Gestion des événements

(suite)

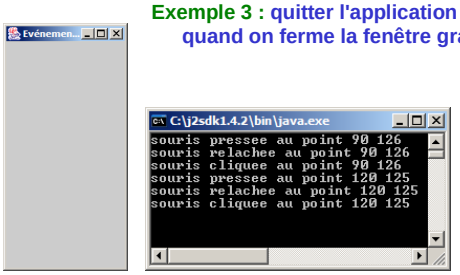
- *Exemple 1 : gérer l'événement "action sur bouton"*
- *Exemple 2 : gérer des événements "souris" de bas niveau*
- *Exemple 3 : quitter l'application quand on ferme une fenêtre graphique*

Classes internes anonymes

- Exemple 4 : écouter plusieurs boutons

Dessiner sur des composants

Exemple 3 : quitter l'application quand on ferme la fenêtre graphique



Créer un écouteur des événements de type WindowEvent (événements de haut niveau) sur cette fenêtre

Interface *java.awt.event.*WindowListener

Method Summary

void	**windowActivated**(WindowEvent e) Invoked when the window is set to be the user's active window, which means the window (or one of its subcomponents) will receive keyboard events.
void	**windowClosed**(WindowEvent e) Invoked when a window has been closed as the result of calling dispose on the window.
void	**windowClosing**(WindowEvent e) Invoked when the user attempts to close the window from the window's system menu.
void	**windowDeactivated**(WindowEvent e) Invoked when a window is no longer the user's active window, which means that keyboard events will no longer be delivered to the window or its subcomponents.
void	**windowDeiconified**(WindowEvent e) Invoked when a window is changed from a minimized to a normal state.
void	**windowIconified**(WindowEvent e) Invoked when a window is changed from a normal to a minimized state.
void	**windowOpened**(WindowEvent e) Invoked the first time a window is made visible.

Si c'est la fenêtre qui écoute ses propres événements :

```
public class Fenetre extends JFrame
                    implements WindowListener
{     ……
  public Fenetre() // constructeur
  {
    addWindowListener(this);
  }

  public void windowClosing (WindowEvent e)
  { System.exit(0); }

  // et les 6 autres méthodes de l'interface
     WindowListener avec un corps vide
……
}
```


Si c'est un objet dédié qui écoute les événements de la fenêtre :

```
public class Fenetre extends JFrame
{
      ……

      public Fenetre() // constructeur
      {
            ……
        Terminator t = new Terminator();
        addWindowListener(t);
       }
……
}
```

Ecrivons la classe Terminator


```
public class Terminator
                implements WindowListener
{
 public void windowClosing (WindowEvent e)
 { System.exit(0); }

// et les 6 autres méthodes avec un corps vide
}
```

Ouf! On peut dériver de la classe WindowAdapter

```
public class Terminator extends WindowAdapter
{
     public void windowClosing (WindowEvent e)
     { System.exit(0); }
}
```

Revenons à la classe Fenetre :

```
public Fenetre() // constructeur
     {
          ……
        Terminator t = new Terminator();
        addWindowListener(t);
     }
```

La référence t n'est pas utile

```
addWindowListener(new Terminator());
```

Le nom Terminator non plus….

on fait de Terminator une classe interne anonyme


```
public class Terminator extends WindowAdapter
{
     public void windowClosing (WindowEvent e)
     { System.exit(0); }
}
```

```
addWindowListener(new Terminator());
```

devient :

```
addWindowListener(new WindowAdapter()
  {
    public void windowClosing (WindowEvent e)
    { System.exit(0); }
  }
);
```


Classes internes

• Classe interne : définie à l'intérieur d'une classe

Intérêts

1. Une instance d'une classe interne a des privilèges : elle peut accéder aux attributs et méthodes de l'instance de la classe englobante qui l'a créée, même s'ils sont privés
2. Une classe interne peut être cachée aux autres classes, même à celles du même package (on peut en faire une classe privée)
3. Les classes internes (et anonymes!) sont très pratiques pour définir des écouteurs d'événements

Attention : complexifie rapidement le code

On restreindra l'utilisation des classes internes au point 3

Retour à l'exemple du bouton (exemple 1)

```
public class PanneauBouton extends JPanel
{
  private JButton b;

  public PanneauBouton()
  {  b = new JButton("Couleurs");
     add(b);
     setBackground(Color.white);
  b.addActionListener(new EcouteBouton(this));
  }
}
```

Notez que l'instance de EcouteBouton
est créée par une instance de PanneauBouton


```
class EcouteBouton implements ActionListener
{
  private static Color[] tCol = {Color.black,
  Color.blue, Color.yellow};
  private int numCol = -1;
  private JPanel p;

  public EcouteBouton(JPanel p)
  { this.p = p; }

  public void actionPerformed(ActionEvent e)
  {  numCol = (numCol + 1) % tCol.length;
     p.setBackground(tCol[numCol]);
  }
}
```

Si EcouteBouton devient une classe interne à PanneauBouton, une instance de EcouteBouton aura accès aux attributs et méthodes de l'instance de PanneauBouton qui l'a créée


```
public class PanneauBouton extends JPanel
{ private JButton b;
  private static Color[] tCol = {Color.black,
  Color.blue, Color.yellow};
  private int numCol = -1;
  public PanneauBouton()
  {  b = new JButton("Couleurs");
     add(b);
     setBackground(Color.white);
     b.addActionListener(new EcouteBouton());
  }
  class EcouteBouton implements ActionListener
  {
     public void actionPerformed(ActionEvent e)
     {     numCol = (numCol + 1) % tCol.length;
           setBackground(tCol[numCol]);
     }
  }
}
```

Puisque la classe EcouteBouton ne sert qu' à créer une instance, on peut en faire une classe interne anonyme :

c'est-à-dire définie « à la volée » lors de la création de l'instance de cette classe

```
b.addActionListener(new EcouteBouton()    );
```

```
class EcouteBouton implements ActionListener
{
      public void actionPerformed(ActionEvent e)
         {  numCol = (numCol + 1) % tCol.length;
            setBackground(tCol[numCol]);
         }
}
```

On remplace le nom de la classe par le nom de la classe dont elle dérive ou de l'interface qu'elle implémente, et on met le « corps » de la classe après les paramètres du constructeur

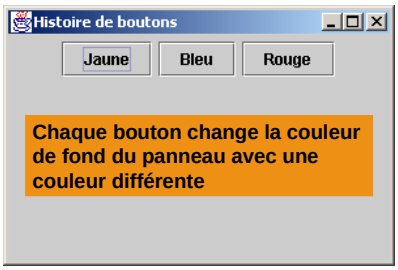
19

```
b.addActionListener(new EcouteBouton () );
```

```
b.addActionListener(new ActionListener()
{
      public void actionPerformed(ActionEvent e)
         {  numCol = (numCol + 1) % tCol.length;
            setBackground(tCol[numCol]);
         }
} );
```

20

Exemple 4 : écouter 3 boutons



Qui écoute ?

1. le panneau
2. un objet d'une classe dédiée écoute les 3 boutons
3. Chaque bouton est écouté par un objet différent

21

Dans les cas 1 et 2 se pose le problème
de la reconnaissance du bouton source de l'événement

Deux méthodes :

- `java.util.EventObject`

```
public Object getSource()
// retourne (une référence sur) l'objet source
```

- `java.awt.event.ActionEvent`

```
public String getActionCommand()
// retourne une chaîne de caractères
// c'est par défaut le label du bouton
// (au moment de l'action)

// on peut modifier cette valeur par défaut par la
// méthode setActionCommand du bouton
```

22

1

```
class PanneauBoutons extends JPanel
                     implements ActionListener
{
   private JButton bJaune, bBleu, bRouge;
   public PanneauBoutons()
   {
       bJaune = new JButton("Jaune"); add(bJaune);
       bBleu = new JButton("Bleu"); add(bBleu);
       bRouge = new JButton("Rouge"); add(bRouge);
       bJaune.addActionListener(this);
       bBleu.addActionListener(this);
       bRouge.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   {
          Object source = e.getSource();
          Color c = null;
          if (source == bJaune) c = Color.yellow;
          else if (source == bBleu) c = Color.blue;
          else c = Color.red;
          setBackground(c);
    }
}
```

23

2

```
class EcouteBoutons implements ActionListener
{ private JPanel p; //référence sur le panneau
  public EcouteBoutons(JPanel p){ this.p = p; }

  public void actionPerformed(ActionEvent e)
  {
     String command = e.getActionCommand();
     Color c = null;
     if (command.equals("Jaune")) c=Color.yellow;
     else if (command.equals("Bleu")) c=Color.blue;
     else c=Color.red;
     p.setBackground(c);

  }
}
```

Dans le constructeur du panneau :

```
EcouteBoutons ecouteur = new EcouteBoutons(this);
bJaune.addActionListener(ecouteur);
bBleu.addActionListener(ecouteur);
bRouge.addActionListener(ecouteur);
```

```
class PanneauBoutons extends JPanel                    3
{
  private JButton bJaune, bBleu, bRouge;
  public PanneauBoutons()
  {
      bJaune = new JButton("Jaune"); add(bJaune);
      bBleu = new JButton("Bleu"); add(bBleu);
      bRouge = new JButton("Rouge"); add(bRouge);
      bJaune.addActionListener(…);
      bBleu.addActionListener(…);
      bRouge.addActionListener(…);
  }
}
```

```
bJaune.addActionListener( new ActionListener()
 { public void actionPerformed(ActionEvent e)
    { setBackground(Color.yellow); }
 }
);
```

Idem pour les autres boutons en changeant la couleur

25

Dessiner sur des composants

[Exemple : une grille de jeu]

Le dessin se fait par un « contexte graphique »

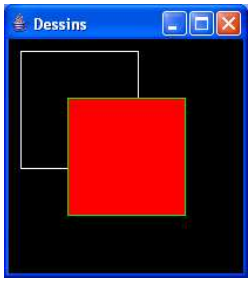
- instance d'une classe dérivée de Graphics
- associé au composant sur lequel on veut dessiner (donc connaissant ses coordonnées à l'écran)
- gérant les outils de dessin : sélection d'une couleur de dessin, d'une police, …
- sachant tracer des formes sur le composant : une ligne, un rectangle, un rectangle plein, une elipse, une chaîne de caractères, …

26

```
Graphics g = panneau.getGraphics();
// demande d'un Graphics associé à « panneau »

g.setColor(Color.white);
g.drawRect(10,10,100,100);
g.setColor(Color.red);
g.fillRect(50,50,100,100);
g.setColor(Color.green);
g.drawRect(50,50,100,100);

g.dispose();
// on rend la ressource
```



Problème : assurer la « permanence » du dessin
[exemple]

27

- A chaque fois qu'un composant graphique a besoin d'être repeint, appel automatique de la méthode :

```
public void paintComponent(Graphics g)
{
  ici, ce qu'il faut faire
  pour repeindre le composant
}
```

- Cas particulier des fenêtres (méthode paint)

Conseil : ne pas dessiner directement dans une fenêtre, mettre un panneau « en toile de fond »

28

```
// exemple du dessin sur le panneau
public void paintComponent(Graphics g)
{
     super.paintComponent(g);
     // appel à la méthode masquée
     g.setColor(Color.white);
     ……
     g.drawRect(50,50,100,100);
}
```

29

- L'objet Graphics est passé en paramètre lors de l'appel – automatique – de la méthode paintComponent

Code hors contrôle du programmeur :

```
Graphics g = panneau.getGraphics();
panneau.paintComponent(g);
g.dispose();
```

Ceci n'est pas fait pas le programmeur.

Ne JAMAIS appeler explicitement paintComponent

30

• Mais parfois on a besoin de demander qu'un composant soit repeint :

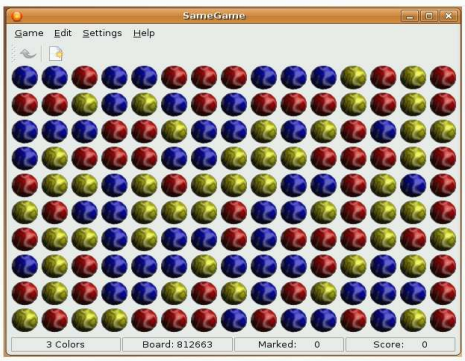
[Exemple]

• Pour demander l'appel de paintComponent :
repaint();

paintComponent sera appelé *dès que possible*

31

Présentation du « gros » TP :



Programmation du jeu SAME

32