Travaux pratiques
MVC

Antoine Pigeau*

Mise à jour: 19 décembre 2019

Modélisation MVC

Objectifs

Le sujet est le développement d'une application en JavaScript.

Le but est de modéliser et de réaliser selon l'architecture MVC une application manipulant un entier borné.

Consignes

Il est très important de comprendre et de savoir justifier le rôle précis de chaque objet. La lisibilité du code participant évidemment à sa compréhension, on respectera plus particulièrement les points suivants :

- le code source sera abondamment commenté ;
- les classes, variables, méthodes, fichiers, etc. seront correctement nommés.

Plus généralement, vous utiliserez les conventions d'écriture de code Java de Sun (`http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html`).

Votre application web sera divisée en plusieurs fichiers (.js, .css et .html).

1 Modèle et Observateur

Voici le cahier des charges de la première application à réaliser. L'objectif de l'application est d'incrémenter/-décrémenter un entier $\in [0, 10]$. La valeur de l'entier est fixée entre 0 et 10. L'objet modèle est responsable de son état interne et ne doit donc jamais accepter de prendre une valeur en dehors des bornes définies.

L'objectif est dans un premier temps de bien comprendre le patron de conception Observateur, l'architecture Modèle-Vue-Contrôleur (MVC) étant basée dessus. Pour cet exercice, vous devez implémenter ce patron. Pour bien comprendre l'exercice, il faut d'abord lire toutes les questions au moins une fois.

1. créer une classe abstraite Observable comprenant une liste d'objets Observer, un booléen state (et son accesseur setChanged(), une méthode addObserver(observer) et une méthode

*à partir d'un sujet de Nicolas Normand

notifyObserver() (ces deux méthodes doivent être implémentées). Un objet Observer est prévenu en appelant sa méthode update. Les observateurs ne sont prévenus que si l'état de l'objet est modifié ;

2. créer une classe Observer avec une méthode update(observable, object) ;
3. créer deux classes ModelInteger et PrintObserver héritant respectivement de Observable et Observer. La classe ModelInteger doit contenir un entier x, une méthode setValue(**int**), cette dernière étant appelée par les deux méthodes plus() et moins() permettant de l'incrémenter ou de le décrémenter.
   Pour une instance de ModelInteger, si son entier x est modifié, l'état de l'objet est modifié et il prévient ensuite ses observateurs. Les observateur ne sont prévenus que si l'état de l'objet observé est modifié.
   Une instance de PrintObserver observe un objet de type ModelInteger. Il affiche simplement sur la console la valeur de l'entier de l'objet observé ;
4. instancier deux objets des classes ModelInteger et PrintObserver, ajouter cette dernière instance dans la liste des observateurs de l'instance de ModelInteger. Vérifier qu'à chaque fois que l'entier est modifié, l'action de l'observateur est bien exécutée ;
5. créer un fichier "js/utils.js" contenant votre implémentation du patron observateur, les classes Observable et Observer

2 Interface graphique

L'interface de base est composée de :

1. un bouton "plus" pour incrémenter ;
2. un bouton "moins" pour décrémenter ;
3. un champ de texte qui affiche la valeur et permet sa modification directe.

L'objectif de cette section est d'ajouter une interface graphique pour manipuler votre compteur :

1. créer une page html avec deux boutons plus et moins
2. ajouter un champ texte affichant la valeur du compteur

2.1 Fenêtre et boutons

1. construire votre interface HTML (sans style pour le moment)
2. réaliser les exercices proposés sur le site suivant pour découvrir les possibilités de Flexbox : http://flexboxfroggy.com/
3. créer un dossier "css" et implémenter votre fichier CSS pour positionner les éléments
4. créer une classe View dans un fichier "js/view.js"
5. dans la classe View, créer des attributs sur chacun des composants de la vue (ceux interagissant avec votre modèle). Faire en sorte que vos attributs soient correctement initialisés
6. créer une classe Controler dans un fichier "js/controler.js", prenant en paramètre votre modèle et créant une instance de la classe View
7. dans la classe Controler, ajouter vos observateurs sur les boutons plus et moins, incrémentant et décrémentant la valeur de l'entier.
8. chaque bouton doit être désactivé lorsque l'action qui y est attachée ne peut être réalisée. Ajouter des observateurs sur votre **modèle**, ayant la tâche d'activer/désactiver les boutons. Cet ajout est à faire dans la classe Controler

Dans la suite du projet :

— la classe Controler contiendra toutes les instanciations des actions et la mise en place des liens Observateur/Observer.
— la classe View contiendra toutes les récupérations dans des attributs des composants de la vue

2.2 Champ de texte

1. ajouter un champ de texte à l'interface.
2. implémenter un observateur sur le modèle qui modifie le texte affiché dans le champ de texte. Tester, la valeur affichée dans le champ de texte doit refléter l'état du modèle.
3. modifier la classe contrôleur pour permettre l'édition de la valeur du modèle à partir du champ de texte. L'application doit maintenant être complètement fonctionnelle : la valeur de l'entier peut être modifiée de trois manières différentes ; et quelle que soit la façon d'altérer le modèle, les trois éléments de l'interface modifient leur apparence de manière appropriée.

3 Extension

L'objectif est tout d'abord d'ajouter un deuxième modèle de type entier que l'on va synchroniser avec le premier. Ensuite, un modèle d'activation/désactivation est intégré. Pour finir, des composants graphiques secondaires permettant de réaliser les opérations plus et moins sur les entiers sont ajoutés.

3.1 Plusieurs modèles

L'application va maintenant utiliser deux entiers bornés contraints l'un par rapport à l'autre (la somme des deux entiers doit toujours être égale à la borne maximale, donc 10). Chacun sera associé à une série de vues (boutons et champ de texte).

1. modifier votre code pour générer deux vues indépendantes pour chacun des modèles
2. créer un observateur chargé de synchroniser les valeurs des deux entiers bornés
3. avez vous fait un copier coller de votre modèle ? un copier coller de tous vos précédentes actions ? vos précédents observateurs ? si vous avez répondu "oui" à une de ces questions, il est nécessaire de revoir votre code

3.2 Activation/désactivation

Pour un modèle d'entier, l'application doit maintenant contenir un checkbox permettant d'activer/désactiver les éléments de l'interface permettant de le modifier. Quand l'activation est à False, les boutons + et − seront désactivés (quand d'autres widgets seront ajoutés, il faudra aussi qu'ils soient désactivés).

1. ajouter un label Activation et un checkbox dans votre interface
2. ajouter un nouveau modèle ModelActivation permettant de gérer l'activation/désactivation. Quand le checkbox est coché, les widgets seront désactivés
3. ajouter les actions et update dans votre architecture
4. la gestion des activations/désactivation étant éparpillée dans vos classes update, utiliser le patron Mediator pour centraliser tout le code gérant cette partie du code

3.3 Interface

1. ajouter des éléments d'interfaces :
   — un menu contenant les actions plus et moins sur chaque modèle (deux possibilités ici, ajouter un menu sur chacun de vos panel ou bien faire un menu général permettant de modifier les deux modèles)
   — un curseurs de défilement (Slider)
   — un menu contextuel contenant les actions plus et moins
2. ajouter les bulles d'aides sur chacune des actions

4 Internationalisation et localisation

Tout ce qui est dépendant d'une langue ou d'un pays doit être chargé au démarrage de l'application plutôt que d'être compilé.

L'i18n de votre application sera réalisée à l'aide de la bibliothèque jQuery.i18n.

4.1 Texte

1. identifier toutes les chaînes de caractères qui sont susceptibles d'apparaître à l'écran.
2. donner un nom officiel à chacune de ces chaînes de caractères. Il sera utilisé comme clé dans les fichiers de propriétés, associé à un texte traduit pour la langue locale.
   Ce nom doit donc être compréhensible par un traducteur quelle que soit sa nationalité.
3. écrire les tableaux json de propriétés donnant l'association nom de chaîne → valeur localisée. Si vous utilisez un serveur, il est possible d'écrire ces tableaux dans des fichiers json
4. charger les valeurs des chaînes au démarrage de l'application

4.2 Objet Format

Les chaînes localisées sont souvent destinées à l'affichage, elles peuvent également servir de paramètres de configuration pour la construction d'objets de formatage dont la classe est prédéterminée (e.g. formats de date, de nombre...), mais il peut être nécessaire de déterminer à l'exécution la classe d'une instance.

1. ajouter la date du jour sur votre page web
2. utiliser un objet Format pour afficher la date dans un format texte correspondant à la langue de l'utilisateur