Poo Laboratoare[1]

Contents

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

Laborator12

Programare Orientată pe Obiecte: Laborator 12

1 Programare cu JTable & JTree

1.1 JTable

Clasa JTable este folosită pentru a afișa și edita tabele de celule în două dimensiuni. Consultați tutorialul How to Use Tables[2] pentru documentație task-oriented și exemple de utilizare.

JTable deține numeroase facilități care permit customizarea după preferințe dar în același timp oferă și opțiuni standard pentru aceste facilități astfel încât tabele simple pot fi create foarte rapid și ușor. De exemplu, un tabel cu 10 linii și 10 coloane se poate obține astfel:

```
TableModel dataModel = new AbstractTableModel() {
       public int getColumnCount() { return 10; }
       public int getRowCount() { return 10;}
       public Object getValueAt( int row, int col ) { return new Integer( row * col ); }
};
JTable table = new JTable( dataModel );
JScrollPane scrollpane = new JScrollPane( table );
```

Când se dorește scrierea de aplicații care folosesc JTable, este necesar să se acorde puțină atenție structurilor de date care vor reprezenta datele din tabel. **DefaultTableModel** este o implementare de model care folosește un vector de vectori de obiecte (de tipul Object) pentru a stoca valorile din celule. La fel cum se pot copia datele dintr-o aplicație în instanța DefaultTableModel, este de asemenea posibil să se 'ascundă' datele în metodele interfeței **TableModel** astfel încât acestea să poată fi transmise direct către JTable, la fel ca în exemplul de mai sus. Această abordare duce deseori la aplicații mai eficiente deoarece modelul este liber să aleagă reprezentarea internă care se potrivește cel mai bine datelor manipulate. Se recomandă folosirea **AbstractTableModel** ca și clasă de bază pentru crearea de subclase, respectiv DefaultTableModel atunci când subclasarea nu este necesară.

JTable folosește exclusiv variabile întregi pentru a referi liniile și coloanele modelului pe care îl afișează. Este folosită metoda **getValueAt(int, int)** pentru a întoarce valorile din model pe parcursul desenării. Coloanele pot fi rearanjate în tabel astfel încât acestea să apară într-o ordine diferită față de cea din model. Acest fapt nu afectează deloc implementarea: atunci când coloanele sunt rearanjate, obiectul de tip JTable menține intern noua ordine și convertește indicii coloanelor înainte de orice interogare a modelului. Așadar, la programarea unui TableModel, nu este necesară ascultarea după evenimente de reordonare de coloane, întrucât modelul va fi interogat în sistemul propriu de coordonate indiferent de ce se întamplă în vizualizare.

În versiunea curentă de Java sunt adăugate metode la clasa JTable care permit acces convenient către nevoi obișnuite de afișare. Noile metode **print()** adaugă cu ușurință suport de printare aplicației ce se dorește a fi dezvoltată. În plus, noua metodă **getPrintable(javax.swing.JTable.PrintMode, java.text.MessageFormat, java.text.MessageFormat)** este disponibilă pentru necesități avansate.

La fel ca pentru toate clasele JComponent, se pot folosi **InputMap** și **ActionMap** pentru a asocia o acțiune cu tastă și a executa acțiunea în condiții specificate.

1.2 JTree

Clasa JTree perminte afișarea datelor ierarhice (sub forma unei schițe). Pentru documentație task-oriented și exemple de utilizare consultați tutorialul How to Use Trees[3].

[2] http://java.sun.com/docs/books/tutorial/uiswing/components/table.html
[3] http://java.sun.com/docs/books/tutorial/uiswing/components/tree.html

Un nod specific poate fi identificat fie printr-un **TreePath** (un obiect care încapsulează nodul și toți strămoșii acestuia), fie prin linia de afișare, unde fiecare linie din zona de afișare conține un singur nod. Un nod expandat este un nod care nu este frunză (metoda **TreeModel.isLeaf(node)** întoarce *false*) și care își va afișa copiii când toți strămoșii săi sunt expandați. Un nod colapsat este un nod care își ascunde copiii. Un nod ascuns este un nod care este situat sub un strămoș colapsat. Toți părinții unui nod care poate fi vizualizat sunt expandați, dar aceștia pot sau nu fi afișați. Un nod afișat se regăsește în zona de afișare și poate fi vizualizat.

Următoarele metode din clasa JTree folosesc cuvântul *visible* pentru a se referi la *afișat*:

- isRootVisible()
- setRootVisible()
- scrollPathToVisible()
- scrollRowToVisible()
- getVisibleRowCount()
- setVisibleRowCount()

Următorul grup de metode folosesc cuvântul *visible* pentru a se referi la *poate fi vizualizat (sub un părinte expandat)*:

- isVisible()
- makeVisible()

Pentru a detecta schimbarea selecției, se va implementa interfața **TreeSelectionListener** și se va adăuga o instanță folosind **addTreeSelectionListener()**. Metoda **valueChanged()** va fi invocată atunci când utilizatorul selectează alt nod, și doar o dată, chiar dacă se efectuează un clic de două ori pe același nod. Cu toate acestea, pentru a face separarea cazurilor de dublu clic, indiferent de selecția anterioară, este recomandată abordarea:

```
final JTree tree = ...  ;
MouseListener ml = new MouseAdapter() {
       public void mousePressed( MouseEvent e ) {
              int selRow = tree.getRowForLocation( e.getX(), e.getY() );
              TreePath selPath = tree.getPathForLocation( e.getX(), e.getY() );
              if( selRow != -1 ) {
                     if( e.getClickCount() == 1 ) {
                            mySingleClick( selRow, selPath );
                     }
                     else if( e.getClickCount() == 2 ) {
                            myDoubleClick(selRow, selPath);
                     }
              }
       }
};
tree.addMouseListener( ml );
```

Pentru afișarea nodurilor complexe (de exemplu, noduri conținând atât text, cât și o icoană) se va implementa interfața **TreeCellRenderer** și se va folosi metoda **setCellRenderer(javax.swing.tree.TreeCellRenderer)**. Pentru a edita astfel de noduri, se va implementa interfața **TreeCellEditor** și se va folosi metoda **setCellEditor(javax.swing.tree.TreeCellEditor)**.

Precizările legate de **InputMap** și **ActionMap** sunt aceleași ca pentru JTable (din secțiunea anterioră).

2 Probleme de laborator

2.1 Problema 1

(0.5 puncte) Să se compileze și să se execute exemplul **TableExample3** anexat la textul laboratorului.

2.2 Problema 2

(1.5 puncte) Să se scrie o clasă **NewTable** pentru afișarea unui tabel (obiect **JTable**) cu atributele fișierelor dintr-un director dat: nume, dimensiune, data ultimei modificari și dacă este director sau nu. Numele directorului se preia dintr-un câmp text. La modificarea câmpului text se va modifica și conținutul tabelului.

2.3 Problema 3

(3 puncte) Să se modifice programul de la punctul anterior prin adăugarea următoarelor componente grafice:

- două etichete JLabel (*row* și *col*)
- două câmpuri text JTextField în care se vor afișa numărului liniei și coloanei selectate de utilizator

Să se definească două clase ascultător compatibile cu interfața **ListSelectionListener**, cu metoda **valueChanged()**, cu argument de tip **ListSelectionEvent**. Metoda extrage numărul liniei sau coloanei cu metoda **getMinSelectionIndex()** și afișează acest număr în câmpul text. Să se adauge cei doi ascultători la obiectele **ListSelectionModel** extrase cu metoda **getSelectionModel()**:

```
ListSelectionModel rowSM = table.getSelectionModel();
ListSelectionModel colSM = table.getColumnModel().getSelectionModel();
```

Să se activeaze selecția de coloane și celule:

```
table.setColumnSelectionAllowed( true );
table.setCellSelectionEnabled( true );
```

Să se compileze și să se execute acest program.

2.4 Problema 4

(2.5 puncte) Să se scrie un program pentru afișarea unui arbore (obiect **JTree**) cu fișierele dintr-un director și din toate subdirectoarele sale. La modificarea câmpului text se modifică și conținutul afișat al arborelui.

```
/* Functia recursiva de creare a arborelui pe baza unui director dat.  */
static void dirlist ( File d, TNode r ) {
       if( !d.isDirectory() ) return;
       File [] files = d.listFiles(); // fisiere din directorul d
       for( int i = 0; i < files.length; i ++ ){
            File file = files[i];
            TNode s = new TNode( file );
            r.add( s );
            dirlist( file, s );
       }
}
```

2.5 Problema 5

(2.5 puncte) Să se adauge programului anterior posibilitatea de selecție a unui nod din arbore, cu afișarea valorii nodului selectat într-un al doilea câmp text.

Pentru a obține nodul selectat se pot folosi mai multe metode:

- metoda **getSelectionPath()** din clasa **JTree** (cu rezultat **TreePath**)
- metoda **getLastSelectedPathComponent()** din clasa **JTree** (cu rezultat **Object**)
- metoda **getPath()** din clasa **TreeSelectionEvent** (cu rezultat **TreePath**)
- metoda **getNewLeadSelectionPath()** din clasa **TreeSelectionEvent**

Clasa **TreePath** corespunde unui vector **Object[ ]**.

2.6 Problema 6 (bonus)

(3 puncte) Să se adauge programului de la punctul 4 un buton de ștergere a nodului selectat (împreună cu tot subarborele său).

Pentru a elimina nodul curent dintr-un arbore se poate:

- obține nodul părinte cu metoda **currentNode.getParent()**
- elimina un fiu din nodul părinte cu metoda **model.removeNodeFromParent(currentNode)**