Module Complément Java Juin 2007
Licence Informatique 3éme année

Examen 2ème session

Durée : 2 heures
Documents autorisés : les polycopiés du cours

Probléme A

Ci-dessous une classe représentant une liste de mots :

```
01 public class ModeleListeMots{
02   private Vector vect;
03   public ModeleListeMots() {
04     vect = new Vector();
05   }
06   public ModeleListeMots(String[]mots) {
07     Arrays.sort(mots);
08     int max = mots.length;
09     vect = new Vector(max);
10     for (int i=0; i<max; i++)
11       vect.add(mots[i]);
12   }
13   public void add(String mot) {
14     if (!vect.contains(mot)) {
15       int max = vect.size();int i=0;
16        while ((i<max) && (mot.compareTo((String) vect.get(i)) >
0)) i++;
17       vect.add(i, mot);
18     }
19   }
20   public String[] toArray() {
21     int max = vect.size(); String[]result = new String[max];
22     for (int i=0; i<max; i++) result[i]=(String)vect.get(i);
23     return result;
24   }
25   public void remove(String mot) {vect.remove(mot);}
26 }
```

Ci-dessous une classe VueConsole qui représente une vue sur un objet ModeleListeMots. Cette vue affiche en ligne de commande la liste des mots.

```
import java.util.*;
public class VueConsole implements Observer {
  public void update(Observable source, Object donnees) {
    if ((source instanceof ModeleListeMots) && (donnees != null)
        && (donnees instanceof String[]))
      System.out.println("----------");
      int max = ((String[])donnees).length;
      for (int i=0; i<max; i++)
        System.out.println(((String[])donnees)[i]);
      System.out.println("----------");
  }
}
```

Ci-dessous une classe ControleurConsole qui permet d'ajouter ou d'enlever des mots dans une liste via la ligne de commande.

```
public class ControleurConsole {
  private ModeleListeMots modele;
  public ControleurConsole(ModeleListeMots modele) {
    this.modele = modele;
  }
  public void run() {
    String rep; boolean encore;
    do {
      rep = Clavier.lireLigne().trim();
      encore = this.operer(rep);
    } while (encore);
  }
  private boolean operer(String control) {
    boolean continuer = true;
    if (control == null)
      return continuer;
    else if (control.indexOf("del ") == 0) {
      String mot = control.substring(4,control.length()).trim();
      if (mot != null)
        this.modele.remove(mot);
    } else if (control.indexOf("add ") == 0) {
      String mot = control.substring(4,control.length()).trim();
      if (mot != null)
        this.modele.add(mot);
    } else  if (control.equals("q"))
      continuer = false;
    return continuer;
  }
}
```

<u>Questions du problème A</u> :

1- Modifier la classe ModeleListeMots pour qu'elle puisse fonctionner avec la VueConsole. Ecrire un programme principal qui crée une instance de ModeleListeMots, y ajoute des mots et en enlève de sorte que l'affichage soit géré par une instance de VueConsole.
2- Modifier le programme principal de la question 1 pour prendre en compte le controleur console : ce programme principal doit créer des instances de ModeleListeMots, VueConsole et ControleurConsole et faire en sorte qu'on puisse ajouter des mots ou en enlever en ligne de commande.

<u>Problème B</u> :

6 philosophes se réunissent pour un banquet philosophique, autour d'une table ronde. Ils ont chacun une assiette mais le philosophe invitant n'a que 3 fourchettes, il pose donc les fourchettes de manière à ce que chaque philosophe ait accès à une fourchette. Chaque philosophe doit donc attendre que la fourchette qu'il peut atteindre soit disponible pour manger. La classe Fourchette suivante modélise les fourchettes, qui peuvent être prises ou libérées par les philosophes.

```
public class Fourchette{
  private boolean utilisee=false;
  public void prend(){
    this.utilisee = true;
  }
  public void libere(){
    this.utilisee = false;
  }
}
```

La classe suivante modélise les philosophes et contient un programme principal.

```
public class Philosophe extends Thread {
  private int id;
  private Fourchette f;
  public Philosophe(int id,Fourchette f){
    this.f=f;this.id=id;
  }
  public void pense(){
    System.out.println("Le philosophe " + id + " pense");
    try{this.sleep(1000);}
    catch(InterruptedException e){}
  }
  public void aFaim(){
    System.out.println("Le philosophe " + id + " a faim");
    f.prend();
  }
  public void mange(){
    System.out.println("Le philosophe " + id + " mange");
    try{this.sleep(1000);}
    catch(InterruptedException e){}
    f.libere();
  }
  public void run(){
    while(true){pense(); aFaim(); mange();}
  }
  public static void main( String [] argv ){
    Fourchette[] fTab = new Fourchette[3];
    for(int i = 0;i<3;i++) fTab[i] = new Fourchette();
    for(int i = 0;i<6;i++){
      Philosophe p = new Philosophe(i,fTab[(i+1)%3]);
      p.start();
    }
  }
}
```

Question du problème B : Tel quel, ce programme marche mal, les philosophes utilisent en même temps les mêmes fourchettes et le banquet risque de tourner au pugilat. Proposer des modifications pour que le banquet se déroule correctement.

<u>Problème C</u> :

Voici ci-dessous le code d'un composant graphique qui affiche un nuancier et permet de sélectionner une couleur.

```
01 /** Cette classe implémente un nuancier permettant, par un clic
02 de souris, de sélectionner une couleur. */
03 public class ComposantNuancier extends Canvas{
04  private int dim;
05  private Color currentColor;
06  private Rectangle[] carreaux;
07  private Color[] couleurCarreaux;
08  private LinkedList ob;
09  /** Constructeur. dim est la résolution du nuancier. */
10  public  ComposantNuancier(int dim){
11    this.dim = dim;
12    this.ob = new LinkedList();
13    this.currentColor = Color.BLACK;
14    this.carreaux  = new Rectangle[dim * dim];
15    this.couleurCarreaux = new Color[dim * dim];
16    for(int i = 0; i < dim; i++){
17      int j;
18      for(j = 0; j < dim / 2; j++)
19       couleurCarreaux[i * dim + j] = Color.getHSBColor((float)i
                                      / dim,1,(float)j / dim * 2);
20      for( ; j < dim; j++)
21       couleurCarreaux[i * dim + j] = Color.getHSBColor((float)i
                        / dim,(float)(dim - 1 - j) / dim * 2,1);
22    }
23    this.enableEvents(AWTEvent.MOUSE_EVENT_MASK);
24  }
25  /** Méthode appelée quand le composant change de dimension */
26  public void setBounds(int x, int y, int width, int height){
27    super.setBounds(x, y, width, height);
28    int largeur = this.getSize().width  / dim;
29    int hauteur = this.getSize().height / dim;
30    for(int i = 0; i < dim; i++)
31      for(int j = 0; j < dim; j++)
32        carreaux[i * dim + j] = new Rectangle(largeur * i,
                                  hauteur * j, largeur, hauteur);
33  }
34
35  public void paint(Graphics gc){
36    for(int i = 0; i < carreaux.length; i++){
37      gc.setColor(couleurCarreaux[i]);
38      gc.fillRect(carreaux[i].x,carreaux[i].y,carreaux[i].width,
                                           carreaux[i].height);
39    }
40  }
41
42  public void processMouseEvent(MouseEvent e){
43    if(e.getID() == MouseEvent.MOUSE_CLICKED){
44      for (int i = 0; i < carreaux.length; i++){
45        if (carreaux[i].contains(e.getX(),e.getY())){
46          this.currentColor = couleurCarreaux[i];
47        }
48      }
49    }
50  }
51
52}
```

Voici maintenant le code d'un modèle de couleur.

```
01  public class MyColorModel extends Observable implements
Observer {
02
03  private int red;
04  private int green;
05  private int blue;
06
07  public MyColorModel(){
08    this.red = 100;
09    this.green = 100;
10    this.blue = 100;
11  }
12
13            public      Color      getColor(){return      new
Color(this.red,this.green,this.blue);}
14
15  public void setColor(Color c){
16   this.red = c.getRed();
17     this.green = c.getGreen();
18     this.blue = c.getBlue();
19     this.setChanged();
20     this.notifyObservers(this.getColor());
21  }
22
23  public void update(Observable o,Object ob){
24    if(ob instanceof Color){
25      this.setColor((Color) ob);
26    }
27  }
28}
```

Questions du problème C :

1- Faites du ComposantNuancier un Java Bean dont la propriété currentColor sera en lecture seulement (read only) et liée (bound).
2- Proposez des modifications de la classe MyColorModel pour que la modification de la couleur dans le ComposantNuancier puisse être répercutée dans le modèle.

Pour ces deux questions, indiquez quelles lignes sont modifiées ou ajoutées dans le code fourni.

Correction problème A :

Question 1 :

Ajouter en ligne 1 : `extends Observable`
Ajouter entre les lignes 17/18 et à la ligne 25 dans le corps de la méthode remove `this.setChanged(); this.notifyObservers(this.toArray());`

Question 2 :

```
public class MVC1 {
   public static void main(String[] args) {
     ModeleListeMots modele = new ModeleListeMots(args);
     VueConsole vue = new VueConsole();
     modele.addObserver(vue);
     modele.add("toto");
     modele.add("titi");
     modele.remove("toto");
     modele.remove("truc");  }
}
```

Question 3 :

```
public class MVC1 {
  public static void main(String[] args) {
    ModeleListeMots modele = new ModeleListeMots(args);
    ControleurConsole controleur = new ControleurConsole(modele);
    VueConsole vue = new VueConsole();
    modele.addObserver(vue);
    controleur.run();
  }
}
```

Correction problème B :

```
public class Fourchette{
  private boolean utilisee=false;
  public void prend(){
    synchronized(this){
     if(utilisee){
          try{ this.wait();}catch(InterruptedException e){}
     this.utilisee = true;
     }
}
  }
  public void libere(){
    synchronized(this){
        this.utilisee = false;
     this.notify();
}
  }
}
```

Correction problème C :

Question 1 :

Ajouter ligne 03 : `implements Serializable`
Ajouter ligne 10 : `private PropertyChangeSupport pcs;`
Ajouter ligne 25 : `pcs = new PropertyChangeSupport(this);`
Ajouter ligne 48 :
```
pcs.firePropertyChange("currentColor",this.currentColor,couleurCar
reaux[i]);
```
Ajouter ligne 53 :
```
public void
addPropertyChangeListener(PropertyChangeListener ecouteur) {
          pcs.addPropertyChangeListener(ecouteur);
     }

     public void
removePropertyChangeListener(PropertyChangeListener ecouteur) {
          pcs.removePropertyChangeListener(ecouteur);
     }
```
Ajouter ligne 51 :
```
public Color getCurrentColor(){return
this.currentColor;}
```

Question 2 :

Ajouter ligne 01 : `,PropertyChangeListener`
Ajouter ligne 27 :
```
public void propertyChange(PropertyChangeEvent e) {
     this.setColor((Color) e.getNewValue());
  }
```