PROGRAMMATION WEB

TD JavaScript

CLASSES & MVC

charles.meunier@u-bourgogne.fr

OBJECTIFS

Découvrir :

- l'utilisation des modules ES6,
- les classes en JavaScript,
- le patron de conception Sujet/Observateur,
- l'architecture Modèle/Vue/Contrôleur,

PREPARATION

- Créez un dossier pour le projet.
- Téléchargez et décompresser dans ce dossier les fichiers du TD « TD – Classes & MVC.zip »
- Ouvrez le projet du dossier avec Visual Studio Code.

MODULES ES6

A L'ANCIENNE

- Créez un dossier « models » dans le dossier « js » de votre projet.
- Créez un fichier « counter.js » dans le dossier « model ».
- Dans le fichier « counter.js » :
  - Créez une variable globale « counter » initialisée à 0.

- Créez un dossier « application » dans le dossier « js » de votre projet.
- Créez un fichier « application.js » dans le dossier « application ».
- Dans le fichier « application.js », affichez le contenu de la variable « counter » dans la console de développement une fois que la page a terminé son chargement.

- Ajoutez les scripts « counter.js » et « application.js » au fichier « index.html » du projet.

- Sélectionnez le fichier « index.html » et cliquez sur « Go Live » dans la partie inférieure droite de la fenêtre Visual Studio Code.

- Affichez la console de développement du navigateur et testez le bon fonctionnement.

AVEC LES MODULES

- A présent, retirez le scripte « counter.js » du fichier « index.html » (uniquement le script « counter.js » !).
- Testez à nouveau le fonctionnement. Que constatez-vous ? Comment l'expliquez-vous ?

- Modifiez la balise script associée à « application.js » dans le fichier « index.html » de la façon suivante :

```
<script src="js/application/application.js" type="module"></script>
```

Ceci indique au navigateur que le script « application.js » sera chargé en tant que module et non plus comme un script indépendant.

- Dans le fichier « application.js » ajoutez la ligne suivante au début du script :

```
import { counter } from "./../models/counter.js"
```

Ceci indique que le module « application.js » souhaite importer l'élément « counter » présent dans le module « counter.js ».

Toutefois pour importer un élément depuis un autre module, il faut que ce dernier le permette.

- Modifiez la déclaration de la variable « counter » dans le fichier « counter.js » de la façon suivante :

```
export let counter = 0;
```

Le mot clé « export » indique que l'élément « counter » peut être importé dans d'autre module pour être utilisé. Sans cela, la variable « counter » ne serait pas accessible pour les autres modules.

- Testez le bon fonctionnement.

UN PEU MIEUX

Vous avez dû voir par le passé que les variables globales sont à proscrire autant que possible car elles sont, la plupart du temps, source d'erreurs.

- Dans le fichier « counter.js », créez une fonction « getCounter » qui retourne la valeur de la variable « counter ».
- Exportez cette fonction en ajoutant le mot clé « export » avant le mot clé « function ».
- Retirez le mot clé « export » devant la déclaration de la variable « counter ».
- Testez le fonctionnement de la page.
- Expliquez le résultat obtenu.
- Corrigez le problème.

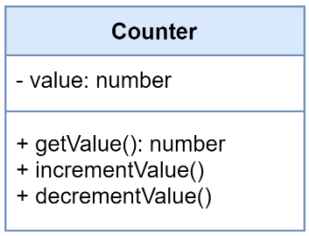
PREMIERE CLASSE

Voici comment créer une classe en JavaScript :

```
//Syntaxe de déclaration d'une classe
class UneClasse
{
    //Déclaration d'un attribut privé (le # devant le nom de l'attribut le rend
    //privé)
    #attrPrive;

    //Déclaration du constructeur
    constructor()
    {
        //this fait référence à l'instance de la classe. Il est nécessaire pour
        //accéder aux attributs internes de la classe et pour faire appel à ses
        //méthodes.
```

```
        this.#attrPrive = 10;
    }

    //Déclaration d'une fonction publique
    fonctionPublique()
    {
        console.log("Une fonction publique");
    }

    //Déclaration d'une fonction privée (le # devant le nom de la fonction la
    //rend privée)
    #fonctionPrivee()
    {
        console.log("Une fonction privée");
    }
}

//Créer une instance de UneClasse :
const uneInstance = new UneClasse();
```

- A partir de la documentation précédente, replacez le code de « counter.js » par celui de la classe suivante :



Note : n'oubliez pas d'initialiser les attributs de la classe dans le constructeur.

- Adaptez le code de « application.js »
- Testez le bon fonctionnement.

GETTER ET SETTER

Il est possible de créer des accesseurs sur des attributs privés offrant une syntaxe plus transparente pour les développeurs :

```
class UneClasse
{
    #attrPrive;
    get attrPrive() { return this.#attrPrive }
    set attrPrive(value) { this.#attrPrive = value }

    …
}

const uneClasse = new UneClasse();
```

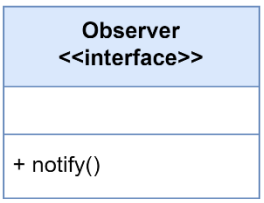
```
//Les getters et setters s'utilisent comme s'il s'agissait de simples attributs
//publiques :

//Utilisation du setter de #attrPrive
uneClasse.attrPrive = 10;

//Utilisateur du getter de #attrPrive
console.log(uneClasse.attrPrive);
```

- Remplacez la méthode « getValue » de la classe « Counter » par un getter.
- Adaptez le code du reste de votre application
- Testez le bon fonctionnement.

SUJET/OBSERVATEUR

Le patron de conception Sujet/Observateur est très utilisé dans les applications disposant d'une interface utilisateur. Il permet d'actualiser les informations affichées à chaque fois que les données du système d'information évoluent.

- Créez un dossier « pattern » dans le dossier « js » de votre projet
- Créez un fichier « observer.js »
- Implémentez l'interface suivante :

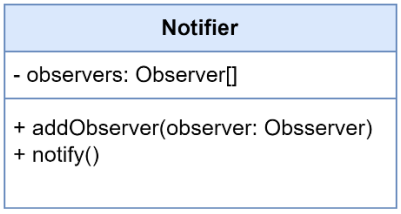


Rappel : en programmation orientée objet, les interfaces sont des classes dont les méthodes sont toutes abstraites.

Note : il n'est pas possible de créer des méthodes abstraites en JavaScript. Pour contourner cette limitation, les interfaces sont créées avec des classes dont les méthodes lèvent des exceptions :

```
class UneClasseAbstraite
{
    methodeAbstraite()
    {
        throw new Error("You have to define this function !");
    }
}
```

- Créez un fichier « notifier.js » dans le dossier « pattern ».
- Implémentez la classe suivante :



- La méthode « addObserver » ajoute l'observateur passé en paramètre au tableau d'observateurs « observers »
- La méthode « notify » appelle la fonction « notify » de chaque observateur du tableau « observers »

Note : n'oubliez pas d'initialiser les attributs de la classe dans le constructeur.

➢ Pensez à exporter les classes Notifier et Observer pour qu'elles puissent être utilisées depuis d'autres modules.

MODELE / VUE / CONTROLEUR

MODELE

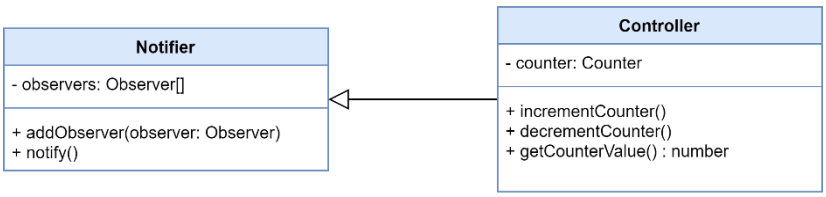
Le modèle correspond au code métier de l'application, c'est-à-dire la base de code propre au système d'information, qui en gère les données et qui est indépendante de la manière dont elles sont représentées.

Ici, la classe « Counter » représentera le code métier de notre application.

CONTROLEUR

Le contrôleur a pour fonction de contrôler le code métier. Il hérite de la classe « Notifier » et notifie ses observateurs à chaque modification des données gérer par le modèle.

➢ Créez un dossier « controllers » dans le dossier « js » de votre projet.
➢ Créez un fichier « controller.js » dans ce dossier.
➢ Implémentez la classe « Controller » qui hérite de « Notifier » :



Note : pour qu'une classe hérite d'une autre classe, on utilise la syntaxe suivante :

```
//La classe UneClassFille hérite de UneClasseMere
class UneClasseFille extends UneClasseMere
{
    constructor()
    {
        //Le constructeur de la classe fille DOIT appeler le constructeur
        //de la classe mère via la fonction super et ce avant tout autre chose
        super();
    }
}
```

Les fonctions « incrementCounter » et « decrementCounter » devront notifier les observateurs du contrôleur que le modèle a changé.

VUE

La vue, quant à elle, s'occupe de faire l'interface entre l'utilisateur et le contrôleur. Elle a en charge l'affichage des informations et la gestion des actions de l'utilisateur (click sur des boutons, saisie de texte, …). En fonction des actions réalisées par l'utilisateur, la vue demande au contrôleur d'effectuer des opérations susceptibles d'impacter le modèle. En retour, le contrôleur notifie la vue que des informations ont changé et la vue se charge d'actualiser l'affichage des données.

- Créez un dossier « views » dans le dossier « js » de votre projet.
- Créez un fichier « view.js » dans ce dossier.
- Implémentez la classe « View » qui implémente l'interface « Observer »

Le constructeur de la classe « View » prend en paramètre un « Controller » auquel la vue va s'abonner.

Le constructeur de la « View » va, par ailleurs, ajouter un gestionnaire d'événements « click » sur le bouton « Incrémenter » qui appellera la méthode « incrementCounter » du contrôleur.

La fonction « notify » de classe « View », héritée de « Observer », devra afficher la valeur du compteur gérer par le contrôleur dans le paragraphe « txt-counter ».

- Dans le fichier « application.js », créez un « Controller » ainsi qu'une « View », une fois que le chargement de la page est terminé.

  Note : pensez à actualiser les imports.

- Testez le bon fonctionnement.

Ici, lorsque l'utilisateur clique sur le bouton « Incrémenter », la vue informe le contrôleur de l'action à réaliser sur le modèle : incrémenter la valeur. Le contrôleur va interagir avec le code métier pour que l'opération souhaitée soit réalisée. Un fois terminée, le contrôleur notifie la vue du changement. Cette dernière actualise alors l'affichage de la page et la nouvelle valeur du compteur apparaît.

- Ajoutez à présent un gestionnaire d'événements « click » sur le bouton « Décrémenter » qui appellera la méthode « decrementCounter » du contrôleur.
- Testez le bon fonctionnement.