

TP n° 4

Préliminaires

Il vous est demandé de lancer votre page avec Google Chrome. Le plus simple est de lancer Chrome puis « Ctrl-O » et d'ouvrir le fichier HTML directement. On suppose que votre répertoire de travail est un dépôt git. Il vous appartient maintenant de faire des `add/commit` judicieux pour sauvegarder l'historique de vos modifications.

Le but de ce TP est de fournir une première interface rudimentaire, permettant de relier les différents composants développés lors des TP précédents : formules, cellules et modèle. Cette interface sera étendue au fur et à mesure. Le code fourni sert de corrigé pour les TP passés et doit être bien compris.

L'architecture générale du code est la suivante :

- Un fichier `page.html` contenant le code HTML
- Un fichier `tableStyle.css` contenant une feuille de style CSS permettant de donner son apparence au tableur. Il n'est pas nécessaire de comprendre ce fichier pour le TP mais il faudra le comprendre pour pouvoir le faire évoluer par la suite.
- Un fichier `lexer.js` contenant le code de l'analyseur lexical. Il n'est pas nécessaire de comprendre ce code.
- Un fichier `formula.js` contenant le code d'analyse syntaxique (algorithme de Dijkstra) et d'évaluation (visiteurs) des formules. Ce code devra être étendu (dans un autre TP), notamment pour rajouter les fonctions (AVERAGE, SUM, COUNT, ...), les références aux cellules et les intervalles de cellules.
- Un fichier `cell.js` contenant la définition du type `Cell`, **à étendre dans ce TP**.
- Un fichier `tableModel.js` contenant la définition du type `TableModel` et les diverses opérations permettant de manipuler le modèle, **à étendre dans ce TP**.
- Un fichier `tableView.js` contenant le code de la *vue*, i.e. la représentation graphique du modèle. Ce code est **à compléter dans ce TP**.
- Un fichier `tableController.js` contenant le code du contrôleur (gestion des événements utilisateurs). Ce code est **à compléter dans ce TP**.

Le fichier `page.html` contient enfin une fonction `init` initialisant un tableur de dimension 50×50 après le chargement de la page.

1 Cell, reloaded

On souhaite étendre le type `Cell` pour supporter les opérations suivantes :

`Cell.prototype.setView(v)` : place l'objet `v` dans la propriété propre `view` de la cellule.

`Cell.prototype.getView()` : renvoie le contenu de la propriété propre `view` de la cellule.

`Cell.prototype.setFormula(s)` : *parse* la chaîne de caractère `s` pour obtenir une formule `f` (objet `Formula`). Place `f` dans la propriété propre `formula` de la cellule et appelle la propriété `setValue` de l'objet en lui passant le résultat de l'évaluation de `f`.

`Cell.prototype.getFormula()` : renvoie le contenu de la propriété propre `formula` de la cellule.

Enfin, l'opération `Cell.prototype.setValue(v)` est modifiée pour appeler la propriété `notify` de l'objet stocké dans `this.view`, en lui passant en argument la cellule elle-même.

2 TableModel, reloaded

On souhaite rajouter deux opérations au type `TableModel` :

TableModel.prototype.forEachRow(f) : appelle la fonction *f* en lui passant successivement en argument tous les noms de lignes du modèle (1, 2, ...).

TableModel.prototype.forEachCol(f) : appelle la fonction *f* en lui passant successivement en argument tous les noms de colonnes du modèle (A, B, ...).

Remarque : il existe des fonctions de conversions des indices de tableaux vers les noms de ligne et colonnes (ces dernières ont été légèrement modifiées par rapport au corrigé donné précédemment).

3 TableView

Le type *TableView* défini dans le fichier *tableView.js* permet de construire une vue à partir d'un modèle en deux temps. Dans un premier temps, le *constructeur*, *TableView* accepte deux arguments *id* et *tableModel*, où *id* représente l'identifiant d'un élément HTML sous-lequel on va créer l'affichage du tableur et *tableModel* est l'objet de type *TableModel* que l'on souhaite afficher. Le constructeur crée la hiérarchie d'éléments ci-dessous :

```
<div id="test">
  <div id="spreadsheet-div">
    <input type="text"></input> <button>&#10003;</button>
    <table>
    </table>
  </div>
</div>
```

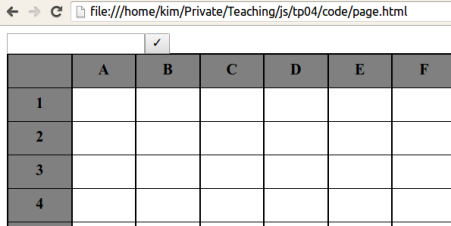
Un élément *div* avec un *id* particulier est créé sous l'élément cible pour servir de conteneur. Sous ce conteneur sont créés trois éléments :

- Une zone de saisie (*input*)
- Un bouton de validation (*button*)
- Une table HTML (*table*), initialement sans contenu.

Chacun de ces éléments, ainsi que le modèle sont stockés dans une propriété propre de l'objet *TableView*, afin de pouvoir y accéder facilement.

Le type *TableView* est ensuite étendu avec un opération *TableView.prototype.createTable()* qui remplit la table HTML en fonction du modèle. Le code HTML généré est le suivant :

```
<table>
  <thead>
    <tr> <th> </th> <th>A</th> <th>B</th> ... </tr>
  </thead>
  <tbody>
    <tr> <td>1</td> <td> </td> <td> </td> ... </tr>
    <tr> <td>2</td> <td> </td> <td> </td> ... </tr>
    <tr> <td>3</td> <td> </td> <td> </td> ... </tr>
    ...
  </tbody>
</table>
```



	A	B	C	D	E	F
1						
2						
3						
4						

Comme on le voit, la table HTML contient d'abord une ligne d'en-tête (*thead*) où chaque case (*th*) excepté la première contient un nom de colonne. La table contient ensuite un élément *tbody*, contenant un ensemble de lignes *tr*. Dans chaque ligne, la première case (*td*) contient le nom de la ligne. Les autres cases représentent le contenu du modèle à la ligne *i*, colonne *j*.

Questions

1. Compléter la méthode `createTable` à l'endroit indiqué pour supprimer tous les éléments se trouvant sous l'élément `table` représentant la table.
2. Compléter la méthode `createTable` à l'endroit indiqué pour ajouter un sous-élément `thead` à l'élément `table`, puis ajouter une ligne (`tr`) à l'élément `thead` et enfin autant de cellules que nécessaire pour afficher les en-têtes de colonne. Il est conseillé d'utiliser l'itérateur `.forEachCol` du `TableModel`. Attention, vous ne devez pas vous préoccuper du style d'affichage dans ce TP, il est fourni et appliqué automatiquement par la feuille de style `tableStyle.css`.
3. Compléter la méthode `createTable` à l'endroit indiqué pour ajouter un sous-élément `tbody`, et autant de lignes que nécessaire (`tr`) à l'élément `tbody`. Chaque ligne contiendra une première cellule (`td`) avec le numéro de la ligne et autant de cellules qu'il y a de colonnes dans la table. Vous pouvez faire une double boucle imbriquée en utilisant `.forEachRow` et `.forEachCol` du modèle.
4. Faites en sorte que chaque élément `td` créé contienne un unique élément texte dont la valeur est celle de la cellule correspondante dans le modèle.
5. Utiliser la technique du *monkey patching* pour ajouter à chaque élément `td` créé (au moment de sa création) :
 - Deux propriétés `row` et `col` contenant respectivement le nom de la ligne et de la colonne qui correspondent à ce `td` dans le modèle
 - Une « méthode » `notify(cell)` prenant une cellule en argument et mettant dans le nœud texte contenu dans le `td` la valeur de la cellule passée en argument.
 - Une « méthode » `isSelected` qui renvoie `true` si le `td` en question possède la classe CSS `selected` et `false` sinon.
 - Une « méthode » `select(b)` qui ajoute la classe `selected` à l'objet `td` si `b` vaut `true` et la retire si `b` vaut `false`.et appeler `cell.setView(td)` pour cet objet `td`, sur la cellule `cell` correspondant dans le modèle. Ainsi, lorsque le contenu d'une cellule du modèle est modifié (au moyen de `cell.setValue(v)`, l'affichage correspondant sera mis à jour (car `setValue` appelle `notify`).

4 TableController

Le constructeur du type `TableController` (fichier `tableController.js`) prend en argument une vue (objet de type `TableView`) et mets en place le code de gestion d'évènement. Lire attentivement le fichier. Le code permettant de sélectionner une case a été rempli, de manière similaire au TP 1.

Questions

1. Compléter le gestionnaire d'évènements pour le click du bouton de validation d'une formule. Lorsque le bouton est cliqué, le gestionnaire récupère l'objet `td` sélectionné (s'il y en a un), récupère le contenu de la zone de saisie (`s`) et récupère la cellule correspondante dans le modèle (`cell`). Si `s` commence par le caractère « = » alors le contenu de la zone de texte (privé du =) est ajouté à la cellule comme une formule (`setFormula`) sinon il est directement ajouté comme valeur. Rattraper les exceptions levées éventuellement par `setFormula` et les afficher dans une boîte de dialogue au moyen de la fonction Javascript `alert`.
2. Faire en sorte que le même gestionnaire d'évènement soit aussi appelé si on presse la touche [enter] du clavier dans la zone de texte.