Programare orientată pe obiecte

Breviar - Laboratorul 11

Mihai Nan
mihai.nan.cti@gmail.com

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2015 - 2016

1 Interfete grafice

1.1 Liste de selectie

Un obiect de tip ***JList*** prezinta utilizatorului un grup de item-uri, afisate in una sau mai multe coloane, pentru a oferi acestuia posibilitatea de a alege.

1.1.1 Crearea unui model

Exista trei metode prin care se poate crea un model de lista:

- ***DefaultListModel*** - aproape totul este gestionat de model pentru a ussura munca programatorului;
- ***AbstractListModel*** - utilizatorul gestioneaza datele si invoca metodele de actionare. Pentru aceasta abordare, programatorul trebuie sa subclaseze ***AbstractListModel*** si sa implementeze metodele ***getSize()*** si ***getElementAt()*** mostenite din interfata ***ListModel***;
- ***ListModel*** - utilizatorul gestioneaza totul.

1.1.2 Initializarea unui liste

Iata un exemplu de cod care creaza si seteaza parametrii unei liste:

Cod sursa Java

```
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));
```

Alti constructori pentru ***JList*** permit initializarea listei dintr-un obiect de tip ***Vector*** sau dintr-un obiect de tipul ***ListModel***. Daca initializarea se face dintr-un obiect de tip ***Vector*** sau dintr-un vector clasic intrinsec, constructorul implicit creeaza un model standard de lista. Modelul standard este imutabil - nu se pot adauga, sterge, inlocui intrari in lista. Pentru a crea o lista a carei intrari pot fi modificate individual, se poate seta modelul listei catre o instanta variabila, cum ar fi ***DefaultListModel***. Modelul unei liste se poate seta cand se creaza lista sau prin apelul metodei ***setModel()***.

Apelul ***setSelectionMode()*** specifica cate intrari din lista poate selecta ultilizatorul si daca acestea trebuie sa fie sau nu continue.

Apelul ***setLayoutOrientation()*** permite afisarea datelor in mai multe coloane. Valoarea ***JList.HORIZONTAL_WRAP*** specifica faptul ca lista ar trebui sa isi afiseze intrarile de la stanga la dreapta inainte de a trece la o noua linie. O alta valoare posibila este ***JList.VERTICAL_WRAP***, care specifica afisarea datelor de sus pana jos (ca de obicei) inainte de a trece la o coloana noua.

In combinatie cu apelul ***setLayoutOrientation()***, invocarea metodei ***setVisibleRowCount(-1)*** ofera listei posibilitatea de afisare a numarului maxim de intrari cuprins in spatiul disponibil pe ecran. O alta utlizare pentru ***setVisibleRowCount()*** este aceea de a indica panoului de scroll-ing asociat listei cate randuri sa afiseze.

1.1.3 Selectarea intrarilor dintr-o lista

O lista utilizeaza o instanta de ***ListSelectionModel*** pentru a-si gestiona selectia. Implicit, modelul de selectie a unei liste permite selectia oricarei combinatii de intrari la un moment dat. Se poate specifica un mod diferit de selectie prin apelul metodei ***setSelectionMode()***:

- ***SINGLE SELECTION*** - numai o singura intrare poate fi selectata la un moment dat. Cand utilizatorul selecteaza o intrare, orice intrare anterior aleasa este deselectata;
- ***SINGLE INTERVAL SELECTION*** - intrari multiple si continue pot fi selectate. Cand utilizatorul incepe un nou interval de selectie, orice intrari anterior alese sunt deselectate;
- ***MULTIPLE INTERVAL SELECTION*** - mod implicit.

Indiferent de modelul de selectie folosit de lista, aceasta genereaza evenimente de selectie de fiecare data cand selectia se schimba. Aceste evenimente pot fi procesate prin adaugarea unui ascultator la lista folosind metoda ***addListSelectionListener()***. Acest ascultator trebuie sa implementeze o metoda: ***valueChanged()***.

Cod sursa Java

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

Multe evenimente de selectie pot fi generate de o singura actiune a utilizatorului, cum ar fi un click. Metoda ***getValuesAdjusting()*** returneaza ***true*** daca utilizatorul inca manipuleaza selectia. Programul de mai sus este interesat doar de rezultatul final al actiunii utilizatorului si, de aceea, metoda ***valueChanged()*** contine secventa de cod doar daca ***getValuesAdjusting()*** intoarce ***false***.

Deoarece lista este in modul ***SINGLE_SELECTION***, acest cod poate utiliza metoda ***getSelectedIndex()*** pentru a obtine indexul intrarii tocmai selectate. ***JList*** pune la dispozitie si alte metode pentru a seta sau a obtine selectia cand se pot alege mai multe intrari. De asemenea, se pot asculta evenimente direct pe modelul de selectie, daca acest lucru este dorit.

1.1.4 Adaugarea si stergerea

Cod sursa Java

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

Programul de mai sus utilizeaza o instanta ***DefaultListModel***. In ciuda numelui, lista nu detine un ***DefaultListModel*** decat daca acest lucru este setat explicit din program. Daca ***DefaultListModel*** nu este potivit nevoilor programatorului, se poate scrie un model customizat dar care trebuie sa adere la interfata ***ListModel***.

Alta metoda de adaugare, cu specificare exacta a pozitiei de inserare, este cea care uziteaza metoda ***insertElementAt***. Stergerea este, de asemenea, foarte simpla, deoarece exista metoda ***remove***.

Cod sursa Java

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

1.1.5 Formatarea celulelor

O lista foloseste un obiect numit ***Cell Renderer*** pentru a-si afisa intrarile. Cel implicit stie sa afiseze siruri de caractere si imagini si afiseaza tipul ***Object*** invocand metoda ***toString()***. Daca se doreste schimbarea modului in care se face afisarea sau daca se doreste un comportament diferit fata de cel oferit de ***toString()***, se poate implementa un ***Cell Renderer*** customizat. Pasii ce trebuie urmati sunt:

- scrierea unei clase care implementeaza interfata ***ListCellRenderer***;
- crearea unei instante de clasa si apelarea metodei ***setCellRenderer()*** pentru lista, folosind instanta ca argument.

1.2 Tabele

Clasa ***JTable*** este folosita pentru a afisa si edita tabele de celule in doua dimensiuni. Consultati tutorialul How to Use Tables pentru documentatie task-oriented si exemple de utilizare.

JTable detine numeroase facilitati care permit customizarea dupa preferinte dar in acelasi timp ofera si optiuni standard pentru aceste facilitati astfel incat tabele simple pot fi create foarte rapid si usor. De exemplu, un tabel cu 10 linii si 10 coloane se poate obtine astfel:

Cod sursa Java

```
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);
```

Cand se doreste scrierea de aplicatii care folosesc ***JTable***, este necesar sa se acorde putina atentie structurilor de date care vor reprezenta datele din table. ***DefaultTableModel*** este o implementare de model care foloseste un vector de vectori de obiecte (de tipul ***Object***) pentru a stoca valorile din celule. La fel cum se pot copia datele dintr-o aplicatie in instanta ***DefaultTableMode***, este, de asemenea, posibil sa se ascunda datele in metodele interfetei ***TableModel*** astfel incat acestea sa poata fi transmise direct catre ***JTable***, la fel ca in exemplul de mai sus. Aceasta abordare duce deseori la aplicatii mai eficiente, deoarece modelul este liber sa aleaga reprezentarea interna care se potriveste cel mai bine datelor manipulate. Se recomanda folosirea ***AbstractTableModel*** ca si clasa de baza pentru crearea de subclase, respectiv ***DefaultTableModel*** atunci cand subclasa nu este necesara.

JTable foloseste exclusiv variabile intregi pentru a referi liniile si coloanele modelului pe care il afiseaza. Este folosita metoda ***getValueAt(int, int)*** pentru a intoarce valorile din model pe parcursul desenarii. Coloanele pot fi rearanjate in tabel astfel incat acestea sa apara intr-o ordine diferita fata de cea din model. Acest fapt nu afecteaza deloc implementarea: atunci cand coloanele sunt rearanjate, obiectul de tip ***JTable*** mentine intern noua ordine si converteste indicii coloanelor inainte de orice interogare a modelului. Asadar, la programarea unui ***TableModel***, nu este necesara ascultarea dupa evenimente de reordonare de coloane, intrucat modelul va fi interogat in sistemul propriu de coordonate indiferent de ce se intampla la vizualizare.

In versiunea curenta de Java sunt adaugate metode la clasa ***JTable*** care permit acces convenient catre nevoi obisnuite de afisare. Noile metode ***print()*** adauga cu usurinta suport de printare aplicatiei ce se doreste a fi dezvoltata. In plus, noua metoda ***getPrintable(javax.swing.JTable.PrintMode, java.text.MessageFormat, java.text.MessageFormat)*** este disponibila pentru necesitati avansate.

La fel ca pentru toate clasele ***JComponent***, se pot folosi ***InputMap*** si ***ActionMap*** pentru a asocia o actiune cu tasta si a executa actiunea in conditii specificate.

1.3 Arbori

Clasa***JTree*** perminte afisarea datelor ierarhice (sub forma unei schite). Pentru documentatie task-oriented si exemple de utilizare consultati tutorialul How to Use Trees.

Un nod specific poate fi identificat fie printr-un ***TreePath*** (un obiect care incapsuleaza nodul si toti stramosii acestuia), fie prin linia de afisare, unde fiecare linie din zona de afisare contine un singur nod. Un nod expandat este un nod care nu este frunza (metoda ***TreeModel.isLeaf(node)*** intoarce ***false***) si care isi va afisa copiii cand toti stramosii sai sunt expandati. Un nod colapsat este un nod care isi ascunde copiii. Un nod ascuns este un nod care este situat sub un stramos colapsat. Toti parintii unui nod care poate fi vizualizat sunt expandati, dar acestia pot sau nu fi afisati. Un nod afisat se regaseste in zona de afisare si poate fi vizualizat.

Urmatoarele metode din clasa ***JTree*** folosesc cuvantul ***visible*** pentru a se referi la afisat:

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

Urmatorul grup de metode folosesc cuvantul ***visible*** pentru a se referi la poate fi vizualizat (sub un parinte expandat):

- isVisible();
- makeVisible();

Pentru a detecta schimbarea selectiei, se va implementa interfata ***TreeSelectionListener*** si se va adauga o instanta folosind ***addTreeSelectionListener()***. Metoda ***valueChanged()*** va fi invocata atunci cand utilizatorul selecteaza alt nod, si doar o data, chiar daca se efectueaza un click de doua ori pe acelasi nod. Cu toate acestea, pentru a face separarea cazurilor de dublu click, indiferent de selectia anterioara, este recomandata abordarea prezentata in codul prezentat mai jos.

Cod sursa Java

```
//final JTree tree = ... ;
MouseListener ml = new MouseAdapter() {
    public void mousePressed(MouseEvent e) {
        int selRow;
        selRow = tree.getRowForLocation(e.getX(),e.getY());
        TreePath selPath;
        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 afisarea nodurilor complexe (de exemplu, noduri continand atat text, cat si o icoaita) se va implementa interfata ***TreeCellRenderer*** si se va folosi metoda ***setCellRenderer(javax.swing.tree.TreeCellRenderer)***. Pentru a edita astfel de noduri, se va implementa interfata ***TreeCellEditor*** si se va folosi metoda ***setCellEditor(TreeCellEditor)***.

Precizarile legate de ***InputMap*** si ***ActionMap*** sunt aceleasi ca pentru ***JTable*** (din sectiunea anteriora).