécanisme de prototypage souvent mal compris par les programmeurs.
- ES5 puis ES6 proposent de nouvelles manières de définir des objets (même si en interne le principe du prototypage est inchangé) afin de rendre l'utilisation des objets plus accessible aux développeurs:
  - `Object.create()` (ES5)
  - Classes (ES6)

Objets et ES5 : `Object`

- Depuis la version ES5 de JavaScript, l'objet prédéfini (*built-in*) `Object` propose un certain nombre de méthodes pour créer et manipuler les prototypes :

  - `Object.create(proto:Object) : Object`
    crée un nouvel objet ayant pour <prototype> l'objet `proto` passé en paramètre

  - `Object.getPrototypeOf(obj : Object) : Object`
    renvoie l'objet <prototype> de l'objet `obj` passé en paramètre

  - `Object.setPrototypeOf(obj: Object, proto: Object)`
    fixe le <prototype> de `obj` avec l'objet `proto`

  - `...` (voir la doc MDN)

Objets et ES6 : Classes

- Avec l'introduction de **`Class`**, ES6 offre une syntaxe compacte pour définir des chaînes de prototypes et qui se rapproche de l'approche plus couramment utilisée dans les langages orientés objets (langages de classes comme Java)

avant ES6

```
function Chat(nom,age,race,poids) {
   this.nom = nom ;
   this.age = age ;
   this.race = race ;
   this.poids = poids;
}

Chat.prototype.miauler = function () {
    console.log(this.nom + "-> Miaou ! Miaou !");
};
Chat.prototype.maigrir = function(deltaPoids) {
     this.poids -= deltaPoids;
};
```

constructeur

⇔

méthodes

ES6 +

```
class Chat {
   constructor(nom,age,race,poids) {
      this.nom = nom ;
      this.age = age ;
      this.race = race ;
      this.poids = poids;
   }
   miauler() {
        console.log(this.nom + "-> Miaou ! Miaou !");
   }
   maigrir(deltaPoids) {
       this.poids -= deltaPoids;
   }
}
```

mot réservé pour identifier la fonction constructeur

Une classe ne peut avoir qu'un seul constructeur

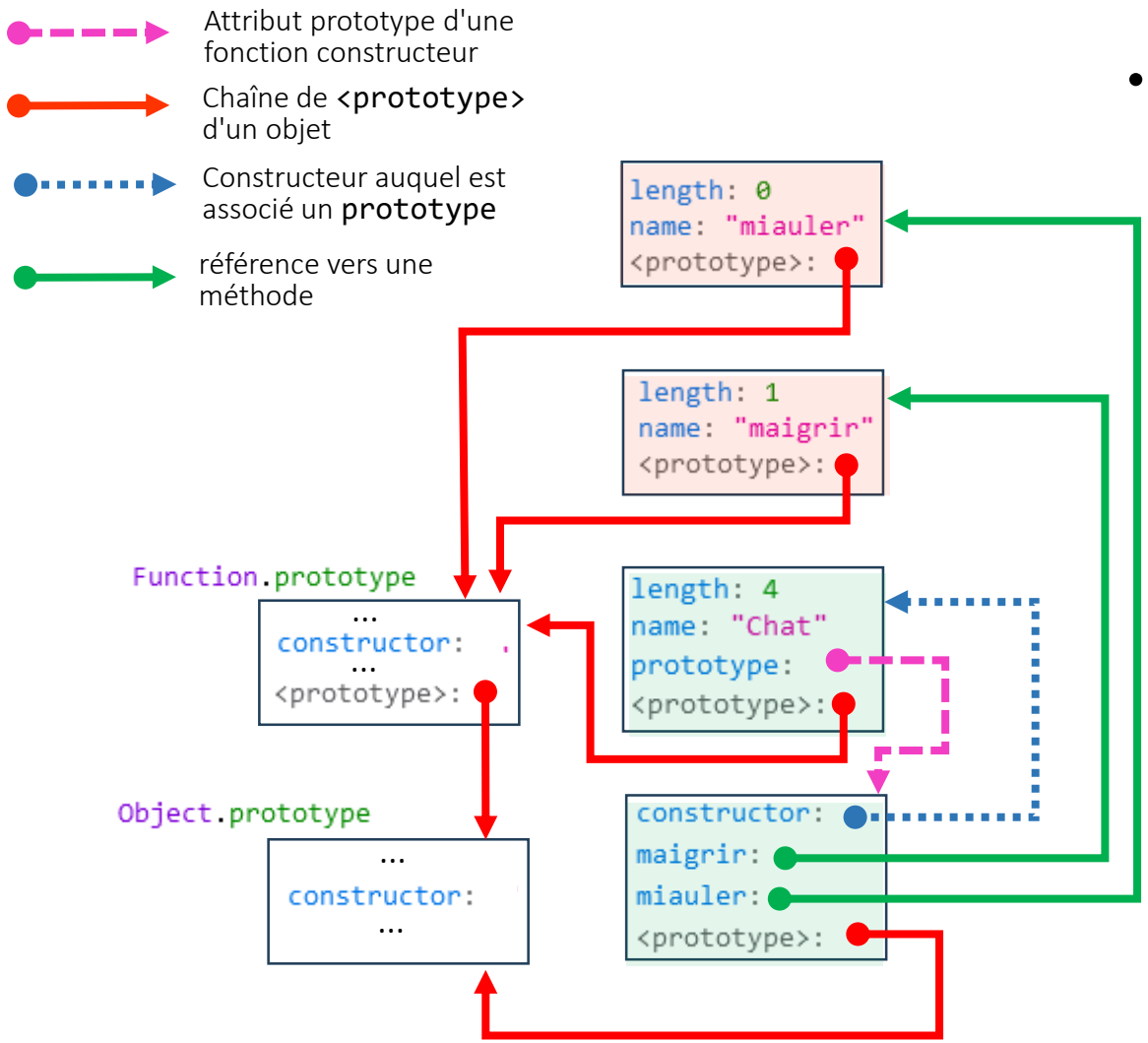


pas de **;** entre les déclarations de méthodes



Voir le code

Objets et ES6 : Classes

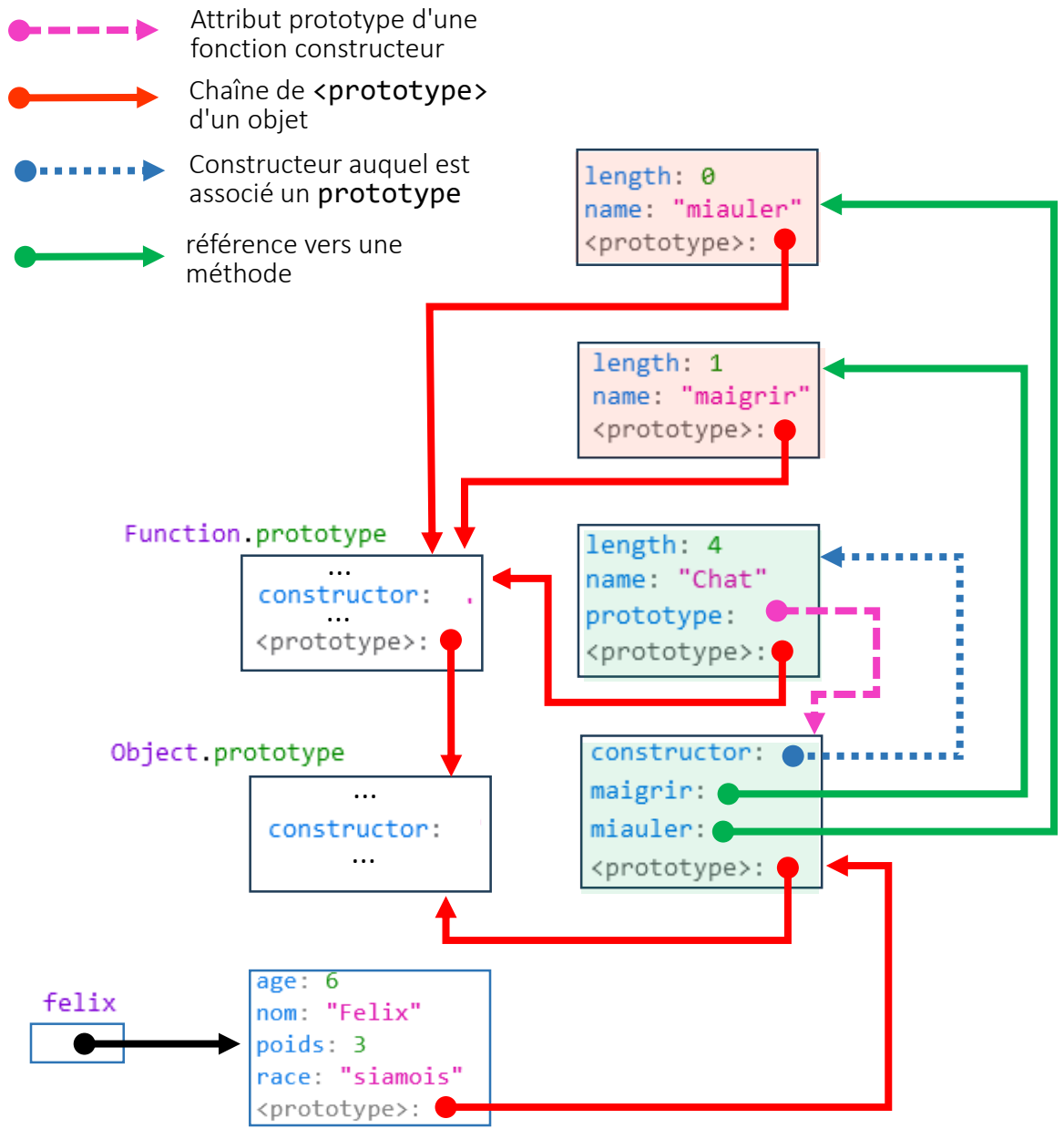


- Lorsqu'une classe est définie, elle est en réalité transformée en une fonction constructeur sous le capot.
- Les méthodes définies dans la classe sont ajoutées au prototype de cette fonction constructeur.

```
class Chat {
    constructor(nom,age,race,poids) {
        this.nom = nom ;
        this.age = age ;
        this.race = race ;
        this.poids = poids;
    }
    miauler() {
        console.log(this.nom + "-> Miaou ! Miaou !");
    }
    maigrir(deltaPoids) {
        this.poids -= deltaPoids;
    }
}
```

Voir le code

Objets et ES6 : Classes



- La création des objets se fait comme précédemment avec l'opérateur **new**

allocation mémoire — exécution de la fonction constructeur

```
let felix = new Chat("Felix",6,"siamois",3);
```

felix

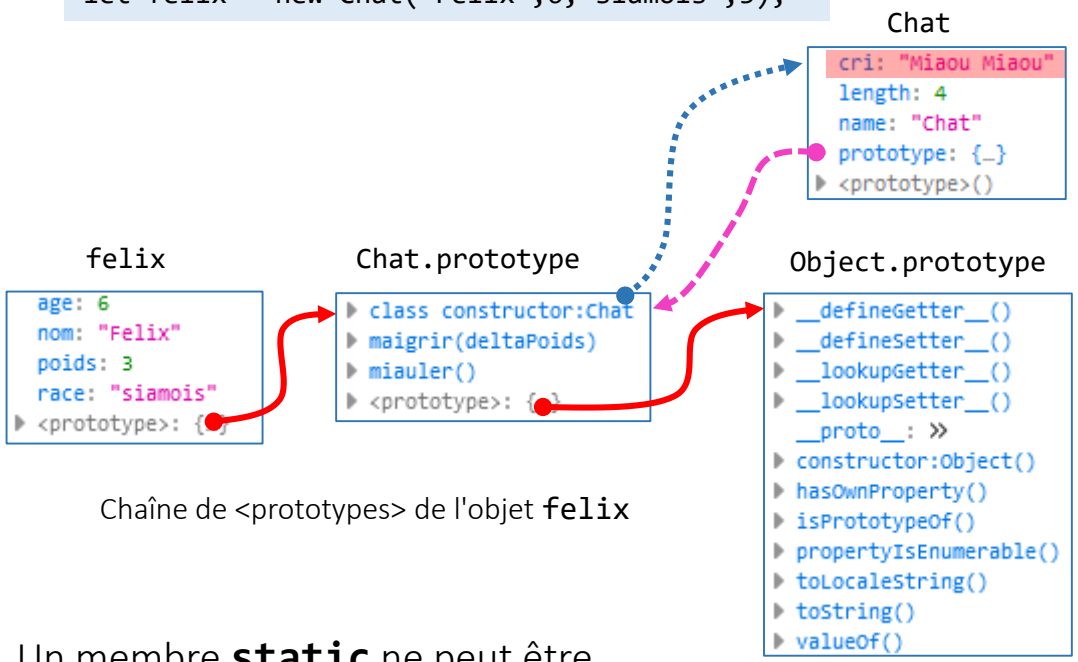
```
class Chat {
    constructor(nom,age,race,poids) {
        this.nom = nom ;
        this.age = age ;
        this.race = race ;
        this.poids = poids;
    }
    miauler() {
        console.log(this.nom + "-> Miaou ! Miaou !");
    }
    maigrir(deltaPoids) {
        this.poids -= deltaPoids;
    }
}
```

Voir le code

Objets et ES6 : Classes – membres statiques

- Propriétés **statiques**
  - Possibilité d'associer des **propriétés** à la classe directement
  - Déclaration de variable préfixée par le mot clé **`static`**
  - Rattaché à la fonction Constructeur et non pas au `prototype` de celle-ci

```
let felix = new Chat("Felix",6,"siamois",3);
```



Chaîne de <prototypes> de l'objet **`felix`**

```
class Chat {
  constructor(nom,age,race,poids) {
     this.nom = nom ;
     this.age = age ;
     this.race = race ;
     this.poids = poids;
  }
  static cri = "Miaou ! Miaou !";
   miauler() {
       console.log(this.nom + "-> " + Chat.cri);
   }
   maigrir(deltaPoids) {
      this.poids -= deltaPoids;
   }
}
```

Voir le code

- Un membre **`static`** ne peut être accédé via une instance de la classe (pas dans la chaîne de <prototypes>)

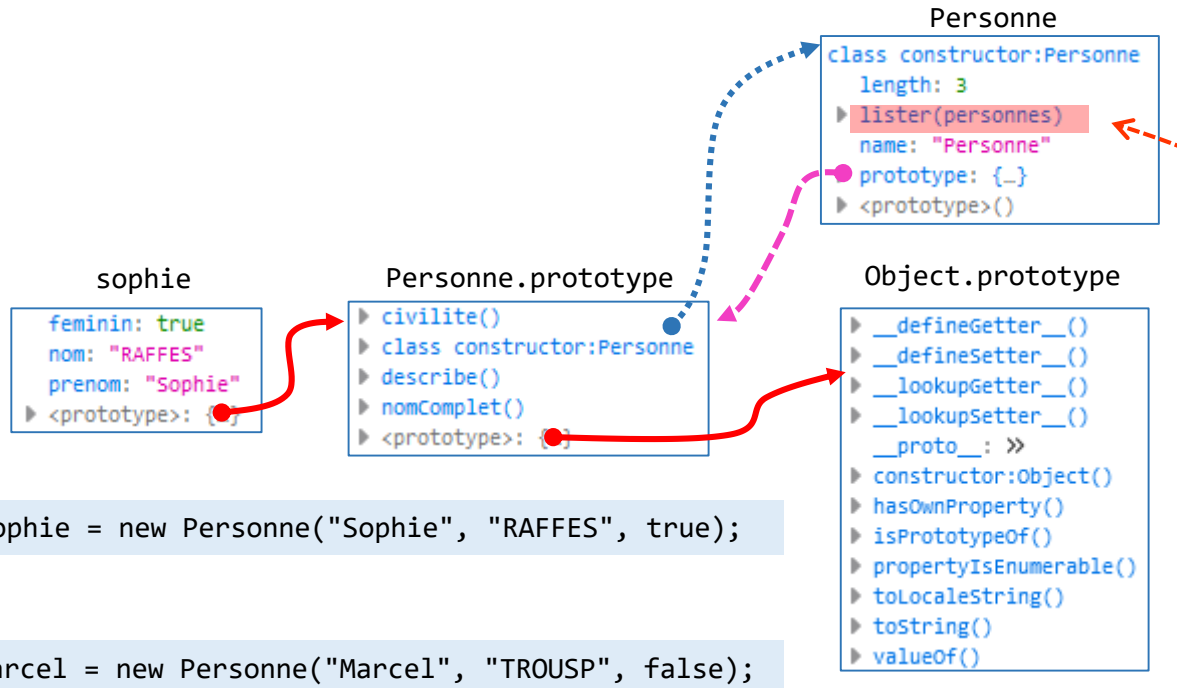
`felix.cri → undefined`

- Un membre **`static`** est accédé via l'identifiant de la classe

`Chat.cri → "Miaou ! Miaou !`

Objets et ES6 : Classes – membres statiques

- Méthodes **statiques**
  - déclaration de méthode préfixée par le mot clé **`static`**
  - fonction rattaché à la fonction constructeur et non pas au `prototype`




```
let sophie = new Personne("Sophie", "RAFFES", true);
```

```
let marcel = new Personne("Marcel", "TROUSP", false);

Personne.lister([sophie, marcel]);
```

→

```
Personne
--------
Mme RAFFES Sophie
M. TROUSP Marcel
```

Pour invoquer la méthode statique il faut passer par l'objet classe (la fonction constructeur)

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin; // true si une femme, false sinon
    }

    nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite()} ${this.nomComplet()}`;
    }

    /**
     * affiche la description des personnes contenues dans un tableau
     * @param {Array[Personne]} personnes le tableau des personnes
     */
    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

Méthode statique se situe non pas au niveau du prototype pour les Personnes mais au niveau de la fonction constructeur **`Personne`**

Voir le code

Objets et ES6 : Classes - *getters* et *setters*

- Accesseurs : fonctions *getters*
  - **get** permet de lier une propriété d'un objet à une fonction qui sera appelée lorsqu'on accédera à la propriété.

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin; // true si une femme, false sinon
    }

    nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite()} ${this.nomComplet()}`;
    }

    /**
     * affiche la description des personnes contenues dans un tableau
     * @param {Array[Personne]} personnes le tableau des personnes
     */
    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

```
let sophie = new Personne("Sophie", "RAFFES", true);
console.log(sophie.civilite()); → 'Mme'
console.log(sophie.nomComplet()); → 'RAFFES Sophie'
```

Méthodes transformées en getters

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    /**
     * affiche la description des personnes contenues dans un tableau
     * @param {Array[Personne]} personnes le tableau des personnes
     */
    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

```
console.log(sophie.civilite);   → 'Mme'
console.log(sophie.nomComplet); → 'RAFFES Sophie'
```

Les getters peuvent être utilisés comme des propriétés

Voir le code

Objets et ES6 : Classes - *getters* et *setters*

- Mutateurs (fonctions *setters*)
  - **set** permet de lier une propriété d'un objet à une fonction qui sera appelée lorsqu'on affectera la propriété.

```
let p1 = new Personne("Sophie", "RAFFES", true);

console.log(p1.nomComplet);     → 'RAFFES Sophie'

p1.nomComplet = "  Maeva   MERLIN";

console.log(p1.prenom);         → 'Maeva'

console.log(p1.nom);            → 'MERLIN'
```

Appel du *setter*

On peut aussi définir getters et setters dans des objets littéraux

```
let date =  {
    jour : '17',
    mois : 'Novembre',
    annee : '2020',
    get dateComplete() {
        return `${this.jour} ${this.mois} ${this.annee}`;
    },
    set dateComplete(dateString) {
        const tokens = dateString.split(/\b\s+(?!$)/);
        this.jour = parseInt(tokens[0]);
        this.mois = tokens[1];
        this.annee = parseInt(tokens[2]);
    }
};
```

Voir le code

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);  // expression régulière*
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

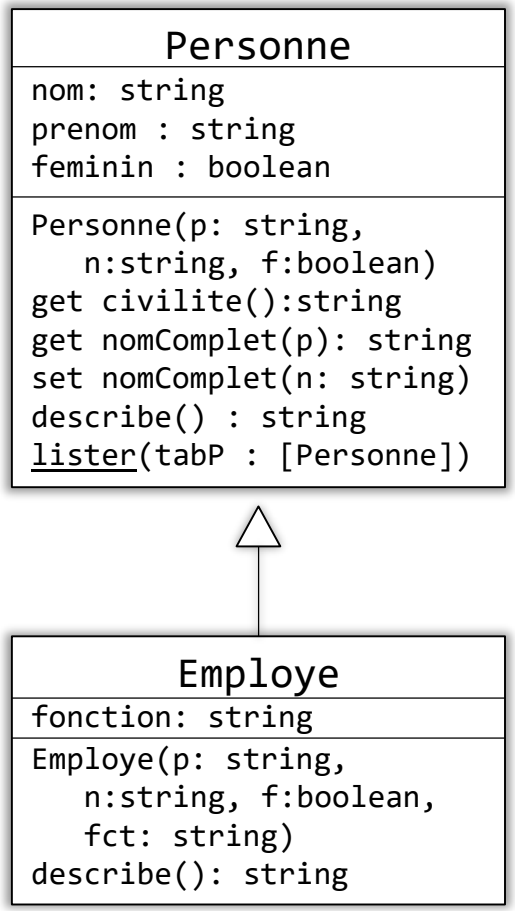
    /**
     * affiche la description des personnes contenues dans un tableau
     * @param {Array[Personne]} personnes le tableau des personnes
     */
    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

Voir le code

*https://blog.abelotech.com/posts/split-string-into-tokens-javascript/

Objets et ES6 : Classes - Héritage

- Héritage – possibilité de définir une classe comme étendant une classe existante (sous classe)



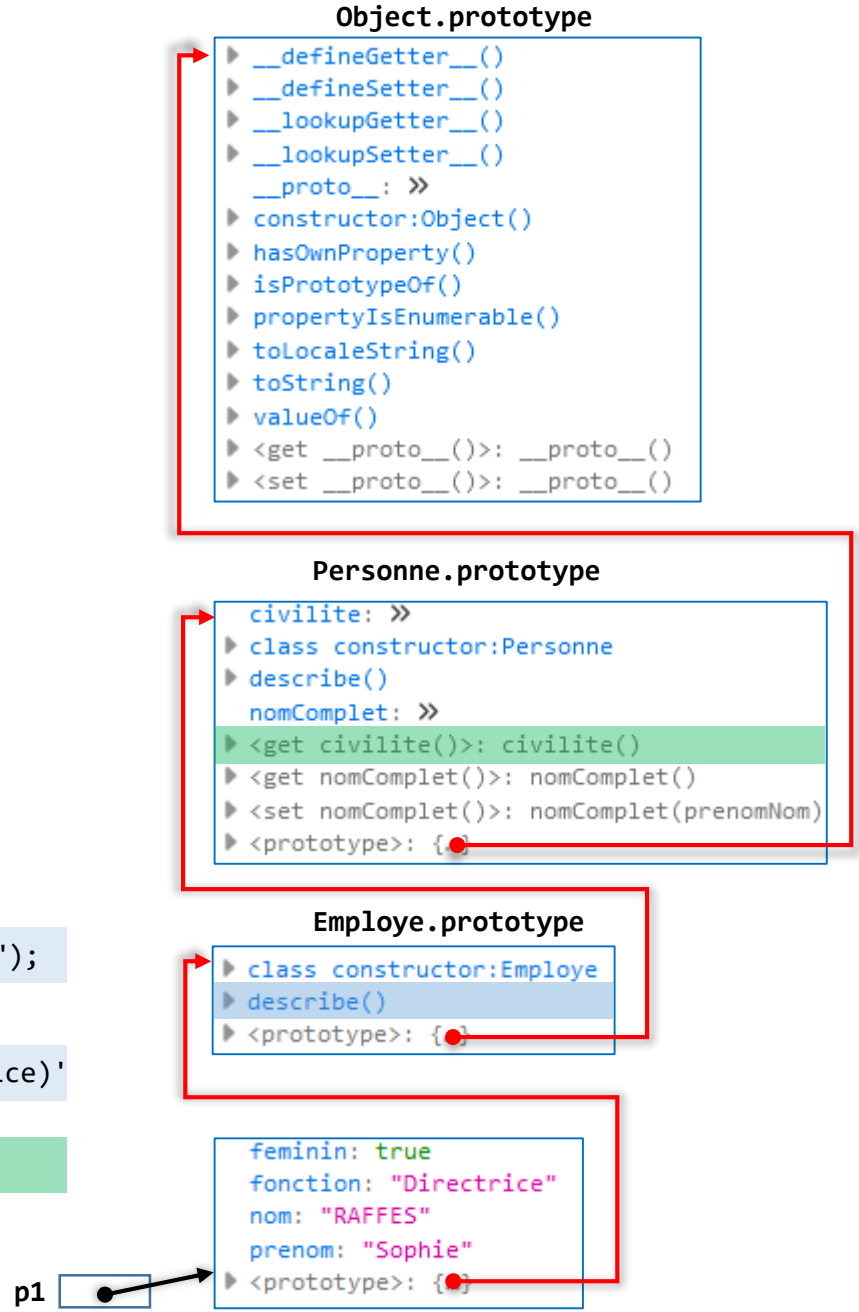
On veut exprimer le fait qu'un employé c'est une personne mais qui en plus a un attribut `fonction`.

Possibilité d'obtenir ce comportement avec les chaînes de prototypes en JavaScript

```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
console.log(p1.describe()); → 'Mme RAFFES Sophie (Directrice)'
```

```
console.log(p1.civilite);   → 'Mme'
```



Objets et ES6 : Classes - Héritage

- Héritage – **`extends`** permet d'exprimer cette relation d'héritage (et de mettre en place la chaîne de prototypes)

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++)
            console.log(personnes[i].describe());
        }
    }
}
```

`super()` appelle le constructeur de la super classe.
Cet appel doit être fait avant d'accéder à `this`

```
class Employe extends Personne {

    constructor(prenom, nom, feminin, fonction) {
        super(prenom, nom, feminin);
        this.fonction = fonction;
    }

    describe() {
        return super.describe() + ` (${this.fonction})`;
    }
}
```

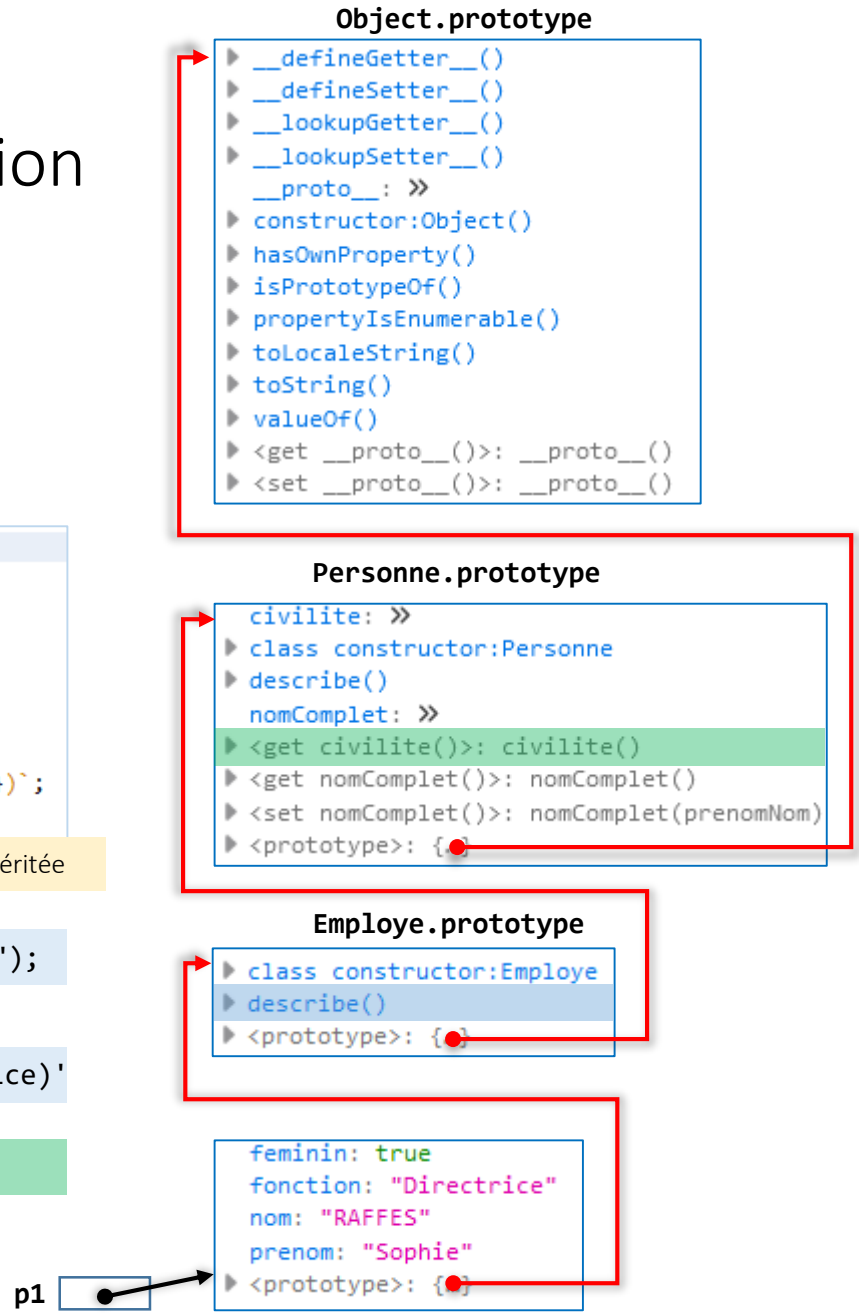
`super.nomMethode()` invoque une méthode héritée

```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
console.log(p1.describe()); → 'Mme RAFFES Sophie (Directrice)'
```

```
console.log(p1.civilite);   → 'Mme'
```

Voir le code



Objets et ES6 : Classes - Héritage

- Héritage – **extends** défini aussi une chaîne de <prototype> au niveau des classes (fonctions constructeurs) → héritage des membres statiques

```
class Personne {
    // ...
    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}

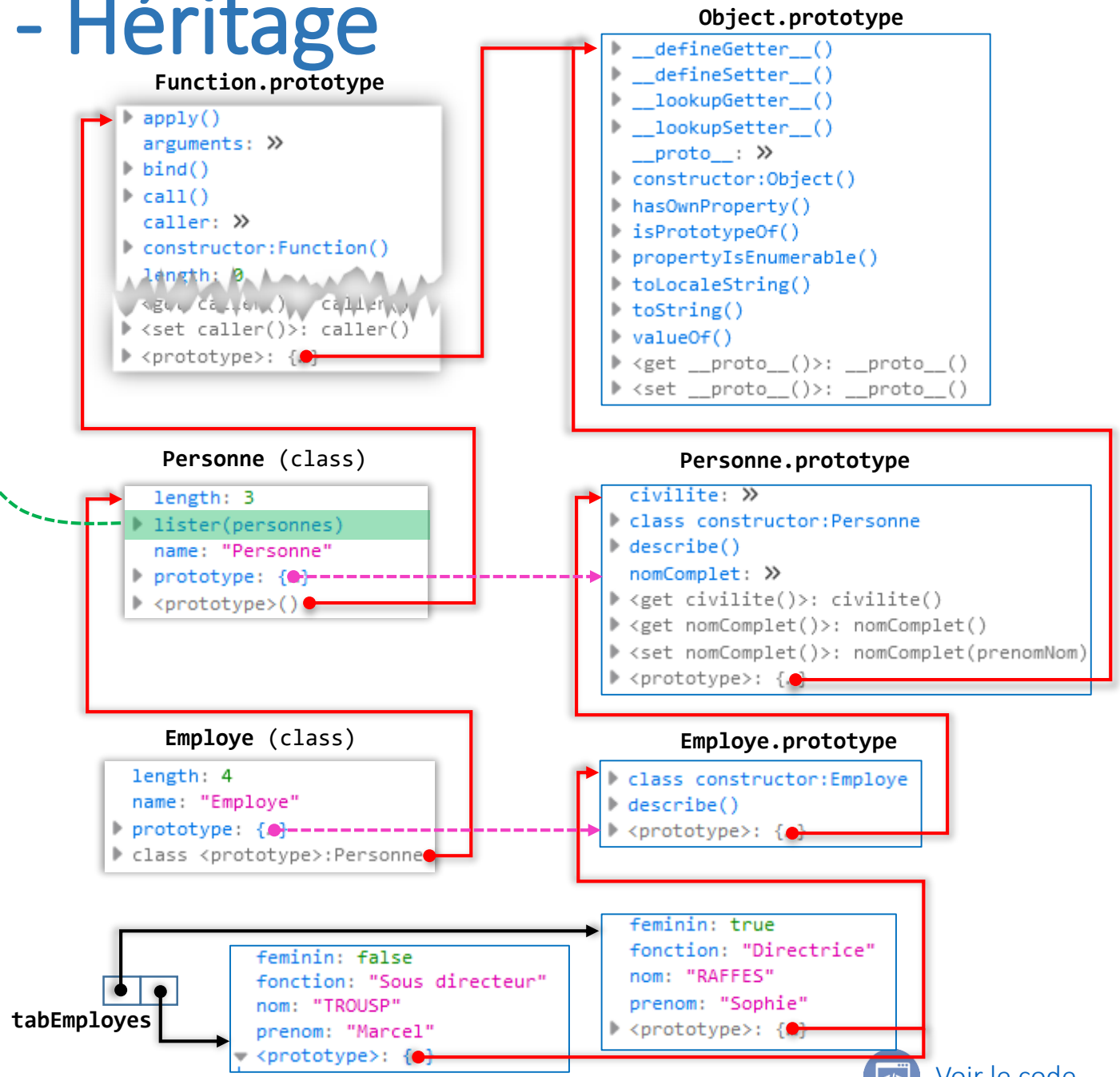
class Employe extends Personne {

    // ...
}
```

```
let tabEmployes = [
  new Employe("Sophie", "RAFFES", true, "Directrice") ,
  new Employe("Marcel", "TROUSP", false, "Sous directeur")
];
```

```
Personne.lister(tabEmployes); → Personne
                                --------
                                Mme RAFFES Sophie (Directrice)
                                M. TROUSP Marcel (Sous directeur)

Employe.lister(tabEmployes);  → Employe
                                --------
                                Mme RAFFES Sophie (Directrice)
                                M. TROUSP Marcel (Sous directeur)
```



Voir le code

© Philippe GENOUD - UGA - Cours Technos Web - M2 CCI - M2 GEOMAS - 2026/2027

Objets : opérateur `instanceof`

- `x instanceof C` renvoie `true` si l'objet référencé par `x` est instance de la classe `C` ou d'une sous classe de `C` (plus précisément si `C.prototype` se trouve dans la chaîne de `<prototypes>` de `x`), `false` sinon

```
let p1 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
p1 instanceof Employe     → true
```

```
p1 instanceof Personne    → true
```

```
p1 instanceof Object      → true
```

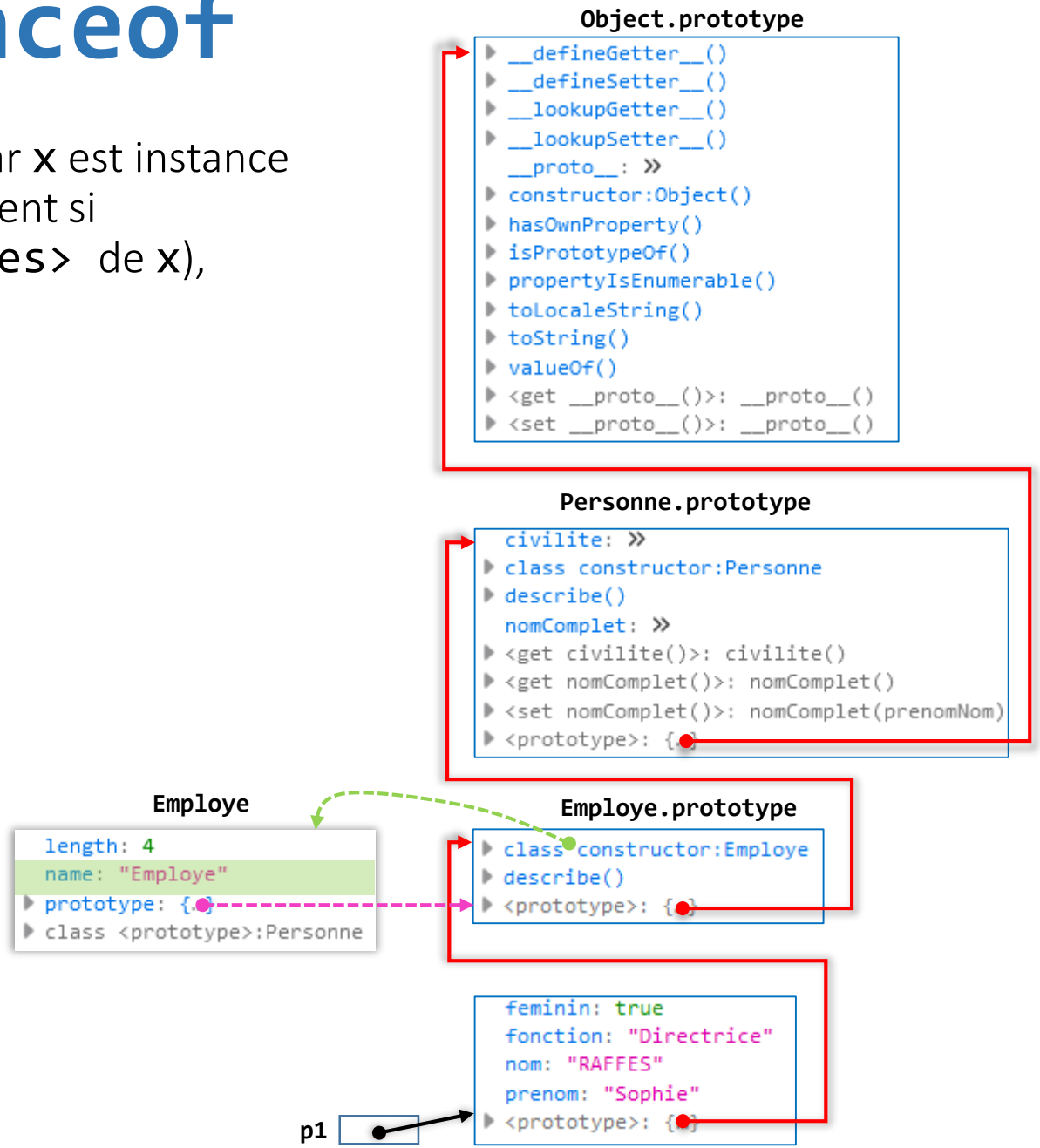
```
let p2 = new Personne("Maeva", "FASFER", true);
```

```
p2 instanceof Employe     → false
```

```
typeof p1 →  object
```

Nom de la classe qui a permis de créer p1

```
p1.constructor.name       → Employe
```



Objets JavaScript

- Quelques liens

  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

  - https://www.w3schools.com/js/js_object_definition.asp
  - https://www.w3schools.com/js/js_class_intro.asp

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

Objets et ES6 : Classes

- Avez-vous compris les objets et chaines de <prototype> ?
  - Exercices
    soit les classes **`Personne`** et **`Employe`** ci-contre
    quel affichage produisent les instructions suivantes ?
    Justifiez vos réponses

```
const p1 = new Personne("Marcel", "TROUSP", false);
const p2 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
console.log(p1.describe());     → ???
```

```
console.log(p2.describe());     → ???
```

```
p1.fonction = 'Sous Directeur';
console.log(p1.describe());     → ???
```

```
delete p2.fonction;
console.log(p2.describe());     → ???
```

```
const p3 = new Personne("J.M.", "FAVRE", false, "Directeur Général");
console.log(p3.describe());     → ???
```

```
const p4 = new Employe("Maeva", "RAFFES", true);
console.log(p4.describe());     → ???
```

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

```
class Employe extends Personne {

    constructor(prenom, nom, feminin, fonction) {
        super(prenom, nom, feminin);
        this.fonction = fonction;
    }

    describe() {
        return super.describe() + ` (${this.fonction})`;
    }
}
```

Objets et ES6 : Classes

- Avez-vous compris les objets et chaines de <prototype> ?
  - Exercices
    soit les classes **`Personne`** et **`Employe`** ci-contre
    quel affichage produisent les instructions suivantes ?
    Justifiez vos réponses

```
const p1 = new Personne("Marcel", "TROUSP", false);
const p2 = new Employe("Sophie", "RAFFES", true, "Directrice");
```

```
console.log(p1.describe());     →  M. TROUSP Marcel
```

```
console.log(p2.describe());     →  Mme RAFFES Sophie (Directrice)
```

```
p1.fonction = 'Sous Directeur';
console.log(p1.describe());     →  M. TROUSP Marcel
```

```
delete p2.fonction;
console.log(p2.describe());     →  Mme RAFFES Sophie (undefined)
```

```
const p3 = new Personne("J.M.", "FAVRE", false, "Directeur Général");
console.log(p3.describe());     →  M. FAVRE J.M.
```

```
const p4 = new Employe("Maeva", "RAFFES", true);
console.log(p4.describe());     →  Mme RAFFES Maeva (undefined)
```

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

```
class Employe extends Personne {

    constructor(prenom, nom, feminin, fonction) {
        super(prenom, nom, feminin);
        this.fonction = fonction;
    }

    describe() {
        return super.describe() + ` (${this.fonction})`;
    }
}
```

Objets et ES6 : Classes - Héritage

- Avez-vous compris les objets et chaines de <prototype> ?
  - Exercices
    soit les classes **Personne** et **Employe** ci-contre
    quel affichage produisent les instructions suivantes ?
    Justifiez vos réponses

```
let tab1 = [
    new Personne("Marcel", "TROUSP", false),
    new Personne("Sophie", "RERHAX", true);
];

let tab2 = [
    new Employe("J.M.", "FAVRE", false, 'Sous Directeur'),
    new Employe("Maeva", "FARRES", true, 'Directrice');
];

let tab3 = [
    new Personne("J.M.", "FAVRE", false),
    new Employe("Maeva", "FARRES", true, 'Directrice');
];
```

```
Personne.lister(tab1);          → ???
```

```
Employe.lister(tab2);           → ???
```

```
Employe.lister(tab3);           → ???
```

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

```
class Employe extends Personne {

    constructor(prenom, nom, feminin, fonction) {
        super(prenom, nom, feminin);
        this.fonction = fonction;
    }

    describe() {
        return super.describe() + ` (${this.fonction})`;
    }
}
```

Objets et ES6 : Classes

- Avez-vous compris les objets et chaines de <prototype> ?
  - Exercices
    soit les classes **Personne** et **Employe** ci-contre
    quel affichage produisent les instructions suivantes ?
    Justifiez vos réponses

```
let tab1 = [
    new Personne("Marcel", "TROUSP", false),
    new Personne("Sophie", "RERHAX", true)
];

let tab2 = [
    new Employe("J.M.", "FAVRE", false, 'Sous Directeur'),
    new Employe("Maeva", "FARRES", true, 'Directrice')
];

let tab3 = [
    new Personne("J.M.", "FAVRE", false),
    new Employe("Maeva", "FARRES", true, 'Directrice')
];
```

```
Personne.lister(tab1);
```

```
Personne
--------
M. TROUSP Marcel
Mme RERHAX Sophie
```

```
Employe.lister(tab2);          → ???
```

```
Employe
--------
M. FAVRE J.M. (Sous Directeur)
Mme FARRES Maeva (Directrice)
```

```
Employe.lister(tab3);
```

```
Employe
--------
M. FAVRE J.M.
Mme FARRES Maeva (Directrice)
```

```
class Personne {
    constructor(prenom, nom, feminin) {
        this.prenom = prenom;
        this.nom = nom;
        this.feminin = feminin;
    }

    get nomComplet() {
        return `${this.nom} ${this.prenom}`;
    }

    set nomComplet(prenomNom) {
        const tokens = prenomNom.split(/\b\s+(?!$)/);
        this.prenom = tokens[0];
        this.nom = tokens[1];
    }

    get civilite() {
        return (this.feminin) ? "Mme" : "M.";
    }

    describe() {
        return `${this.civilite} ${this.nomComplet}`;
    }

    static lister(personnes) {
        console.log(this.name);
        console.log("--------");
        for (let i = 0; i < personnes.length; i++) {
            console.log(personnes[i].describe());
        }
    }
}
```

```
class Employe extends Personne {

    constructor(prenom, nom, feminin, fonction) {
        super(prenom, nom, feminin);
        this.fonction = fonction;
    }

    describe() {
        return super.describe() + ` (${this.fonction})`;
    }
}
```