LABO 8 – TABLEAUX
Informatique 1

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

But du laboratoire (4 heures)

1. Dans ce laboratoire, vous allez pouvoir exercer les tableaux ainsi que les objets en réalisant une application de gestion de sièges pour un cinéma. Avec cette application, il sera possible de réserver des sièges, de savoir combien de sièges sont occupés à tout instant, etc… De plus, cette application sera couplée avec un affichage textuel pour pouvoir visualiser les places prises ou non dans la salle. Cela vous permettra d'utiliser des tableaux de différents types dans une application réelle et de réviser les notions liées aux objets et aux classes.
2. La durée estimée pour réaliser ce laboratoire est de **quatre périodes**.
3. Vous pouvez trouver cette donnée sous forme électronique se trouve sur le site web du cours (http://inf1.begincoding.net/). Vous y trouverez également le corrigé de ce labo à la fin de celui-ci.

Partie 1 - Introduction

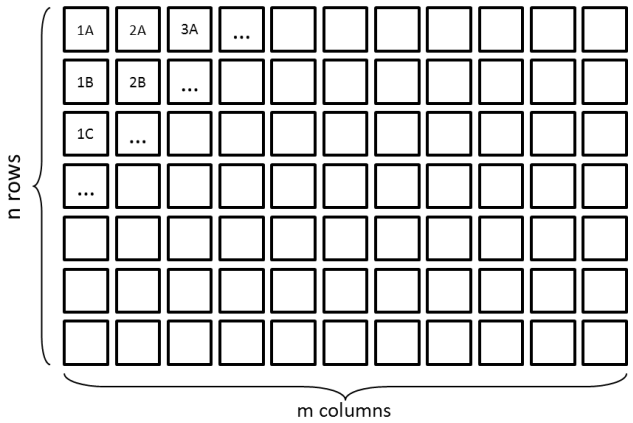
Tâche 0

1. Écrivez une méthode `even_array` prenant comme argument un entier `n`. Cette méthode doit retourner un tableau de `n` entiers ayant comme valeurs les `n` premiers nombres pairs.
2. Appelez cette méthode et affichez le contenu du tableau sur la console.

Partie 2 - Cinéma

Pour modéliser notre cinéma, nous allons utiliser deux classes principales, à savoir la classe *Seat* et la classe *Theater*.

1. La classe *Seat* est relativement simple. Elle possède un attribut, `busy`, de type `boolean`. Cet attribut est *public* et donc modifiable depuis l'extérieur de la classe. Le constructeur d'un *Seat* oblige également à définir la position de ce siège dans le cinéma. Une position de siège est spécifiée par une lettre (correspondant à la rangée – `row`, à stocker comme *char*) et un chiffre (la colonne dans laquelle le siège se trouve – `column` à stocker comme `int`). La position de siège ne peut pas changer durant la durée de vie d'un siège.
2. La classe *Theater* quant à elle contient principalement un tableau d'objets de la classe *Seat* qui représentent la salle (nommé `seats`). Dans cette labo, nous partons du principe que la salle est rectangulaire est qu'elle possède un nombre n de rangées (*jamais plus grand que 20*) et un nombre m de colonnes de sièges disposés comme suit :



Tâche 1

1. Écrivez la représentation UML d'un siège et implémentez la classe *Seat* correspondante.
2. Créez la classe *Theater* avec uniquement le tableau de sièges pour l'instant.
3. Ajoutez maintenant le constructeur de *Theater* qui doit permettre de spécifier la taille de la salle sous la forme de deux entiers. Dans le constructeur vous allez devoir créer les instances de tous les sièges et leur assigner un

numéro de colonne et une lettre pour la ligne. Utilisez pour cela deux boucles imbriquées. A chaque fois que vous créez un siège, affichez ses coordonnées ligne–colonne sous la forme 1A, 4E…

4. Créez la classe *TheaterApplication* qui va contenir le *main*. Créez-y une instance de *Theater* (taille 5x10) et assurez-vous que le constructeur créée bien tous les sièges dans le constructeur correctement, notamment en créant les positions correctement. Afin de tester, créez une méthode `getSeat(int row, int column)` qui retourne le siège se trouvant à la position [0][0] du tableau. Ses coordonnées doivent être 1A. Le siège à la position [2][3] du tableau doit avoir les coordonnées 4C.

Notez au passage comment les coordonnés du tableau commencent à [0][0].

Remarque : notez bien que dans le code nous travaillerons avec les coordonnées du tableau directement et non avec les coordonnées des sièges (par ex. 3B).

Tâche 2

Nous allons maintenant ajouter les méthodes de base permettant de réserver un siège et de voir si des sièges sont occupés ou libres.

1. Ajoutez une méthode `isSeatBusy(int row, int column)` qui retourne `true` si un siège est occupé et `false` autrement. Faites attention à vérifier que les coordonnées passées en paramètres font partie du cinéma. Si tel n'est pas le cas, indiquez par un message que le siège est invalide et retournez `true`.
2. Ajoutez une méthode `occupySeat(int row, int column)` qui rend le siège à la position [x, y] occupé s'il est libre. Si le siège est déjà occupé, indiquez-le directement à l'aide d'un message dans la console et retournez la valeur `false`. Si le siège est libre, la méthode retourne `true`. Dans votre implémentation, utilisez la méthode `isSeatBusy` que vous avez créée ci-dessus.
3. Selon vous, le tableau `seats` doit-être déclaré `public` ou `private` ? Pourquoi ?
   La question sera posée en classe donc préparez votre réponse.

Tâche 3

Nous allons maintenant passer à la représentation textuelle simple du cinéma. Pour ce faire, nous allons utiliser la console.

Implémentez la méthode `toString()` qui affiche un cinéma comme suit (ceci est un cinéma de 4 rangées de 9 sièges, les sièges occupés étant marqués par un X).

```
Theater seats occupation:

   0 1 2 3 4 5 6 7 8
0      X X       X
1
2
3      X
```

Testez le fonctionnement de votre classe *Theater* et des méthodes que vous avez implémentées jusqu'à maintenant.

Tâche 4

1. Ajoutez maintenant une méthode retournant le nombre de sièges occupés dans le cinéma. Nommez cette méthode `numberOfBusySeats()`. Ajoutez également une méthode permettant de connaître le nombre de sièges totaux (occupés ou non) dans la salle. A l'aide de ces deux méthodes, créez une troisième méthode affichant l'occupation de la salle sous la forme :

   ```
   Theater occupation : 12 / 200
   ```

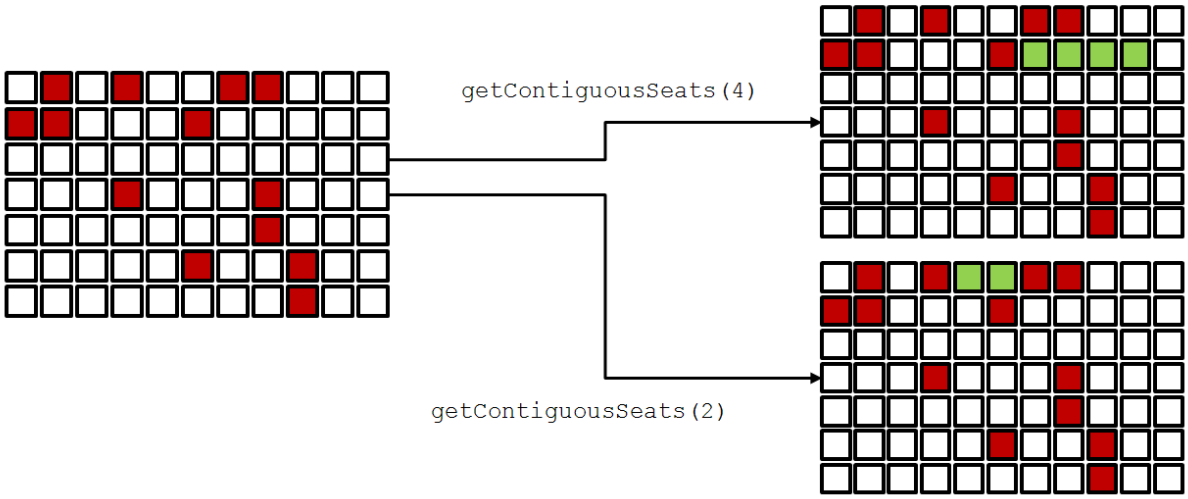
2. Pour tester votre programme complet, définissez un cinéma de 10 rangées de 20 places. Allouez d'abord quelques sièges au hasard à l'aide de la méthode `occupySeat`.
3. Vérifiez que le nombre de places est correct au fur et à mesure de l'exécution. Vous pouvez également afficher le cinéma complet comme suit :

   ```
   Theater myTheater = new Theater(10, 20);
   System.out.println(myTheater);
   ```

Tâche 5 / Facultatif, plus difficile

Maintenant que les mécanismes de base de la salle sont mis en place, nous pouvons ajouter des nouvelles possibilités à la salle.

1. La première chose que nous voulons obtenir c'est un système permettant de réserver automatiquement un nombre *p* de sièges se trouvant sur une MÊME rangée. Pour ce faire, vous devez écrire une méthode `getContiguousSeats` prenant en argument le nombre de sièges à réserver et qui retourne un tableau contenant les sièges réservés. L'état des sièges réservé passe également automatiquement à occupé. Voici un exemple de ce que retourne cette méthode dans un cinéma partiellement rempli :



Un peu de théorie : la référence *null*

Que se passe-t-il s'il n'est pas possible de trouver un nombre de sièges suffisant ? Que retourne donc la méthode qui est censée retourner un vecteur de *Seat* (càd quelque chose de la forme `Seat[]`). La solution est en fait dans ce cas là d'utiliser le type `null`. Il s'agit en fait d'une référence nulle qui est compatible avec toutes les classes. Dans notre exemple, s'il n'est pas possible de trouver des sièges libres, retournez simplement `null` (en écrivant `return null;`).

Que se passe-t-il maintenant lorsque on utilise la méthode `getContiguousSeats(int)` et qu'un assignement de siège n'est pas possible ? La valeur `null` étant retournée, il suffit de vérifier la valeur de retour pour voir si un assignement a été possible ou non :

```
Seat[] reservedSeats = myTheater.getContinuousSeats(1);

if(reservedSeats != null){
	System.out.println("Got the contiguous seats : ");

	// Display the seats location
	for(int i = 0; i < reservedSeats.length; i++){
		System.out.print("- " + reservedSeats[i].column + reservedSeats[i].row + "\n");
	}
}
else
{
	System.out.println("Could not get contiguous seats");
}
```