JAVASCRIPT
POO

1

INTRODUCTION

Le concept général de la POO permet de regrouper les données et les instructions de manipulation en unités conceptuelles.

La **programmation orientée objets** encapsule les données, nommées *attributs*, et les méthodes, qui constituent les *comportements* dans des objets. Les données et les méthodes d'un objet sont intimement liées.

2

AVANTAGES DES OBJETS

- L'implémentation d'une classe peut être changée sans que les appels à celle-ci aient à être modifiés.
- La programmation par objets permet de créer du code très modulaire et réutilisable.
- Elle permet de penser en termes d'objets du domaine de l'application.

3

TERMINOLOGIES

- Classe: type de données défini par le programmeur.
- Objet: instance de la classe. On défini la classe une seule fois et on l'utilise plusieurs fois pour créer des objets.
- Donnée membre: aussi appelée attribut ou propriété et représente une valeur faisant partie des données de la classe.
- Fonction membre: aussi appelée méthode et représente une fonction permettant d'agir sur les données de la classe.
- Classe parent: c'est la classe utilisée pour dériver une nouvelle classe. On l'appelle aussi classe de base.
- Classe enfant: c'est la nouvelle classe dérivée à partir d'une classe parent.

4

ENCAPSULATION

- L'encapsulation consiste à cacher les détails internes utiles au fonctionnement et à l'implémentation du type de données.
- Il est également possible de forcer la modification des données en passant par un mutateur (fonction de modification) qui permettra de valider le changement avant qu'il soit effectuer.
- De la même manière il est possible de forcer la lecture en passant par un accesseur (fonction de lecture).

5

HÉRITAGE

- C'est la base des notions de réutilisation de composants logiciels.
- L'idée est de pouvoir définir (dériver) une nouvelle classe en se servant d'une classe existante (base).
- La classe dérivée hérite des membres de la classe de base tout en lui ajoutant de nouveaux.
- Il s'agit d'écrire de nouvelles classes plus spécifiques en se servant des définitions de base.

6

- Par exemple, nous pouvons dériver une nouvelle classe Employe en se servant de la classe de base Personne.
- Permet de définir la relation “est un”. Par exemple, un employé est une personne ou encore un étudiant est une personne. Un cercle est une forme géo…

7

CONSTRUCTEURS/DESTRUCTEURS

- Un constructeur servira à initialiser le contenu d’un objet et même dans certains cas à allouer l’espace nécessaire aux données membres de l’objet.
- Le constructeur est appelé automatiquement lors de la création de l’objet, il est alors impossible de l’oublier.
- Un destructeur permet de libérer les ressources utilisées par un objet.

8

Exemple en PHP

```
class NomDeLaClasse {
      var $propriete1;
      var $propriete2;
      function __construct() {
            //constructeur
      }
      function fonction() {
         // méthode
     }
}
```

Note: avant PHP5 le constructeur utilisait le même nom que la classe. Mais on n'utilise plus cette méthode.

9

La classe Employe

```
class Employe {
  var $nom;
  var $salaire;
  function __construct($n, $s) {
    $this->nom = $n;
    $this->salaire = $s;
  }
  function toHTML() {
    return "<strong>Le nom:</strong><em>$this->nom</em>".
       "<strong>sal:</strong><em>$this->salaire</em>";
  }
}
$bob = new Employe( "Bob", 45000 );
echo $bob->toHTML();
```

10

EXEMPLE EN JAVA

```
public class Film
{
     // déclarations des constantes de classe
     public static final char GÉNÉRAL = 'G',
                              ADULTE = 'A';

     // déclarations des attributs de classe
     private static int nbFilms = 0;

     // déclarations des attributs d'instance
     private String titre;
     private char cote;
     private boolean comique;

     // déclaration du constructeur sans paramètre
     public Film()
     {
           titre = "";
           cote = ' ';
           comique = false;
           nbFilms++;
     }

     // déclaration du constructeur avec 3 paramètres
     public Film(String unTitre, char uneCote, boolean c)
     {
           titre = unTitre;
           cote = uneCote;
           comique = c;
           nbFilms++;
     }
```

11

Classe Date

```
public class Date
{
    private int jour;
    private int mois;
    private int an;

    // constructeur par défaut
    public Date()
    {
        jour = 1;
        mois = 1;
        an   = 2007;
    }

    // constructeur d'initialisation
    public Date(int j, int m, int a)
    {
        jour = j;
        mois = m;
        an   = a;
    }

    public String toString()
    {
        String message = jour + "/" + mois + "/" + an;
        return message;
    }
} // fin Date
```

Date
-jour : int -mois : int -an : int
+Date() +Date(j : int, m : int, a : int) +toString() : String

NOTIONS AVANCÉES DE POO

- Membres private,protected,public
- Héritage multiple (cas C++)
- Interfaces
- Constantes
- Classes abstraites
- Utilisation des méthodes de la classe de base
- Surcharge des méthodes

13

MÉTHODES

Une méthode est une fonction qui peut être invoquée via l'objet issu de la fonction constructeur.

Ici aussi, le mot clé this référence l'objet qui invoque la méthode.

14

Cas1 : méthode externe à la classe

```
//Méthode
  function Calculer()
  {
  return this.attrib1 + this.attrib2;
  };
  // Constructeur
  function maClasse(arg1,arg2)
  {
  //Initialisation des propriétés de l'objet
  this.attrib1 = arg1;
  this.attrib2 = arg2;

  //Définition des méthodes de l'objet
  this.getCalculer = Calculer;
} appel : var result = maClasse.getCalculer();
```

15

Cas2 : méthode interne à la classe

```
// Constructeur
  function maClasse(arg1,arg2)
  {
  //Initialisation des propriétés de l'objet
  this.attrib1 = arg1;
  this.attrib2 = arg2;

  //Définition des méthodes de l'objet
  this.getCalculer = function ()
  {
  return this.attrib1 + this.attrib2;
  };
  } appel : var result = maClasse.getCalculer();
```

16

Cas3 : utilisation de prototype (à voir plus loin)

La classe

```
function Dinosaure(){
        this.sexe;
        this.couleur;
        this.age;
        this.affame;
```

17

Constructeur par défaut

```
this.Dinosaure=function(sexe, couleur, age, affame){
   if (arguments.length == 0){
        this.sexe=null;
        this.couleur=null;
        this.age=0;
        this.affame=false;
  }
```

18

CONSTRUCTEUR DE COPIE

```
else
  if (arguments[0] instanceof Dinosaure){

        this.sexe=arguments[0].sexe;
        this.couleur=arguments[0].couleur;
        this.age=arguments[0].age;
        this.affame=arguments[0].affame;
  }
```

19

CONSTRUCTEUR PARAMÈTRÉ

```
else{
        this.sexe=sexe;
        this.couleur=couleur;
        this.age=age;
        this.affame=affame;
  }
}//fin des constructeurs
```

20

MÉTHODES «GET» ET «SET»

```
this.getSexe=function(){
   return this.sexe;
}
this.getCouleur=function(){
   return this.couleur;
}
this.getAge=function(){
   return this.age;
}
this.getAffame=function(){
   return this.affame;
}
```

21

SUITE

```
this.setSexe=function(sexe){
   this.sexe=sexe;
}
this.setCouleur=function(couleur){
   this.couleur=couleur;
}
this.setAge=function(age){
   this.age=age;
}
this.setAffame=function(affame){
   this.affame=affame;
}
```

22

AUTRES MÉTHODES

```
this.affiche=function(){
        return "Sexe = "+this.sexe+"<br>"+"Couleur =
  "+this.couleur+"<br>"+"Age = "+this.age+"<br>"+"Affamé = "+this.affame;
}
this.nourrir=function(){
   var message;
   if (this.affame){
        message="Miam, à manger !";
        this.affame=false;
   }
   else
        message="Non merci, j'ai déjà mangé !";
   return message;
}
this.feter=function(){
  this.age++;
}
}//fin de la classe Dinosaure
```


UTILISATION DE LA CLASSE DINOSAURE

```
var resultat="";
var dino=new Dinosaure();
dino.Dinosaure("mâle","rouge",7,true);
var dinette=new Dinosaure();
dinette.Dinosaure("femelle","jaune",15,false);
var dirien=new Dinosaure();
dirien.Dinosaure();
resultat+="Caractéristiques de l'objet dino : <br><br>"+dino.affiche();
resultat+="<br><br>État de l'estomac de dino : "+dino.nourrir();
resultat+="<br><br>"+dino.affiche();
resultat+="<br><br>État de l'estomac de dino : "+dino.nourrir();
dino.feter();
dino.setCouleur("vert");
dinette.setCouleur("rose");
resultat+="<br><br>"+dino.affiche();
resultat+="<br><br>"+dinette.affiche();
resultat+="<br><br>"+dirien.affiche();
document.write(resultat);
```

RÉSULTAT

Caractéristiques de l'objet dino :

Sexe = mâle
Couleur = rouge
Age = 7
Affamé = true

État de l'estomac de dino : Miam, à manger !

Sexe = mâle
Couleur = rouge
Age = 7
Affamé = false

État de l'estomac de dino : Non merci, j'ai déjà mangé !

25

SUITE

Sexe = mâle
Couleur = vert
Age = 8
Affamé = false

Sexe = femelle
Couleur = rose
Age = 15
Affamé = false

Sexe = null
Couleur = null
Age = 0
Affamé = false

26

DESTRUCTEUR

En JavaScript, il n'est pas possible de créer une fonction « *destructeur* » pour supprimer un objet.

Mais l'opérateur delete permet de supprimer des propriétés, des variables (*sauf celles déclarées avec l'instruction var*).

Le mot clé null peut également être utilisé pour supprimer une référence.

```
obj = new Object();
delete obj;// Supprime obj
obj = null;// Vide obj
```

```
var obj = new Object();
delete obj;// Ne supprime pas obj
obj = null;// Vide obj
```

27

PROTOTYPE

Les techniques utilisées ici, pour définir les méthodes de la classe *Dinosaure*, présentent un inconvénient .

Effectivement, pour définir les fonctions en tant que méthode de la « classe d'objets » *Dinosaure*, ont les a attribuées à des propriétés déclarées dans le constructeur. Ceci implique que pour chaque instance de l'objet, les méthodes occuperont un certain espace mémoire, et ce, avant même de savoir si l'instance utilisera la ou les méthodes.

C'est là qu'intervient l'objet prototype…

28

L'objet prototype est une propriété de l'objet Object. L'objet Object peut être vu comme une superclasse dont héritent tous les objets JavaScript. L'objet Function hérite donc, lui aussi, de cette super classe et de fait, de sa propriété prototype.

Les instances d'objet créées par une fonction constructeur héritent de l'ensemble des propriétés du prototype.

Noter, qu'à la lecture d'une propriété d'un objet, JavaScript recherche d'abord cette propriété dans l'objet et si elle ne si trouve pas, il passe au prototype.

29

CRÉATION D'UNE CLASSE

- Il est tout à fait possible de définir des classes en se basant uniquement sur le concept des fonctions, mais, dans ce cas, le code de chaque méthode serait dupliqué à chaque instanciation d'un nouvel objet alourdissant d'autant la place réservée en mémoire.

```
function maClasse() {
  this.attribut1;
  this.attribut2;
  this.methodeA = function() {
        // code
  }
  this.methodeB = function() {
        // code
  }
}
```

30

Comment faire alors? Mais oui bien sur, en se basant sur l'objet prototype. En effet, la technique consiste à créer une classe de base. Ensuite celle-ci est enrichie grâce au prototype ce qui permet de ne charger en mémoire qu'une seule partie de l'objet. En se basant sur le schéma précédent cela donnerait :

```
function maClasse() {
   this.attribut1;
   this.attribut2;
}

maClasse.prototype.methodeA() = function() {
   // code
}

maClasse.prototype.methodeB() = function() {
   // code
}
```

31

Reprise de la classe «Donosaure» en utilisant prototype

```
function Dinosaure(){
        this.sexe;
        this.couleur;
        this.age;
        this.affame;
}//fin de la classe Dinosaure
```

32

CONSTRUCTEUR PAR DÉFAUT

```
Dinosaure.prototype.Dinosaure=function(sexe, couleur,
  age, affame){
   if (arguments.length == 0){
        this.sexe=null;
        this.couleur=null;
        this.age=0;
        this.affame=false;
  }
```

33

CONSTRUCTEUR DE COPIE

```
else
  if (arguments[0] instanceof Dinosaure){

        this.sexe=arguments[0].sexe;
        this.couleur=arguments[0].couleur;
        this.age=arguments[0].age;
        this.affame=arguments[0].affame;
  }
```

34

Constructeur paramètré

```
else{
        this.sexe=sexe;
        this.couleur=couleur;
        this.age=age;
        this.affame=affame;
  }
}//fin des constructeurs
```

35

Méthodes «get» et «set»

```
Dinosaure.prototype.getSexe=function(){
   return this.sexe;
}
Dinosaure.prototype.getCouleur=function(){
   return this.couleur;
}
Dinosaure.prototype.getAge=function(){
   return this.age;
}
Dinosaure.prototype.getAffame=function(){
   return this.affame;
}
```

36

SUITE

```
Dinosaure.prototype.setSexe=function(sexe){
   this.sexe=sexe;
}
Dinosaure.prototype.setCouleur=function(couleur){
   this.couleur=couleur;
}
Dinosaure.prototype.setAge=function(age){
   this.age=age;
}
Dinosaure.prototype.setAffame=function(affame){
   this.affame=affame;
}
```

37

AUTRES MÉTHODES

```
Dinosaure.prototype.affiche=function(){
        return "Sexe = "+this.sexe+"<br>"+"Couleur =
   "+this.couleur+"<br>"+"Age = "+this.age+"<br>"+"Affamé = "+this.affame;
}
Dinosaure.prototype.nourrir=function(){
   var message;
   if (this.affame){
        message="Miam, à manger !";
        this.affame=false;
   }
   else
        message="Non merci, j'ai déjà mangé !";
   return message;
}
Dinosaure.prototype.feter=function(){
   this.age++;
}
```

38

Utilisation de la classe Dinosaure

```
var resultat="";
var dino=new Dinosaure();
dino.Dinosaure("mâle","rouge",7,true);
var dinette=new Dinosaure();
dinette.Dinosaure("femelle","jaune",15,false);
var dirien=new Dinosaure();
dirien.Dinosaure();
resultat+="Caractéristiques de l'objet dino : <br><br>"+dino.affiche();
resultat+="<br><br>État de l'estomac de dino : "+dino.nourrir();
resultat+="<br><br>"+dino.affiche();
resultat+="<br><br>État de l'estomac de dino : "+dino.nourrir();
dino.feter();
dino.setCouleur("vert");
dinette.setCouleur("rose");
resultat+="<br><br>"+dino.affiche();
resultat+="<br><br>"+dinette.affiche();
resultat+="<br><br>"+dirien.affiche();
document.write(resultat);
```

39

Types prédéfinies et prototypes

Les classes natives de JavaScript ont aussi une propriété prototype.

Il est donc possible de définir de nouvelles méthodes (et, pourquoi pas, redéfinir des méthodes existantes) pour ces classes.

Pour l'exemple, on va d'ajouter une nouvelle méthode à l'objet String : *String.getReverse()*. Cette méthode sera chargée de retourner la chaîne string à l'envers.

40

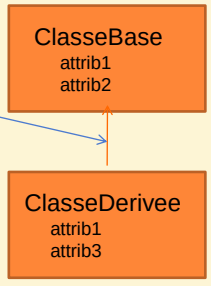
```
String.prototype.getReverse = function()
{
var str = "";
for(var i = this.length; i>-1; i--)
str += this.substr(i,1);
return str;
};

var myString = "Ceci est un exemple !";
var result = myString.getReverse();
```

41

HÉRITAGE (CAS1)

```
function ClasseBase()
{
this.attrib1 = null;
this.attrib2 = "valeur2";
};
ClasseBase.prototype.methodeA = function()
{
alert("Méthode A exécutée !");
};
ClasseDerivee.prototype = new ClasseBase();
ClasseDerivee.prototype.constructor =
ClasseDerivee;
function ClasseDerivee(param)
{
this.attrib3 = param;
this.methodeB = function()
{
alert(
"Méthode B exécutée ! \n"
+ this.attrib1 + ' - '
+ this.attrib2 + ' - '
+ this.attrib3
);
};
};
```

```
var monObject = new
ClasseDerivee("valeur3");
monObject.attrib1 = "valeur1";
monObject.methodeA();
monObject.methodeB();
alert(monObject.constructor);
```



ClasseDerivee.prototype.constructor = ClasseDerivee;, permet de modifier la référence à la fonction constructeur. En effet, si on supprime cette ligne, la ligne *alert(myObject.constructor);* retournera *ClasseBase* ! Chaque objet possède une propriété constructor héritée de l'objet prototype. Cette propriété peut se révéler très utile pour déterminer le type d'un objet.

42

HÉRITAGE PAR « COPIE » (CAS2)

```
function ClasseBase(param1,param2){
this.attrib1 = param1;
this.attrib2 = param2;
this.methodeA = function()
{
alert("Méthode A exécutée !");
};
};
function ClasseDerivee(param){
this.parent = ClasseBase;
this.parent("valeur1","valeur2");
delete this.parent;
this.attrib3 = param;
this.methodeB = function(){
alert(
"Méthode B exécutée ! \n"
+ this.attrib1 + ' - '
+ this. attrib2 + ' - '
+ this. attrib3
);};};
```

```
var monObject = new ClasseDerivee("valeur3");
monObject.methodeA();
monObject.methodeB();
alert(monObject.constructor);
```

Il est primordial que ces trois lignes soient déclarées au tout début de la fonction constructeur.

- Le constructeur de la classe *ClasseBase* est appelé sur la propriété temporaire *parent* comme une méthode.
- La seconde ligne permet de passer des paramètres à la fonction construteur de la classe *ClasseBase*.
- La dernière détruit la propriété temporaire.

43

HÉRITAGE MULTIPLE PAR « CALL » (CAS3)

```
function ClasseBase(param1,param2){
 this.property_1 = param1;
 this.attrib2 = param2;
 this.methodeA = function()
 {
 alert("Méthode A exécutée !");
 };
};
ClasseBase.prototype.methodeC = function(){
 alert("Méthode C exécutée !");
};
ClasseAbstract = function(){
 this.methodeD = function()
 {
 alert("Méthode D exécutée !");
 };
 this.methodeE = function(){
 alert("Cette méthode doit être implémentée !");
 };
};
```

```
function ClasseDerivee(param){
 ClasseBase.call(this, "valeur1","valeur2");
 ClasseAbstract.call(this);
 this.attrib3 = param;
 this.methodeB = function(){
 alert(
 "Méthode B exécutée ! \n"
 + this.attrib1 + ' - '
 + this.attrib2 + ' - '
 + this.attrib3
 );
 };
};
var monObject = new ClasseDerivee("valeur3");
monObject.methodeA();
monObject.methodeB();
ClasseBase.prototype.methodeC.call(monObject);
alert(monObject.constructor);
```

Dans cet exemple, la fonction call() permet d'invoquer la fonction constructeur ClasseBase sur l'objet courant this, comme si la fonction ClasseBase était une méthode de ClasseDerivee.

Le premier argument de la fonction call() doit être l'objet sur lequel la méthode sera appliquée. Les arguments suivants sont passés à la fonction ClasseBase comme paramètres. Seul l'argument représentant l'objet est obligatoire.

44

POLYMORPHISME

Polymorphisme d'héritage

Définition :

Une classe dérivée qui substitue l'implémentation d'une méthode, issue de sa classe de base, par sa propre implémentation.

45

```
Animal = function()
{
this.dormir = function()
{
alert("BZzzz...")
};

this.parle = function()
{
alert(" Rien à dire!");
};
};
```

FIN POO

```
Chien = function()
{
Animal.call(this);
this.parle = function()
{
alert("Ouarf! Ouarf!");
};
}

var obj = new Chien();
obj.parle();
obj.dormir();
```

46