Le JavaScript orienté objet pour débutants

Précédent Aperçu : Objects Suivant

Après avoir parcouru les fondamentaux, nous allons aborder en détail le JavaScript orienté objet (JSOO). Cet article présente une approche simple de la programmation orientée objet (POO) et détaille comment JavaScript émule des classes objet au travers des méthodes constructeur et comment instancier ces objets.

Pré-requis :	Connaissances de base en informatique et compréhension des notions HTML et CSS, notions de JavaScript (voir Premiers pas et Blocs de construction)
Objectif :	Comprendre les concepts de base derrière la programmation orientée objet et comment ils s'appliquent à JavaScript (« tout est objet ») et comment créer des constructeurs et instancier des objets.

La programmation orientée objet de loin

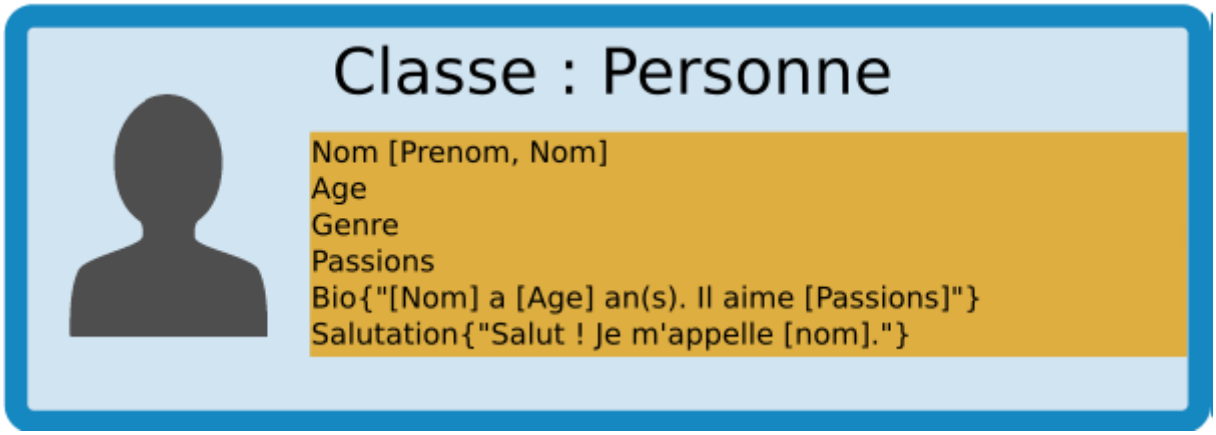
Pour commencer, concentrons-nous une vision simplifiée et haut niveau de la programmation objet (POO). On parle d'une vision simplifiée étant donnée que la POO peut devenir très vite complexe et qu'être exhaustif rendrait probablement la découverte plus confuse et difficile qu'autre chose. L'idée de base de la POO consiste à utiliser des objets pour modéliser les objets du monde réel que l'on souhaite représenter dans nos programmes et/ou de fournir un moyen simple d'accéder à une fonctionnalité qu'il serait difficile d'utiliser autrement.

Les objets peuvent contenir des données et du code représentant de l'information au sujet de la chose que l'on essaie de modéliser ainsi que des fonctionnalités ou un comportement que l'on souhaite lui appliquer. Les données (et bien souvent les fonctions) associées à un objet peuvent être stockées (le terme officiel est **encapsulé**) à l'intérieur d'un paquet objet. Il est possible de donner un nom spécifique à un paquet objet afin d'y faire référence, on parle alors d'**espace de noms** ou *namespace*, il y sera ainsi plus facile de le manipuler et d'y accéder. Les objets peuvent aussi servir pour stocker des données et les transférer facilement sur un réseau.

Définissons un modèle objet

Nous allons voir un programme simple qui affiche des information au sujet des élèves et des professeurs d'une école. Nous allons aborder la théorie de la programmation orientée objet de manière générale sans l'appliquer à un langage particulier.

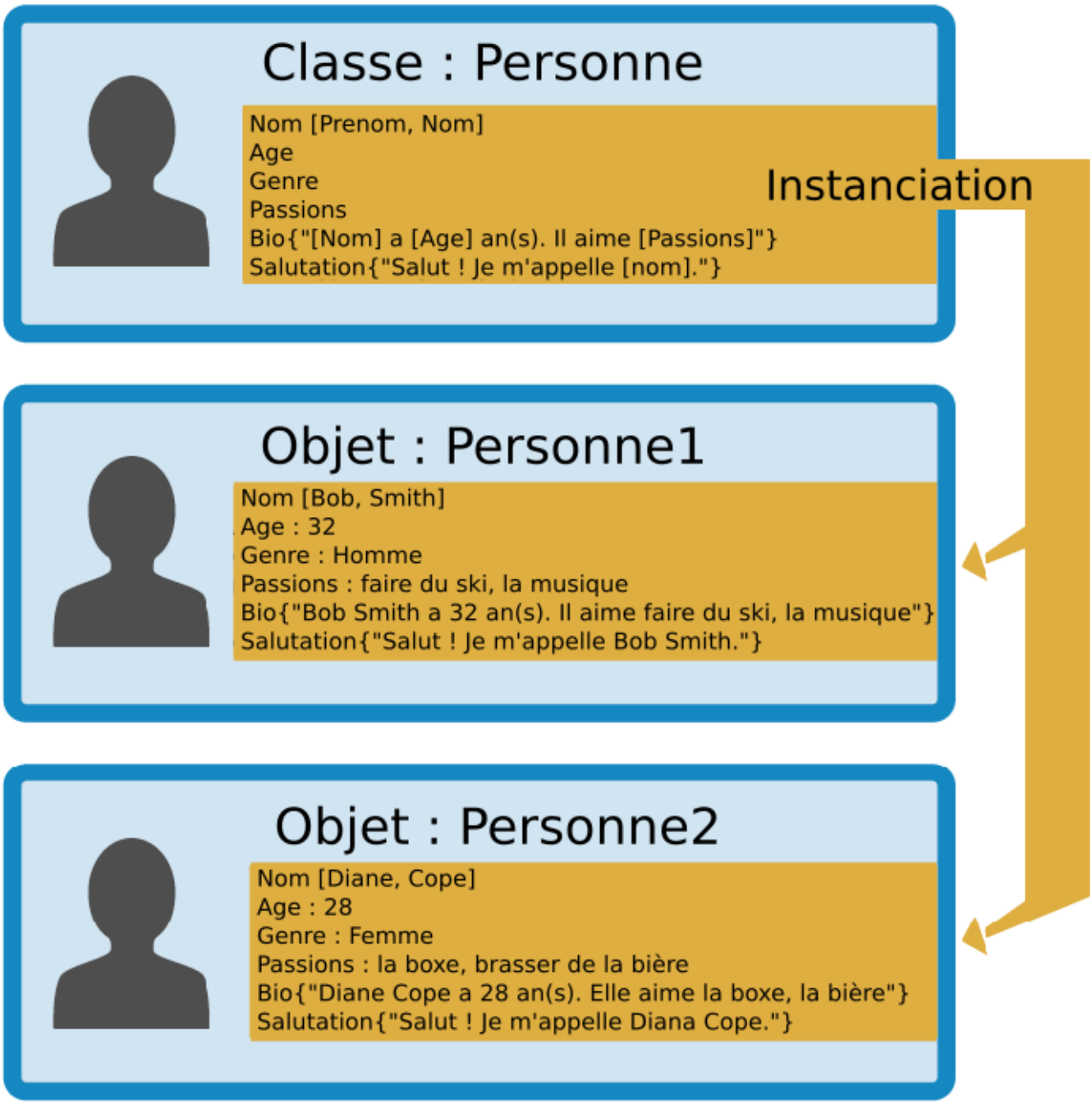
Pour débuter, nous pouvons réutiliser l'objet Personne que nous avons créé dans notre premier article, il définit un ensemble de données et actions d'une personne. Il existe tout un tas de choses que nous pourrions savoir au sujet d'une personne (leur adresse, leur taille, leur pointure, leur ADN, leur numéro de passeport, traits particuliers…). En l'occurrence, nous souhaitons uniquement afficher leur nom, leur âge, leurs passions, écrire une petite introduction à leur sujet en utilisant ces données et leur apprendre à se présenter. On parle alors d'**abstraction** : créer un modèle simplifié de quelque chose de complexe mais qui ne contient que les aspects qui nous intéressent. Il sera alors plus simple de manipuler ce modèle objet simplifié dans le cadre de notre programme.



Dans plusieurs langages de POO, la définition d'un objet est appelé une **classe** (comme on le verra ci-après, JavaScript se base sur un mécanisme et une terminologie différente). En réalité ce n'est pas vraiment un objet mais plutôt un modèle qui définit les propriétés que notre objet doit avoir.

Créons des objets

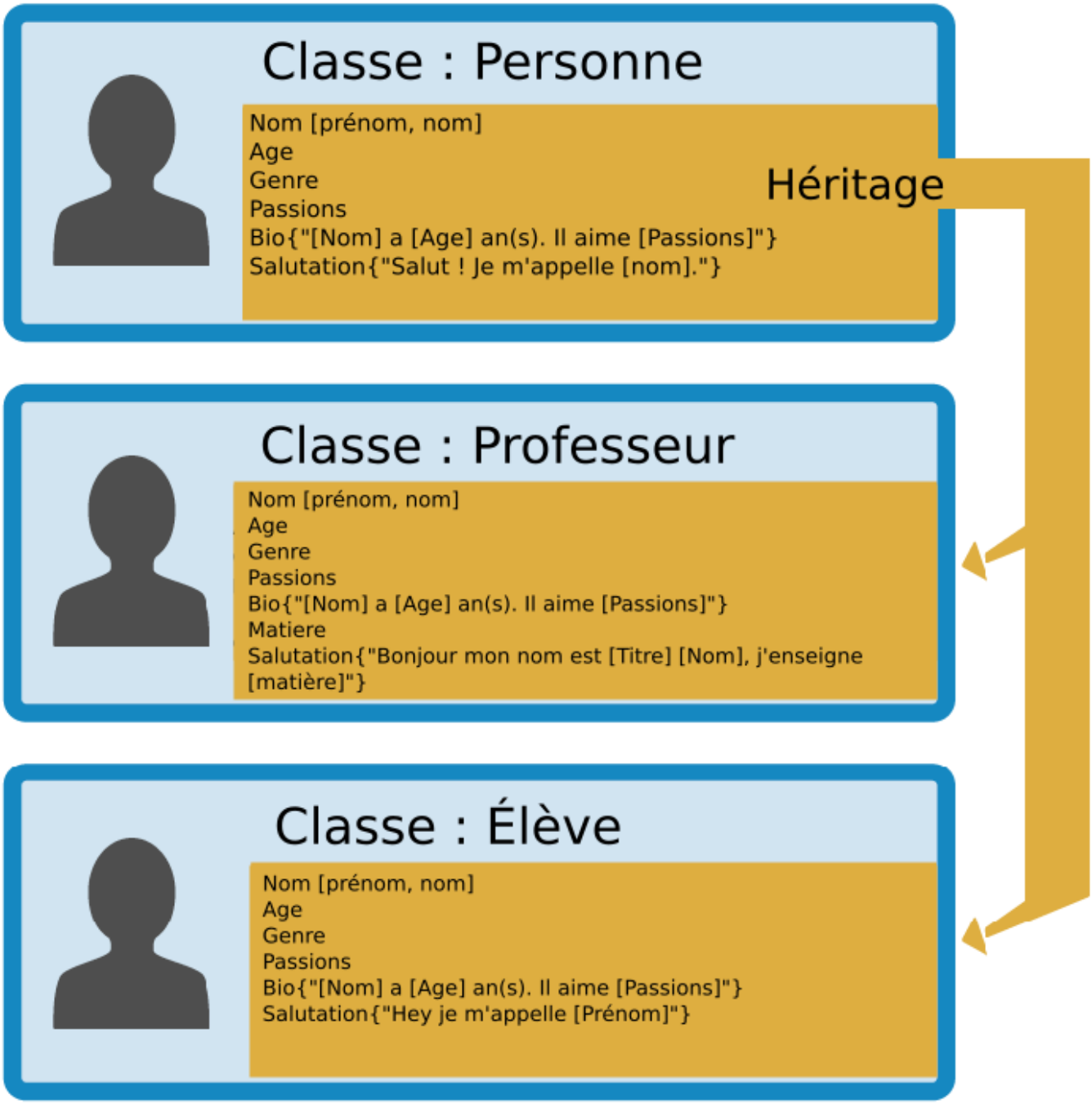
À partir de notre classe, nous pouvons créer des objets, on parle alors d'**instancier des objets**, une classe objet a alors **une instance**. Il s'agit d'objets qui contiennent les données et attributs définis dans une classe. À partir de notre classe Personne, nous pouvons modéliser des personnes réelles :



Lorsque l'instance d'un objet est créé, on appelle la **fonction constructeur** de la classe pour la créer. On parle d'**instanciation** d'un objet — l'objet ainsi créé est **instancié** à partir de la classe.

Classes filles

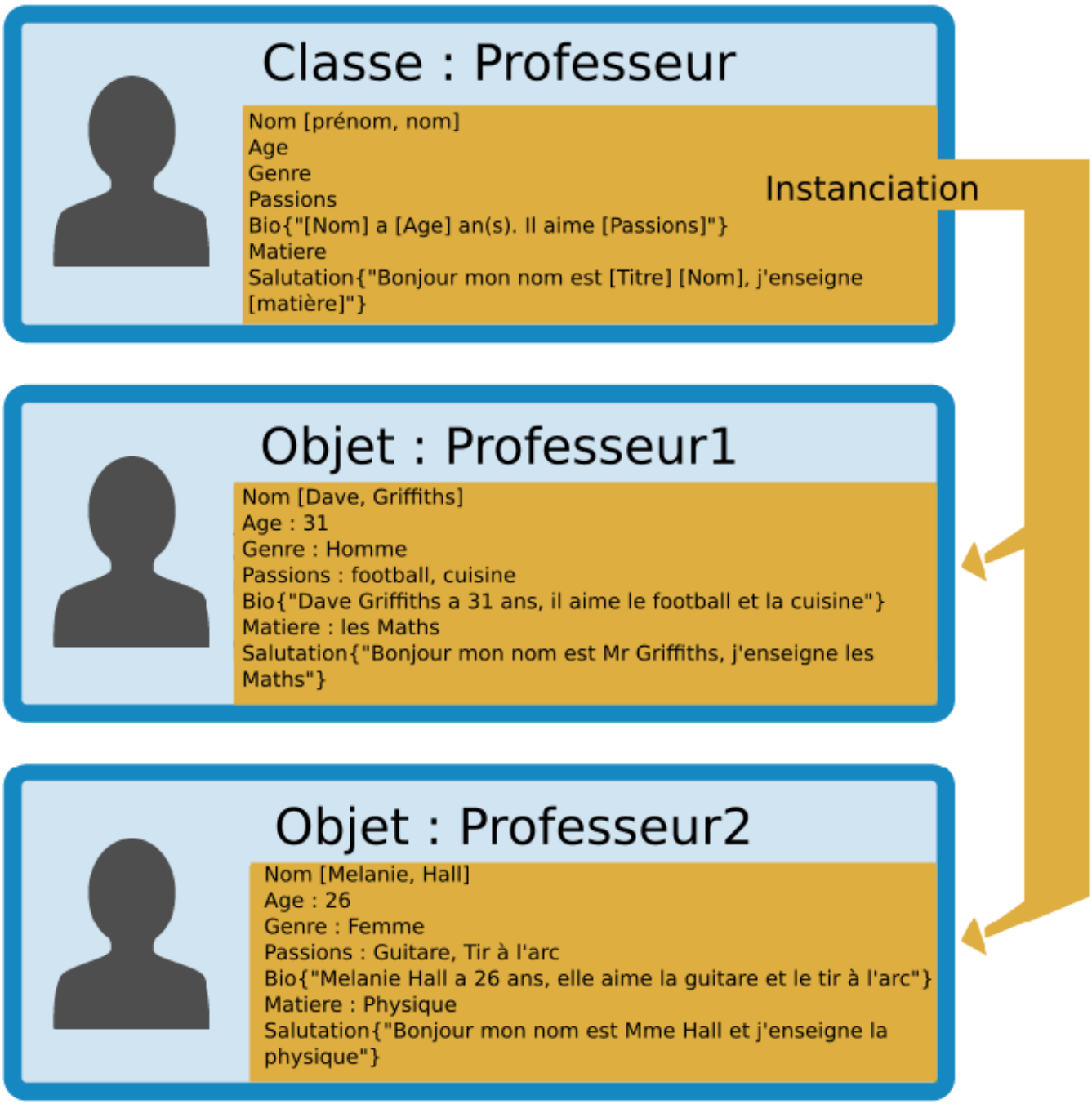
Pour notre exemple, nous n'allons pas nous contenter de personnes génériques — nous pourrions utiliser des professeurs, des étudiants, qui sont des types un peu plus spécifiques de personnes. En POO, il est possible de créer de nouvelles classes à partir d'autres classes — ces **classes filles** nouvellement créées peuvent **hériter** des propriétés et des attributs de leur **classe mère**. Il est donc possible d'avoir des attributs partagés à l'ensemble des classes plutôt que de les dupliquer. Si besoin, il est possible d'ajouter des fonctions et attributs spécifiques sur chaque classe fille.



Cela s'avère très utile puisque les étudiants et les professeurs se ressemblent sur de nombreux aspects : ils ont un nom, un genre, un age, il est donc utile de ne définir ces attributs qu'une seule fois. Il est aussi possible de redéfinir le même attribut dans différentes classes étant donné que l'attribut appartiendra à chaque fois à un nom d'espace différent. On pourra ainsi avoir différentes formes de salutations : « Hey, je m'appelle [prénom] » pour les étudiants (« Hey je m'appelle Sam ») tandis que les professeurs pourront dire quelque chose d'un peu plus soutenu comme « Bonjour, mon nom est [Titre][Nom] et j'enseigne [matière] » par exemple « Bonjour mon nom est M. Griffiths et j'enseigne la chimie ».

> **Note** : On parle de **polymorphisme**, lorsque des objets réutilisent la même propriété, mais c'est juste pour info, vous embêtez pas.

Une fois la classe fille créée il est alors possible de l'instancier et de créer des objets. Par exemple :



Dans la suite de l'article, nous nous intéresserons à la mise en œuvre de la programmation orientée objet (POO) au sein de JavaScript.

Constructeurs et instances d'objet

Certains disent que le JavaScript n'est pas vraiment un langage de programmation orienté objet — Il n'existe pas, en JavaScript d'élément `class` pour créer des classes alors que c'est

le cas dans plusieurs langages orientés objet. JavaScript quant à lui, utilise des fonctions spéciales appelées **constructeurs** pour définir les objets et leurs propriétés. Ces constructeurs s'avèrent utiles, puisque bien souvent, nous ne savons pas combien d'objets nous allons définir, les constructeurs nous permettent de créer autant d'objets que nécessaire et d'y associer des données et des fonctions au fur et à mesure.

Lorsqu'un objet est instancié à partir d'une fonction constructeur, les fonctions de la classe ne sont pas copiées directement dans l'objet comme dans la plupart des langages orientés objet (OO). En JavaScript, les fonctions sont liées grâce à une chaîne de référence appelée chaîne prototype (voir Prototypes Objet). Il ne s'agit donc pas d'une véritable instanciation au sens strict puisque JavaScript utilise un mécanisme différent pour partager des fonctionnalités entre les objets.

> **Note** : Ne pas être un "langage classique de POO" n'est pas nécessairement un défaut. Comme nous le mentionnions au début de l'article, la POO peut très vite devenir compliquée et JavaScript, grâce à ses différences parvient à utiliser certains concepts avancés tout en restant abordable.

Voyons comment créer des classes via les constructeurs et les utiliser pour instancier des objets en JavaScript. Nous allons commencer par faire une copie locale du fichier oojs.html que nous avons vu dans notre premier article sur les objets.

Un exemple simple

1. Tout d'abord ; voyons comment définir une personne au travers d'une fonction classique. Vous pouvez ajouter l'exemple ci-dessous dans votre code existant :

```
function creerNouvellePersonne(nom) {
  var obj = {};
  obj.nom = nom;
  obj.salutation = function() {
    alert('Salut ! Je m\'appelle ' + this.nom + '.');
  };
  return obj;
}
```

2. Vous pouvez désormais créer une personne en appelant cette fonction, essayez en copiant les lignes suivantes dans la console JavaScript de votre navigateur :

```
var salva = creerNouvellePersonne('Salva');
```

```
salva.nom;
salva.salutation();
```

Ça fonctionne bien, mais on peut améliorer notre exemple. Si l'on sait que l'on va créer un objet, pourquoi créer un objet vide pour l'utiliser ensuite ? Heureusement, JavaScript est là et possède des fonctions adaptées comme les constructeurs. À l'abordage !

3. Remplacez la fonction précédent par celle-ci :

```
function Personne(nom) {
  this.nom = nom;
  this.salutation = function() {
    alert('Bonjour ! Je m\'appelle ' + this.nom + '.');
  };
}
```

Le constructeur est l'équivalent JavaScript d'une classe. Il possède l'ensemble des fonctionnalités d'une fonction, cependant il ne renvoie rien et ne crée pas d'objet explicitement. Il se contente de définir les propriétés et les méthodes associées. Il y a aussi l'utilisation du mot-clé `this`, ce mot-clé sert au sein d'une instance qui sera créée à y faire référence, ainsi l'attribut nom sera, pour l'instance, égal au nom passé en argument de la fonction constructrice, la méthode `salutation()` retournera elle aussi le nom passé en argument de la fonction constructrice.

> **Note** : Les fonctions de type constructeur commencent généralement par une majuscule. Cette convention d'écriture permet de repérer les constructeurs plus facilement dans le code.

Comment pouvons-nous utiliser un constructeur ?

1. Ajoutez les lignes suivantes au code déjà existant :

```
var personne1 = new Personne('Bob');
var personne2 = new Personne('Sarah');
```

2. Enregistrez votre code et relancez le dans votre navigateur puis essayez d'entrer les lignes suivantes dans la console :

```
personne1.nom
personne1.salutation()
personne2.nom
personne2.salutation()
```

Pas mal ! Vous voyez désormais que nous avons deux nouveaux objets sur cette page, chaque objet étant stocké dans un nom d'espace différent, pour y accéder il faut utiliser `personne1` et `personne2` pour préfixer les fonctions et attributs. Ce rangement permet de ne pas tout casser et de ne pas rentrer en collision avec d'autres fonctionnalités. Cependant les objets disposent du même attribut `nom` et de la même méthode `salutation()`. Heureusement, les attributs et les méthodes utilisent `this` ce qui leur permet d'utiliser les valeurs propres à chaque instance et de ne pas les mélanger.

Revoyons l'appel au constructeur :

```
var personne1 = new Personne('Bob');
var personne2 = new Personne('Sarah');
```

Dans chaque cas, le mot clé `new` est utilisé pour dire au navigateur que nous souhaitons définir une nouvelle instance, il est suivi du nom de la fonction que l'on utilise et de ses paramètres fournis entre parenthèses, le résultat est stocké dans une variable. Chaque instance est créée à partir de cette définition :

```
function Personne(nom) {
  this.nom = nom;
  this.salutation = function() {
    alert('Bonjour ! Je m\'appelle ' + this.nom + '.');
  };
}
```

Une fois les objets créés, les variables `personne1` et `personne2` contiennent les objets suivants :

```
{
  nom: 'Bob',
  salutation: function() {
    alert('Bonjour ! Je m\'appelle ' + this.nom + '.');
  }
}

{
  nom: 'Sarah',
  salutation: function() {
    alert('Bonjour ! Je m\'appelle ' + this.nom + '.');
  }
```

```
}
```

On peut remarquer qu'à chaque appel de notre fonction constructrice nous définissons `salutation()` à chaque fois. Cela peut être évité via la définition de la fonction au sein du prototype, ce que nous verrons plus tard.

Créons une version finalisée de notre constructeur

L'exemple que nous avons utilisé jusqu'à présent était destiné à aborder les notions de base des constructeurs. Créons un constructeur digne de ce nom pour notre fonction constructrice `Personne()`.

1. Vous pouvez retirer le code que vous aviez ajouté précédemment pour le remplacer par le constructeur suivant, c'est la même fonction, ça reste un constructeur, nous avons juste ajouté quelques détails :

```
function Personne(prenom, nom, age, genre, interets) {
  this.nom = {
    prenom,
    nom
  };
  this.age = age;
  this.genre = genre;
  this.interets = interets;
  this.bio = function() {
    alert(this.nom.prenom + ' ' + this.nom.nom + ' a ' + th
  };
  this.salutation = function() {
    alert('Bonjour ! Je m\'appelle ' + this.nom.prenom + '.
  };
};
```

2. Vous pouvez ajouter la ligne ci-dessous pour créer une instance à partir du constructeur :

```
var personne1 = new Personne('Bob', 'Smith', 32, 'homme', [
```

Vous pouvez accéder aux fonctions des objets instanciés de la même manière qu'avant :

```
personne1['age']
personne1.interets[1]

```

```
personne1.bio()
// etc.
```

Note : Si vous avez du mal à faire fonctionner cet exemple, vous pouvez comparez votre travail avec notre version (voir oojs-class-finished.html (vous pouvez aussi jeter un œil à la démo)

Exercices

Vous pouvez démarrer en instanciant de nouveaux objets puis en essayant de modifier et d'accéder à leurs attributs respectifs.

D'autre part, il y a quelques améliorations possibles pour notre méthode `bio().` En effet elle ne retourne que le pronom *He* (Il en français), même si l'instance est une femme ou possède un autre genre. La biographie n'inclut que deux passions, même s'il y en a plus dans la liste. Essayez d'améliorer cette méthode. Vous pouvez mettre votre code à l'intérieur d'un constructeur (vous aurez probablement besoin d'un peu de logique et de boucles). Réflechissez à l'organisation des phrases en fonction du genre et du nombre de passions listées dans la liste.

Note: Si vous êtes bloqués, nous avons mis une réponse possible sur notre dépôt GitHub (la démo) —tentez d'abord l'aventure avant d'aller regarder la réponse !

D'autres manières d'instancier des objets

Jusque là nous n'avons abordé que deux manières différentes pour créer une instance d'objet, la déclarer de manière explicite et en utilisant le constructeur.

Elles sont toutes les deux valables, mais il en existe d'autres. Afin que vous les reconnaissiez lorsque vous vous baladez sur le Web, nous en avons listées quelques unes.

Le constructeur Object()

Vous pouvez en premier lieu utiliser le constructeur `Object()` pour créer un nouvel objet. Oui, même les objets génériques ont leur propre constructeur, qui génère un objet vide.

1. Essayez la commande suivante dans la console JavaScript de votre navigateur :

```
var personne1 = new Object();
```

2. On stocke ainsi un objet vide dans la variable personne1. Vous pouvez ensuite ajouter des attributs et des méthodes à cet objet en utilisant la notation point ou parenthèses comme vous le souhaitez.

```
personne1.nom = 'Chris';
personne1['age'] = 38;
personne1.salutation = function() {
  alert('Bonjour ! Je m\'appelle ' + this.nom + '.');
};
```

3. Vous pouvez aussi passer un objet en paramètre du constructeur `Object()`, afin de prédéfinir certains attributs et méthodes.

```
var personne1 = new Object({
  nom: 'Chris',
  age: 38,
  salutation: function() {
    alert('Bonjour ! Je m\'appelle ' + this.nom + '.');
  }
});
```

Via la méthode create()

JavaScript intègre directement une méthode appelée `create()`. Elle permet d'instancier un objet à partir d'un objet existant.

1. Essayez d'ajouter la ligne suivante dans votre console JavaScript :

```
var personne2 = Object.create(personne1);
```

2. Mantenant :

```
personne2.nom
personne2.salutation()
```

`personne2` a été créée à partir de `personne1` — et elle possède les mêmes propriétés. C'est assez utile puisque ça permet de créer de nouvelles instances sans devoir définir un constructeur. D'autre part, les constructeurs permettent de structurer le code : vous pouvez

avoir l'ensemble de vos constructeurs au même endroit et ensuite créer les instances étant donné qu'elles proviennent du constructeur associé.

L'inconvénient de `create()` est qu'elle n'est pas supportée par IE8. Ainsi, utiliser les constructeurs peut s'avérer peu judicieux lorsqu'il s'agit de supporter les anciens navigateurs Web ou si vous n'avez besoin que de quelques instances d'objet. Faites comme vous le souhaitez.

Nous verrons les détails et les effets de `create()` plus tard.

Résumé

Cet article a présenté une vue simplifiée de la programmation orientée objet. Tout n'y a pas été détaillé mais ça vous permet d'avoir une idée de ce que nous faisons. Nous avons ensuite pu voir comment JavaScript s'appuyait sur un certain nombre de principes orienté objet tout en ayant un certain nombre de particularités. Nous avons aussi vu comment implémenter des classes en JavaScript via la fonction constructeur et les différentes possibilités permettant de générer des instances d'objets.

Dans le prochain article, nous explorerons le monde des prototypes objet en JavaScript.

← Précédent ↑ Aperçu : Objects Suivant →