Programmation Objet

Examen

3 heures - documents écrits autorisés
jeudi 31 janvier 2002

- la consultation des annexes est probablement indispensable
- il est possible d'utiliser les réponses à une question non traitée pour résoudre les autres questions

Exercice 1 : Un peu de codage

Le but de cet exercice est de permettre le codage et le décodage d'un texte décrit par une chaîne de caractères ne pouvant être composée que des caractères minuscules correspondant aux 26 lettres de l'alphabet. Les codes utilisés seront très simples puisqu'il s'agit des codes par permutation où chacune des 26 lettres est codée par l'une des 26 lettres et il n'existe pas deux lettres avec le même codage.
Nous définirons 2 classes. La classe `Code` pour représenter le code et la classe `Codage` qui permettra de réaliser le codage d'un texte pour un code donné. Toutes les classes définies dans cet exercice appartiendront à un paquetage `crypto`.

Q 1 . Définition de la classe `Code`

Les lettres seront représentées par des objets `Character`. Le code sera mémorisé dans une table `tableCode` associant un `Character` au `Character` représentant son codage.

Q 1.1. Définissez l'attribut `tableCode` ainsi que le constructeur de la classe `Code`.

Q 1.2. Définissez une méthode `initCode` qui initialise la valeur de `tableCode` en remplissant la table avec les associations caractère ↔ caractère codé tirées aléatoirement.
On pourra générer une liste contenant les 26 caractères et piocher (en supprimant) dedans pour générer la table.
Rappel : le type primitif `char` est un type "entier", il est donc possible d'écrire :

```
for(char c = 'a'; c <= 'z'; c++) {
   System.out.println(c); // affiche chacun des 26 caractères
}
```

Q 1.3. Définissez une méthode `codeLettre` qui pour une lettre (`Character`) en donne la version codée.

Q 1.4. L'inverse d'un code est le code pour lequel `tableCode` définit les associations inverses de caractères. Définissez une méthode `codeInverse` qui retourne l'objet `Code` inverse du code courant.

Q 2 . Définition de la classe `Codage`

Q 2.1. Un objet `Codage` est défini par une instance de `Code`. Définissez les attributs de la classe `Codage` ainsi que le (ou les) constructeur(s) pour cette classe.

Q 2.2. Définissez les méthodes :

- `public String crypte(String texte) { ... }` : qui crypte le texte `texte` à l'aide du code.
- `public String decrypte(String texte) { ... }` : qui décrypte le texte supposé codé par le code (ce qui revient à le coder par le code inverse).

Q 3 . Utilisation

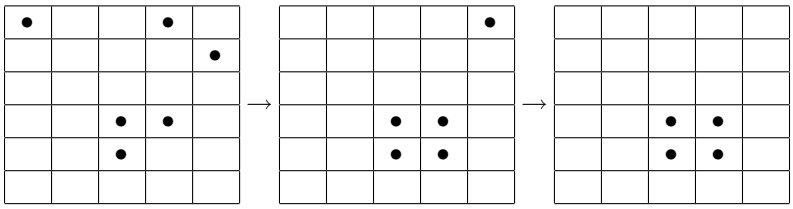
Créez une méthode statique `main` qui crée un code, l'initialise, affiche la version cryptée du texte passé en premier paramètre de la ligne de commande, puis vérifie que la méthode `decrypte` permet de retrouver le texte initial.

Exercice 2 : Jeu de la Vie

Le but de cet exercice est l'écriture d'un programme permettant d'exécuter le célèbre "jeu de la vie".
Le "jeu de la vie" consiste à faire évoluer un ensemble de cellules réparties sur une grille rectangulaire torique. À chaque cycle l'état de toutes les cellules est évalué simultanément et elles évoluent comme suit :

- Une cellule inactive s'active si elle est entourée d'exactement 3 cellules actives.
- Une cellule active ne survit que si elle est entourée de 2 ou 3 cellules actives.

Voici un exemple d'évolution sur 2 cycles (la situation d'arrivée est stable) :



Pour programmer ce problème, nous allons modéliser les cellules par des objets d'une classe `Cell`, la grille de cellules sera représentée par un objet de la classe `Environment` et enfin une classe `GameOfLife` qui gèrera l'évolution de l'environnement et son affichage. Ces classes appartiendront à un paquetage `jeudelavie`.
Pour cette réalisation, nous allons nous appuyer sur le paquetage `grid` déjà utilisé en TD pour le programme `wator`. Les interfaces `grid.Grid` et `grid.GridDisplayer`, ainsi que la classe `grid.Position` sont rappelées en annexe.

Q 1 . la classe `Cell`

Q 1.1. Définir des méthodes `isActive`, `setActive` et `setInactive` qui permettent respectivement, de savoir si la cellule est active, d'activer la cellule et de désactiver la cellule (faire les déclarations annexes nécessaires)

Q 1.2. Définissez un constructeur pour les instances de la classe `Cell`.

Q 1.3. Afin de permettre l'affichage d'une cellule, en mode texte ou graphique, on souhaite disposer des deux méthodes suivantes :

```
/** returns the color of this cell
* @return the color of this cell
*/
public java.awt.Color getColor() {
...
}
/** returns the char associated to this cell
* @return the char associated to this cell
*/
public char getChar() {
...
}
```

On décide qu'une cellule active sera caractérisée par le caractère `'*'` et la couleur `java.awt.Color.blue` et une cellule inactive par le caractère espace (`' '`) et la couleur blanche. Après avoir défini 4 attributs de classe constants `ACTIVE_COLOR`, `INACTIVE_COLOR`, `ACTIVE_CHAR`, `INACTIVE_CHAR`, compléter le code des méthodes `getColor()` et `getChar()`.

Q 2 . la classe `Environment`

La classe `Environment` devra implémenter l'interface `Grid`.

Q 2.1. Un objet de cette classe sera défini par sa hauteur h, sa largeur l et un tableau de $h \times l$ instances de `Cell`.
Définissez les attributs nécessaires à la définition d'une instance de `Environment` ainsi que le (ou les) constructeur(s) nécessaire(s).

Q 2.2. Faites les déclarations nécessaires pour le respect de l'interface `Grid` par la classe `Environment`.

Q 2.3. Définissez la méthode `getCellAtPosition` qui permet de récupérer la cellule de l'environnement à une position (instance de la classe `Position`) donnée.

Q 2.4. Définissez la méthode `setCellAtPosition` qui permet de fixer l'objet cellule (instance de `Cell`) qui se trouve à une position donnée de la grille.

Q 2.5. Définissez la méthode `numberOfActiveNeighbours` qui pour une position donnée, retourne le nombre de cellules actives parmi ses 8 voisines (rappel : l'environnement est un tore !).

Q 3 . la classe `GameOfLife`

Complétez le code suivant :

```
...
public class GameOfLife {

   private Environment environment; // l'environnement torique
   private GridDisplayer displayer; // pour l'affichage

   public GameOfLife (int width, int height) {
      environment = new Environment(width, height);
   }

   /** initialise l'environnement avec des cellules dont
   * l'état initial est tiré aléatoirement
   */
   public void randomInit() {
      ...
   }

   /** fixe le "gridDisplayer"
   */
   public void setGridDisplayer(GridDisplayer displayer) {
      ...
   }
   /** execute le jeu de la vie pendant nbSteps cycles
   * et affiche l'environnement après chaque cycle
   */
   public void execute(int nbSteps) {
      ...
   }

   /** execute un seul cycle du jeu de la vie.  Toutes les
   * cellules évoluent simultanément.
   */
   private void executeOneCycle() {
      ...
   }
}// GameOfLife
```

On attire votre attention sur la méthode `executeOneCycle` dans laquelle il faut gérer l'évolution **simultanée** de toutes les cellules de l'environnement.

Q 4 . On dispose d'une classe `GraphicalGridDisplayer` implémentant l'interface `GridDisplayer` et permettant la visualisation d'une grille dans une fenêtre graphique.

Écrire une méthode `main` qui permet d'exécuter (et de visualiser l'exécution) pendant 100 cycles d'un jeu de la vie sur une grille 30×30.

Annexes

Extraits de la documentation de l'API java. Tout ce qui est présenté ici n'est pas forcément nécessaire à la résolution des exercices. De même, certaines méthodes présentées en cours ou largement utilisées en TD peuvent ne pas avoir été reprises.

classe `java.lang.Character`

`Character(char value)` Constructs a Character object and initializes it so that it represents the primitive value argument.]

`char charValue()` Returns the value of this Character object.

classe `java.lang.Math`

`static double random()` Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

classe `java.lang.StringBuffer`

`StringBuffer(String str)` Constructs a string buffer so that it represents the same sequence of characters as the string argument; in other words, the initial contents of the string buffer is a copy of the argument string.

`StringBuffer append(char c)` Appends the string representation of the char argument to this string buffer.

`String toString()` Converts to a string representing the data in this string buffer.

classe `java.util.ArrayList`

`boolean add(Object o)` Appends the specified element to the end of this list.

`Object get(int index)` Returns the element at the specified position (`index`) in this list.

`Object remove(int index)` Removes the element at the specified position (`index`) in this list. Removed element is returned.

classe `java.util.HashMap`

`boolean containsKey(Object key)` Returns true if this map contains a mapping for the specified key.

`boolean containsValue(Object value)` Returns true if this map maps one or more keys to the specified value.

`Object get(Object key)` Returns the value to which this map maps the specified key.

`Set keySet()` Returns a set view of the keys contained in this map.

`Object put(Object key, Object value)` Associates the specified value with the specified key in this map.

`Object remove(Object key)` Removes the mapping for this key from this map if present.

`int size()` Returns the number of key-value mappings in this map.

`Collection values()` Returns a collection view of the values contained in this map.

fichier `Grid.java`

```
package grid;
/** a Grid is an area of width times height boxes.  Each box can have a
* specific color and a char to be displaid in it */
public interface Grid {

   /** returns the width of the grid
   * @return the width of the grid
   */
   public int getWidth();
   /** returns the height of the grid
   * @return the height of the grid
   */
   public int getHeight();
   /** returns the color of the box at position p
   * @param p the position
   * @return the color
   */
   public java.awt.Color getColorAtPosition(Position p);
   /** returns the char to be displaid in the box at position p
   * @param p the position
   * @return the char to be displaid
   */
   public char getCharAtPosition(Position p);
}// Grid
```

fichier `GridDisplayer.java`

```
package grid;

/** interface of a displayer of a Grid object
*/
public interface GridDisplayer {
   /** displays the grid
   * @param grid the grid to be displaid
   * @param msg a msg
   */
   public void display(Grid grid, String msg);
}// GridDisplayer
```

fichier `Position.java`

```
package grid;

/** a position in a grid
*/
public class Position {

   private int x;
   private int y;

   public Position (int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() { return x; }
   public int getY() { return y; }
   public String toString() {
      return "("+x+","+y+")";
   }
}// Position
```