Web (Persistance)

Andrea G. B. Tettamanzi
Université de Nice Sophia Antipolis
Département Informatique
andrea.tettamanzi@unice.fr

CM - Séance 12

Programmation Orientée Objet en JavaScript

Plan

- Objets, prototypes et héritage
- Encapsulation en JavaScript
- Constructeurs et création d’objets

Objets

- Un objet JavaScript peut être pensé comme un ensemble de couples (propriété, valeur) que l'on peut représenter avec la notation intuitive

  propriété : valeur
- qui est aussi utilisé par la syntaxe du langage pour définir des objets littéraux
- Propriétés
  - Propres : apparaissent explicitement dans un objet
  - Héritées : ne font pas explicitement partie de l'objet, mais sont dérivées implicitement d'autres objets (dits prototypes)
- Une propriété peut prendre des valeurs de n'importe quel type
- Les objets JavaScript sont des tables de hachage

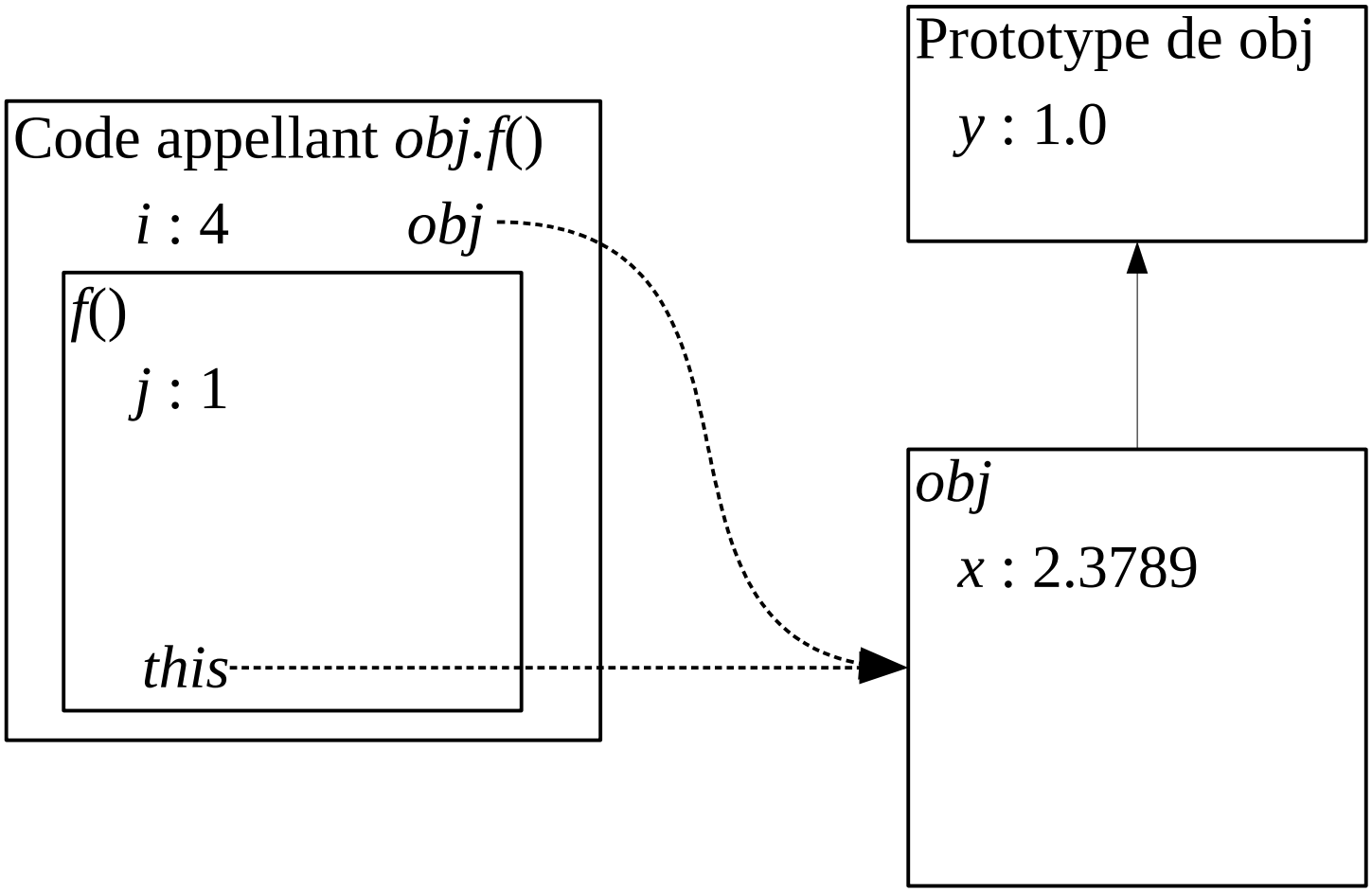
Méthodes

- Une propriété peut avoir une fonction comme valeur
- On parle alors d'une méthode
- Lorsqu'une méthode d'un objet est appelée, la variable spéciale « this » passe à référencer l'objet
- De ce fait, le code contenu dans la méthode peut accéder aux propriétés de l'objet à travers de la variable « this »

Portée lexicale

- Lorsqu'une fonction est appelée, son code est exécutée dans un « environnement lexical »
- Si la fonction est appelée comme méthode d'un objet, la variable « this » passe à référencer cet objet ; sinon, elle référence l'objet global
- Les identifiants définis dans les environnements lexicaux extérieurs (par exemple, celui du code qui a appelé la fonction) sont dans la portée

Portée lexicale



Encapsulation

- Principe fondamental de la POO
  - cacher les détails de la représentation des données
  - empêcher l'accès aux données par un autre moyen que des méthodes.
  - ⇒ l'interface n'expose jamais les données mais seulement des méthodes
- Certains langages POO (p.ex., C++, Java) disposent de limitateurs d'accès aux données et méthodes d'une classe :
  - Publique – accessible partout
  - Protégée – accessible par une sous-classe uniquement
  - Privée – accessible dans la classe elle-même uniquement

Encapsulation en JavaScript

- En JavaScript, les propriétés (et donc aussi les méthodes) d'un objet sont toutes des membres publiques
  - n'importe quelle portion de code peut y avoir accès et les modifier
- Cela n'exclue pas d'appliquer le principe d'encapsulation
  - Comme bonne pratique de programmation
  - En faisant preuve d'auto-discipline
  - Contournement toujours possible, mais à éviter
- Cette forme « primitive » d'encapsulation se base sur
  - Accesseurs (*getters*) : méthodes qui lisent un attribut
  - Mutateurs (*setters*) : méthodes qui modifient un attribut

Propriétés et méthodes privées

- Les variables ordinaires du constructeur peuvent être utilisées comme des membres privés des objets construits
  - ces variables restent attachées à l'objet,
  - mais ne sont pas visibles (= accessibles) à son extérieur
  - c’est le cas aussi pour les paramètres passés au constructeur
- De plus, elle ne sont pas visibles aux méthodes ordinaires de l’objet construit
- Pourtant, elle sont accessibles aux fonctions internes du constructeur
  - Ces fonctions sont aussi invisibles à l’extérieur
  - On pourrait les appeler « méthodes privées »

Exemple

```
function Conteneur(param) {
     function dec() {
          if (secret > 0) {
               secret -= 1;
               return true; }
          else return false; }
     this.a = param;
     var secret = 3;
     var that = this;
}
```

Méthodes privilégiées

- Une méthode privilégiée
  - a le droit d'accéder aux propriétés et méthodes privées
  - est accessible aux méthodes ordinaires
  - est accessible au code externe
- On peut effacer ou remplacer une méthode privilégiée
  - mais on ne peut pas la modifier
  - ou la forcer a révéler ses secrets
- Définition d'une méthode privilégiée : dans le constructeur,
  - on affecte une function interne du constructeur à une propriété publique (dans l'espace lexical de this)

Exemple

```
function Conteneur(param) {
  function dec() {
    if (secret > 0) {
      secret -= 1;
      return true; }
    else return false; }
  this.service = function () {
    return dec() ? that.a : null; };
  this.a = param;
  var secret = 3;
  var that = this;
}
```

Prototypes et héritage

- Chaque objet a un lien interne vers son prototype
- Un prototype est un objet et peut avoir un prototype à son tour
- On parle alors de chaîne de prototypes d'un objet
- Étant donné un objet o, la valeur o.x d’une propriété x est déterminée comme suit :
  - si x est une propriété propre de o, o.x renvoie la valeur de la propriété x en o ;
  - sinon, la chaîne des prototypes de o est parcourue jusqu’au premier prototype p où x est définie ; dans ce cas, o.x renvoie la valeur de p.x ;
  - si, en parcourant la chaîne des prototypes, l’objet « null » est atteint, o.x renvoie « undefined », la valeur indéfinie.

Création d'objets

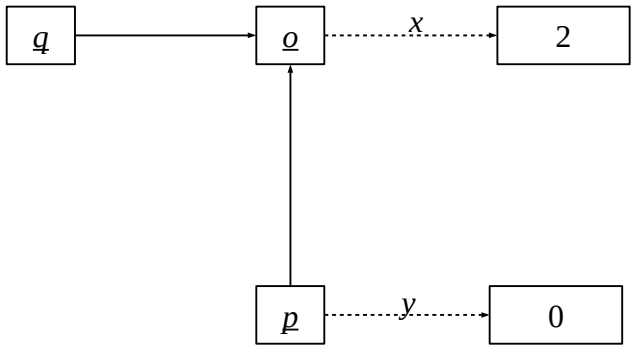
- Un objet peut être créé de plusieurs façons, parmi lesquelles
  - Syntaxiquement, à l'aide d'une notation littérale :

    obj = { x : 5, y : 10 } ;
  - Par des constructeurs
    - Un constructeur, en JavaScript, est une fonction qui est appelée avec l'opérateur « new »

      new f(…) ;
    - Renvoie un nouvel objet moulé sur l'objet f.prototype
  - Per Object.create(*proto*).

Exemple

```
o = { x : 2 };
function f() { … }
f.prototype = o;
p = new f();
p.y = 0;
q = new f();
```



Merci de votre attention