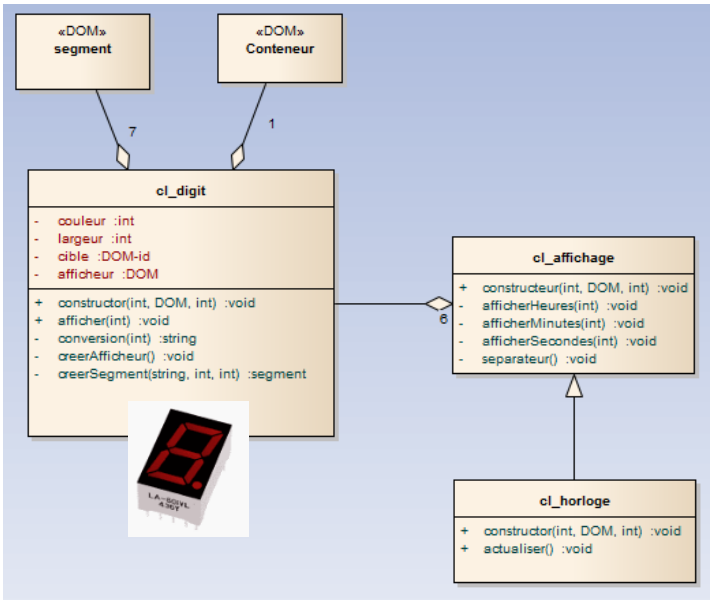
TP JavaScript

Les Objets ES6

But du TP : réaliser une horloge avec des afficheurs 7 segments, sous forme de bibliothèque objet JavaScript. La classe *cl_digit* pourra être réutilisée pour d'autres applications utilisant des affichages de ce type.

Structure de l'application :



Utilisation finale envisagée :

```
<div id="pendule"></div>

<script>
	var h = new cl_horloge(150, 'pendule', 'red');
	setInterval( () => h.actualiser(), 1000);
</script>
```

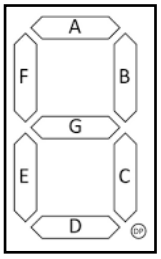
La fonction *setInterval* sert à lancer une action à intervalle régulier. Ici, toutes les 1s (1000ms), on lance la méthode *actualiser* de l'objet créé.

150 : la taille en pixel
'pendule ' : le ID de la balise où afficher l'horloge
'red' : la couleur des chiffres.

1 La classe `cl_digit` :

C'est le plus gros morceau de l'application … Cette classe va créer l'affichage un module 7-segments inspiré ceux utilisés en électronique.

Nous allons utiliser la technique des : Une « conteneur » qui contiendra 7 pour les 7 segments. Les tailles et positions des segments seront fixées par CSS.



☞ Les et le CSS seront générés par le JavaScript !

1.1 Structure vide d'une classe JavaScript (ECMA Script 6):

Ci-dessous le code du « squelette » de notre classe.
On observe la *méthode* **constructor** qui correspond au constructeur de l'objet créé à partir de cette classe. On y trouve le stockage des arguments dans des attributs de l'objet et la définition d'autres attributs. Le mot *this* indique que l'attribut appartient à la classe :

```
class cl_digit {

	constructor (largeur, parentID, couleur) {
		this.cible = document.querySelector('#'+parentID);
		this.largeur = largeur;
		this.couleur = couleur ;
		this.afficheur = document.createElement('div');
	}

	creerAfficheur() {

		// Ajouter les segments ici

		this.cible.appendChild(this.afficheur); // Affichage du résultat
	}

	afficher(num) {
		let motif = this.conversion(num);

		// Ajouter code allumage segment ici
	}

	creerSegment(sens, haut, gauche) {
		var seg = document.createElement('div');

		// Ajouter code réglage ici

		return seg;
	}

	conversion (num) {
		var motif = "";

		return motif;
	}
}
```

NB : L'attribut *this.afficheur* représente une conteneur qui contiendra les des segments.
A la fin de la méthode *creerAfficheur()* on ajoute cette au document à la position définie par *this.cible*, c'est-à-dire une balise html repérée par son *id*. Elle devient visible.

TRAVAIL : Créez un fichier *cl_digit.js* contenant le code ci-dessus.

1.2 La méthode `creerAfficheur()`

On commence par effectuer les réglages communs :

```
		this.afficheur.style.position = 'relative';
		this.afficheur.style.width = this.largeur + 'px';
		this.afficheur.style.height = (this.largeur * 1.8) + 'px';
		this.afficheur.style.margin = '5px';
```

Pour l'opération répétitive de création des 7 segments, on utilise la méthode **`creerSegment`** de la classe :

```
let seg = this.creerSegment('H', '1%', '10%');       //a
this.afficheur.appendChild(seg);
seg = this.creerSegment('V', '5%', '90%');           //b
this.afficheur.appendChild(seg);
seg = this.creerSegment('V', '52%', '90%');          //c
this.afficheur.appendChild(seg);
seg = this.creerSegment('H', '95%', '10%');          //d
this.afficheur.appendChild(seg);
seg = this.creerSegment('V', '52%', '1%');           //e
this.afficheur.appendChild(seg);
seg = this.creerSegment('V', '5%', '1%');            //f
this.afficheur.appendChild(seg);
seg = this.creerSegment('H', '48%', '10%');          //g
this.afficheur.appendChild(seg);
```

TRAVAIL : Complétez la méthode *creerAfficheur* avec les éléments fournis ci-dessus.

1.3 La méthode `creerSegment(sens, haut, gauche)`

Si le sens du segment est Horizontal (H), le réglage de taille est : width=80% et height=5%
Si le sens du segment est Vertical (V), le réglage de taille est : width=8% et height=43%
Haut et gauche représentent les positions du segment dans son conteneur.

Autre réglages communs à un segment :

```
seg.style.position = 'absolute';
seg.style.borderRadius = '25%';
seg.style.backgroundColor = this.couleur;
seg.style.opacity = '0.02';
```

NB: L'opacité à 0.02 permettra de voir le segment même s'il n'est pas « allumé ».

TRAVAIL : Complétez cette méthode qui doit fixer les réglages d'un segment.

1.4 Premiers essais

Voici le **module de test** de cette classe : un fichier HTML (avec ses balises habituelles !).
TRAVAIL : créez un fichier HTML correctement structuré comme on vous a appris …

Ajoutez une cible :

```
<div id="pos1"></div>
```

Ajoutez la bibliothèque dans la zone *head* :

```
<script src="cl_digit.js"></script>
```

Et un script en fin de *body* :

```
var seg1 = new cl_digit(50, 'pos1', 'red');
seg1.creerAfficheur() ;
var seg2 = new cl_digit(50, 'pos1', 'red');
seg2.creerAfficheur() ;
```


Vous devriez voir 2 afficheurs en exécutant cette page HTML !
Pour l'instant, ils s'affichent l'un sous l'autre mais ce sera corrigé plus tard dans la classe *cl_affichage*.

1.5 La méthode `afficher(num)`

L'objectif est d'allumer certains segments en fonction du chiffre à afficher.
C'est la méthode ***conversion()*** qui jouera le rôle de « décodeur ».
Elle renverra un « motif » sous forme d'une suite de 0 et de 1 qui représentent les segments éteints ou allumés.

Pour vos essais, comme la méthode *conversion()* n'existe pas encore, vous pouvez créer la variable « motif » avec une valeur fixe :

Pour le chiffre 2 : `let motif = "1101101";`

Ensuite, il faut parcourir cette variable tableau « motif[ ]» en même temps que l'on parcoure les segments de l'afficheur. Si le motif est à 1, on met l'opacité à 1. Si le motif est à 0, on met l'opacité à 0.02

Quelques indications :

- Pour connaitre l'état souhaité pour le segment A (position 0), on appelle la variable : `motif[0]`
- Pour parcourir les 7 segments (qui n'ont ni attribut *id* ni attribut *name*) nous allons utiliser la ligne suivante :

```
let segments = this.afficheur.querySelectorAll('div');
```

Cette ligne produit un tableau (nommé *segments*) qui contient une liste des contenues dans le conteneur *afficheur* ; donc nos 7 segments, dans le même ordre que lors de la création, c-à-d de A à G.
Par exemple, pour allumer le segment A (position 0 dans le tableau), il faut écrire :

```
segments[0].style.opacity = '1';
```

Il suffit ensuite de parcourir ce tableau avec une boucle ***for*** pour répéter l'opération.

```
for (let i = 0; i < segments.length; i++) {
        // « i » représente le n° de segment, motif[i] l'état souhaité
}
```

TRAVAIL : Complétez la méthode *afficher()* avec les informations ci-dessus, et ajoutez le code à l'intérieur de la boucle ***for*** pour allumer les segments en fonction des valeurs contenues dans *motif*.

Enfin, dernier TRAVAIL : Créez la méthode ***conversion(num)*** et testez le fonctionnement de votre bibliothèque avec le module de test HTML.

2 La classe `cl_affichage()`

Cette classe va organiser les 6 afficheurs pour les présenter sous forme d'une horloge HH :MM :ss

Principe de l'alignement : On utilise le positionnement FLEX :

Extrait du constructeur :

```
let cible = document.querySelector('#'+parentID);
cible.setAttribute('style', 'display:flex; flex-direction:row; align-items: flex-end');
```

Effet : La cible (en mode Flex) affichera son contenu en ligne (row), aligné par le bas (flex-end).

2.1 La méthode `separateur()`

Code de la méthode *separateur* (et d'une méthode accessoire *point*) :

```
separateur (largeur, parentID, couleur) {
        let sep = document.createElement('div');
        sep.setAttribute('style', 'display:flex; flex-direction:column;
                                        align-items: center; justify-content: space-around');
        sep.style.height = (largeur * 1.8) + 'px';
        sep.style.width = (largeur / 3 ) + 'px';
        sep.appendChild( this.point(largeur/10, couleur) );
        sep.appendChild( this.point(largeur/10, couleur) );
        let cible = document.querySelector('#'+parentID);
        cible.appendChild(sep);
}

point (largeur, couleur) {
        var boule = document.createElement('div');
        boule.style.borderRadius = '50%';
        boule.style.backgroundColor = couleur;
        boule.style.width = largeur + 'px';
        boule.style.height = largeur + 'px';
        boule.style.opacity = '1';
        return boule;
}
```

Effet : La *sep* (en mode Flex) affichera son contenu en colonne (column), centré (center) et réparti sur la hauteur (space-around). Les 2 points seront l'un sur l'autre et centrés.

TRAVAIL : créez la classe *cl_affichage* (dans un fichier séparé *cl_affichage.js*) avec les informations ci-dessus. Le constructeur doit contenir la création des 6 digits et des 2 séparateurs.
Modifiez le HTML de test pour vérifier l'affichage.

2.2 Finaliser la classe :

TRAVAIL : Créer les méthodes AfficherHeures/Minutes/Secondes. Ces méthodes font la même chose, sur des digits différents : La variable reçue en argument est séparée en 2 : la partie dizaine et la partie unité.
Exemple de code : Si 'i' est la variable argument :

```
let dizaine = parseInt(i /10);
let unite = i - (10 * dizaine);
```

3 La classe `cl_horloge` :

Cette classe hérite de *cl_affichage*.
La classe *cl_affichage* peut servir pour une horloge, un chronomètre, un compte à rebours…
La classe *cl_horloge* ajoute à *cl_affichage* les fonctions pour gérer l'heure.

Voici le code complet : Le mot clé *extend* indique l'héritage.
Le mot *super* se réfère à la classe mère.
Dans le constructeur, on transmet au constructeur de la classe mère les informations dont elle a besoin pour s'initialiser.

```
class cl_horloge extends cl_affichage {

	constructor (parentID, taille, couleur) {
		super(parentID, taille, couleur);
	}

	actualiser() {
		let now = new Date(); // date actuelle (avec l'heure)

		let i = now.getHours();    // On extrait l'heure exacte
		super.afficherHeures(i);
		i = now.getMinutes();      // On extrait les minutes
		super.afficherMinutes(i);
		i = now.getSeconds();      // On extrait les secondes
		super.afficherSecondes(i);
	}
}
```