Les Classes

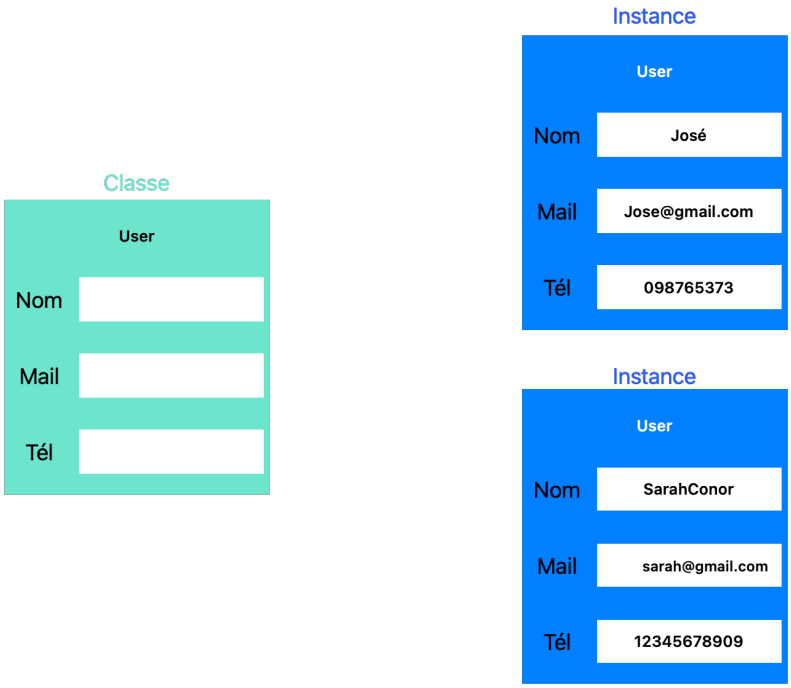
Pré requis : Les objets

Les classes sont un élément essentiel de le P.O.O (Programmation orientée objet), pour résumer le concept les classes vont nous permettre de créer plus facilement et rapidement plusieurs objets avec des caractéristiques, d'architectures similaires.

Par exemple si une application gère des utilisateurs on peut définir une classe « user » et pour chaque user on gère les données suivantes : un nom, un mail, un num de téléphone.

Pour créer / **construire** des nouveaux user, le système de class utilise une fonction qui s'appelle **constructor** (**construct** dans d'autres langages)

On va pouvoir plus facilement créer de nouveaux user (des nouvelles instances de la classe user)



Syntaxe

Donc côté code, on crée notre class UserProfile, pour pouvoir créer des nouvelles instance on renseigne les données dont on a besoin, que l'on recevra en paramètre (nameUser, mialUser, phoneUser)

Le mot clé this va représenter l'objet courant, celui que l'on est entrain de créer, c'est le contexte. Ici pour résumer on assigne aux propriété de notre classe les valeur qu'on va recevoir en paramètre

```
class UserProfile {
    constructor(nameUser, mailUser, phoneUser) {
        this.nameUser = nameUser;
        this.mailUser = mailUser;
        this.phoneUser = phoneUser;
    }
    getProfileInfo() {
        return `infos de l'utilisateur :
            son nom : ${this.nameUser}
            son mail : ${this.mailUser}
            son Tél : ${this.phoneUser}`;
    }
}
```

Bonus : on a écrit une fonction au sein de la classe, c'est une méthode de classe et elle ne pourra s'utiliser QUE sur des objets (des nouvelles instances) de cette classe

Une fois qu'on a définit la structure de notre classe on va pouvoir utiliser le constructeur pour créer un nouvel utilisateur en faisant new UserProfile( )

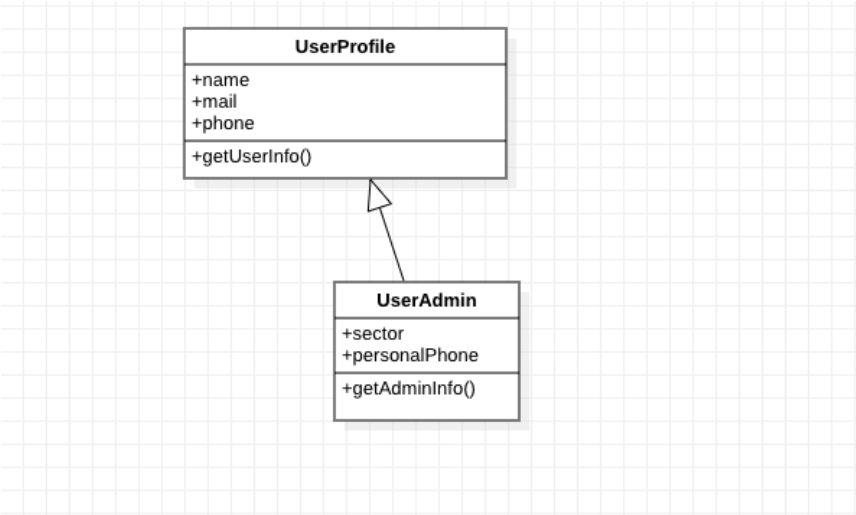
```
const exampleUser1 = new UserProfile("José", "jose@gmail.com",
« 09876543");
```

```
const exampleUser2 = new UserProfile("Sarah", "sarah@gmail.com",
"063736252");
exampleUser2.getProfileInfo();
```

Dans notre cas Il n'y aura que sur des new UserProfile que l'on pourra utiliser la méthode getProfileInfo( ).

L'héritage

Avec les classes on peut également profiter d'un système d'héritage, cela signifie que nous pouvons étendre (**extends**) les propriétés, les méthodes d'une classe vers une autre,
Par exemple dans notre application on gère déjà des utilisateurs, mais on veut aussi gérer des utilisateurs un peu plus spécifiques : des Admin, les admins ils auraient les mêmes propriétés que les utilisateurs (un nom, un mail, un téléphone) mais avec des informations en plus (le **secteur** dans lequel l'admin travaille, et son **Téléphone personnel**)
on crée une nouvelle classe « enfant » qui hérite des propriétés et des méthodes d'une classe parent.



⚠️ une classe enfant peut hériter d'une classe parent mais l'inverse n'est pas possible.
Sur une instance de **UserAdmin** on pourra utiliser **getProfileInfo**( ) mais sur une instance de **UserProfile** on ne peut pas utiliser **getAdminInfo**( ).

Dans le code : on va utiliser **extends** et **super( )**

```
class UserProfile {
    //! Pas besoin de déclarer function devant le constructor et méthodes
    constructor(nameUser, mailUser, phoneUser) {
        this.nameUser = nameUser;
        this.mailUser = mailUser;
        this.phoneUser = phoneUser;
    }
    getProfileInfo() {
        return `infos de l'utilisateur :
            son nom : ${this.nameUser}
            son mail : ${this.mailUser}
            son Tél : ${this.phoneUser}`;
    }
}

const exampleUser1 = new UserProfile("José", "jose@gmail.com", "09876543");
const exampleUser2 = new UserProfile("Sarah", "sarah@gmail.com", "063736252");
exampleUser2.getProfileInfo();

class UserAdmin extends UserProfile{
    constructor(unNom,unMail,unPhone,sector,personnalPhone){
        super(unNom,unMail,unPhone); //! Appel au constructor du parent
        this.sector = sector;
        this.personnalPhone = personnalPhone;
    }
    getAdminInfo(){
        return `infos de l'utilisateur :
        son nom : ${this.nameUser}
        son secteur d'intervention : ${this.sector}
        son Tél Personnel : ${this.personnalPhone}`;
    }
}

const exampleAdmin1 = new UserAdmin('Jacky','jack@gmail.com','012345678','administration','0987654323');

console.log(exampleAdmin1.getAdminInfo());
```

Exo Class IMC

Créer un programme permettant de Calculer l'IMC d'une personne

TODO :
- Créer une classe Imc avec un constructeur qui recevra un nom, un poids, une taille
- Créer une fonction de calcul d'IMC, qui retourne le résultat du calcul (à 2 nombre après la virgule si possible)
- Créer une fonction d'affichage « display », elle a pour rôle d'afficher en console :
  Le nom de la personne, son poids, sa taille et son imc dans une phrase
- En dehors de la class (donc dans le programme principal), récupérer la variable list (un tableau de nouvelle instances de la class) (voir discord ou 👇)
- Trouver un moyen de parcourir les éléments dans la variable list, sur chaque element utiliser la fonction display

En console du navigateur :

```
Sébastien Chabal (135 kg, 1.7 M) a un IMC de: 46.71
Escaladeuse (45 kg, 1.68 M) a un IMC de: 15.94
JOJO  (300 kg, 2 M) a un IMC de: 75.00
Gontrand  (90 kg, 1.75 M) a un IMC de: 29.39
Colonel Clock  (200 kg, 1.75 M) a un IMC de: 65.31
JOsiane de la Vega (99 kg, 1.55 M) a un IMC de: 41.21
```

Programme Principal (en dehors de la classe)

```
// //* progr principal -> on fait l'injection des données
let list = [
  new Imc("Sébastien Chabal", 135, 1.7),
  new Imc("Escaladeuse", 45, 1.68),
  new Imc("JOJO ", 300, 2),
  new Imc("Gontrand ", 90, 1.75),
  new Imc("Colonel Clock ", 200, 1.75),
  new Imc("JOsiane de la Vega", 99, 1.55),
];
//*Boucle qui parcourt list pour utiliser display()
????.?????????((???????) ?? ???.????());
```

Auteur : Jean-François Pech
Relu, validé & visé par :
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création : 03/03/2023
Date révision : 10/03/2023

Exo Class PME

Gérer une PME

CDC

Un Salarié a un nom, prénom, âge, salaire mensuel
Il est payé sur N mois.
En plus il y a XXX de charges

Une Pme c'est un nom, une équipe de plusieurs salariés
Grace à ses ventes elle a des revenus R
Mais aussi … :
- des frais fixes FF (impôts etc…)
- Des frais d'achats de matériel et de logiciels FA

TODO :
- Créer une classe Pme et une classe Employee
- Utiliser des fonctions
- Faire le bilan annuel de l'entreprise et l'afficher en console.

Détails :
- 3 salariés qui gagnent par mois : 2000, 3000 et 4000 euros
- R = 300000 (trois cent mille)
- FF = 20000 (vingt mille)
- FA = 50000 (cinquante mille)
- N = 12
- XXX = 90%

En console du navigateur :

```
------------------MA PME-----------------
Ma Petite Entreprise -  : Cout Initial : 70000
Ma Petite Entreprise -  : Cout Total Equipe : 205200
Ma Petite Entreprise -  : VENTES : 300000
Ma Petite Entreprise -  : BILAN : 24800
```

```
// // Scénario
const pme = new Pme (
  //Le nom entreprise
    "Ma Petite Entreprise - ",
    //L'equipe de salariés (un tableau)
    [new Employee ("Duval", "Paul", 30, 2000),
    new Employee ("Durand", "Alain", 40, 3000),
    new Employee ("Dois", "Sylvia", 50, 4000),],
     //le revenu , frais fixe, frais d'achat
    300000,
    20000,
    50000);
pme.bilanCalculed();
```

Exo Class COMPTES BANCAIRES.

Enoncé

Gérer des compte en banque

Consignes

- Créer une classe CompteBancaire avec des méthodes de crédit, de retrait, de visualisation de l'état du compte bancaire (en console), on doit pouvoir aussi faire un virement d'un membre à un autre.
- Générer une exception pour ne pas dépasser le solde (pas de retrait ou de virement qui dépassent le solde du compte bancaire)

Détails

Faire une scénario avec gestion de 3 comptes crédités à 1000 € chacun (Alex, Clovis, Marco)
Puis Alex retire 100
Puis Marco fait un virement de 300 à Clovis
Enfin Alex tente un retrait de 1200
Afficher tous les soldes finaux.
Ces compte sont placés dans un tableau associatif de clients

En console :

```
Ajout de: 1000 pour: Alex
Ajout de: 1000 pour: Clovis
Ajout de: 1000 pour: Marco
Retrait de: 100 pour: Alex
Virement de: 300 de: Marco vers: Clovis
Ajout de: 300 pour: Clovis
Retrait de: 300 pour: Marco
----->Alex, retrait de: 1200 refusé avec solde: 900
titulaire: Alex, solde: 900
titulaire: Clovis, solde: 1300
titulaire: Marco, solde: 700
```

Des ressources :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Working_with_objects
https://www.pierre-giraud.com/javascript-apprendre-coder-cours/gestion-erreur-exception-try-catch/
https://www.pierre-giraud.com/javascript-apprendre-coder-cours/gestion-erreur-exception-try-catch/
https://javascript.info/try-catch

Auteur : Jean-François Pech
Relu, validé & visé par :
Jérôme CHRETIENNE
Sophie POULAKOS
Mathieu PARIS
Date création : 03/03/2023
Date révision : 10/03/2023

```
// Main, gère 3 comptes bancaires dans un tableau associatif
const lesComptes = {
Alex: new CompteBancaire("Alex"),
Clovis: new CompteBancaire("Clovis"),
Marco: new CompteBancaire("Marco"),
};

// lecture tableau associatif ou Objet["truc"]
// Crédite et décrit chaque compte
for (let key in lesComptes) lesComptes[key].crediter(1000);

// un retrait de 100 par Alex
????????????????;
// un petit virement: Marco Vire 300 à Clovis
?????????????;
// un petit retrait incorrect (doit déclencher erreur custom) :
// Alex fait un retrait de 1200
?????;
// bilan : faire une description de tous les comptes en console (ou DOM ?)
??????;
```

La guerre des langages

⚔️ En programmation on peut distinguer à peu près tous les langage en 2 familles, les langages basés sur les **classes** et ceux basés sur les **prototypes**.
Js est un langage orienté objet basé sur les prototypes. (C'est pour cela que l'on dit qu'en Javascript TOUT est Objet)

Le JavaScript est un langage objet basé sur les prototypes. Cela signifie que le JavaScript ne possède qu'un type d'élément : **les objets** et que tout objet va pouvoir partager ses propriétés avec un autre, c'est-à-dire servir de prototype pour de nouveaux objets.
L'héritage en JavaScript se fait en remontant la chaine de prototypage.

En plus de la manière déclarative (créer un variable et lui assigner directement une valeur) dans Javascript on va retrouver également un système de constructor pour créer tout type d'objet

Exemple de fiche récapitulative des types d'objets en JS (non exhaustif) :

On peut déclarer soit en utilisant le constructor new Array( ), mais il faut penser à stocker dans une variable, nativement dans JS pour chaque objet on aura des propriétés de base, ici length commun à plusieurs type et JS propose aussi des fonctions de base, utiles pour manipuler chaque type d'objets. (Toujours avoir le réflexe d'aller consulter la documentation, NE PAS ré inventer la ROUE)

Violet : créer une variable (via constructor ou en mode déclaratif)
Jaune : exemple d'une propriété
Blanc : des fonctions de bases