TP JS 3 - Objets

Décompressez l'archive déposée sur Moodle pour ce TP. Le dossier résultant contient différents fichiers à réutiliser ou à compléter. Pensez à consulter le site MDN, et pour visualiser ce qui est attendu, ce démonstrateur.

Rappel sur objets et prototypes en JS

```
// constructeur
function Foo(y) {
  // propriété en propre pour chaque objet construit avec Foo
  this.y = y;
}
// propriété héritée par tout objet construit avec Foo
Foo.prototype.x = 10;
// méthode héritée par tout objet construit avec Foo
Foo.prototype.calculate = function (z) {
    return this.x + this.y + z;
    };
var b = new Foo(20);
var c = new Foo(30);
b.calculate(30); // 60
c.calculate(40); // 80
console.log(
  b.__proto__ === Foo.prototype, // true
  c.__proto__ === Foo.prototype, // true
  b.constructor === Foo, // true
  c.constructor === Foo, // true
  Foo.prototype.constructor === Foo, // true
  b.calculate === b.__proto__.calculate, // true
  b.__proto__.calculate === Foo.prototype.calculate // true
);
```

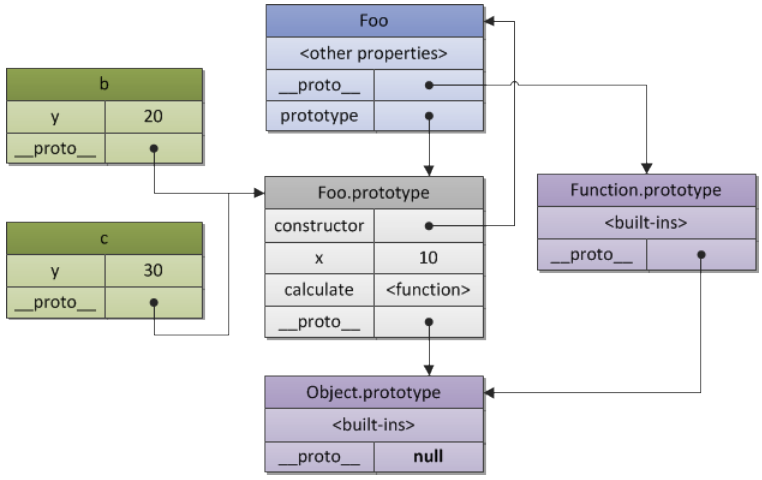

FIGURE 1 – Liens entre objets et prototypes

Exercice 1. Construction d'objects

Complétez **dalton.html** pour y importer en mode différé les fichiers **dalton1.js**, **dalton2.js** et **dalton3.js** dans cet ordre. Complétez ensuite **dalton1.js** pour répondre aux questions de cet exercice.

1. Créez un constructeur `Dalton` modélisant un "Dalton" par son prénom.

2. Créez un Dalton `averell` de prénom "Averell".

3. Créez une fonction `log` prenant un objet en paramètre et l'affichant à la console ainsi que sa propriété `prénom`. Appelez la avec `averell`.

4. Créez un Dalton avec `Object.create` et fixez son prénom à "Jack". Affichez cet objet avec `log`.

5. Créez un objet littéral `joe` ayant "Joe" pour valeur de la propriété `prénom`. Affectez un objet anonyme de type `Dalton` à son prototype. L'afficher avec `log`.

6. Créez un objet `william` à partir de la chaîne au format JSON `'{ "prénom" : "William" }'` en utilisant la méthode `JSON.parse`. Modifiez son prototype comme ci-dessus. L'afficher avec `log`.

7. `"{ 'prénom' : 'William' }"`, `'{ "prénom" : William }'` et `'{ "prénom" : "William", }'` sont-elles des chaînes au format JSON ?

8. Comparez la propriété `prototype` du constructeur `Dalton` avec le prototype objet de chacun des 4 objets Dalton en y accédant avec `Object.getPrototypeOf()`. Remontez la chaîne de prototype si nécessaire pour obtenir l'égalité.

9. Invoquez la méthode `hasOwnProperty` sur l'objet `william` pour tester si la propriété `prénom` est une propriété (en) propre de cet objet. Invoquez `Object.getOwnPropertyNames` avec le paramètre `william` pour récupérer ses propriétés propres. Invoquez `Object.keys` avec le paramètre `william` pour récupérer ses propriétés propres et énumérables (itérables). Itérez sur toutes les propriétés énumérables (propres ou héritées) de l'objet `william` avec une boucle `for ... in`.

Exercice 2. Prototypes

Complétez **dalton2.js** pour répondre aux questions de cet exercice.

1. Ajoutez aux prototypes des objets construits avec `Dalton` une propriété `nom` égale à "Dalton".

2. Ajoutez aux prototypes des objets construits avec `Dalton` une méthode `afficher` qui est sans arguments et affiche leurs nom et prénom dans la console.

3. Créez un tableau `daltons` contenant les 4 objets `DALTON`. Invoquez la méthode `afficher` sur ses éléments en itérant sur `daltons` avec une boucle `for ... of`. Même question mais en utilisant la méthode `map` (héritées par les objets `Array`).

4. Invoquez la méthode `hasOwnProperty` sur l'objet `william` pour vérifier que `nom` est une propriété héritée par cet objet. Itérez sur l'objet `william` avec une boucle `for ... in` pour afficher clé et valeur de ses propriétés énumérables. Itérez sur le tableau `daltons` avec une boucle `for ... in` pour afficher clé et valeur de ses propriétés énumérables. Que représentent les clés affichées ?

5. Détruisez la propriété `nom` du constructeur `DALTON` et réaffichez les objets `DALTON` du tableau.

6. Affichez parmi les 4 objets `DALTON` du tableau ceux dont le prénom commence par la lettre `J`. Chainez les méthodes de tableaux `filter` et `map` à cet effet.

Exercice 3.

Complétez **dalton3.js** pour répondre aux questions de cet exercice.

1. Créez un constructeur `Famille` prenant une chaîne `nom` en paramètre, et définissant pour "ses" objets une propriété `nom` égale à cette chaîne si elle définie ou à la chaîne vide sinon, et une propriété `membres` égale au tableau vide.

2. Construisez un objet `DALTON` de nom `Dalton` avec `Famille`.

3. Ajoutez aux prototypes des objets construits avec `Famille` une méthode `ajouter` prenant un objet en paramètre et ajoutant sa propriété `prénom` en fin du tableau `membres`.

4. Invoquez cette méthode sur `DALTON` pour y ajouter `averell`.

5. Invoquez cette méthode sur `DALTON` pour y ajouter `jack` en utilisant la méthode `call` héritées par les objets `Function`.

6. Invoquez cette méthode sur `DALTON` sur chacun des éléments du tableau `[joe,william]` en utilisant la méthode `map` héritées par les objets `Array`.

7. Ajoutez aux prototypes des objets créés avec `Famille` une méthode `afficher` affichant leur `nom` et les prénoms de leurs `membres`. L'invoquer sur `DALTON`.

Exercice 4.

Complétez **personne.html** pour y importer en mode différé les fichiers **personne.js** et **professeur.js** dans cet ordre. Complétez ensuite **personne.js** pour répondre aux questions de cet exercice.

1. Créez un constructeur `Personne` modélisant une personne par son identité (nom et prénom), son âge et ses centres d'intérêts. Ce constructeur fournira deux méthodes produisant des alertes telles que celles illustrées en Figure 2 et 3.

- `salutation` qui affiche une alerte du type `Bonjour, je m'appelle X`;
- `bio` qui affiche une alerte du type `Nom : X, Prénom : Y, Age : nn, Centres d'intérets : I1, I2,... et In`.

FIGURE 2 – Salutations

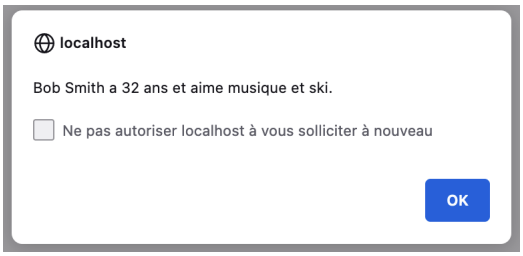

FIGURE 3 – Bio

2. Créez un objet avec ce constructeur, en examinez le contenu et le prototype.

3. Ajoutez une méthode `aurevoir` à la propriété `prototype` du constructeur produisant une alerte telle qu'illustrée en Figure 4. Testez cette méthode sur l'objet précédent et en réexaminez le prototype.

FIGURE 4 – Adios

Exercice 5. Héritage prototypique

Complétez **professeur.js** pour répondre aux questions de cet exercice.

1. On modélisera un professeur comme une personne enseignant une matière. Créez un constructeur `Professeur` qui, par héritage prototypique, hérite de `Personne` et ajoute une propriété spécifique `matière`. L'affichage attendu est illustré en Figure 5.

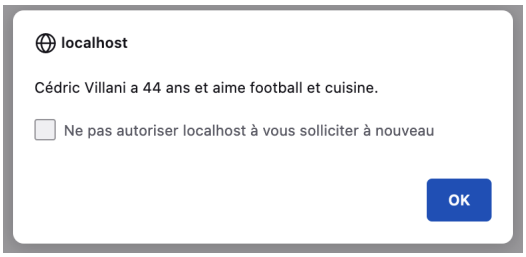

FIGURE 5 – Bio

2. Ajoutez au prototype de `Professeur` une méthode `saluer` qui affiche nom et matière enseignée. L'affichage attendu est illustré en Figure 6.

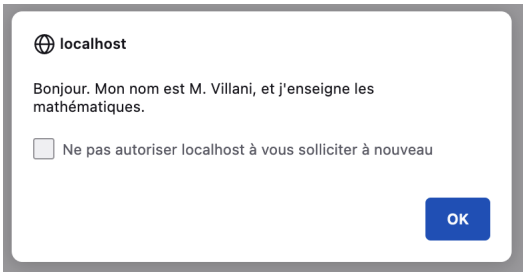

FIGURE 6 – Salutations