Développement orienté objet

Classes et méthodes

Approche orientée objet

- Lier les données et les traitements dans des objets (encapsulation)
- La plupart des calculs est exprimée en termes d'opérations sur les objets

Ces approches sont facilités par l'utilisation de langages de programmation orienté objet

Bonne nouvelle : TypeScript est un langage orienté objet

Méthodes

Une **méthode** est une fonction qui est associée à une classe.

Dans le cours d'initiation, on a vu des exemples de méthodes : les méthodes `push` et `pop` pour les tableaux par exemple.

Les méthodes sont donc sémantiquement identiques à des fonctions mais avec deux différences syntaxiques :

- les méthodes sont définies dans la définition des classes pour rendre explicite le lien entre la classe et la méthode
- la syntaxe pour appeler une méthode est différente de celle pour appeler une fonction

Exemple : fonction vs méthode

Considérons la classe suivante :

```
class Temps {
  heure: number;
  minute: number;
  seconde: number;
}

let chrono = new Temps();
chrono.heure = 9;
chrono.minute = 45;
chrono.seconde = 0;
```

On peut faire une fonction qui affiche le temps sous la forme : `hh:mm:ss`.

```
function affichageTemps(t: Temps): void {
  let ch = "";
  if (t.heure < 10) ch += "0";
  ch += t.heure + ":";
  if (t.minute < 10) ch += "0";
  ch += t.minute + ":";
  if (t.seconde < 10) ch += "0";
  ch += t.seconde;

  console.log(ch);
}

affichageTemps(chrono);
```

```
09:45:00
```

L'approche pour faire une méthode est de mettre la définition de `affichageTemps` dans la classe :

```
class Temps {
  heure: number;
  minute: number;
  seconde: number;

  affichageTemps(): void {
    let ch = "";
    if (this.heure < 10) ch += "0";
    ch += this.heure + ":";
    if (this.minute < 10) ch += "0";
    ch += this.minute + ":";
    if (this.seconde < 10) ch += "0";
    ch += this.seconde;

    console.log(ch);
  }
}
```

Remarques :

1. La définition d'un méthode est identique à celle d'une fonction mis à part l'omission du mot-clef `function`
2. Une méthode a directement accès aux attributs de l'objet, il n'y a donc pas à lui donner l'objet en paramètre (on peut lui passer des paramètres si nécessaire)
3. Pour accéder aux attributs de l'objet, on les préfixe du mot clef `this` avec une notation pointée

Pour appeler la méthode, il faut utiliser une notation pointée sur un objet de la classe et `()` comme pour une fonction :

```
chrono.afficheTemps();
```

```
09:45:00
```

Ajoutons maintenant les méthode `temps2Number` et `number2Temps` qui reprennent le principe des fonctions faite en TD :

```
class Temps {
  heure: number;
  minute: number;
  seconde: number;

  affichageTemps(): void {
    let ch = "";
    if (this.heure < 10) ch += "0";
    ch += this.heure + ":";
    if (this.minute < 10) ch += "0";
    ch += this.minute + ":";
    if (this.seconde < 10) ch += "0";
    ch += this.seconde;

    console.log(ch);
  }

  temps2Number(): number {
    return this.heure * 60 * 60 + this.minute * 60 + this.seconde;
  }

  number2Temps(tempsEnSeconde: number): void {
    this.heure = Math.floor(tempsEnSeconde / 3600);
    tempsEnSeconde %= 3600;
    this.minute = Math.floor(tempsEnSeconde / 60);
    this.seconde = tempsEnSeconde % 60;
  }
}

console.log("Temps en seconde =", chrono.temps2Number());
chrono.number2Temps(chrono.temps2Number());
chrono.affichageTemps();
```

```
Temps en seconde = 35100
09:45:00
```

Remarque : ici la fonction `number2Temps` modifie les attributs de l'objet. On pourrait parfaitement faire une méthode qui retourne un nouvel objet `Temps` :

```
number2NouveauTemps(tempsEnSeconde: number): Temps {
  let temps = new Temps();

  temps.heure = Math.floor(tempsEnSeconde / 3600);
  tempsEnSeconde %= 3600;
  temps.minute = Math.floor(tempsEnSeconde / 60);
  temps.seconde = tempsEnSeconde % 60;

  return temps;
}
```

Ajoutons maintenant une méthode qui incrémente le temps selon d'un nombre de secondes passées en paramètre :

```
class Temps {
  heure: number;
  minute: number;
  seconde: number;

  affichageTemps(): void {
    let ch = "";
    if (this.heure < 10) ch += "0";
    ch += this.heure + ":";
    if (this.minute < 10) ch += "0";
    ch += this.minute + ":";
    if (this.seconde < 10) ch += "0";
    ch += this.seconde;

    console.log(ch);
  }

  temps2Number(): number {
    return this.heure * 60 * 60 + this.minute * 60 + this.seconde;
  }

  number2Temps(tempsEnSeconde: number): void {
    this.heure = Math.floor(tempsEnSeconde / 3600);
    tempsEnSeconde %= 3600;
    this.minute = Math.floor(tempsEnSeconde / 60);
    this.seconde = tempsEnSeconde % 60;
  }

  incrémente(nbreSecondes: number): void {
    let secondes = this.temps2Number();
    secondes += nbreSecondes;
    this.number2Temps(secondes);
  }
}
```

```
chrono.incrémente(1337);
chrono.affichageTemps();
```

```
10:07:17
```

Remarque : pour appeler une méthode dans une autre méthode de la classe, il faut la préfixer avec `this`.

On peut passer un objet en paramètre et l'utiliser directement. On peut aussi retourner des objets.

Remarque : ici on utilise des objets `Temps` mais cela pourrait être autre chose)

```
class Temps {
  heure: number;
  minute: number;
  seconde: number;

  affichageTemps(): void {
    let ch = "";
    if (this.heure < 10) ch += "0";
    ch += this.heure + ":";
    if (this.minute < 10) ch += "0";
    ch += this.minute + ":";
    if (this.seconde < 10) ch += "0";
    ch += this.seconde;

    console.log(ch);
  }

  temps2Number(): number {
    return this.heure * 60 * 60 + this.minute * 60 + this.seconde;
  }

  number2Temps(tempsEnSeconde: number): void {
    this.heure = Math.floor(tempsEnSeconde / 3600);
    tempsEnSeconde %= 3600;
    this.minute = Math.floor(tempsEnSeconde / 60);
    this.seconde = tempsEnSeconde % 60;
  }

  incrémente(nbreSecondes: number): void {
    let secondes = this.temps2Number();
    secondes += nbreSecondes;
    this.number2Temps(secondes);
  }

  copie(): Temps {
    let temps = new Temps();

    temps.heure = this.heure;
```

```
    temps.minute = this.minute;
    temps.seconde = this.seconde;

    return temps;
  }

  avant(chrono: Temps): boolean {
    return this.temps2Number() < chrono.temps2Number();
  }
}

let chrono = new Temps();
chrono.heure = 9;
chrono.minute = 45;
chrono.seconde = 0;

let nouveauChrono = chrono.copie();
nouveauChrono.incrémente(1000);
console.log(chrono.avant(nouveauChrono));
```

```
true
```

Constructeur

Un constructeur peut être vu comme une *sorte* de méthode appelée lorsque l'on crée un objet à partir d'une classe avec l'opérateur `new`. Les classes ont un constructeur par défaut qui ne fait rien sauf affecter les valeurs que l'on a donné par défaut aux attributs. Si aucune valeur n'est donnée, l'attribut prend la valeur `undefinied`.

```
class Point {
  x = 0;
  y: number;
}

let p = new Point();

console.log("p.x =", p.x, "p.y =", p.y);
```

```
p.x = 0 p.y = undefined
```

Il est possible de redéfinir le constructeur :

```
class Point {
  x: number;
  y: number;

  constructor() {
    this.x = 0;
    this.y = 0;
  }
}

let p = new Point();

console.log("p.x =", p.x, "p.y =", p.y);
```

```
p.x = 0 p.y = 0
```

Remarques :

- le constructeur s'appelle `constructor` mais son appel via `new` se fait par le nom de la classe
- on ne doit pas mettre de type de retour
- on ne peut pas appeler le constructeur autrement que via `new`

```
p.constructor();
```

```
p.constructor();
  ^
TypeError: Class constructor Point cannot be invoked without 'new'
```

On peut aussi redéfinir le constructeur en lui passant des paramètres :

```
class Point {
  x: number;
  y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }
}

let p = new Point(1, 2);

console.log("p.x =", p.x, "p.y =", p.y);
```

```
p.x = 1 p.y = 2
```

La déclaration de ce nouveau constructeur empêche l'utilisation d'un constructeur sans paramètre :

```
let p = new Point();
```

```
5   constructor(x: number, y: number) {
                ~~~~~~~~~
An argument for 'x' was not provided.
```

On peut vouloir un constructeur avec des paramètres ainsi qu'un constructeur par défaut. Par exemple pour la classe `Point`, on peut vouloir initialiser `x` et `y` à 0 si aucune valeur n'est donnée. Pour avoir ce comportement, on peut donner des valeurs par défaut aux paramètres :

```
class Point {
  x: number;
  y: number;

  constructor(x = 0, y = 0) {
    this.x = x;
    this.y = y;
  }
}

let p1 = new Point();
let p2 = new Point(1, 2);

console.log("p1.x =", p1.x, "p1.y =", p1.y);
console.log("p2.x =", p2.x, "p2.y =", p2.y);
```

```
p1.x = 0 p1.y = 0
p2.x = 1 p2.y = 2
```

Remarques :

- on peut affecter des valeurs par défaut aux paramètres des fonctions et des méthodes aussi
- si on affecte une valeur par défaut, le type est déduit implicitement