**Définissez des méthodes d'instance et des champs**

**Tirez parti des classes avec des méthodes d'instance**

Si vous revenez au chapitre sur les classes, nous les avons utilisées pour créer des objets ayant certaines propriétés. Maintenant que vous avez commencé à découvrir les fonctions, vous pouvez ajouter des **méthodes d'instance** à ces classes, pour augmenter leur puissance et leur utilité.

Une méthode d'instance est une fonction faisant partie d'une classe, et qui agit sur une instance de cette classe. Imaginons une classe  BankAccount  (compte bancaire) :

class BankAccount {

constructor(owner, balance) {

this.owner = owner;

this.balance = balance;

}

}

Vous pouvez ensuite créer une instance de cette classe appelée  newAccount  (nouveau compte) :

*const* newAccount = new BankAccount("Will Alexander", 500);

N'oubliez pas qu'un objet (une instance d'une classe est un objet) est un type par référence, donc vous pouvez toujours apporter des modifications à l'instance de  newAccount  — la partie *constante* désigne une **référence** à cette instance.

Telle quelle, l'instance n'est pas très utile. Vous pourriez afficher son solde à la console par  newAccount.balance, mais si on pensait plutôt à ajouter une mise en forme ? Vous pouvez pour cela ajouter une méthode à la classe !

class BankAccount {

constructor(owner, balance) {

this.owner = owner;

this.balance = balance;

}

showBalance() {

*console*.log("Solde: " + this.balance + " EUR");

}

}

La nouvelle méthode ci-dessus, déclarée par son nom suivi par  (), utilise le mot clé  this  pour accéder au champ  balance  de l'instance, et l'afficher sur la console avec une mise en forme supplémentaire. Ceci signifie que vous pouvez utiliser la notation dot sur l'instance  newAccount  pour appeler sa méthode  showBalance()  :

*const* newAccount = new BankAccount("Will Alexander", 500);

newAccount.showBalance(); // imprime "Solde: 500 EUR" à la console

Vous pouvez aussi ajouter des méthodes capables de modifier les propriétés de l'instance. Dans ce cas, ajoutez les méthodes  deposit()  (dépôt) et  withdraw()  (retrait) à la classe, en n'oubliant pas que les deux ont besoin d'un paramètre  amount  (montant) (parce que vous devez savoir combien déposer ou retirer !) :

class BankAccount {

constructor(owner, balance) {

this.owner = owner;

this.balance = balance;

}

showBalance() {

*console*.log("Solde: " + this.balance + " EUR");

}

deposit(amount) {

*console*.log("Dépôt de " + amount + " EUR");

this.balance += amount;

this.showBalance();

}

withdraw(amount) {

if (amount > this.balance) {

*console*.log("Retrait refusé !");

} else {

*console*.log("Retrait de " + amount + " EUR");

this.balance -= amount;

this.showBalance();

}

}

}

La méthode  deposit()  affiche le montant du dépôt sur la console, ajoute le montant au solde de l'instance, puis appelle la méthode  showBalance()  de l'instance, imprimant ainsi le nouveau solde sur la console.

Dans le corps d'une classe, le mot clé *this* fait référence à l'**instance** créée de la classe. Dans cet exemple, il fait référence à  newAccount.

La méthode  withdraw()  vérifie si le montant demandé est supérieur au solde actuel. Si c'est le cas, il refuse le retrait ; sinon, il l'autorise, en soustrayant le montant du solde et en affichant le nouveau solde sur la console.

**Pratiquez : les méthodes instance**

Voyons maintenant comment utiliser des méthodes d'instance pour enregistrer l'application de diffusion vidéo.

Rendez-vous sur cet[éditeur CodePen](https://codepen.io/nicolaspatschkowski/pen/eYNKJmP).

Plutôt qu'utiliser une fonction pour calculer la note moyenne pour une série, on a décidé que la classe Show (ou "Série") devrait gérer les nouvelles notes automatiquement.

Comme vous pouvez le constater, la classe  Show  contient un Array  ratings  , pour les notes, qui est initialisé vide, et une propriété  averageRating  (note moyenne) de zéro.

1. Déclarez une méthode instance dans la classe  Show  appelée  addRating()  . Elle prendra une  rating  (la note) qui sera un number entre 1 et 5 comme paramètre.
2. La méthode  addRating()  doit :

* ajoutez la note reçue au tableau ratings de l'instance
* recalculer la valeur  averageRating  de l'instance

Cliquez le bouton "Check ratings" pour vérifier si les séries se mettent à jour correctement. Vous pouvez cliquer plusieurs fois le bouton pour voir l'évolution des notes moyennes.

**Solution :**

[Voici un nouveau CodePen avec une solution](https://codepen.io/nicolaspatschkowski/pen/poJYedM) à l’exercice.

**Découvrez les méthodes statiques**

Il existe un type spécial de méthode pouvant être ajouté à une classe : **statique.** Elle est différente des méthodes d'instance parce qu'elle n'est **pas** liée à une instance particulière d'une classe, mais à la classe elle-même. Utilisez-la pour créer des méthodes **helper** où vous n'aurez pas besoin d'une instance d'une classe pour les utiliser.

Par exemple, en JavaScript, l'objet  Math  contient beaucoup de méthodes utiles :

*const* randomNumber = Math.random(); // crée un nombre aléatoire sur l'intervalle [0, 1]

*const* roundMeDown = Math.floor(495.966); // arrondit vers le bas à l'entier le plus proche, renvoie 495

Vous n'avez pas besoin de créer par  new  une instance de l'objet  Math  pour utiliser ces méthodes ; il suffit de les appeler sur l'objet  Math  global.

Ces deux méthodes ont été utilisées dans l'exercice précédent en arrière-plan pour créer un nombre aléatoire d'évaluations pour les composants de la série !

Vous pouvez créer vos propres méthodes statiques par le mot clé  static  !

Observons la classe suivante :

class BePolite {

static sayHello() {

*console*.log("Hello!");

}

static sayHelloTo(name) {

*console*.log("Hello " + name + "!");

}

static add(firstNumber, secondNumber) {

return firstNumber + secondNumber;

}

}

BePolite.sayHello(); // imprime "Hello!""

BePolite.sayHelloTo("Will"); // imprime "Hello Will!""

*const* sum = BePolite.add(2, 3); // sum = 5

Vous n'avez pas besoin d'ajouter un  constructor  à votre classe, car vous n'allez pas l'instancier. Vous avez une méthode qui :

* imprime un message générique ;
* accepte un argument et l'utilise pour créer un message ;
* renvoie une valeur à partir des arguments que vous lui envoyez.

Toutes ces fonctionnalités pourraient être des **fonctions**, maisl'avantage d'utiliser des méthodes de classe statique est par exemple de pouvoir les regrouper par catégorie ou par type.

**En résumé**

Dans ce chapitre, nous nous sommes intéressés à deux types de méthodes de classe :

* les méthodes d'**instance**, qui agissent sur les instances individuelles d'une classe ;
* les méthodes **statiques**, qui ne s'appuient pas sur une instance d'une classe.