

Les fiches récap de l'école O'clock



Les objets en JS

Dernière modification: 8 août 2022

Les objets en JS

Historique, naissance, concept

Dans les années 1960, **la programmation structurée** se développe avec l'informatique. Ce type de programmation utilise les structures de contrôles *while*, *repeat*, *for*, *if .. then .. else* et les *fonctions* (appelées également *procédures*). On utilise les langages de programmation procéduraux. On parle alors de **programmation procédurale**, soit l'enchaînement de portions de code et des appels à des fonctions, qui peuvent elles-mêmes appeler d'autres fonctions.

Avant l'invention de l'orienté objet, certains projets ne pouvaient plus fonctionner avec ces méthodes. **La programmation orientée objet** a été créée (dans les années 1980 et 1990) pour dépasser certaines barrières techniques. Elle a prit les meilleures idées de la programmation structurée et les a combinées avec plusieurs nouveaux concepts. Le résultat donna une nouvelle façon d'organiser son programme.

On peut résumer ainsi deux types d'organisation d'un programme :

- **Autour du code** (ce qui se passe) : **programmation structurée/procédurale**.
- **Autour de la donnée** (ce qui est affecté) : **programmation orientée objet**.

« Data first » vs « Code first » : la POO travaille autour de la donnée. L'idée principale

est que la donnée contrôle l'accès au code. Dans un langage orienté-objet, nous définissons la donnée et les fonctions qui ont la permission d'agir sur cette donnée. Ainsi, un type de donnée définit précisément quelles sortes d'opérations peuvent être appliquées à ces données.

Pour supporter le principe de la programmation orientée objet, tous les langages supportant POO ont 3 traits en commun :

- L'encapsulation (voir plus bas)
- L'héritage
- Le polymorphisme

Nous détaillerons les deux derniers dans la fiche récap' suivante.

Les avantages de la POO

- Clarté fonctionnelle du code.
- Protège la cohérence des données et des traitements.
- Modulaire.
- Réutilisable.

La POO est une approche par entités, avec leur contexte, leurs données, leurs méthodes.

Qu'est ce qu'un objet ?

Objet

Représentation d'une entité matérielle ou immatérielle qui possède des propriétés (données) ou actions (code, méthodes). *Quelques exemples : une personne, un animal, un élève, un compte bancaire, une requête en base de données.*

Attribut

Caractère propre à un objet : on parle aussi de ses « propriétés ».

Exemple : une personne possède différents attributs : son nom, sa date de naissance, ses coordonnées, ..., un compte bancaire : montant disponible, numéro de compte, ..., une requête en base de données : la chaîne de

la requête.

Méthode

Une méthode est une action applicable à un objet.

Exemple : manger, dormir, parler, ..., créditer, débiter, afficher le solde, ..., préparer une requête, lier un paramètre, exécuter la requête, retourner le résultat.

Classe

Modèle qui définit la structure d'un objet, ses attributs, ses méthodes. Les objets sont créés (« instanciés ») à partir de ce modèle. Ils ont pour **type** le nom de la classe à partir de laquelle ils sont créés.

Exemple : Jacques, François, Emmanuel, Jean-Luc sont des Personnes. Elles disposent d'un nom, d'un âge, ..., peuvent marcher, manger, dormir.

Définition et utilisation

Définition d'une classe en JS

class

```
// fichier myclass.js
// le nom d'une classe s'écrit en CamelCase
// On définit une classe par fichier qui est nommé comme la classe

class MyClass
{
    // Propriété/Attribut/Donnée
    property;

    // Méthode/Action/Code
    myMethod()
    {
        // ...
    }
}
```

L'instanciation

Créer une instance (un objet) de la classe *MyClass* avec **new** :

```
const instance = new MyClass();
```

`instance` est un objet de classe *MyClass*.

Les attributs

Déclarer les attributs d'une classe :

```
class Person
{
    name = 'John Doe';
}
```

Utiliser (lire, assigner une valeur) :

```
const instance = new Person();

console.log(instance.name); // John Doe

instance.name = 'Michaël Jackson';

console.log(instance.name); // Michaël Jackson
```

Opérateur objet (object operator) : `.`

Les méthodes

Déclarer les méthodes d'une classe :

```
class Person
{
    name = 'John Doe';

    hello()
    {
        console.log('Oh, hello!');
    }
}
```

Appeler (exécuter) une méthode publique :

```
const anonymous = new Person;

console.log(anonymous.name); // John Doe

anonymous.hello();
```

this

Variable accessible *dans le contexte de l'objet*, référence à l'objet lui-même :

```
class Person
{
  name = 'John Doe';

  hello()
  {
    console.log('Oh, hello, my name is ' + this.name);
  }
}
```

Visibilité

- **publique** autorise l'accès direct aux propriétés et aux méthodes de l'objet depuis « l'extérieur ».
- **privée** Les champs d'instance privés sont déclarés avec `#nomdeavar`. Le `#` fait partie du nom lui-même et est également utilisé pour la déclaration et l'accès. L'encapsulation est imposée par la langue. C'est une erreur de syntaxe de faire référence à des variables `#` (donc privées) qui ne sont pas dans la portée.

Encapsulation

En programmation, l'encapsulation désigne le principe de regrouper des données avec un ensemble de méthodes permettant de les lire ou de les manipuler, empêchant l'accès aux données par un autre moyen que les services proposés. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.



Par défaut, une variable déclarée dans une classe est accessible et modifiable (comme un objet basique en fait)



```
class Person
{
  #secret = "Ceci est un secret";

  showSecret()
  {
    return this.#secret;
  }
}


const dude = new Person();

console.log(dude.showSecret()); // Ceci est un secret
console.log(dude.#secret); // Uncaught SyntaxError: Private field
'#secret' must be declared in an enclosing class
```

Getters et Setters (accesseurs et mutateurs)

Méthodes standard qui permettent un accès contrôlé aux propriétés privées.

Définition manuelle




```
class Person {  
  // Propriété  
  #secret = 123;  
  
  // Getter  
  get secret()  
  {  
    return this.#secret;  
  }  
  
  // Setter  
  set secret(newSecret)  
  {  
    // Contrôle sur la donnée fournie  
    if(!isNaN(parseInt(newSecret))) {  
      // Modification de la propriété  
      this.#secret = newSecret;  
    }  
  }  
}
```

Constructeur

Méthode magique d'un objet qui est appelée automatiquement à l'instanciation de l'objet.

Déclarer le constructeur n'est pas obligatoire mais cela permet d'exécuter certaines actions à l'instanciation: initialiser les valeurs.



```
class MyClass
{
  #config;

  constructor(config)
  {
    this.#config = config;
  }

  get config(name)
  {
    if(typeof this.#config[name] !== 'undefined') {
      return this.#config[name];
    }
  }
}
```

Méthodes et propriétés statiques

Les méthodes et propriétés statiques sont partagées par toutes les instances, et accessibles directement à partir du nom de la classe.

statique == de classe



```
// constante de classe
// (en majuscule par convention) et en dehors de la classe mais dans
// le même fichier
const DEFAULT_NAME = "John Doe";
class Person
{
    // propriété statique
    static #majority;

    // méthode statique
    static get majority()
    {
        return Person.#majority;
    }

    static get DEFAULT_NAME() {
        return DEFAULT_NAME;
    }

    static set majority(nb)
    {
        if(!isNaN(parseInt(nb)))
            Person.#majority = nb;
    }
}

console.log('Le nom par défaut d\'une personne est ' +
Person.DEFAULT_NAME); // Le nom par défaut d'une personne est John
Doe
Person.majority = 18;
console.log('En France, l\'âge de la majorité est ' +
Person.majority); // En France, l'âge de la majorité est 18
```

Sources

- [Cours O'clock](#)
- [Programmation structurée sur Wikipédia](#)
- [Programmation procédurale sur Wikipédia](#)
- [Programmation objet sur Wikipédia](#)

Ressources

- [Cheat Sheet complète](#)
- [Cheat Sheet complète jolie](#)