Laborator 11:

Evenimente generate de componente AWT

Întocmit de: Adina Neculai

Îndrumător: Asist. Drd. Gabriel Danciu

4 decembrie 2011

I. NOŢIUNI TEORETICE

Unei componente grafice îi este asociată:

- o formă (ceea ce este afişat pe ecran);
- o serie de evenimente la care poate răspunde (mişcări de mouse, apăsare de taste);
- un model obiectual de responsabil cu prelucrarea evenimentului (ceea ce face componenta propriu-zis).

A. Ce este un eveniment?

La acţiunea utilizatorului asupra unei componente grafice, JVM generează aşa numitele *evenimente*. Evenimentul este de fapt o instanţă a unei clase Java, instanţă care conţine o serie de informaţii despre acţiune.

B. Modelul de evenimente

Modelul de lucru cu evenimente prezentat în această documentaţie (modelul jdk1.1) conţine o sursă, care poate genera un eveniment, şi unul sau mai mulţi delegaţi. *Sursa* poate fi oricare dintre componentele grafice. *Delegaţii*, numiţi şi handleri, sunt cei responsabili de răspunsul la eveniment (cei care prelucrează informaţia provenită de la eveniment). Mai mult, ei sunt definiţi de programator şi datorită acestui model de implementare a evenimentelor, pot fi orice ce clasă ce implementează o interfaţă corespunzătoare evenimentului respectiv. Pe lângă implementarea clasei delegat, programatorul este responsabil de asocierea acesteia cu sursa de evenimente. Această asociere se poate face printr-o metodă a sursei (a componentei grafice), metodă dependentă de tipul delegatului ce trebuie conectat.

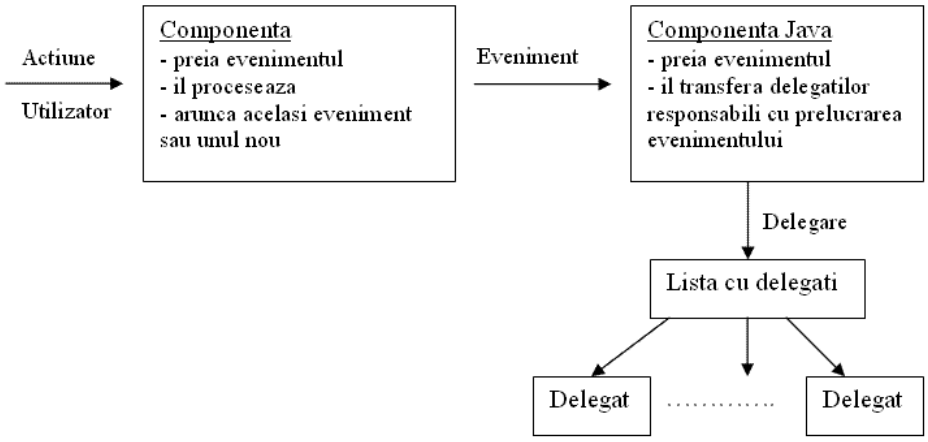

Figura 1: Modelul de evenimente jdk1.1

C. Tipuri de evenimente

Diferite surse (componente grafice) pot genera diferite tipuri de evenimente.

- *ActionEvent*: indică dacă a avut loc o acţiune definită de o componentă;
- *ComponentEvent*: clasa rădăcină a evenimentelor; indică dacă o componentă şi-a modificat dimensiunea, vizibilitatea sau dacă a fost mutată;
- *ContainerEvent*: specifică dacă s-a modificat conţinutul unui container în urma adăugării sau eliminării unei componente;
- *KeyEvent*: precizează dacă s-a apăsat sau nu o tastă;
- *MouseEvent*: precizează dacă a avut loc o acţiune de genul apasarea unei taste a mouse-ului, şamd.;
- *MouseMotionEvent*: precizează dacă a avut loc o acţiune de genul mişcare de mouse;
- *WindowEvent*: eveniment generat în momentul în care s-a deschis/inchis, activat/dezactivat o fereastră.

Pentru a răspunde unui eveniment este necesară crearea unuia sau mai multor handlere de eveniment (delegaţi) şi asocierea lor cu componenta corespunzătoare. Pentru crearea unui handler e nevoie de implementarea unei anumite interfeţe (în funcţie de tipul evenimentului) într-o anumită clasă. Asocierea handler - componentă se face prin intermediul unor metode ale clasei *Component*, în funcţie de tipul evenimentului.

Componentă grafică	**Interfaţă**	**Metode de asociere handler - componentă**
Component	ComponentListener	addComponentListener(ComponentListener l)
	KeyListener	addKeyListener(KeyListener l)
	MouseListener	addMouseListener(MouseListener l)
	MouseMotionListener	addMouseMotionListener(MouseMotionListener l)
Button	ActionListener	addActionListener(ActionListener l)
CheckBox	ItemListener	addItemListener(ItemListener l)
Choice		
List	ActionListener	addActionListener(ActionListener l)
	ItemListener	addItemListener(ItemListener l)
	ComponentListener	cele de la clasa Component
TextField	ActionListener	addActionListener(ActionListener l)
	TextListener	addTextListener(TextListener l)
	ComponentListener	cele de la clasa Component
Window	ComponentListener	cele de la clasa Component
Frame	ContainerListener	addContainerListener(ContainerListener l)
Dialog	WindowListener	addWindowListener(WindowListener l)
FileDialog		
MenuItem	ActionListener	addActionListener(ActionListener l)
Menu		
PopupMenu		

Interfaţă	**Metode ce trebuie implementate**
ActionListener	actionPerformed(ActionEvent e)
ComponentListener	componentResized(ComponentEvent e)
	componentMoved(ComponentEvent e)
	componentShown(ComponentEvent e)
	componentHidden(ComponentEvent e)
MouseListener	mouseClicked(MouseEvent e)
	mousePressed(MouseEvent e)
	mouseReleased(MouseEvent e)
	mouseEntered(MouseEvent e)
	mouseExited(MouseEvent e)
MouseMotionListener	mouseDragged(MouseMotionEvent e)
	mouseMoved(MouseMotionEvent e)
KeyListener	keyTyped(KeyEvent e)
	keyPressed(KeyEvent e)
	keyReleased(KeyEvent e)
WindowListener	windowOpened(WindowEvent e)
	windowClosed(WindowEvent e)
	windowActivated(WindowEvent e)
	etc.

D. Componenta FileDialog

Clasa *FileDialog* este o clasă derivată din clasa *Dialog*, iar aceasta din urmă derivată din clasa *Window*. *FileDialog* este aşadar o fereastră, fereastră utilizată pentru selecţia unui fişier. Aceasta afişează un arbore ce reprezintă structura ierarhică a directoarelor şi fişierelor şi permite utilizatorului căutarea în această structură şi selecţia unui fişier.

Există două tipuri de ferestre *FileDialog*: fereastră de tipul *load* şi una de tipul *save*. Primul tip permite doar selecţia unui fişier existent, pe când al doilea tip oferă şi posibilitatea creării unui fişier nou. O dată un fişier selectat, fereastra se închide automat.

Pentru a şti mai multe despre această clasă se recomandă citirea API-ului.

E. Componente AWT de lucru cu meniuri

Clasa **MenuBar** este utilizată pentru a crea o bară de meniuri. Această bară de meniuri poate fi asociată doar unei ferestre grafice de tipul *Frame* utilizând metoda *setMenuBar(MeniuBar mb)* a clasei *Frame*. O bară de meniuri este utilizată ca şi cadru în care se adaugă componente de tip *Menu*. API-ul clasei se află aici.

Clasa **MenuItem** reprezintă clasa ce implementeză un meniu. La selecţia unui meniu acesta generează un eveniment de tipul *ActionEvent* care poate fi preluat de un handler de tipul *ActionListener*. API-ul clasei se află aici.

Clasa **Menu** este derivată din *MenuItem* şi este componenta ce oferă posibilitatea creării unei structuri verticale de meniuri. În plus faţă de clasa *MenuItem*, clasa *Menu* poate crea o componentă cadru în care se pot adăug alte meniuri (componente de tip *MenuItem*). API-ul clasei se află aici.

Clasa **MenuShortcut** nu reprezintă o componentă grafică, dar este corelată lucrului cu meniuri. Aceasta este utilizată pentru a asocia unei componente de tip *MenuItem* o combinaţie de taste. API-ul clasei se află aici.

II. PREZENTAREA LUCRĂRII DE LABORATOR

A. Evenimentul ActionEvent

Componenta Button creează imaginea unui buton asupra căruia se poate acţiona prin apăsare (cu ajutorul mouse-ului sau a tastaturii). La ”apăsarea” acestui tip de componentă se va genera un eveniment de tipul *ActionEvent* care poate fi tratat de programator printr-un handler înregistrat la această componentă. Handler-ul în acest caz este reprezentat de clasa *MyButtonListener*.

După instanţierea unui obiect de acest tip, programatorul trebuie să asocieze respectivul obiect instanţiat cu componenta Button (*addActionListener()*). Observaţi că înaintea acestui pas, componentei Button i s-a setat un şir care va fi înscris în obiectul eveniment de tip *ActionEvent* ce se generează când se apasă butonul. Acest lucru a fost realizat prin apelarea metodei *setActionCommand()*.

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

```
2  import java.awt.event.ActionListener;
3
4  public class TestButton {
5
6     public static void main(String[] args) {
7        Frame f = new Frame("Fereastra Buton");
8        f.setLayout(new FlowLayout());
9
10       Button b = new Button("Bonjour");
11       f.add(b);
12       b.setActionCommand("Bonjour");
13
14       ActionListener actionListener = new MyButtonListener();
15       b.addActionListener(actionListener);
16
17       b = new Button("Good Day");
18       f.add(b);
19       b.addActionListener(actionListener);
20
21       b = new Button("Aurevoir");
22       f.add(b);
23       b.setActionCommand("Exit");
24       b.addActionListener(actionListener);
25
26       f.pack();
27       f.setVisible(true);
28    }
29 }
```

Pentru crearea handler-ului sau delegatului acestui eveniment, programatorul trebuie să implementeze interfaţa *ActionListener* într-o clasă (*MyButtonListener*). Metoda ce trebuie suprascrisă în acest caz este *actionPerformed()*. De asemenea, observaţi folosirea metodei *getActionCommand* care returnează informaţia setată în clasa *TestButton*.

```
1  import java.awt.event.*;
2
3  public class MyButtonListener implements ActionListener {
4     public void actionPerformed(ActionEvent ae) {
5        String s = ae.getActionCommand();
6        if (s.equals("Exit")) {
7           System.exit(0);
8        } else if (s.equals("Bonjour")) {
9           System.out.println("Good Morning");
10       } else {
11          System.out.println(s + " clicked");
12       }
13    }
14 }
```

B. Evenimentul KeyEvent

Clasa următoare creează o etichetă şi o componentă de tip text pe care le adaugă într-o fereastră. La apasarea oricărei taste se declansează un eveniment de tipul KeyEvent. Evenimentul este tratat de către handler-ul *MyKeyListener* şi a fost asociat componentei de tip text prin intermediul metodei *addKeyListener()*.

```
1  import java.awt.*;
2  import java.awt.event.KeyListener;
3
4  public class TestKey {
5
6     public static void main(String[] args) {
7        Frame f = new Frame("Fereastra keyEvent");
8
9        Label label = new Label("Scrie: ");
10       TextField txtField = new TextField(20);
11       KeyListener keyListener = new MyKeyListener();
12       txtField.addKeyListener(keyListener);
13
14       f.add(txtField, BorderLayout.CENTER);
15       f.add(label, BorderLayout.NORTH);
16
17       f.pack();
18       f.setVisible(true);
19    }
20 }
```

Întrucât handler-ul implementează interfaţa *KeyListener* trebuie să-i suprascrie toate metodele. S-a scris cod doar pentru metoda *keyPressed()* care se va apela doar în momentul apăsării unei taste.

```
1  import java.awt.event.*;
2
3  public class MyKeyListener implements KeyListener {
4
5     @Override
6     public void keyPressed(KeyEvent keyEvent) {
7        char i = keyEvent.getKeyChar();
8        String str = Character.toString(i);
9        System.out.println("Key pressed: "+ str);
10
11    }
12
13    @Override
14    public void keyReleased(KeyEvent arg0) {}
15    @Override
16    public void keyTyped(KeyEvent arg0) {}
17 }
```

C. Evenimentul MouseEvent

Exemplul de mai jos crează două etichete şi un buton. La apăsarea butonului se va afişa în consolă suma numerelor conţinute de etichete.

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

public class TestMouseClick {
   public static void main(String[] args) {
      Frame f = new Frame("Fereastra mouse click");

      Label labelY = new Label("2");
      f.add(labelY, BorderLayout.NORTH);
      Label labelX = new Label("1");
      f.add(labelX, BorderLayout.CENTER);

      Button button = new Button("Click Me");
      MouseListener mouseListener = new MyMouseListener(labelX, labelY);
      button.addMouseListener(mouseListener);
      f.add(button, BorderLayout.SOUTH);

      f.pack();
      f.setVisible(true);
   }
}
```

Clasa handler se numeşte *MyMouseListener* şi suprascrie toate metodele interfeţei *MouseListener*, dar scrie cod efectiv doar pentru metoda care se declanşează în momentul apăsării unei taste a mouse-ului. Această metodă preia informaţiile conţinute de etichetele transmise ca parametru la instanţierea handler-ului şi afişează în consolă.

```
import java.awt.Label;
import java.awt.event.*;

public class MyMouseListener implements MouseListener{
   Label labelX, labelY;

   public MyMouseListener(Label x, Label y){
      this.labelX = x;
      this.labelY = y;
   }

   @Override
   public void mouseClicked(MouseEvent mouseEvent) {
      int x = Integer.parseInt(labelX.getText());
      int y = Integer.parseInt(labelY.getText());
      System.out.println(x+" + "+y+" = "+(x+y));
   }

   @Override
   public void mouseEntered(MouseEvent arg0) {}
   @Override
   public void mouseExited(MouseEvent arg0) {}
```

```
23     @Override
24     public void mousePressed(MouseEvent arg0) {}
25     @Override
26     public void mouseReleased(MouseEvent arg0) {}
27 }
```

D. Componenta FileDialog

La pornirea aplicaţiei se va deschide o fereastră *FileDialog* de tipul *load* cu numele ”Choose a file”. Structura de directoare din care se porneşte este ”D:\”.

```
1  import java.awt.*;
2
3  public class TestFileDialog {
4     public static void main(String[] args) {
5        Frame frame = new Frame("Test fileDialog");
6        FileDialog fileDialog = new FileDialog(frame, "Choose a file", FileDialog.LOAD);
7        fileDialog.setDirectory("D:\\");
8        frame.addWindowListener(new MyFileDialogListener(fileDialog));
9        frame.setVisible(true);
10    }
11 }
```

În urma implementării interfeţei *WindowListener* trebuie suprascrise o sumedenie de metode (vezi al doilea tabel din sectiunea I C). Cum exemplul are scris cod doar pentru metoda *windowOpened()* care se va apela în momentul în care se deschide fereastra, dorim să nu mai suprascriem toate celelalte metode asa cum au fost definite in exemplele anterioare. Astfel că se moşteneşte clasa *WindowAdapter*. Clase de tipul *Adapter* există pentru fiecare eveniment în parte.

```
1  import java.awt.FileDialog;
2  import java.awt.event.*;
3  /*am folosit clasa WindowAdapter pentru a suprascrie doar anumite metode si nu toate cum am fi fost
4  obligati daca implementam interfata WindowListener*/
5  public class MyFileDialogListener extends WindowAdapter {
6     FileDialog fileDialog;
7
8     public MyFileDialogListener(FileDialog fileDialog) {
9        this.fileDialog = fileDialog;
10    }
11
12    @Override
13    public void windowOpened(WindowEvent e) {
14       fileDialog.setVisible(true);
15       String fileName = fileDialog.getFile();
16       if (fileName == null)
17          System.out.println("You cancelled the choice");
18       else
19          System.out.println("You chose " + fileName);
20    }
21 }
```

E. Componente AWT de lucru cu meniuri

Clasa de mai jos realizează o bara de meniuri care conţine elementele următoare: *File* cu submeniul format din: Open, Exit şi *Edit* cu submeniul format din Undo. Atât Exit cât şi Undo au şi un shortcut creat cu clasa *MenuShortcut*. Explicaţiile suplimentare le aveţi trecute în comentariu.

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  public class TestMenu {
5     public static void main(String[] args) {
6        Frame frame = new Frame("Meniu");
7        //se seteaza dimendiunea ferestrei
8        frame.setSize(100, 100);
9
10       //s-a creat si s-a adaugat o bara de meniuri in fereastra
11       MenuBar myMenuBar = new MenuBar();
12       frame.setMenuBar(myMenuBar);
13
14       //se creeaza componente Menu care se adauga in bara de meniuri
15       Menu myFileMenu = new Menu("File");
16       Menu myEditMenu = new Menu("Edit");
17       myMenuBar.add(myFileMenu);
18       myMenuBar.add(myEditMenu);
19
20       //se creeaza obiecte de tipul MenuItem
21       MenuItem myFileOpenMenuItem = new MenuItem("Open...");
22       /*se foloseste clasa MenuShortcut pt crearea unei combinatii de taste pt accesarea
23        componentei MenuItem Exit*/
24       MenuItem myFileExitMenuItem = new MenuItem("Exit", new MenuShortcut(
25             KeyEvent.VK_X));
26       MenuItem myEditUndoMenuItem = new MenuItem("Undo", new MenuShortcut(
27             KeyEvent.VK_Z));
28
29       //se adauga fiecare componente MenuItem in componenta Menu corespunzatoare
30       myFileMenu.add(myFileOpenMenuItem);
31       myFileMenu.addSeparator();
32       myFileMenu.add(myFileExitMenuItem);
33       myEditMenu.add(myEditUndoMenuItem);
34
35       //se seteaza informatiile ce vor fi transmise handler-ului o data cu declansarea evenimentului
36       myFileOpenMenuItem.setActionCommand("open");
37       myFileExitMenuItem.setActionCommand("exit");
38       myEditUndoMenuItem.setActionCommand("undo");
39
40       //se asociaza componentele grafice cu handler-ul MyActionListener
41       ActionListener actionListner = new MyActionListener();
42       myFileOpenMenuItem.addActionListener(actionListner);
43       myFileExitMenuItem.addActionListener(actionListner);
44       myEditUndoMenuItem.addActionListener(actionListner);
45
46       frame.setVisible(true);
47    }
48 }
```

Clasa handler *MyActionListener* :

```
1  import java.awt.event.*;
2
3  public class MyActionListener implements ActionListener {
4
5     @Override
6     public void actionPerformed(ActionEvent e) {
7        String cmd = e.getActionCommand();
8        if (cmd.equals("open")) {
9           System.out.println("open");
10       } else if (cmd.equals("exit")) {
11          System.exit(0);
12       } else if (cmd.equals("undo")) {
13          System.out.println("undo");
14       }
15    }
16 }
```

III. TEMĂ

1. Continuaţi exerciţiul 3 din Laborator10.pdf în felul următor:

   - Implementaţi funcţionalităţile calculatorului (operaţiile aritmetice specificate). Calculatorul va efectua operaţiile atât la apasarea butoanelor cu ajutorul mouse-ului cât şi la apăsarea tastelor corespunzătoare de pe tastatură.
   - Pe lângă cerinţele de implementare ale interfeţei specificate în laboratorul anterior, adăugaţi o bară de meniu asemănătoare cu cea a calculatorului din Windows. (File: Exit Ctrl+X; Edit: Copy Ctrl+C, Paste Ctrl+V).

Pentru nota 10 implementaţi şi funcţionalităţile meniului.