Poo Laboratoare[1]

Contents

[1] http://www.google.ro/

Laborator11

Programare Orientată pe Obiecte: Laborator 11

1 Liste de selecție JList

Un obiect de tip JList prezintă utilizatorului un grup de item-uri, afișate în una sau mai multe coloane pentru a oferi acestuia posibilitatea de a alege.

1.1 Crearea unui model

Există trei metode prin care se poate crea un model de listă:

- **DefaultListModel** - aproape totul este gestionat de model pentru a ușura munca programatorului
- **AbstractListModel** - utilizatorul gestionează datele și invocă metodele de "acționare". Pentru această abordare, programatorul trebuie să subclaseze AbstractListModel și să implementeze metodele **getSize()** și **getElementAt()** moștenite din interfața ListModel
- **ListModel** - utilizatorul gestionează totul

1.2 Inițializarea unei liste

Iată un exemplu de cod care creează și setează parametrii unei liste:

```
list = new JList( data ); //data e de tipul Object[ ]
list.setSelectionMode( ListSelectionModel.SINGLE_INTERVAL_SELECTION );
list.setLayoutOrientation( JList.HORIZONTAL_WRAP );
list.setVisibleRowCount( -1 );
...
JScrollPane listScroller = new JScrollPane( list );
listScroller.setPreferredSize( new Dimension( 250, 80 ) );
```

Codul pasează un vector către constructorul listei. Vectorul conține șiruri de caractere, care în exemplul de față reprezintă nume de băieți.

Alți constructori pentru JList permit inițializarea listei dintr-un **obiect de tip Vector** sau dintr-un **obiect de tipul ListModel**. Dacă inițializarea se face dintr-un obiect de tip Vector sau dintr-un vector clasic, constructorul implicit creează un model standard de listă. Modelul standard este imuabil - nu se pot adăuga, șterge, înlocui intrări în listă. Pentru a crea o listă a cărei intrări pot fi modificate individual, se poate seta modelul listei către o instanță variabilă, cum ar fi **DefaultListModel**. Modelul unei liste se poate seta când se creează lista sau prin apelul de metodă **setModel()**.

Apelul **setSelectionMode()** specifică câte intrări din listă poate selecta ultilizatorul și dacă acestea trebuie să fie sau nu contigue.

Apelul **setLayoutOrientation()** permite afișarea datelor în mai multe coloane. Valoarea **JList.HORIZONTAL_WRAP** specifică faptul că lista ar trebui să își afișeze intrările de la stânga la dreapta înainte de a trece la o nouă linie. O altă valoare posibilă este **JList.VERTICAL_WRAP**, care specifică afișarea datelor de sus până jos (ca de obicei) înainte de a trece la o coloană nouă.

În combinație cu apelul **setLayoutOrientation()**, invocarea metodei **setVisibleRowCount(-1)** oferă listei posibilitatea de afișare a numărului maxim de intrări cuprins în spațiul disponibil pe ecran. O altă utlizare pentru **setVisibleRowCount()** este aceea de a indica panoului de scroll-ing asociat listei câte rânduri să afișeze.

1.3 Selectarea intrărilor dintr-o listă

O listă utilizează o instanță de **ListSelectionModel** pentru a-și gestiona selecția. Implicit, modelul de selecție a unei liste permite selecția oricărei combinații de intrări la un moment dat. Se poate specifica un mod diferit de selecție prin apelul metodei **setSelectionMode()**:

- **SINGLE_SELECTION** - numai o singură intrare poate fi selectată la un moment dat. Când utilizatorul selectează o intrare, orice intrare anterior aleasă este deselectată
- **SINGLE_INTERVAL_SELECTION** - intrări multiple și contigue pot fi selectate. Când utilizatorul începe un nou interval de selecție, orice intrări anterior alese sunt deselectate
- **MULTIPLE_INTERVAL_SELECTION** - mod implicit. Orice combinație de intrări poate fi selectată. Utilizatorul trebuie să deselecteze explicit intrări.

Indiferent de modelul de selecție folosit de listă, aceasta generează evenimente de selecție de fiecare dată când selecția se schimbă. Aceste evenimente pot fi procesate prin adăugarea unui ascultător la listă folosind metoda **addListSelectionListener()**. Acest ascultător trebuie să implementeze o metodă: **valueChanged()**.

Iată un exemplu:

```
public void valueChanged( ListSelectionEvent e ) {
       if( e.getValueIsAdjusting() == false ) {
              if( list.getSelectedIndex() == -1 ) {
                     fireButton.setEnabled( false );
              } else {
                     fireButton.setEnabled( true );
              }
       }
}
```

Multe evenimente de selecție pot fi generate de o singură acțiune a utilizatorului, cum ar fi un clic cu mouse-ul. Metoda **getValueIsAdjusting()** returnează **true** dacă utilizatorul încă manipulează selecția. Programul de mai sus este interesat doar de rezultatul final al acțiunii utilizatorului și de aceea metoda **valueChanged()** conține secvență de cod doar dacă **getValueIsAdjusting()** întoarce **false**.

Deoarece lista este în modul **SINGLE_SELECTION**, acest cod poate utiliza metoda **getSelectedIndex()** pentru a obține indexul intrării tocmai selectate. JList pune la dispoziție și alte metode pentru a seta sau a obține selecția când se pot alege mai multe intrări. De asemenea, se pot asculta evenimente direct pe modelul de selecție, dacă acest lucru este dorit.

1.4 Adăugarea și ștergerea

```
listModel = new DefaultListModel();
listModel.addElement( ‘‘Debbie Scott’’ );
listModel.addElement( ‘‘Scott Hommel’’ );
listModel.addElement( ‘‘Alan Sommerer’’ );
list = new JList( listModel );
```

Programul de mai sus utilizează o instanță DefaultListModel. În ciuda numelui, lista nu deține un DefaultListModel decât dacă acest lucru este setat explicit din program. Dacă DefaultListModel nu este potivit nevoilor programatorului, se poate scrie un model customizat dar care trebuie să adere la interfața ListModel.

Altă metodă de adăugare, cu specificare exactă a poziției de inserare, este:

```
listModel.insertElementAt( employeeName.getText(), index );
```

Ștergerea este de asemenea, foarte simplă:

```
listModel.remove( index );
```

Indiferent dacă sunt adăugate, șterse sau modificate intrări, modelul listei declanșează evenimente. Pentru mai multe informații despre acest subiect consultați adresa[2].

1.5 Formatarea celulelor

O listă folosește un obiect numit **Cell Renderer** pentru a-și afișa intrările. Cel implicit știe să afișeze șiruri de caractere și imagini și afișează tipul **Object** invocând metoda **toString()**. Dacă se dorește schimbarea modului în care se face afișarea sau dacă se dorește un comportament diferit față de cel oferit de **toString()**, se poate implementa un **Cell Renderer customizat**. Pașii ce trebuie urmați sunt:

- scrierea unei clase care implementează interfața **ListCellRenderer**
- crearea unei instanțe de clasă și apelarea metodei **setCellRenderer()** pentru listă, folosind instanța ca argument

1.6 Jlist API

Pentru vizualizarea completă a API-ului consultați documentația oficială de la Sun[3].

2 Probleme de laborator

2.1 Problema 1

(2 puncte) Să se scrie un program care permite introducerea unui nume de director într-un câmp text și afișează lista fișierelor din director într-o listă de selecție **JList** (dacă există acel director). Lista de selecție este reinițializată la fiecare modificare a câmpului text (nu se prelungește cu alte nume de fișiere). Constructorul obiectului JList primește un obiect **Vector**.

2.2 Problema 2

(2.5 puncte) Să se adauge programului anterior încă un câmp text în care se va afișa numele fișierului selectat de utilizator (prin clic pe mouse) din lista de selecție (cu toate fișierele din director). Fiecare câmp text este însoțit de o etichetă (mai exact *Directory*, repectiv *Selected*). Cele două câmpuri și cele două etichete se plasează într-un panou separat (cu așezare GridLyout). Așezarea în fereastra principală este FlowLayout. Pentru a evita producerea unei excepții se va adăuga la începutul metodei **valueChanged()** din obiectul ascultător **ListSelectionListener** secvența:

```
if( list.isSelectionEmpty() ) return; // Jlist list
```

Să se adauge apoi o clasă **Renderer** astfel încât atunci când o celulă este selectată să apară alb pe roșu, iar când nu este selectată verde pe alb.

2.3 Problema 3

(2 puncte) Să se modifice programul anterior prin înlocuirea câmpului text pentru fișierul selectat cu o altă listă JList, la care se adaugă fișierele selectate din prima listă. Pentru a evita afișarea repetată a fișierelor selectate (la apăsarea și ridicarea butonului de mouse) se va introduce în metoda **valueChanged()** secvența:

[2] `http://java.sun.com/docs/books/tutorial/uiswing/events/listdatalistener.html`
[3] `http://java.sun.com/javase/6/docs/api/javax/swing/JList.html`

```
if( e.getValueIsAdjusting() ) return; // ListSelectionEvent e
```

2.4 Problema 4

(2 puncte) Să se modifice programul de la punctul 2 prin adăugarea unui buton de ștergere a fișierelor selectate (din lista tuturor fișierelor din directorul specificat), astfel ca în lista afișată să rămână numai fișierele dorite. Se va actualiza și un câmp text cu numărul fișierelor rămase (cele afișate în listă). Se poate elimina a doua listă de selecție. Fișierele selectate nu vor fi șterse efectiv de pe disc, ci numai din lista afișată.

2.5 Problema 5

(1.5 puncte) Să se modifice programul de la punctul 4 prin înlocuirea obiectului **Vector** cu un obiect **DefaultListModel** pentru a memora fișierele selectate din lista afișată. Acest obiect este transmis la construirea unui obiect JList.

2.6 Problema 6 (bonus)

(2 puncte) Să se adauge programului de la punctul 2 posibilitatea de a naviga prin sistemul de fișiere: la selectarea unui subdirector se mută numele acestuia în câmpul text și se afișează în JList conținutul său. Pentru revenire la directorul părinte se afișează în JList și numele acestuia ca **..** (două puncte).