IFT1015 Programmation 1

programmation orientée objets en Javascript

Pascal Vincent et Aaron Courville

Université de Montréal

Paradigme de programmation orienté objet

- Programmation impérative (ou procédurale): programme vu comme un ensemble de fonctions qui s'appellent mutuellement

- Programmation orientée objet: programme vu comme un ensemble d'objets qui communiquent (s'envoient des messages)

- Un objet peut communiquer avec un autre en appelant une méthode (fonction) associée à l'autre objet.

Objets

- Objet composite = structure = enregistrement
- (en JavaScript) = tableau associatif
- Un objet possède des propriétés nommées (aussi appelés attributs ou champs)
- La valeur d'une propriété (attribut) d'un objet peut être de n'importe quel type.
- L'ensemble des valeurs des attributs d'un objet constitue son état (qui peut changer)

```
var veh = { marque: "honda",age: 5 };
var coord = { x: 3, y: 5, z: 2*10 };
var naiss = { annee: 1995, mois: 12, quant: 9 };

print( veh.age ); // affiche 5
veh.age = veh.age+1;
print( veh.age ); // affiche 6
```

Objets

Exemple:

```
var mon_perroquet = { nom: "Coco",
                      age: 3,
                      altitude: 0 };
```

On aurait aussi pu écrire:

```
var mon_perroquet = {};

mon_perroquet.nom = "Coco";
mon_perroquet.age = 3;
mon_perroquet.altitude = 0;
```

Ou bien:

```
var mon_perroquet = new Object();

mon_perroquet.nom = "Coco";
mon_perroquet.age = 3;
mon_perroquet.altitude = 0;
```

Méthodes des objets

- En programmation objet les objets regroupent
  des attributs/champs/propriétés, et des fonctions (appelées méthodes) pour manipuler ce genre d'objet.
- Les méthodes accèdent ou modifient souvent les attributs de l'objet.

Définissons un «objet»:

```
var mon_perroquet = { nom: "Coco",
                      age: 3,
                      altitude: 0 };
```

Approche procédurale classique: fonctions globales

```
var repete = function(perroquet, phrase) {
  print( perroquet.nom + " dit " + phrase );   };

var envoleToi = function(perroquet, altitude){
  perroquet.altitude = altitude;  };

repete(mon_perroquet, "Bonjour les pirates!");
envoleToi(mon_perroquet, 20);
```

Déplaçons une fonction ***dans*** l'objet...

```
mon_perroquet.envoleToi = function(perroquet, altitude){
  perroquet.altitude = altitude;  };

// la syntaxe d'appel est différente, et logique...
mon_perroquet.envoleToi(mon_perroquet, 20); // on semble se répéter!
```

Syntaxe d'appel des méthodes

- Les fonctions classiques s'invoquent avec la syntaxe:

  nom_de_fonction(paramètres)

- Les méthodes d'un objet s'invoquent avec la syntaxe:

  objet.nom_de_fonction(paramètres)

- On dit qu'on appelle la méthode **«sur»** l'objet.

Appel de méthode et this

- **objet.**nom_de_fonction**(** **paramètres** **)**
- L'objet sur lequel on appelle une méthode (à gauche du .) est semblable à un *paramètre implicite* additionnel (en plus des paramètres entre parenthèses).
- **Dans** la **définition** de la méthode on peut y accéder par l'identifiant ***this***.
- ***this*** réfère à l'objet sur lequel la méthode a été apelée.

```
mon_perroquet.envoleToi = function(altitude){
  this.altitude = altitude;  };

mon_perroquet.envoleToi(20);     // on ne se répète plus!
```

Et si on voulait créer plusieurs objets du même type ? (ex: Perroquet)

- Si on voulait créer 10 ou 1000 perroquets différents...
- Avec l'approche précédente, il faudrait pour chacun **répéter:**
- La liste des propriétés
- L'ajout des méthodes
- Meilleure approche en JS (pour éviter de répéter 10 fois): ***fonction Constructeur*** et ***prototype***

Constructeur

- On peut définir une fonction spéciale (un *constructeur*) pour construire et initialiser des objets d'un certain type.

```
var Perroquet = function(nom, age_en_annees) {
  this.nom = nom;
  this.age = age_en_annees;
  this.altitude = 0; };
```

- On peut ensuite facilement créer plusieurs objets de ce type, à l'aide du mot-clé **new**. Syntaxe: new Constructeur(paramètres)

```
// Créer des perroquets:

var mon_perroquet = new Perroquet("Coco",3);
var jaco = new Perroquet("Jaco",0);
```

Constructeurs et classes

- L'orienté objet de nombreux langages (Java, C++, Python...) est basé sur des «Classes».
- Pas JavaScript: c'est un langage objet basé sur des «prototypes». Il n'y a pas de vraies «classes».
- Mais on peut voir un constructeur comme correspondant à une «Classe»: il permet de créer (construire) autant d'objets que l'on veut du même genre (~classe) ex: plusieurs Perroquets.
- Convention recommandée: nommez vos fonctions constructeurs avec une Majuscule initiale.

Ajouter des méthodes

- L'exemple de constructeur qu'on a vue initialise les **propriétés** de tout objet qu'il construit.

- On pourrait aussi ajouter les **méthodes** à l'objet depuis le code du constructeur avec ex:

  this.nom_de_methode = function(...) { ... };

- Mais il y a une meilleure manière...

Ajouter des méthodes

- **Manière recommandée:** ajouter les méthodes au *prototype* de la fonction constructeur:

Soit le constructeur

```
var Perroquet = function(nom, age_en_annees) {
  this.nom = nom;
  this.age = age_en_annees;
  this.altitude = 0; };
```

On peut rajouter des méthodes à son «prototype»

```
Perroquet.prototype.envoleToi = function(altitude){
  this.altitude = altitude;  };
```

Les objets céés avec **`new NomDuConstructeur(...)`** reçoivent en partage toutes les méthodes (et propriétés) présentes dans **`NomDuConstructeur.prototype`**.

Résumé: exemple complet

```
// Constructeur
var Perroquet = function(nom, age_en_annees) {
  this.nom = nom;
  this.age = age_en_annees;
  this.altitude = 0; };

// Méthodes (ajoutees au prototype du constructeur)
Perroquet.prototype.envoleToi = function(altitude){
  this.altitude = altitude;  };

Perroquet.prototype.repete = function(phrase) {
  print( this.nom + " dit " + phrase );   };

// Ex d'utilisation: on crée deux objets de type Perroquet
var mon_perroquet = new Perroquet("Coco",3);
var jaco = new Perroquet("Jaco",0);
// On appelle des méthodes sur ces objets
mon_perroquet.envoleToi(20);
jaco.repete("Coco s'envole!");
```

Précision: syntaxe de définition de fonction

- Deux syntaxes équivalentes pour définir des fonctions:

  `var nomDeFonction = function(paramètres) { ... };`

  Ou

  `function nomDeFonction(paramètres) { ... }`

- La 2ème est plus proche de la syntaxe Java/C/...

- Mais pour ajouter *directement* une méthode à un prototype on peut seulement utiliser la première:

  `NomConstructeur.prototype.nomDeMethode = function(paramètres) { ... };`

Précision: paramètres d'une fonction

- En JavaScript vous pouvez appeler une fonction avec un nombre d'arguments (paramètres) différent de ceux que vous avez déclaré
- La liste des arguments est accessible par la variable ***arguments*** (un tableau ordinaire)

```
var f = function(x, y) { print(arguments); };

f(3, 5); // affiche 3, 5
f(3); // affiche 3
f(3, "Bonjour", 5, true, -0.25);
 // affiche 3,Bonjour,5,true,-0.25
```

- Cela permet d'écrire des fonctions qui peuvent gérer un nombre variable d'arguments.

Héritage, polymorphisme

- Des fonctionnalités orienté objet plus avancées (telles que l'*héritage*) sont aussi possibles en JavaScript mais dépassent le cadre de ce cours.

  (Si ça vous intéressés, voir ex. https://developer.mozilla.org/en-US/docs/JavaScript/Introduction_to_Object-Oriented_JavaScript)

- Vous les verrez en Java dans le cours Programmation 2