Programmation orientée objet par prototype avec jQuery

par Daniel Hagnoul (Mon cahier d'exercices sur jQuery & Co)

Date de publication : 28 septembre 2010

Dernière mise à jour :

Je suis persuadé que jQuery est un outil fantastique pour la manipulation du DOM, mais cela ne représente que 80 % de la programmation en JavaScript. Pour le solde, la Programmation Orientée Objet (POO) répond à l'essentiel des besoins. Faut-il pour autant recourir obligatoirement à une autre bibliothèque JavaScript comme Dojo ? Non, je crois qu'il suffit d'un outil léger, permettant l'héritage simple.

L'outil, l'objet $.dvjhClass, permet de réunir dans un même corpus la fonction d'initialisation et son prototype.

(1) Il simplifie l'usage de la méthode d'héritage simple décrite par Thierry Templier au chapitre 1.4 de la deuxième partie de son tutoriel : "Programmation orientée objet avec le langage JavaScript".

I - Introduction

I-A - Introduction

Je résumerai le titre par le vocable POOPJ, prononcez "pop j".

Je suis persuadé que jQuery est un outil fantastique pour la manipulation du DOM, mais cela ne représente que 80 % de la programmation en JavaScript. Pour le solde, la Programmation orientée objet (POO) répond à l'essentiel des besoins. Faut-il pour autant recourir obligatoirement à une autre bibliothèque JavaScript comme Dojo ? Non, je crois qu'il suffit d'un outil léger, permettant l'héritage simple, c'est pourquoi j'ai créé POOPJ.

$.dvjhClass est l'outil qui permet d'écrire du code poopj. C'est un objet JSON comprenant trois méthodes : _create(), _base() et toString(). L'outil permet de réunir dans un même corpus la fonction d'initialisation et son prototype.

(1) $.dvjhClass simplifie l'usage de la méthode d'héritage simple décrite par Thierry Templier au chapitre 1.4 de la deuxième partie de son tutoriel : "Programmation orientée objet avec le langage JavaScript".

Cette technique d'héritage ne permet pas le polymorphisme (objX instanceof objY), mais c'est la plus appropriée dans la plupart des cas. Si le polymorphisme et l'héritage multiple vous manquent, le tutoriel de Thierry Templier décrit d'autres techniques d'héritage.

Le code poopj n'est qu'un emballage qui rend l'Orienté Objet par Prototype (OOP) plus séduisant, et facilite son utilisation. Mais tous les avantages du OOP, tous ses travers et toutes ses insuffisances sont toujours là.

Nous examinerons d'abord, avec l'aide de plusieurs exemples simples, comment générer du code poopj avec §.dvjhClass._create(). Nous regarderons ensuite, très succinctement, le code de l'objet $.dvhClass qui ne présente aucune difficulté particulière pour un initié.

J'ai écrit l'outil en jQuery, mais vous pouvez utiliser ou publier une version purement JavaScript dans le respect des termes de la licence.

I-B - Quelques rappels sur la POO

La notion de classe n'existe pas en JavaScript, mais le besoin de qualifier la chose existant bien, je désigne le code généré par $.dvjhClass._create() sous le vocable "fonction classe".

(2) Je cite Grady Booch : "La programmation orientée objet est une méthode d'implémentation dans laquelle des programmes sont organisés comme des ensembles coopérants d'objets, chacun représentant une instance d'une certaine classe, et toutes les classes sont des membres d'une hiérarchie de classes unifiées par des relations d'héritage."

En POO, les mots instance et objet sont synonymes.

L'héritage simple représente une hiérarchie d'abstractions, dans laquelle une "fonction classe" fille hérite d'une "fonction classe" mère. Typiquement, une "fonction classe" fille enrichit ou redéfinit la structure et le comportement de la "fonction classe" mère. (Cette phrase a été adaptée à mes besoins, l'originale est de Grady Booch (2).)

On doit distinguer trois sortes de relation entre les objets. La relation "est un" qui correspond à l'héritage, la relation "à un" qui correspond à un composant de l'objet et la relation "utilise un" qui correspond à un autre objet. Exemple : "Médor est un chien", "Médor à une queue" et "Médor utilise un collier".

Choisir entre les relations "à un" et "utilise un" dépend aussi du contexte de votre application. Exemple : "L'avion à un moteur" et "L'avion utilise un moteur". Dans l'absolu, il s'agit d'une relation "utilise un" mais si votre application s'intéresse surtout à la navigation aérienne, il n'est pas utile de développer une hiérarchie de classes pour décrire les moteurs d'avions.

(3) Il existe des règles à suivre pour obtenir un code POO de qualité. Mais, sans le support des mécanismes intégrés dans un véritable langage POO, elles peuvent paraître impossibles à satisfaire. Détrompez-vous, même en poopj, il est tout à fait possible d'appliquer une grande partie des règles de bases dans le but d'obtenir une bonne conception orientée objet.

Une "fonction classe" doit avoir une responsabilité unique. Elle ne doit regrouper que les fonctionnalités nécessaires à la réalisation d'un seul but.

Si le but à atteindre est complexe, construisez une hiérarchie de classes.

Toutes les "fonctions classes" d'une hiérarchie doivent offrir une interface commune. Dans cette interface, la dénomination et la signature (nombre et ordre des paramètres) des accesseurs, des mutateurs et des méthodes doivent être identiques.

Une "fonction classe" doit être conçue de telle sorte qu'elle puisse intégrer une nouvelle méthode et qu'elle puisse devenir la "fonction classe" mère d'une hiérarchie de "fonctions classes" filles sans qu'il soit nécessaire de modifier le code existant.

Une "fonction classe" mère peut-être abstraite. C'est même souvent une nécessité, car l'implémentation d'une méthode peut dépendre du contexte d'une "fonction classe" fille.

Une "fonction classe" ne doit pas tester (instanceof) l'objet qu'elle utilise, car cet objet doit pouvoir être une instance de n'importe quelle "fonction classe" d'une hiérarchie de "fonctions classes". Bien entendu, cela n'est possible que si toutes les "fonctions classes" d'une hiérarchie possèdent une interface commune.

L'état d'un objet est défini par la valeur de ses attributs, on dit qu'un objet change d'état lorsque l'on modifie la valeur d'un attribut de l'objet. Avant et après chaque changement d'état, on doit s'assurer, par des tests, que l'objet est toujours dans un état cohérent.

Une "fonction classe" fille ne doit jamais assouplir les contraintes imposées par sa "fonction classe" mère.

Dans un système de "fonctions classes" collaborant entre elles dans un même but, il peut sembler logique de trouver des dépendances entre les "fonctions classes". Mais pour des raisons de réutilisabilité et de maintenance du code, vous devez réduire ces dépendances au minimum. Pensez à séparer les couches métier, interface utilisateur et contrôle (M.V.C.). Même en JavaScript on peut satisfaire largement à cette règle en utilisant les événements.

Le code poopj, comme n'importe quel code JavaScript, doit être placé dans un fichier séparé.

Du point de vue de la conception et de la maintenance, ne placez jamais plusieurs "fonctions classes" sans rapport entre elles dans le même fichier.

Du point de vue de l'utilisateur, dans le cas d'un système de "fonctions classes" collaborant entre elles dans un même but ou d'une hiérarchie de "fonctions classes", livrez un seul fichier compressé (minified), en POO on parle de module distinct.

I-C - Les prérequis

Une bonne maîtrise des bases : CSS, HTML, JavaScript, et la notation JSON.

Les langages véritablement conçus pour la XOO (X pour analyse, conception et programmation. C++, Java, C#, etc.), n'en requièrent pas moins des programmeurs rigoureux, méticuleux et soigneux. JavaScript ne possédant aucun des mécanismes évolués et des garde-fous des véritables langages XOO, le codage et l'utilisation du poopj ne reposent que sur les épaules des programmeurs rigoureux, méticuleux et soigneux.

Si le programmeur qui a conçu les "fonctions classes" est celui qui les utilise, ou si le programmeur qui les utilise a les mêmes qualités, tout va bien.

Aux autres, aux amateurs de la bidouille, "du truc qui", aux amateurs des amas de codes jetés au hasard sans ordre ni méthode, aux oublieux des points virgules et en notation JSON des virgules, je prédis les pires catastrophes.

En bref, si vous codez comme un cochon le résultat sera porcin !

C'est également une des raisons pour laquelle j'ai écrit l'outil en jQuery, car les bons programmeurs jQuery ont nécessairement acquis une partie des qualités requises.

I-D - Utilité de l'héritage

(4) Voici, sortit de son contexte et dans une traduction approximative, ce qu'en pense Douglas Crockford : "Je n'ai jamais trouvé nécessaire d'utiliser l'héritage. L'idée est importante, mais elle semble être inutile en JavaScript. Je considère aujourd'hui mes premières tentatives pour intégrer l'héritage en JavaScript comme une erreur."

La complexification des codes JavaScript allant croissant, j'espère que les principes de rigueur, d'ordre et de méthode s'appliqueront bientôt au JavaScript comme ils se sont appliqués, avec succès, dans d'autres langages. Je crois que l'orienté objet par prototype (OOP) est un outil qui peut être utilisé dans ce but. Par l'expérience acquise en C++, je me vois mal utiliser la programmation orientée objet (POO) en étant privé de l'héritage simple.

II - Construction d'une fonction classe poopj

Téléchargement de POOPJ.zip

La "fonction classe" classique et son prototype sont réunis dans le même corpus, un objet anonyme.

Squelette d'une fonction classe poopj

```
try {
  var FuncClass = $.dvjhClass._create( {

  // contenu de l'objet anonyme

} );
}
catch(err){
  alert(err);
}
```

Le contenu de la "fonction classe" classique devient la méthode _builder() et les méthodes du prototype s'écrivent à la suite du constructeur. L'ensemble sera désigné comme une "fonction classe" poopj.

Construction d'une fonction classe poopj (poopj1.html)

```
try {
  var Personne = $.dvjhClass._create({
    _builder: function(prenom, nom){
      this.prenom = prenom || "";
      this.nom = nom || "";
      this.adresse = "Belgique";
```

Construction d'une fonction classe poopj (poopj1.html)

```
    },
    getAdresse: function(){
      return "J'habite en " + this.adresse;
    },
      toString: function(){
      return "Personne : ";
    }
    });

  var moi = new Personne("Pierre", "Dupond");

  /*
   * Je vous conseille l'usage intensif de Firebug.
   *
   * Tous mes exemples envoient leurs résultats dans
   * la console Firebug.
   *
   * J'utilise la méthode toString() pour donner
   * un nom à l'objet.
   */
   // Personne : { prenom="Pierre",  nom="Dupond",  more...}
   console.log(moi);

   console.log(moi.getAdresse());
}
catch(err){
   alert(err);
}
```

III - Accesseurs et mutateurs. Attribut et méthode statiques

Toutes ces possibilités existent déjà dans le système OOP.

Pour l'attribut this.numero :

1 L'accesseur, getNumero(), est une méthode qui permet d'obtenir la valeur de l'attribut ;
2 Le mutateur, setNumero(value), est une méthode qui permet de changer la valeur de l'attribut this.numero.

Une méthode (idem pour l'attribut) statique est une méthode qui est rattachée à la "fonction classe". Exemples célèbres : Math.PI et Math.floor(value).

Utilisation d'un mutateur (poopj2.html)

```
try {
  var Personne = $.dvjhClass._create({
 _builder: function(prenom, nom, numero){
  this.prenom = prenom || "";
  this.nom = nom || "";
  this.setNumero(numero);
 },
 setNumero: function(value){
  var n = parseInt(value, 10);

  if (!isNaN(n) && n >= 100){
   this.numero = n;
  } else {
   throw("Erreur, numero est inférieur à 100");
  }
 },
 toString: function(){
  return "Personne : ";
 }
  });

  var dvjh = new Personne("Daniel", "Hagnoul", 999);
```

Utilisation d'un mutateur (poopj2.html)

```
  console.log(dvjh);

  dvjh.setNumero(-1);
}
catch(err){
  alert(err);
}
```

Dans l'exemple précédent, le mutateur setNumero(value) teste la validité du numéro et déclenche une exception en cas d'erreur.

Le constructeur fait appel au mutateur, this.setNumero(numero), et l'attribut this.numero est créé dans le mutateur.

Même si cela fonctionne, le fait de ne pas déclarer tous les attributs d'instance dans le constructeur déconcertera le prochain utilisateur de la "fonction classe" poopj.

L'utilisation d'une méthode statique, permet de déclarer l'attribut dans le constructeur.

Utilisation d'un mutateur et d'une méthode statique (poopj3.html)

```
try {
  var Personne = $.dvjhClass._create({
 _builder: function(prenom, nom, numero){
  this.prenom = prenom || "";
  this.nom = nom || "";
  this.numero = Personne.TestNumero(numero);
 },
 setNumero: function(value){
  this.numero = Personne.TestNumero(value);
 },
 toString: function(){
  return "Personne : ";
 }
  });

  /*
  * Méthode statique
  */
  Personne.TestNumero = function(value){
 var n = parseInt(value, 10);

 if (!isNaN(n) && n >= 100){
  return n;
 } else {
  throw("Erreur, numero est inférieur à 100");
   }
  };

  /*
   * Vous pouvez contrôler le contenu du prototype
   * de la fonction classe poopj.
   *
   * La méthode statique est extérieure au prototype.
   */
  console.log(Personne, Personne.prototype);

  var dvjh = new Personne("Daniel", "Hagnoul", 999);

  console.log(dvjh, dvjh.numero);

  dvjh.setNumero(-1);
}
catch(err){
  alert(err);
}
```

IV - Rendre une "fonction classe" poopj abstraite

On peut rendre une "fonction classe" non instanciable en passant l'option _abstract à true.

L'option _abstract

```
try {
  var funcClass = $.dvjhClass._create( {

    // contenu du premier objet anonyme

  },
  {

 /*
  * contenu du second objet anonyme
  *
  * $.dvjhClass définit les options : _abstract,
  * _auteur, _copyright,  et _version
  * mais il n'utilise que la première.
  * Vous pouvez modifier les autres à votre
  * convenance et en utiliser d'autres.
  */

 _abstract: true

  } );
}
catch(err){
 alert(err);
}
```

Cette possibilité ne doit pas être négligée :

1 On peut écrire une "fonction classe" abstraite, c'est-à-dire une "fonction classe" qui n'implémente pas toutes les méthodes qui sont nécessaires à son fonctionnement. C'est la "fonction classe" fille (la classe qui hérite) qui a la responsabilité de fournir l'implémentation des méthodes manquantes ;
2 On peut écrire l'équivalent d'une interface en écrivant une "fonction classe" abstraite qui ne contient qu'un squelette, c'est-à-dire des attributs non initialisés et des méthodes non implémentées ;
3 On peut écrire l'équivalent d'une "fonction classe" Math, c'est-à-dire une "fonction classe" non instanciable et constituée entièrement d'attributs et de méthodes statiques.

Une fonction classe abstraite (poopj4.html)

```
try {
  var Personne = $.dvjhClass._create({
 _builder: function(prenom, nom){
  this.prenom = prenom || "";
  this.nom = nom || "";

  /*
   * On doit enregistrer l'adresse de chaque
   * instance de la fonction classe Personne.
   *
   * Mais on délègue cette responsabilité à
   * la classe qui héritera de celle-ci.
   */
  this.adresse = ""; // valeur non disponible
 },
 getAdresse: function(){
  return this.adresse;  // valeur non disponible
 },
 setAdresse: function(value){
  // implémentation non disponible
 },
 toString: function(){
  return "Personne : ";
```

Une fonction classe abstraite (poopj4.html)

```
  }
   },
   {
  _abstract: true

   });

   /*
    * Dans le cas d'une classe abstraite on choisit de
    * retourner un objet vide (new Object) plutôt que
    * de déclencher une exception, car dans ce cas
    * l'opérateur new construit un objet invalide.
    */
   var dvjh = new Personne("Daniel", "Hagnoul");

   console.log(typeof dvjh, dvjh);
}
catch(err){
   alert(err);
}
```

Une fonction classe interface

```
try {
   var IEtudiant = $.dvjhClass._create({
  _builder: function(prenom, nom, adresse, numero,
                     classe, interne){
   this.prenom = "";
   this.nom = "";
   this.adresse = "";
   this.numero = "";
   this.classe = "";
   this.interne = "";
  },
  getPrenom: function(){},
  getNom: function(){},
  getAdresse: function(){},
  getNumero: function(){},
  getClasse: function(){},
  getInterne: function(){},
  setPrenom: function(value){},
  setNom: function(value){},
  setAdresse: function(value){},
  setNumero: function(value){},
  setClasse: function(value){},
  setInterne: function(value){},
  toString: function(){
   return "IEtudiant : ";
  }
   },
   {
  _abstract: true
   });
}
catch(err){
   alert(err);
}
```

Une fonction classe statique

```
try {
   var PIERRE = $.dvjhClass._create({
  _builder: function(){
   // même vide, le constructeur doit toujours exister
  },
  toString: function(){
   return "PIERRE : ";
  }
   },
```

Une fonction classe statique

```
  {
 _abstract: true
  });

  PIERRE.prenom = "Pierre";
  PIERRE.nom = "Dupond";
  PIERRE.rue = "27 Rue de la Joie";
  PIERRE.ville = "Tristesseville";
  PIERRE.fonction = "Directeur exécutif";
  PIERRE.telephone = "+32 844 256 612";
  PIERRE.email = "direction@supertrust.com";
  PIERRE.vacances = function(date){
 // retourne true si PIERRE est en vacances à cette date
 return AgendaPierre("vacances", date);
  };
  PIERRE.rendezvous = function(type, visiteurs, date){
 // retourne true si PIERRE accepte ce rendez-vous
 return AgendaPierre(type, visiteurs, date);
  }

  console.log(PIERRE, PIERRE.prototype);
}
catch(err){
  alert(err);
}
```

V - Tout est public, on ne peut donc rien protéger ?

Certes, en JavaScript tout est objet et tout est public, mais comme en OOP classique une fonction classe poopj peut avoir dans son constructeur des variables et des méthodes privées ou protégées.

Cette possibilité est très importante pour renforcer la sécurité d'une fonction classe. Mais on ne doit pas en abuser, car elle impacte les performances et la consommation mémoire si la fonction classe est instanciée à de multiples reprises. En effet, les variables et les méthodes privées ou protégées sont initialisées par le constructeur.

Des variables et des méthodes privées ou protégées (poopj5.html)

```
try {
  var Limite = $.dvjhClass._create({
 _builder: function(nombre){
  this.nombre = parseInt(nombre, 10) || 0;

  /*
   * Limite est une valeur privée, elle n'est
   * utilisable qu'à l'intérieur du constructeur.
   */
  var limite = 9999;

  /*
   * Multiplicateur est une valeur privée qui deviendra une
   * valeur protégée par l'intermédiaire d'un accesseur
   * et d'un mutateur.
   */
  var multiplicateur = 2;

  /*
   * L'accesseur et le mutateur sont des fonctions protégées.
   * On peut accéder à une fonction protégée, on peut
         * la remplacer par une autre du même nom, mais on ne
         * peut pas altérer son contenu.
   */
  this.getMultiplicateur = function(){
   return multiplicateur;
  }

  this.setMultiplicateur = function(value){
```

Des variables et des méthodes privées ou protégées (poopj5.html)

```
    var n = parseInt(value, 10);

    if (n > 0 && n < 6){
     multiplicateur = n;
    } else {
     throw("Erreur, tentative d'assigner une valeur
                hors limites (1 à 5 inclus) au multiplicateur.")
    }
   }

   /*
    * Une fonction privée.
    *
    * Rappel, dans une fonction privée this est windows.
    */
   var self = this;

   var calcul = function(){
    var n = self.nombre * multiplicateur;

    if (n > limite){
     n = limite;
    } else if (n < 0){
     n = 0;
    }

    return n;
   }

   /*
    * Résultat du calcul pour this.nombre
    *
    * N'écrivez jamais "this.resultat = calcul();"
    * sinon n'importe qui, n'importe où, peut
    * écrire : "this.resultat = 120000;"
    */
   this.getResultat = function(){
    return calcul();
   }
  },
  toString: function(){
   return "Limite";
  }
   });

   var limite = new Limite(17);

   console.log(limite, limite.getResultat());

   limite.setMultiplicateur(5);

   console.log(limite, limite.getResultat());
}
catch(err){
   alert(err);
}
```

VI - Exemple, le patron de conception observateur

Mise en oeuvre du patron de conception observateur (poopj6.html)

```
try {
   /*
    * Mise en oeuvre du patron de conception observateur.
    *
    * Voir le tutoriel de Thierry Templier, troisième
    * partie page 13
    */
```

Mise en oeuvre du patron de conception observateur (poopj6.html)

```
   function enregistrerObservateur(objet, methode, observateur){
  var objMethode = objet[methode];

  objet["__" + methode] = objMethode;

  objet[methode] = function(){
   observateur();
   objMethode.apply(this, arguments);
  }
   }

   var Personne = $.dvjhClass._create({
  _builder: function(prenom, nom){
   this.prenom = prenom;
   this.nom = nom;
  },
  adresse: function(rue, ville){
   var strN = this.prenom + " " + this.nom;
   var strV = rue + ", " + ville;

   console.log(strN, strV);
  }
   });

   var dvjh = new Personne("Daniel", "Hagnoul");

   enregistrerObservateur(dvjh, "adresse", function(){
  dvjh.nom = "DVJH";
   });

   dvjh.adresse("Chaussée de Tongres 207", "Bavais");

   console.log(dvjh);
}
catch(err){
   alert(err);
}
```

VII - "Fonction classe" poopj, un exemple plus élaboré

On construit un système du genre MVC :

- M pour les objets du modèle ;
- V pour l'objet responsable de l'affichage des résultats ;
- C pour l'objet qui contrôle le programme.

Table est la "fonction classe" de type V. L'objet instancié affichera la table à la demande, elle contiendra les informations qui lui auront été communiquées par l'objet C.

Pair est une "fonction classe" de type M. L'objet instancié communiquera à l'objet C le résultat de son calcul.

Impair est une autre "fonction classe" de type M. L'objet instancié communiquera à l'objet C le résultat de son calcul.

Total est la "fonction classe" de type C. L'objet instancié jouera le rôle d'initiateur et de contrôleur du programme. Il crée une instance des objets M et de l'objet V.

L'objet C reçoit les résultats envoyés par les objets M, il les traite et envoie le résultat à l'objet V. Lorsque le temps est venu, il demande à l'objet V d'afficher les résultats.

Comme il se doit, les "fonctions classes" poopj sont définies dans un fichier JavaScript indépendant.

Contenu du fichier JS (poopj7.js)

Contenu du fichier JS (poopj7.js)

```
try {
  /*
   * On construit un système type MVC.
   * M pour les objets du modèle.
   * V pour l'objet responsable de l'affichage des résultats.
   * C pour l'objet qui contrôle le programme.
   *
   * Table est la fonction classe V.
   *
   * L'objet instancié affichera la table à la demande, elle
   * contiendra les informations qui lui auront été communiquées
   * par l'objet C.
   */
  var Table = $.dvjhClass._create({
    _builder: function(){
        this.eventName = "tableEvent.Table";
        this.th = null;
        this.trs = [];

        // bonne pratique car jQuery peut modifier this
        var self = this;

        $(this).bind(self.eventName, self.tableEventHandler);
    },
    tableEventHandler: function(event){
        if (event.dvjh && event.dvjh.length == 1){
            if (this.th == null) {
                this.th = event.dvjh.p1Name;
            }

            this.trs.push(event.dvjh.p1Data);
        }

        return false;
    },
    print: function(id){
        // suppprime une éventuelle table précédente
        $("#"+id).find(".tablesorter").remove();

        if (this.th != null && this.trs[0] != null){
            var self = this;
            var t0 = this.trs[0][3];
            var tableID = "vue" + t0;
            var deltaT = [];
            var n = this.trs.length;

            deltaT.push(0);

            for(var i = 1; i < n; i++){
                deltaT.push(this.trs[i][3] - t0);
            }

            var tdlength = this.th.length;
            var trlength = this.trs.length;

            var array = ["<table id='" + tableID +
                         "' class='tablesorter'>"];

            var arrayHead = ["<thead><tr>"];

                for(var i = 0; i < tdlength; i++) {
                    arrayHead.push("<th>" + self.th[i] + "</th>");
                }

            arrayHead.push("<th>Delta T ms</th></tr></thead>");

            var arrayFoot = ["<tfoot><tr>"];

                for(var i = 0; i < tdlength; i++) {
                    arrayFoot.push("<th>" + self.th[i] + "</th>");
```

Contenu du fichier JS (poopj7.js)

```
                }

            arrayFoot.push("<th>Delta T ms</th></tr></tfoot>");

            array.push(arrayHead.join(''), arrayFoot.join(''),
                       "<tbody>");

                for(var j = 0; j < trlength; j++) {
                    array.push("<tr>");

                    for(var i = 0; i < tdlength; i++) {
                        array.push("<td>" + self.trs[j][i] + "</td>");
                    }

                    array.push("<td>" + deltaT[j] + "</td></tr>");
                }

            array.push("</tbody></table>");

            $("#"+id).append(array.join(''));

            $("#"+tableID).tablesorter({
                sortList: [[3,0]],
                widgets: ['zebra'],
                headers: {
                    4:{sorter: false}
                }
            });
        } else {
            $("#"+id).html("<p>Il n'y a aucune information" +
              " à afficher !</p>");
        }
    },
    toString: function(){
        return "Table : ";
    }
});

/*
 * Pair est une fonction classe M.
 *
 * L'objet instancié communiquera à l'objet C
 * le résultat de son calcul.
 */
var Pair = $.dvjhClass._create({
    number: 0,
    _builder: function(number){
        this.number = number || this.number;
    },
    add: function(obj){
        this.number++;

        if ((this.number % 2 == 0) && (obj.eventName)){
            var objEvent = new $.Event(obj.eventName);

            /*
             * On adopte la convention suivante :
             * event.dvjh.length donne le nombre de params : p1,
             * p2, etc.
             * event.dvjh.p1Name donne le(s) libellé(s) du param p1
             * event.dvjh.p1Data contient la/les donnée(s) du
             * param p1.
             */
            objEvent.dvjh = {
                length: 1,
                p1Name: "Pair",
                p1Data: this.number
            };

            $(obj).trigger(objEvent);
        }
```

Contenu du fichier JS (poopj7.js)

```
    },
    toString: function(){
        return "Pair : ";
    }
});

/*
 * Impair est une fonction classe M.
 *
 * L'objet instancié communiquera à l'objet C
 * le résultat de son calcul.
 */
var Impair = $.dvjhClass._create({
  number: 0,
  _builder: function(number){
      this.number = number || this.number;
  },
  add:function(obj){
      this.number++;

      if ((this.number % 2 != 0) && (obj.eventName)){
          var objEvent = new $.Event(obj.eventName);

          objEvent.dvjh = {
              length: 1,
              p1Name: "Impair",
              p1Data: this.number
          };

          $(obj).trigger(objEvent);
      }
  },
  toString: function(){
      return "Impair : ";
  }
});

/*
 * Total est la fonction classe C.
 *
 * L'objet instancié jouera le rôle d'initiateur et de contrôleur
 * du programme. Il crée une instance des objets M et de l'objet V.
 *
 * Il questionne les objets M et stockent leurs
 * résultats dans l'objet V. Lorsque le temps est venu
 * il demande à l'objet V d'afficher les résultats.
 */
var Total = $.dvjhClass._create({
  _builder: function(milliseconds, divID){
      this.eventName = "totalEvent.Total";
      this.number = 0;
      this.source = "";
      this.total = 0;
      this.timeStamp = 0;
      this.output = new Table();
      this.pair = new Pair();
      this.impair = new Impair();
      this.milliseconds = milliseconds || 1000;
      var self = this;

      $(this).bind(self.eventName, self.totalEventHandler);

      var pairInterval  = window.setInterval(function(){
          self.pair.add(self);
      }, 30);

      var impairInterval = window.setInterval(function(){
          self.impair.add(self);
      }, 40);

      window.setTimeout(function(){
```

Contenu du fichier JS (poopj7.js)

```
            window.clearInterval(pairInterval);
            window.clearInterval(impairInterval);
            self.output.print(divID);
        }, this.milliseconds);
    },
    totalEventHandler: function(event){
        if (event.dvjh && event.dvjh.length == 1){
            this.number = event.dvjh.p1Data;
            this.source = event.dvjh.p1Name;
            this.total += this.number;
            this.timeStamp = event.timeStamp;

            if (this.output){
                var self = this;

                var outputEvent = new $.Event(self.output.eventName);

                outputEvent.dvjh = {
                    length: 1,
                    p1Name: ["Total", "Nombre", "Source",
                             "TimeStamp ms"],
                    p1Data: [this.total, this.number, this.source,
                             this.timeStamp]
                };

                $(this.output).trigger(outputEvent);
            }
        }

        return false;
    },
    toString: function(){
        return "Total : ";
    }
  });
}
catch(err){
  alert(err);
}
```

Contenu de la page web (poopj7.html)

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="Author" content="Daniel Hagnoul">
  <title>POOPJ</title>
  <style>
 /* BASE */
 body {
  background-color:#dcdcdc;
  color:#000000;
  font-family:sans-serif;
  font-size:medium;
  font-style:normal;
  font-weight:normal;
  line-height:normal;
  letter-spacing:normal;
 }
 h1,h2,h3,h4,h5 {
  font-family:serif;
 }
 div,p,h1,h2,h3,h4,h5,h6,ul,ol,dl,form,table,img {
  margin:0px;
  padding:0px;
 }
 p {
  padding:6px;
 }
```

Contenu de la page web (poopj7.html)

```
ul,ol,dl {
 list-style:none;
 padding-left:6px;
 padding-top:6px;
}
li {
 padding-bottom:6px;
}

/* dvjh */
h1 {
 text-align:center;
 font-style:italic;
 text-shadow: 4px 4px 4px #bbbbbb;
}
h2 {
 text-align:center;
}
div#conteneur {
 width:95%;
 height:auto;
 margin:12px auto;
 padding:6px;
 background-color:#FFFFFF;
 color:#000000;
 border:1px solid #666666;
}
div#affiche {
 clear:both;
 margin:40px;
 padding:6px;
 border:1px solid #999999;
 background-color:#FFFFFF;
 color:#000000;
}

/* TABLE */
table.dvjhTable {
 width:95%;
 margin:12px auto;
 /* table-layout = problèmes */
 empty-cells:show;
 border-collapse:collapse;
 border-spacing:0px;
 border-width:1px;
 border-style:solid;
 border-color:#666666;
 background-color:#CDCDCD;
 color:#000000;
 font-size:0.9em;
 font-style:normal;
 font-weight:normal;
 line-height:normal;
 letter-spacing:normal;
}
table.dvjhTable caption {
 caption-side:top;
 padding-top:6px;
 padding-bottom:6px;
 text-align:center;
 font-size:1.2em;
 font-style:oblique;
 font-weight:bold;
 line-height:normal;
 letter-spacing:0.2em;
 color:#000000;
}
table.dvjhTable thead tr th {
 border-width:1px;
 border-style:solid;
 border-color:#999999;
```

Contenu de la page web (poopj7.html)

```
 background-color: #e6eeee;
 color:#000000;
 padding:6px;
 text-align:center;
 font-size:0.9em;
 font-style:normal;
 font-weight:bold;
 line-height:1.8em;
 letter-spacing:normal;
}
table.dvjhTable tfoot tr th {
 border-width:1px;
 border-style:solid;
 border-color:#999999;
 background-color: #e6eeee;
 color:#000000;
 padding:6px;
 text-align:left;
 font-size:0.9em;
 font-style:italic;
 font-weight:normal;
 line-height:1.8em;
 letter-spacing:normal;
}
table.dvjhTable thead tr .header {
 background-image:url(../images/bg.gif);
 background-repeat: no-repeat;
 background-position:right;
 cursor: pointer;
 min-width:80px;
}
table.dvjhTable thead tr .headerSortUp {
 background-image: url(../images/asc.gif);
}
table.dvjhTable thead tr .headerSortDown {
 background-image: url(../images/desc.gif);
}
table.dvjhTable thead tr .headerSortDown, table.dvjhTable thead tr .headerSortUp {
 background-color: #8dbdd8;
}
table.dvjhTable tbody th {
 border-width:1px;
 border-style:solid;
 border-color:#999999;
 background-color:#e6eeee;
 color:#000000;
 vertical-align:middle;
 padding:6px;
 text-align:left;
 font-size:0.9em;
 font-style:normal;
 font-weight:normal;
 line-height:1.2em;
 letter-spacing:normal;
}
table.dvjhTable tbody td {
 border-width:1px;
 border-style:solid;
 border-color:#999999;
 background-color:#FFFFFF;
 color:#000000;
 vertical-align:middle;
 padding:6px;
 text-align:left;
 font-size:0.9em;
 font-style:normal;
 font-weight:normal;
 line-height:1.2em;
 letter-spacing:normal;
}
table.dvjhTable tbody tr.dvjhTableOdd td {
```

Contenu de la page web (poopj7.html)

```
   background-color:#FDFFD9;
  }

  /* TEST */
   </style>
   <script charset="utf-8"
           src="../lib/jqueryui/js/jquery-1.4.2.min.js"></script>
   <script charset="utf-8" src="../lib/dvjh/Tablesorter.js"></script>
   <script charset="utf-8" src="../lib/dvjh/poopj.js"></script>
   <script charset="utf-8" src="poopj7.js"></script>
   <script>
  $(function(){
    try {
   /*
    * L'utilisateur peut appeler plusieurs fois le programme,
    * mais il ne doit y avoir qu'un objet contrôleur.
    *
    * L'utilisateur crée et initialise l'objet contrôleur en
    * cliquant sur le bouton Afficher.
    *
    * L'objet contrôleur se charge du bon fonctionnement du
          * programme et de l'affichage des résultats.
    *
    * L'affichage des résultats nécessite le plugin
          * jquery.tablesorter.min.js
    */
   var objTotal = null;

   $("#btn").click(function(){

    if (objTotal != null) {
     objTotal = null;
    }

    objTotal = new Total($("#choix").val(), "affiche");
   });
    }
    catch(err){
   alert(err);
    }
  });
   </script>
 </head>
 <body>
  <h1>POOPJ</h1>
  <div id="conteneur">
   <select id="choix">
    <option value="1000" selected="selected">1000 ms</option>
    <option value="1500">1500 ms</option>
    <option value="2000">2000 ms</option>
    <option value="2500">2500 ms</option>
    <option value="3000">3000 ms</option>
    <option value="3500">3500 ms</option>
    <option value="4000">4000 ms</option>
   </select>
   <button id="btn" type="button">Afficher</button>
   <div id="affiche"></div>
  </div>
 </body>
 </html>
```

VIII - Mise en oeuvre de l'héritage simple

Squelette d'une classe mére et de sa classe fille

```
try {
  var FuncClassMere = $.dvjhClass._create({
 _builder: function(){
  // le constructeur de la fonction classe mère
```

Squelette d'une classe mére et de sa classe fille

```
  },
  toString: function(){
   return "FuncClassMere : ";
  }
   });

   var FuncClassFille = $.dvjhClass._create(FuncClassMere, {
  _builder: function(){
   this._base(arguments.callee, FuncClassMere, arguments);

   // le constructeur de la fonction classe fille
  },
  toString: function(){
   return "FuncClassFille : ";
  }
   });
 }
 catch(err){
   alert(err);
 }
```

Pour la "fonction classe" fille, le premier argument de $.dvjhClass._create() devient la "fonction classe" poopj mère et le corpus de la "fonction classe" fille devient le second argument de la méthode _create().

La première ligne du constructeur d'une "fonction classe" fille doit toujours être this._base(arguments.callee, FuncClassMere, arguments).

L'insertion de FuncClassMere comme premier argument de la méthode _create(), permet l'héritage du prototype de la "fonction classe" mère.

En utilisant "arguments.callee", on permet à une fonction anonyme de se référencer elle-même, on peut ainsi déclencher l'initialisation récursive des "fonctions classes" mères.

L'existence de this._base() est une contrainte acceptable en regard des avantages qu'il procure. J'ai essayé d'autres solutions, mais elles étaient toutes plus complexes à mettre en oeuvre au niveau de l'objet $.dvjhClass et plus coûteuses pour l'objet instancié et son prototype.

Mise en oeuvre de l'héritage simple (poopj8.html)

```
try {
   var Personne = $.dvjhClass._create({
  _builder: function(prenom, nom){
   this.prenom = prenom;
   this.nom = nom;
  },
  toString: function(){
   return "Personne : ";
  }
   });

   var Etudiant = $.dvjhClass._create(Personne, {
  _builder: function(prenom, nom, numero){
   this._base(arguments.callee, Personne, arguments);
   this.numero = numero;
  },
  toString: function(){
   return "Etudiant : ";
  }
   });

   var Chanteur = $.dvjhClass._create(Personne, {
  _builder: function(prenom, nom, voix){
   this._base(arguments.callee, Personne, arguments);
   this.voix = voix;
  },
```

Mise en oeuvre de l'héritage simple (poopj8.html)

```
  toString: function(){
   return "Chanteur : ";
  }
   });

   var Interne = $.dvjhClass._create(Etudiant, {
  _builder: function(prenom, nom, numero, chambre){
   this._base(arguments.callee, Etudiant, arguments);
   this.chambre = chambre;
  },
  toString: function(){
   return "Interne : ";
  }
   });

   var Tenor = $.dvjhClass._create(Chanteur, {
  _builder: function(prenom, nom, voix){
   this._base(arguments.callee, Chanteur, arguments);
   this.ton = "grave";
  },
  toString: function(){
   return "Tenor : ";
  }
   });

   /*
    * L'ordre des arguments du constructeur dans une fonction classe
    * fille doit bien entendu être rigoureusement identique à celui
    * de la fonction classe mère sous peine d'erreurs dans
    * l'initialisation des attributs de la classe mère.
    */
   var moi = new Personne("Daniel", "Hagnoul");
   var etudiant = new Etudiant("Pierre", "Dupond", 111);
   var chanteur = new Chanteur("Henri", "Latour", "soprano");
   var interne = new Interne("Jaques", "Lemaitre", 111, "royale");
   var tenor = new Tenor("Gustave", "Lefort", "tenor");

   console.log(moi);
   console.log(etudiant);
   console.log(chanteur);
   console.log(interne);
   console.log(tenor);
}
catch(err){
   alert(err);
}
```

La technique d'héritage mise en oeuvre ne permet pas le polymorphisme (objX instanceof objY), mais c'est la plus appropriée dans la plupart des cas. Si le polymorphisme et l'héritage multiple vous manquent, le tutoriel de Thierry Templier décrit d'autres techniques d'héritage.

L'objet _super stocke une copie des méthodes du prototype de la "fonction classe" mère.

Appel d'une méthode de la fonction classe mère (poopj9.html)

```
try {
   var Personne = $.dvjhClass._create({
  _builder: function(prenom, nom){
   this.prenom = prenom || "";
   this.nom = nom || "";
   this.adresse = "Belgique";
  },
  getAdresse: function(){
   return "J'habite en " + this.adresse;
  },
  toString: function(){
   return "Personne";
  }
```

Appel d'une méthode de la fonction classe mère (poopj9.html)

```
  });

  var Etudiant = $.dvjhClass._create(Personne, {
 _builder: function(prenom, nom, numero){
  this._base(arguments.callee, Personne, arguments);
  this.numero = numero || -1;
 },
 getAdresse: function(){
  /*
   * Les méthodes de l'objet _super doivent toujours s'exécuter
   * dans le contexte de l'objet appelant.
   *
   * On peut utiliser call(this, param1, param2, etc.)
         * ou apply(this, arguments)
   */
  return this._super.getAdresse.call(this) +
               ". Cette adresse est valable !\n";
 },
 toString: function(){
  return "Etudiant";
 }
  });

  var dvjh = new Personne("Daniel", "Hagnoul");
  console.log(dvjh.getAdresse());

  var pierre = new Etudiant("Pierre", "Dubois", 427);
  console.log(pierre.getAdresse());

  console.log(dvjh, pierre);

  console.log(pierre._super.getAdresse.call(pierre));
}
catch(err){
  alert(err);
}
```

La méthode toString() et l'objet _super peuvent être utilisés dans le code d'une "fonction classe" ou par l'objet instancié.

On peut redéfinir les options _auteur, _copyright et _version mais on ne doit pas toucher à l'option _abstract. On peut créer et utiliser d'autres options.

Toucher à _create(), _builder() ou _base() ne conduira qu'à la catastrophe.

IX - Exemple, le patron de conception template

Mise en oeuvre du patron de conception template (poopj10.html)

```
try {
  /*
   * Mise en oeuvre du patron de conception template.
   *
   * Voir le tutoriel de Thierry Templier, troisième
   * partie page 13
   */
  var ParcoursListe = $.dvjhClass._create({
 _builder: function(){
  // Même vide, le _buider doit toujours exister !
 },
 creerSousListe: function(liste){
  var n = liste.length;
  var nouvelleListe = [];

  for (var indice = 0; indice < n; indice++){
   var valeur = liste[indice];
```

Mise en oeuvre du patron de conception template (poopj10.html)

```
    var nouvelleValeur = this.traiterElement(indice, valeur);

    if (nouvelleValeur != null){
     nouvelleListe.push(nouvelleValeur);
    }
   }

   return nouvelleListe;
  }
   },
   {
  _abstract: true
   });

   var FiltrageListe = $.dvjhClass._create(ParcoursListe, {
  _builder: function(valeurMinimum){
   this._base(arguments.callee, ParcoursListe, arguments);
   this.valeurMinimum = valeurMinimum;
  },
  traiterElement: function(indice, valeur){
   if (valeur >= this.valeurMinimum){
    return valeur;
   }
  }
   });

   var filtrage = new FiltrageListe(10);

   var liste = [10, 1, 4, 13, 14];

   var nouvelleListe = filtrage.creerSousListe(liste);

   console.log(nouvelleListe);
}
catch(err){
   alert(err);
}
```

X - Héritage simple, un exemple plus élaboré

(5) Je me suis inspiré des tutoriels : "Créer une table HTML éditable" et "Créer une table HTML éditable - HtmlEditTable v2.0" de Nourdine FALOLA. Je dis inspiré, car je n'ai pas repris toutes les possibilités, ni implémenté les mêmes codes.

Téléchargement de table-poopj.zip

L'exemple se compose de six "fonctions classes" : TABLE, SORTTABLE, DvjhTable, DvjhSortTable, DvjhInput et DvjhEdit.

TABLE et SORTTABLE sont deux "fonctions classes" abstraites avec des attributs et des méthodes statiques, ce qui permet de personnaliser, adapter ou modifier le comportement d'une table sans toucher au code des "fonctions classes" tables.

On peut construire une table simple (DvjhTable) ou une table triable (DvjhSortTable, utilise le plug-in tablesorter. Une version de tablesorter très légèrement modifiée pour prendre en compte la virgule et la date dd/mm/yyyy).

En utilisant DvjhEdit -- new DvjhEdit(meTable); (DvjhEdit crée une instance de DvjhInput) -- on peut éditer les cellules d'une table, la valeur n'est modifiée que si elle passe au travers d'un test de cohérence.

XI - Le point délicat du système poopj

Les habitués des objets JSON auront remarqué que je n'ai pas parlé de la possibilité d'ajouter des attributs dans l'objet anonyme.

Je déconseille fortement l'usage de cette possibilité aux nouveaux utilisateurs de l'outil, car elle est très dangereuse.

Dangereuse pour deux raisons :

1 On risque d'assimiler, de considérer comme analogue, les attributs du _builder et ceux de l'objet anonyme. Certes, l'objet instancié aura accès aux deux de la même manière, mais ceux du _builder sont initialisés lors de l'instanciation et les autres ne le sont pas ;

2 Lorsqu'un attribut de l'objet anonyme contient du texte ou un nombre, on peut ne s'apercevoir de rien, mais s'il contient un objet (rappel : un Array est un objet) toutes les instances de la "fonction classe" font référence au même objet.

Les attributs de l'objet anonyme (poopj11.html)

```
try {
  /*
   * Si vous définissez des attributs pour l'objet
   * anonyme, soyez prévenus des conséquences.
   *
   * Lorsque l'attribut est une objet il est
   * unique pour toutes les instances.
   */
  var Personne = $.dvjhClass._create({
 pays: "France",
 tab: [],
 obj: {},
 _builder: function(prenom, nom){
  this.prenom = prenom || "";
  this.nom = nom || "";
 },
 toString: function(){
  return "Personne : ";
 }
  });

  var dvjh = new Personne("Daniel", "Hagnoul");
  dvjh.pays = "Belgique";
  dvjh.obj["secret"] = 10;
  dvjh.tab.push(5);

  var pierre = new Personne("Pierre", "Durand");
  pierre.pays = "Findland";
  pierre.obj["valeur"] = function(){return 267;};
  pierre.tab.push(6);

  console.log("dvjh et pierre pays : ", dvjh.pays, pierre.pays);
  console.log("dvjh et pierre tab : ", dvjh.tab, pierre.tab);
  console.log("dvjh et pierre obj : ", dvjh.obj, pierre.obj);
  console.log("dvjh et pierre obj.valeur : ",
              dvjh.obj.valeur(), pierre.obj.valeur());

  var Etudiant = $.dvjhClass._create(Personne, {
 _builder: function(prenom, nom, numero){
  this._base(arguments.callee, Personne, arguments);
  this.numero = numero || -1;
 },
 toString: function(){
  return "Etudiant : ";
 }
  });

  var etudiant = new Etudiant("Jean", "Dupond", 214);
```

Les attributs de l'objet anonyme (poopj11.html)

```
   console.log("etudiant pays", etudiant.pays);
   console.log("etudiant tab", etudiant.tab);
   console.log("etudiant obj", etudiant.obj);
   console.log("etudiant obj.valeur", etudiant.obj.valeur());
}
catch(err){
   alert(err);
}
```

Mais la possibilité pour toutes les instances de stocker des valeurs dans un même objet peut également être un plus.

Les attributs de la clôture jQuery (poopj12.html)

```
try {
  /*
   * Si vous maîtrisez les attributs publics de l'objet anonyme,
   * vous êtes qualifié pour jouer avec la clôture jQuery.
   *
   * Attention, tout est commun à toutes les instances !!!
   */
  var Personne = (function($){
 var pays = "France";
 var tab = [];
 var obj = {};

 var hello = function(){
  return "Bonjour ! " + pays;
 };

 return $.dvjhClass._create({
  _builder: function(prenom, nom){
   this.prenom = prenom || "";
   this.nom = nom || "";
  },
  getPays: function(){
   return pays;
  },
  setPays: function(value){
   pays = value;
  },
  getObj: function(value){
   if (typeof value === "string"){
    return obj[value];
   }

   return obj;
  },
  setObj: function(nom, value){
   obj[nom] = value;
  },
  getTab: function(value){
   if (typeof value === "number"){
    return tab[value];
   }
   return tab;
  },
  setTab: function(value){
   tab.push(value);
  },
  print: function(){
   return hello();
  },
  toString: function(){
   return "Personne : ";
  }
 });
  })(jQuery);

  var dvjh = new Personne("Daniel", "Hagnoul");
```

Les attributs de la clôture jQuery (poopj12.html)

```
  console.log(dvjh);

  dvjh.setPays("Belgique");
  dvjh.setObj("secret", 10);
  dvjh.setTab(5);

  console.log(dvjh.getPays());
  console.log(dvjh.getObj());
  console.log(dvjh.getTab());
  console.log(dvjh.print());

  var pierre = new Personne("Pierre", "Durand");

  console.log(pierre);

  pierre.setPays("Findland");
  pierre.setObj("valeur", function(){return 267;});
  pierre.setTab(6);

  console.log(pierre.getPays());
  console.log(pierre.getObj());
  console.log(pierre.getObj("valeur"));
  console.log(pierre.getTab());
  console.log(pierre.print());

  var Etudiant = $.dvjhClass._create(Personne, {
 _builder: function(prenom, nom, numero){
  this._base(arguments.callee, Personne, arguments);
  this.numero = numero || -1;
 },
 toString: function(){
  return "Etudiant : ";
 }
  });

  var etudiant = new Etudiant("Jean", "Dupond", 214);

  console.log(etudiant.getPays());
  console.log(etudiant.getObj());
  console.log(etudiant.getTab());
  console.log(etudiant.print());

  // erreur, undefined !
  console.log(etudiant.pays);
}
catch(err){
  alert(err);
}
```

XII - Le prix à payer pour l'utilisation du système poopj

Les méthodes de l'objet $.dvjhClass sont présentent dans chaque objet instancié : _create(), _builder(), _base(), toString(), _options et en cas d'héritage l'objet _super() qui contient les méthodes originales de la classe mère.

La taille de l'objet _super dépend bien entendu du poids des méthodes de la classe mère, _builder() est le constructeur de la fonction classe poopj, le reste est très léger et l'impact de sa présence sur les performances est négligeable.

XIII - $.dvjhClass

$.dvjhClass est l'outil qui permet d'écrire du code poopj. C'est un objet JSON comprenant trois méthodes : _create(), _base() et toString(). L'outil permet de réunir dans un même corpus la fonction d'initialisation et son prototype.

(1) $.dvjhClass simplifie l'usage de la méthode d'héritage simple décrite par Thierry Templier au chapitre 1.4 de la deuxième partie de son tutoriel : "Programmation orientée objet avec le langage JavaScript".

Cette technique d'héritage ne permet pas le polymorphisme (objX instanceof objY), mais c'est la plus appropriée dans la plupart des cas. Si le polymorphisme et l'héritage multiple vous manquent, le tutoriel de Thierry Templier décrit d'autres techniques d'héritage.

La notion de classe n'existe pas en JavaScript, mais le besoin de qualifier la chose existant bien, je désigne le code généré par $.dvjhClass._create() sous le vocable "fonction classe poopj".

J'ai testé plusieurs ajouts à $.dvjhClass, mais je les ai tous abandonnés, ils n'apportaient rien de plus au système POOPJ et ils pesaient lourd dans l'occupation mémoire des objets instanciés.

Formaliser la notion d'interface, de propriété, d'accesseurs et de mutateurs en n'ayant pas de support pour ces notions dans les mécanismes du langage n'apporte qu'une satisfaction intellectuelle encombrante.

Tel qu'il est, je crois que l'objet $.dvjhClass est le meilleur compromis entre les services rendus et l'encombrement mémoire dû à son utilisation.

$.dvjhClass (lib/dvjh/poopj.js)

```
$.dvjhClass = {
  _create: function(){
    var classMere = null;
    var corpus = null;
    var options = {
        _abstract: false,
        _auteur: "Auteur : Daniel Hagnoul " +
        "(http://www.developpez.net/forums/u285162/danielhagnoul/)",
        _copyright: "Creative Commons License : " +
    "Attribution-Share Alike 2.0 Belgium " +
    "(http://creativecommons.org/licenses/by-sa/2.0/be/legalcode.fr)",
        _version: "$.dvjhClass, Daniel Hagnoul, v1.0.0 2010-08-16"
    }

    // si le premier argument est une fonction
    if ($.isFunction(arguments[0])){
        classMere = arguments[0];

        // il doit y avoir un objet comme deuxième argument
        if ($.isPlainObject(arguments[1])){
            corpus = arguments[1];
        } else {
            throw(dvjhExceptions.create1);
        }

        // il peut y avoir un objet comme troisième argument
        if ($.isPlainObject(arguments[2])){
            $.extend(options, arguments[2]);

            if (typeof options._abstract != "boolean"){
                options._abstract = true;
            }

        } else if (arguments[2]){
            throw(dvjhExceptions.create2);
        }
    } else if ($.isPlainObject(arguments[0])){
        // le premier argument est un objet
        corpus = arguments[0];

        // il peut y avoir un objet comme second argument
        if ($.isPlainObject(arguments[1])){
            $.extend(options, arguments[1]);

            if (typeof options._abstract != "boolean"){
```

$.dvjhClass (lib/dvjh/poopj.js)

```
                options._abstract = true;
            }

        } else if (arguments[1]){
            throw(dvjhExceptions.create3);
        }
    } else {
        // le premier argument n'est pas un objet
        throw(dvjhExceptions.create4);
    }

    if (!corpus._builder){
        throw(corpus + dvjhExceptions.corpus1);
    }

    if (classMere && !classMere.prototype._builder){
        throw(classMere + dvjhExceptions.classMere1);
    }

    // fonction à partir de laquelle le nouvel objet est instancié
    function build(){
        try {
            /*
             * Dans le cas d'une classe abstraite
             * ( _abstract == true) on choisit de retourner
             * un objet vide (new Object) plutôt que de
             * déclencher une exception, car dans ce cas
             * l'opérateur new construit un objet invalide.
             */
            if (this._options._abstract){
                return {};
            }

            this._builder.apply(this, arguments);
        }
        catch(err){
            alert(err);
        }
    }

    if (classMere){
        // incorporer classMere en premier
        $.extend(build.prototype, classMere.prototype);

        // conserver un accès aux méthodes originales de classMere
        build.prototype._super = {};

        for (var item in classMere.prototype){
            if (item.slice(0,1) != "_"){
                build.prototype._super[item] =
                                classMere.prototype[item];
            }
        }
    }

    // incorporer l'objet $.dvjhClass et le corpus
    $.extend(build.prototype, this, corpus);

    // incorporer les options
    build.prototype._options = options;

    // retourne la fonction à partir de laquelle
    // le nouvel objet est instancié
    return build;
},
_base: function(doNotTouchMe_YouIdiot, classMere, args){
    /*
     * doNotTouchMe_YouIdiot est arguments.callee
     *
     * arguments.callee est l'équivalent de this pour une
     * fonction anonyme
```

$.dvjhClass (lib/dvjh/poopj.js)

```
       *
       * il permet l'initialisation des classes mères par un
       * appel récursif
      */
      if (!classMere.prototype._builder){
          throw(classMere + dvjhExceptions.classMere1);
      }

      // initialisation de la classe mère
      classMere.prototype._builder.apply(this, args);
    },
    toString: function(){
      return this._options._version;
    }
  };
```

XIV - Remerciements

Je remercie très sincèrement les relecteurs **littledaem** et **jacques_jean** pour leur excellent travail.

1 : **Programmation orientée objet avec le langage JavaScript par Thierry Templier**
2 : Conception orientée objets et applications par Grady Booch, Addison-Wesley, 1992.
3 : **Bonnes pratiques objet en .net : Introduction aux principes SOLID par Philippe Vialatte**
4 : **Classical Inheritance in JavaScript par Douglas Crockford**
5 : **Créer une table HTML éditable par Nourdine FALOLA**
Créer une table HTML éditable - HtmlEditTable v2.0 par Nourdine FALOLA