USTL Licence STA Mention Informatique Semestre 4

UE Programmation Orientée Objet

Examen seconde session – septembre 2009

2 heures - notes et polycopiés de cours/TD/TP autorisés
livres, calculatrices et portables interdits
dictionnaires de langue autorisés pour candidats étrangers

▷ Les deux exercices sont totalement indépendants et l'ordre dans lequel ils apparaissent ne préjuge pas de leur difficulté réelle ou supposée.

Exercice 1 : Le *taquin* est ce jeu formé d'un rectangle de n sur p emplacements carrés. Tous ces emplacements sauf un, que nous appellerons "*trou*" par la suite, sont occupés par une tuile.
Une tuile peut glisser verticalement ou horizontalement, à la seule condition qu'elle se trouve à côté du *trou*. On peut considérer que le déplacement d'une telle tuile située à côté du *trou* consiste en fait à déplacer le trou et ainsi à échanger cette tuile avec le *trou*. Il y a donc au maximum quatre directions possibles pour le déplacement du *trou* : vers le haut, le bas, la gauche ou la droite.
Dans la configuration initiale les tuiles forment un *motif particulier*, dans lequel le *trou* occupe la position en "bas à droite". Chaque tuile "porte" donc une partie de ce motif. Les tuiles peuvent donc être numérotées de manière ordonnée en accord avec ce motif particulier, la tuile en haut à gauche du motif a le numéro 0, les tuiles sont ensuite numérotées de manière incrémentale de gauche à droite et de haut en bas (voir Figure 1).

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	T

Figure 1: Numérotation des tuiles pour un taquin 5×4 dans la configuration du motif particulier . Le *trou* est symbolisé par T.

Pour le jeu, les tuiles sont mélangées, c'est-à-dire qu'on réalise une séquence aléatoire de déplacements. Le but du jeu est alors de retrouver le motif particulier initial en réalisant une suite de déplacements de tuiles.

On s'intéresse dans cet exercice à la modélisation de ce jeu. Les types demandés par la suite sont à définir dans un paquetage `taquin`.

Les tuiles sont définies par leur numéro dans la configuration du motif initial. Un objet particulier constant de type `Tuile` appelé `TROU` est défini dans cette classe.

Q 1 . Donnez le code java d'une classe `Tuile` correspondant à cette description.

La classe `Taquin` représente un jeu de taquin. Une instance de cette classe est définie par la largeur et la hauteur du taquin et caractérisée par les tuiles qui le composent, rangées dans un tableau, ainsi que par les coordonnées en ligne et colonne du trou.

Q 2 . Donnez le code java permettant de définir les attributs de la clase `Taquin` ainsi que le code java du constructeur de cette classe qui prend en paramètre la largeur et la hauteur du taquin et qui ordonne les tuiles, et donc les numérote, selon le motif particulier comme décrit précédemment (à la création le taquin est dans la configuration initiale).

Q 3 . Donnez le code java d'une méthode `equals` de la classe `Taquin`, on dira que 2 taquins sont égaux si ils sont de même taille et si leurs tuiles sont dans la même configuration : chaque tuile numérotée est à la même position dans les deux taquins.

Q 4 . Définissez une méthode `nbCasesEnPlace` qui renvoie le nombre de cases qui dans l'état courant du taquin sont à leur place par rapport à la configuration initiale.

Q 5 . Définissez une méthode `bienRange` de la classe `taquin` qui renvoie *vrai* si et seulement si toutes les tuiles du taquin occupent la position qui est prévue se trouve dans la configuration initiale.

Q 6 . Définissez un type énuméré `DirectionDeplacement` qui définit les quatre directions de déplacement possibles mentionnées précédemment pour le *trou*.

Q 7 . Définissez la méthode `deplacement` de la classe `taquin` dont la documentation est :

```
/** Réalise, quand il est possible, le déplacement du trou dans
 *   la direction de deplacement indiquee
 * @param depl la direction du déplacement du trou à réaliser
 * @exception DeplacementImpossibleException si le déplacement
 *   n'est pas possible dans la direction demandée.
 */
```

La classe `DeplacementImpossibleException` est supposée définie dans le paquetage `taquin` et son constructeur ne prend pas de paramètre.

Q 8 . Définissez une méthode `melanger` dans la classe `Taquin` qui consiste à mélanger le taquin en réalisant 100 déplacements du *trou* dans une direction tirée aléatoirement à chaque déplacement (les tentatives de déplacements impossibles ne sont pas comptabilisées dans les 100).

Vous trouverez en annexe un extrait de la documentation de la classe `java.util.Random`.

Q 9 . Définissez un nouveau constructeur de la classe `Taquin` qui permet de créer un taquin à partir de la donnée d'un tableau de tuiles (non nécessairement rangées selon le motif particulier).

Q 10 . Définissez la méthode `successeurs` de la classe `Taquin` qui a pour résultat la liste des objets de la classe `Taquin` qu'il est possible d'obtenir à partir du taquin courant après un seul déplacement du *trou* dans chacune des directions possibles (voir Figure 2).

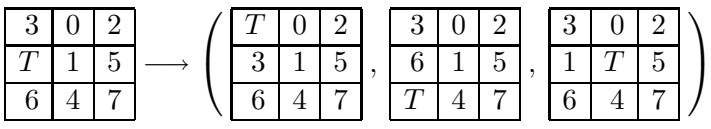
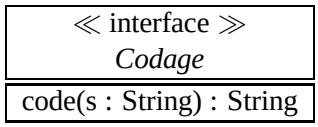


Figure 2: Le taquin de gauche a pour successeurs les trois taquins de droite, atteignables après les déplacements du *trou* vers le haut, le bas et la droite. Le déplacement vers la gauche n'est pas possible.

Exercice 2 : Le codage d'une chaîne de caractères consiste à la convertir en une autre chaîne de caractères. Différents algorithmes de codage sont envisageables. Les codages sont définies par l'interface `Codage` :



On s'intéresse dans la suite de cet exercice à un type de codage en particulier : celui qui consiste à remplacer une lettre par une autre. On va ainsi définir la classe `CodeSimple` qui réalise ce codage.
Cette classe définit la table de conversion des caractères à l'aide d'une **table de hachage** qui associe à un caractère un autre caractère qui en est sa version codée. Le codage d'une chaîne de caractères par ce code consiste donc à remplacer chacun des caractères de cette chaîne par sa version codée. Si un caractère n'est pas présent dans la table, il est laissé inchangé par le code. On suppose le code injectif, c'est-à-dire qu'il n'y a pas deux caractères qui ont pour version codée le même caractère (on ne vérifiera pas ce point dans la suite).

Q 1 . Donnez le code java de la classe `CodeSimple`. Le constructeur de cette classe prendra en paramètre la table du codage des caractères.

Q 2 . Le code “inverse” est le code qui consiste à coder la version codée d’un caractère par le caractère lui-même.

C’est-à-dire que si dans un code le caractère c_1 a pour version codée le caractère c_2, alors dans le code inverse le caractère c_2 a pour version codée le caractère c_1.

Donnez le code java de la méthode de signature :

```
public CodeSimple codeInverse()
```

dont le résultat est le `CodeSimple` qui est le code inverse de `this`.

Annexe

Classe `java.util.Random`

Random() Creates a new random number generator.

int nextInt(int n) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator’s sequence.

Classe `java.lang.String`

char charAt(int index) Returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.