TP nº 2

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.

Important En début de séance, ouvrir un terminal, entrer :

```
gedit .bashrc
```

Ajouter en fin de fichier les lignes :

```
export EDITOR="gedit -s"
```

Sauver le fichier et relancer la session.

1 Modèle

Le but de cet exercice est de créer une première version du modèle (au sens MVC). On se limite pour l'instant à une feuille de clacul (l'application finale en contiendra plusieurs, affichées dans des onglets). On va donc devoir implémenter deux types d'objets :

`Cell` : représente le contenu d'une cellule

`TableModel` : représente un tableau de cellule

1.1 Git

Pour toute la suite du cours et des TP, nous utiliserons le même répertoire de code. Le logiciel `git` nous permettra de conserver l'historique des modifications.

1. Choisir un répertoire de travail **vide** (par exemple un répertoire nouvellement créé). Exécuter `git init` depuis le terminal, dans ce répertoire.
2. Récupérer les fichier `page.html` depuis la page du cours et le sauver dans ce répertoire (bouton droit, enregistrer sous pour télécharger le fichier).
3. Créer deux fichiers vide `cell.js` et `tableModel.js`
4. Exécuter la commande `git add *` dans le répertoire, suivi de `git commit`. Ajouter, **en dehors des lignes commençant par un #** un message de commit (par exemple « Ajout des premiers fichiers »), sauver et quitter l'éditeur.

1.2 cell.js

1. Ajouter dans le fichier `cell.js` un constructeur pour un objet `Cell` prenant un argument `v` et initialisant une unique propriété `value` à `v`. Si `v` n'est pas défini, alors `value` est initialisé à la chaîne vide.
2. Effectuer `git add cell.js` suivi de `git commit`. Ajouter, **en dehors des lignes commençant par un #** un message de commit (par exemple « Ajout du constructeur de Cell »), sauver et quitter l'éditeur.
3. Ajouter à `Cell` deux méthodes, `getValue()` et `setValue(v)` permettant respectivement de récupérer et mettre à jour la valeur de la propriété `value`.
4. Faire de nouveau un `git add cell.js`, `git commit` en entrant de nouveau un message.

1.3 tableModel.js

On souhaîte encapsuler dans un objet Javascript `TableView` la notion de de feuille de calcul.

1. Ajouter au fichier `tableModel.js` un constructeur `TableModel(w, h)` qui initialise les propriétés `width`, `height` et `cells` de la manière suivante :
   — Si `w` a une valeur fausse ou `w` est inférieur à `1`, alors `this.width` vaut `1` sinon `this.width` vaut la partie entière (inférieure) de `w`
   — Si `h` a une valeur fausse ou `h` est inférieur `1`, alors `this.height` vaut `1` sinon `this.height` vaut la partie entière (inférieure) de `h`
   — `this.cells` contient un tableau de taille `this.height` dont chaque case contient un tableau de taille `this.width` dont chaque case contient un objet `Cell` (contenant la chaîne vide).
   La partie entière inférieure de e s'obtient avec `Math.floor(`e`)`.
2. Ouvrir le fichier `page.html` dans Chrome et lancer la console Javascript (Ctrl-Shift-J). Tester les deux classes crées dans la console (créer une cellule et utiliser ses méthodes, créer plusieurs tables de tailles différentes et visualiser le contenu des tableaux au moyen de la console).
3. Une fois que vos classes sont bien testées, faire un `git add` pour chaque fichier modifié et `git commit`. La commande `git status` vous permet de voir quels sont les fichiers avec des modifications non enregistrées dans `git`.
4. Dans un tableur, les lignes sont usuellement numérotées à partir de 1 et les colonnes sont indexées par des lettres : `A`, `B`, ..., `Z`, `AA`, `AB`, ldots, `AZ`, `BA`, ... Rajouter à votre classe les méthodes suivantes. On prendra bien soin de ne pas **poluer** l'objet global avec des fonctions auxiliaires (cf cours). Vous penserez à faire les `git add/commit` adéquats entre chaque ajout de méthode dans votre fichier.

   `getWidth()` : renvoie le nombre de colonnes de la table

   `getHeight()` : renvoie le nombre de lignes de la table

   `firstLine()` : renvoie le premier numéro de ligne (`1`) sous forme d'une chaîne de caractère si la table n'est pas vide, `""` sinon

   `firstColumn()` : renvoie la première lettre de colonne (`A`) sous forme d'une chaîne de caractère si la table n'est pas vide, `""` sinon

   `lastLine()` : renvoie le dernier numéro de ligne sous forme d'une chaîne de caractère si la table n'est pas vide, `""` sinon

   `lastColumn()` : renvoie le nom de la dernière colonne sous forme d'une chaîne de caractère si la table n'est pas vide, `""` sinon

   `getCell(c, r)` : renvoie la cellule se trouvant la la colonne `c` (donné sous forme de chaîne à partir de `A`) et à la ligne `r` (chaîne contenant un entier à partir de `1`). Si les coordonnées ne sont pas valides, renvoie `undefined`.

   `getCellAtIdx(i, j)` : renvoie la cellule se trouvant la la colonne `i` et à la ligne `j` (`i` et `j` sont des entiers à partir de `0`).

   `insertLineAtIdx(i)` : Insère une ligne à l'indice `i` (à partir de `0`).

   `insertLineBefore(r)` : Insère une ligne avant la ligne `r` donnée comme une chaîne contenant un entier à partir de `1`.

   `insertLineAfter(r)` : Insère une ligne après la ligne `r` donnée comme une chaîne contenant un entier à partir de `1`.

   `insertColumnAtIdx(j)` : Insère une colonne à l'indice `j` (à partir de `0`).

   `insertColumnBefore(c)` : Insère une colonne avant la ligne `r` donnée comme une chaîne contenant un entier à partir de `A`.

   `insertColumnAfter(c)` : Insère une colonne après la ligne `r` donnée comme une chaîne contenant un entier à partir de `A`.