Licence Info 3éme année - module Programmation Objet 2

Examen – janvier 2013 – session 2 – durée : 2h

Documents autorisés : les polycopiés du cours.

Partie A : Gestion d'une dictionnaire *(environ 10 points)*

Nous reprenons l'exercice de la gestion d'un dictionnaire. Cette application est composée d'une classe Dictionnaire.java comme modèle, une classe VueGraphik comme vue et une classe ContrôleurGraphik comme contrôleur.

Nous donnons le code de la classe DictionnaireGestion1.java :

```
import java.util.*;
public class DictionnaireGestion1 {
  public static void main(String[] args) {
    final Dictionnaire dict = new Dictionnaire();
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
   ControleurGraphik controlGra = new ControleurGraphik(dict);
   VueGraphik vueGra = new VueGraphik(dict);
   dict.addObserver(vueGra) ;
   JFrame frame = new JFrame("vue graphique");
   JPanel panel = new JPanel();
   panel.add(controlGra);
   // The splitPane is split horizontally – the two components panel and vueGra
appear side by side
   JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, panel,
vueGra);
   // The user can collapse (and then expand) either of the components with a
single click
   splitPane.setOneTouchExpandable(true);
   // Make 50% of the space go to left component panel
   splitPane.setDividerLocation(0.5);
   frame.getContentPane().add(splitPane);
   frame.setSize(200,100);
   frame.setVisible(true);
      }
   });
  }
}
```

Nous donnons le code de la classe Dictionnaire.java :

```
import java.util.*;
public class Dictionnaire extends Observable
{
  /** la liste des mots*/
  private List<String> dict;
  /** constructeur sans parametre */
  public Dictionnaire() {
    dict = new ArrayList<String>();
  }
  /** ajoute un mot s'il n'existe pas */
```

```
  public void add(String mot) {
    if ((mot != null) && !mot.equals("") && !dict.contains(mot)) {
      int max = dict.size();
      int i=0;
      while ((i<max) && (mot.compareTo((String)dict.get(i)) >0))
        i++ ;
      dict.add(i, mot);
      this.setChanged();
      this.notifyObservers(this.toArray());
    }
  }
  /** fournit une copie tableau */
  public String[] toArray() {
    String [] t = {};
    return dict.toArray(t);
  }
  /** supprime un mot s'il existe */
  public void remove(String mot) {
    if (mot != null)
      if (dict.remove(mot)) {
        this.setChanged();
        this.notifyObservers(this.toArray());
      }
  }
}
```

Question A1 :

Ecrire la classe ControleurConsole.java qui permet un contrôle du modèle à partir du clavier :
elle affiche le menu suivant : add mot, del mot, q(uitter)
puis attend les commandes que tape l'utilisateur.

Question A2 :

Ecrire la classe VueConsole.java qui fournit une vue en console du modèle. Elle affiche la liste des mots du dictionnaire ainsi, si le dictionnaire contient abc, bonjour, def :

```
----------
abc
bonjour
def
----------
```

Question A3 :

Modifier la classe DictionnaireGestion1.java en ajoutant le contrôleur et la vue en console. Nous donnons une exécution de cette application comme le suivant :

```
$java DictionnaireGestion2
----------
----------
add mot, del mot, q(uitter)
add abc
----------
abc
----------
add mot, del mot, q(uitter)
add def
```

```
----------
abc
def
----------
add mot, del mot, q(uitter)
add bonjour
----------
abc
bonjour
def
----------
add mot, del mot, q(uitter)
del abc
----------
bonjour
def
----------
add mot, del mot, q(uitter)
```

Partie B : Généricité, Exception et Thread *(environ 10 points)*

Question B.1 : Voici un bouton clignotant : au premier clic, il se met à clignoter du jaune au bleu selon le nombre de secondes donné à son instanciation.
Voici le code du bouton clignotant :

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class BoutonClignotant extends JButton {
  private int periode;
  private class Clignotement implements Runnable {
    public void run() {
      for (;;)
        try {
          BoutonClignotant.this.setBackground(Color.BLUE);
          Thread.sleep(periode);
          BoutonClignotant.this.setBackground(Color.YELLOW);
          Thread.sleep(periode);
        } catch (InterruptedException e) { }
    }
  };
  public BoutonClignotant(int periode, String texte) {
    super(texte);
    this.periode = periode * 1000;
    this.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          // a completer ….
        }
      }
    );
  }
}
```

Voici une classe de test de notre bouton clignotant :

```
import java.awt.*;
```

```
import javax.swing.*;
public class Test extends JFrame {
  public static void main (String [] args)  {
    SwingUtilities.invokeLater( new Runnable() {
      public void run() { new Test(); }
    });
  }
  public Test() {
    super("test du bouton clignotant");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel panel = new JPanel();
    JLabel labelFixeVert = new JLabel("  vert  ");
    labelFixeVert.setForeground(Color.GREEN);
    panel.add(labelFixeVert);
    BoutonClignotant bouton = new BoutonClignotant(2, "attention");
    panel.add(bouton);
    JLabel labelFixeRouge = new JLabel("  rouge  ");
    labelFixeRouge.setForeground(Color.RED);
    panel.add(labelFixeRouge);
    this.setContentPane(panel);
    pack();
    setVisible(true);
  }
}
```

Les questions sont indépendantes :

1. La classe BoutonClignotant n'est pas complètement écrite.
2. Complétez le code manquant.
3. La classe interne Clignotement expose un risque d'inter blocage. Veuillez le corriger.

Question B.2 : Nous avons besoin d'une classe générique qui permette d'indexer n'importe classe d'élément par des clés : une sorte de super tableau associatif.
La seule contrainte sur le type des clefs est qu'elles soient comparable.
Les éléments sont accessibles par leur clef. 2 éléments ne peuvent pas avoir la même clef.

Voici un programme d'utilisation de ce "super tableau associatif" : c'est le tableau des nombre de jour de chaque mois de l'année ; la clef est le nom du mois.

```
public class Test2 {
  public static void main (String [] args)  {
    Indexation<String,Integer> nombreJourParMois = new
Indexation<String,Integer>();
    nombreJourParMois.add(new ItemIndexable<String,Integer>("janvier", 31));
    nombreJourParMois.add(new ItemIndexable<String,Integer>("février", 28));
    nombreJourParMois.add(new ItemIndexable<String,Integer>("mars", 31));
    nombreJourParMois.add(new ItemIndexable<String,Integer>("avril", 30));
    nombreJourParMois.add(new ItemIndexable<String,Integer>("mai", 31));
    // ...
    System.out.println("nombreJourParMois : "+nombreJourParMois);
    System.out.println("nombreJourParMois.get(\"mars\")) : "
                       +nombreJourParMois.get("mars"));
```

```
    System.out.println("nombreJourParMois.remove(\"février\"))");
    nombreJourParMois.remove("février");
    System.out.println("nombreJourParMois.add(\"février, 29\"))");
    nombreJourParMois.add(new ItemIndexable<String,Integer>("février", 28));
    System.out.println("nombreJourParMois.get(\"février\")) : "
                       +nombreJourParMois.get("février"));
  }
}
```

Voici le résultat de l'exécution :

```
$ java Test2
nombreJourParMois : [ , ( avril , 30 ), ( février , 28 ), ( janvier , 31 ),
( mai , 31 ), ( mars , 31 ) ]
nombreJourParMois.get("mars")) : 31
nombreJourParMois.remove("février"))
nombreJourParMois.add("février, 29"))
nombreJourParMois.get("février")) : 28
```

Voici la classe ItemExecutable :

```
public class ItemIndexable<T extends Comparable, U>
{
  private T clef;
  private U element;
  public ItemIndexable(T clef, U element) throws NullPointerException {
    if ((clef == null) || (element == null))
      throw new NullPointerException("clef ou élément null");
    this.clef = clef;
    this.element = element;
  }
  public T getClef() {
    return this.clef;
  }
  public U getElement() {
    return this.element;
  }
  public String toString() {
    return "( "+this.clef.toString()+" , "+this.element.toString()+" )";
  }
}
```

Voici la classe Indexation :
L'implémentation se fait avec une liste ordonnée des "itemIndexables" selon l'ordre des clefs. Pour la recherche, l'ajout et la suppression, un algorithme de recherche dichotomique est mis en œuvre dans la méthode private pos(clef). La méthode pos est faite pour vous simplifier la programmation !

```
1  import java.util.*;
2  public class Indexation....
3  {
4    private .... liste;
5    public Indexation() {
6      ...
7    }
8    public String toString() {
9      ...
```

```
10
11    }
12    /** cherche l'index de la clef
13     *  @return un entier positif ou nul si trouvé
14     *   sinon un négatif dont la valeur absolue est l'index d'insertion
éventuelle + 1
15     */
16    private int pos(T clef) throws NullPointerException {
17      if (clef == null)
18        throw new NullPointerException("clef nulle");
19      int fin = liste.size() - 1;
20      if (fin == -1)
21        return -1;
22      int debut = 0;
23      int milieu;
24      T clefMilieu;
25      do {
26        milieu = (debut + fin) / 2;
27        clefMilieu = liste.get(milieu).getClef();
28        if (clefMilieu.equals(clef))
29          return milieu;
30        else
31          if (clef.compareTo(clefMilieu) > 0)
32            debut = milieu + 1;
33          else
34            fin = milieu - 1;
35      } while (debut <= fin);
36      return -debut-1;
37    }
38    ...
39
40
```

Les questions pour compléter la classe Indexation:

1. ligne 2 : indiquer la classe générique
2. ligne 4 : quel type de liste ?
3. Ligne 6 : compléter le constructeur
4. lignes 9-10 : compléter la méthode toString()
5. ajouter les méthodes get(), add(), remove() que nous avons utilisez dans la classe Test2.