INSTITUT LIMAYRAC

Projet : JS Events

Programmation objet

Objectifs	
Mettre en œuvre la programmation événementielle en JavaScript au travers d'un projet de simulation d'un plafonnier de voiture à partir d'une modélisation objet fournie. Langages retenus : HTML et JavaScript	
Prérequis	**Ressources à disposition / Environnement**
Cours sur les processus, chapitre 4, programmation événementielle en JS	Environnement : PC avec éditeur de texte Sublime Text, Notepad++ ou autre éditeur de code. Navigateur : Mozilla Firefox ou Google Chrome.

On vous propose de développer une page HTML / JavaScript qui simule le fonctionnement d'un plafonnier de voiture en appliquant les principes de la programmation orientée objet. Pour cela, un diagramme de classes est fourni dans la page suivante.

Au fur et à mesure de l'avancement du projet, vous pourrez vous appuyer sur cette documentation :

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes

pour comprendre comment sont implémentés les concepts de la programmation orientée objet en JavaScript.

Travail demandé

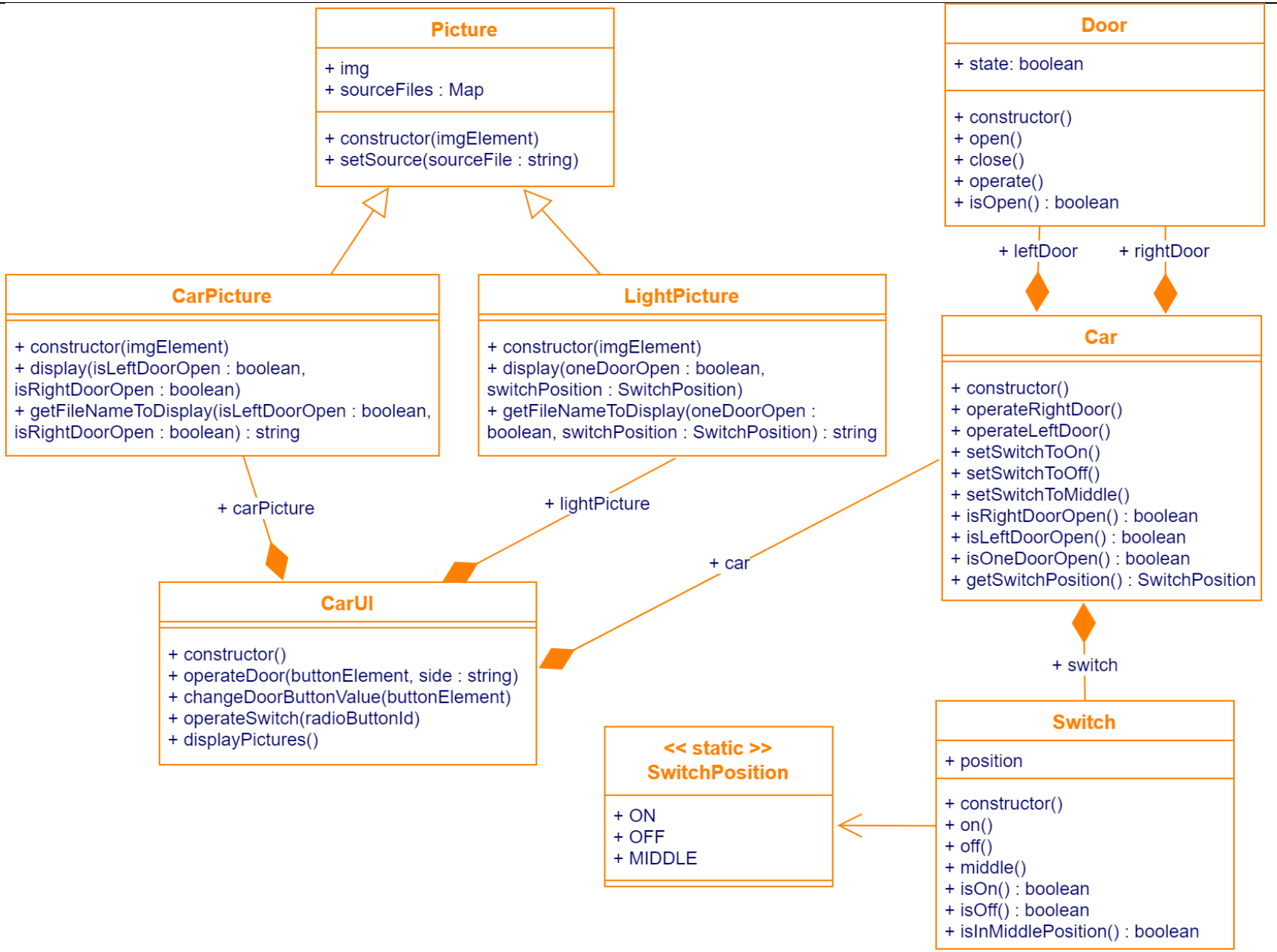
Copier les fichiers `CarUI.html` et `CarUI.css`. Vous complèterez ce fichier `html` au fur et à mesure du codage des classes dans les fichiers `.js`. Il contient déjà l'inclusion du fichier `Door.js` à titre d'exemple.

Copier également le dossier `Pictures` dans le même dossier que les fichiers de la page HTML. Il contient les images du projet que vous devrez utiliser.

Étape 1 : classe Door

Dans la classe `Door`, l'attribut `state` représente l'état de la porte, ouverte ou fermée. Dans l'état initial, la porte est fermée. La méthode `operate()` ouvre la porte si elle est fermée et inversement.

Coder la classe `Door` dans le fichier `Door.js`


Picture
+ img
+ sourceFiles : Map
+ constructor(imgElement)
+ setSource(sourceFile : string)
Door
+ state: boolean
+ constructor()
+ open()
+ close()
+ operate()
+ isOpen() : boolean
+ leftDoor
+ rightDoor
CarPicture
+ constructor(imgElement)
+ display(isLeftDoorOpen : boolean, isRightDoorOpen : boolean)
+ getFileNameToDisplay(isLeftDoorOpen : boolean, isRightDoorOpen : boolean) : string
LightPicture
+ constructor(imgElement)
+ display(oneDoorOpen : boolean, switchPosition : SwitchPosition)
+ getFileNameToDisplay(oneDoorOpen : boolean, switchPosition : SwitchPosition) : string
Car
+ constructor()
+ operateRightDoor()
+ operateLeftDoor()
+ setSwitchToOn()
+ setSwitchToOff()
+ setSwitchToMiddle()
+ isRightDoorOpen() : boolean
+ isLeftDoorOpen() : boolean
+ isOneDoorOpen() : boolean
+ getSwitchPosition() : SwitchPosition
+ carPicture
+ lightPicture
+ car
CarUI
+ constructor()
+ operateDoor(buttonElement, side : string)
+ changeDoorButtonValue(buttonElement)
+ operateSwitch(radioButtonId)
+ displayPictures()
+ switch
<< static >>
SwitchPosition
+ ON
+ OFF
+ MIDDLE
Switch
+ position
+ constructor()
+ on()
+ off()
+ middle()
+ isOn() : boolean
+ isOff() : boolean
+ isInMiddlePosition() : boolean

Étape 2 : classe SwitchPosition

Comme le type énuméré n'existe pas en JavaScript, la classe `SwitchPosition` contient seulement 3 attributs statiques qui représente les 3 positions possibles de l'interrupteur. Elle permettra d'utiliser dans le code sans erreur de syntaxe les valeurs associées aux positions.

Coder la classe `SwitchPosition` dans le fichier `SwitchPosition.js`

Étape 3 : classe Switch

Dans la classe `Switch`, l'attribut `position` doit être initialisé à `OFF`.

Coder la classe `Switch` dans le fichier `Switch.js`

Étape 4 : classe Car

Dans le constructeur de la classe `Car`, il faut créer les attributs `leftDoor`, `rightDoor` et `Switch`. Les méthodes `operateRightDoor()` et `operateLeftDoor()` permettent d'ouvrir chaque porte si elle est fermée et inversement. La méthode `getSwitchPosition()` renvoie une variable de type `SwitchPosition` qui indique la position de l'interrupteur.

Coder la classe `Car` dans le fichier `Car.js`

Étape 5 : classe CarUI (1/2)

On peut commencer à coder la classe `CarUI`. Dans le constructeur, il faut créer l'attribut `car`.

La méthode `changeDoorButtonValue()` prend en paramètre un objet nommé `buttonElement` qui représente le bouton de porte sur lequel on a cliqué. Elle échange le texte de ce bouton de `Ouvrir` à `Fermer` et inversement.

La méthode `operateSwitch()` reçoit en paramètre `radioButtonId` qui est l'identifiant HTML du bouton radio sur lequel on a cliqué. Elle permet de changer l'état de l'interrupteur.

Enfin, la méthode `operateDoor()` prend en paramètre le bouton de porte sur lequel on a cliqué (`buttonElement`) et le côté où cela s'est produit (par exemple `left` ou `right`). Elle appelle la méthode `changeDoorButtonValue()` et ouvre la porte si elle est fermée et inversement.

Coder l'initialisation de l'attribut `car` et les méthode exposées ci-dessus de la classe `CarUI` dans le fichier `CarUI.js`

Étape 6 : classe Picture

La classe `Picture` a un attribut nommé `img`, qu'il faut initialiser avec un objet image du DOM. Cet objet est passé en paramètre au constructeur. Elle a également un attribut nommé `sourceFiles` de type `Map` qui représente sous la forme d'un dictionnaire les différents fichiers source d'une image. Il faut le créer dans le constructeur de `Picture`, mais cette liste sera initialisée dans les classes dérivées `LightPicture` et `CarPicture`.

Pour comprendre le fonctionnement d'un dictionnaire, examiner le premier exemple de : https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Keyed_collections

La méthode `setSource()` met à jour le fichier source de l'attribut `img` avec le paramètre `sourceFile`.

Coder la classe `Picture` dans le fichier `Picture.js`

Étape 7 : classe CarPicture

La classe `CarPicture` hérite de `Picture`. Un exemple d'héritage en JavaScript est fourni ici : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes#Cr%C3%A9er_une_sous-classe_avec_extends

Le paramètre `imgElement` du constructeur est un objet de type image du DOM. Dans le constructeur, il faut initialiser la liste des fichiers source pour l'image en remplissant l'attribut hérité `sourceFiles`. Il faut également initialiser l'image avec l'image de la voiture toutes portes fermées.

La méthode `getFileNameToDisplay()` renvoie le nom du fichier image à utiliser en fonction de ses deux paramètres `isLeftDoorOpen` et `isRightDoorOpen`. Pour cela, elle utilise le dictionnaire `sourceFiles`.

Enfin, la méthode `display()` affiche l'image en modifiant la source déterminée par l'appel à `getFileNameToDisplay()`.

Coder la classe `CarPicture` dans le fichier `CarPicture.js`

Étape 8 : classe LighPicture

La classe `LightPicture` hérite de `Picture` et a un comportement symétrique à `CarPicture`. Seuls les paramètres des méthodes internes sont différents.

Coder la classe `LightPicture` dans le fichier `LightPicture.js`

Étape 9 : classe CarUI (2/2)

On peut terminer le codage de la classe `CarUI`.

Dans le constructeur, il faut créer les deux attributs `lightPicture` et `carPicture` en récupérant depuis le DOM les images du plafonnier et de la voiture. Il faut également cocher le bouton radio de l'interrupteur, en position OFF.

La méthode `displayPictures()` appelle simplement la méthode `display()` de chaque image de la classe.

Il faut à présent ajouter les *Event Handlers* pour donner un comportement dynamique à la page `carUI.html`. On choisit de les ajouter dynamiquement dans le constructeur de la classe à l'aide de la méthode `addEventListener()`.

Pour les deux boutons qui ont pour identifiant HTML `leftDoor` et `rightDoor`, ajouter pour l'événement `click` la méthode `operateDoor()`.

Pour les trois boutons radio qui ont pour identifiant HTML `on`, `off` et `middle`, ajouter pour l'événement `click` la méthode `operateSwitch()`.

Pour cela, il faudra soit attacher explicitement un contexte à une méthode (avec `bind()`) soit utiliser une *Arrow Function* :

https://www.digitalocean.com/community/conceptual_articles/understanding-this-bind-call-and-apply-in-javascript

Pour aller plus loin sur les *Arrow Functions* :
https://www.digitalocean.com/community/tutorials/understanding-arrow-functions-in-javascript

Terminer le codage de la classe `CarUI` comme indiqué ci-dessus, en complétant le fichier `CarUI.js`

Étape 10 : instancier la classe CarUI

Il ne reste plus qu'une seule chose à faire : dans le `body` de la page HTML, instancier la classe `CarUI` une fois la page chargée, c'est-à-dire lors de la survenue de l'événement `onload`.

Le constructeur ajoutera alors les *Event Handlers* aux boutons de la page et elle sera prête à fonctionner.