TEHNOLOGII JAVA
PENTRU DEZVOLTAREA APLICAȚIILOR
LUCRARE DE LABORATOR 15

JTree. JTable. Meniuri. JToolBar

I. SCOPUL LUCRĂRII

Lucrarea de față are rolul de a prezenta şi familiariza studentul cu modul de construire a unei interfețe grafice utilizator folosind pachetul de clase **javax.swing.** Se vor prezenta câteva componente vizuale utile, împreună cu modul de creare şi utilizare a acestora.

La sfârşitul acestei lucrări, studentul va avea posibilitatea să scrie programe Java în care să utilizeze noțiunile învățate.

II. NOȚIUNI TEORETICE

1. JTree

Vezi noțiunile teoretice prezentate la curs.

Să considerăm următorul exemplu - o instanță **JTree** conținând setul complet de cărți al unui pachet de cărți de joc. Să observăm utilizarea clasei **DefaultMutableTreeNode**, care reprezintă un nod cu scop general al arborelui. Fiecare nod fiu este inserat în nodul părinte utilizându-se metoda **add()**, şi fiecare nod fiu poate fi la rândul său nod părinte pentru alte noduri. Rădăcina este un nod special pentru că este părintele ultim al tuturor celorlalte noduri din arbore. Nodul rădăcină este inserat în modelul de date al arborelui, care pentru exemplul nostru este o instanță a clasei **DefaultTreeModel**.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
class Tree extends JFrame
{
private JPanel topPanel;
private JTree tree;
private JScrollPane scrollPane;
public Tree()
{
setTitle( "More Advanced Tree Application" );
setSize( 300, 100 );
setBackground( Color.gray );
// Creeaza un panou pentru a stoca toate celelalte componente
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
```

```
getContentPane().add( topPanel );
// Creeaza datele pentru arbore
DefaultMutableTreeNode root= new DefaultMutableTreeNode( "Deck" );
DefaultMutableTreeNode itemClubs=new DefaultMutableTreeNode( "Clubs" );
addAllCard( itemClubs );
root.add( itemClubs );
DefaultMutableTreeNode itemDiamonds=
                                new DefaultMutableTreeNode("Diamonds");
addAllCard( itemDiamonds );
root.add( itemDiamonds );
DefaultMutableTreeNode itemSpades=new DefaultMutableTreeNode("Spades");
addAllCard( itemSpades );
root.add( itemSpades );
DefaultMutableTreeNode itemHearts=new DefaultMutableTreeNode("Hearts");
addAllCard( itemHearts );
root.add( itemHearts );
// Creeaza un model pentru arbore
DefaultTreeModel treeModel = new DefaultTreeModel( root );
tree = new JTree( treeModel );
// Add the list box to a scrolling pane
scrollPane = new JScrollPane();
scrollPane.getViewport().add( tree );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
// metoda care adauga intregul set de carti de joc la nodul curent
// al arborelui
public void addAllCard( DefaultMutableTreeNode suit )
{
suit.add( new DefaultMutableTreeNode( "Ace" ) );
suit.add( new DefaultMutableTreeNode( "Two" ) );
suit.add( new DefaultMutableTreeNode( "Three" ) );
suit.add( new DefaultMutableTreeNode( "Four" ) );
suit.add( new DefaultMutableTreeNode( "Five" ) );
suit.add( new DefaultMutableTreeNode( "Six" ) );
suit.add( new DefaultMutableTreeNode( "Seven" ) );
suit.add( new DefaultMutableTreeNode( "Eight" ) );
suit.add( new DefaultMutableTreeNode( "Nine" ) );
suit.add( new DefaultMutableTreeNode( "Ten" ) );
suit.add( new DefaultMutableTreeNode( "Jack" ) );
suit.add( new DefaultMutableTreeNode( "Queen" ) );
suit.add( new DefaultMutableTreeNode( "King" ) );
}
public static void main( String args[] )
{
Tree mainFrame = new Tree();
mainFrame.setVisible( true );
}
}
```

Se poate utiliza un model de date definit de programator (se creează o nouă clasă pentru acest model care extinde **DefaultTreeModel**). De asemenea, se poate defini o clasă responsabilă pentru desenarea fiecărui nod din arbore. Să observăm următorul exemplu.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;
class Tree1 extends JFrame
{
private JPanel topPanel;
private JTree tree;
private JScrollPane scrollPane;
public Tree1()
{
setTitle( "Custom Rendered Tree Application" );
setSize( 300, 200 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
DefaultMutableTreeNode root= new DefaultMutableTreeNode( "Deck" );
DefaultMutableTreeNode itemClubs=new DefaultMutableTreeNode( "Clubs" );
addAllCard( itemClubs );
root.add( itemClubs );
DefaultMutableTreeNode itemDiamonds=
                                new DefaultMutableTreeNode("Diamonds");
addAllCard( itemDiamonds );
root.add( itemDiamonds );
DefaultMutableTreeNode itemSpades=new DefaultMutableTreeNode("Spades");
addAllCard( itemSpades );
root.add( itemSpades );
DefaultMutableTreeNode itemHearts=new DefaultMutableTreeNode("Hearts");
addAllCard( itemHearts );
root.add( itemHearts );
DefaultTreeModel treeModel = new DefaultTreeModel( root );
tree = new JTree( treeModel );
// informeaza arborele ca este desenat utilizand o clasa definita de
// programator
tree.setCellRenderer( new CustomCellRenderer() );
scrollPane = new JScrollPane();
scrollPane.getViewport().add( tree );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
public void addAllCard( DefaultMutableTreeNode suit )
{
suit.add( new DefaultMutableTreeNode( "Ace" ) );
suit.add( new DefaultMutableTreeNode( "Two" ) );
suit.add( new DefaultMutableTreeNode( "Three" ) );
suit.add( new DefaultMutableTreeNode( "Four" ) );
suit.add( new DefaultMutableTreeNode( "Five" ) );
```

```
suit.add( new DefaultMutableTreeNode( "Six" ) );
suit.add( new DefaultMutableTreeNode( "Seven" ) );
suit.add( new DefaultMutableTreeNode( "Eight" ) );
suit.add( new DefaultMutableTreeNode( "Nine" ) );
suit.add( new DefaultMutableTreeNode( "Ten" ) );
suit.add( new DefaultMutableTreeNode( "Jack" ) );
suit.add( new DefaultMutableTreeNode( "Queen" ) );
suit.add( new DefaultMutableTreeNode( "King" ) );
}
public static void main( String args[] )
{
Tree1 mainFrame = new Tree1();
mainFrame.setVisible( true );
}
}

class CustomCellRenderer extends JLabel implements TreeCellRenderer
{
private ImageIcon deckImage;
private ImageIcon[] suitImages;
private ImageIcon[] cardImages;
private boolean bSelected;
public CustomCellRenderer()
{
// Incarca imaginile
deckImage = new ImageIcon( "deck.gif" );
suitImages = new ImageIcon[4];
suitImages[0] = new ImageIcon( "clubs.gif" );
suitImages[1] = new ImageIcon( "diamonds.gif" );
suitImages[2] = new ImageIcon( "spades.gif" );
suitImages[3] = new ImageIcon( "hearts.gif" );
cardImages = new ImageIcon[13];
cardImages[0] = new ImageIcon( "ace.gif" );
cardImages[1] = new ImageIcon( "two.gif" );
cardImages[2] = new ImageIcon( "three.gif" );
cardImages[3] = new ImageIcon( "four.gif" );
cardImages[4] = new ImageIcon( "five.gif" );
cardImages[5] = new ImageIcon( "six.gif" );
cardImages[6] = new ImageIcon( "seven.gif" );
cardImages[7] = new ImageIcon( "eight.gif" );
cardImages[8] = new ImageIcon( "nine.gif" );
cardImages[9] = new ImageIcon( "ten.gif" );
cardImages[10] = new ImageIcon( "jack.gif" );
cardImages[11] = new ImageIcon( "queen.gif" );
cardImages[12] = new ImageIcon( "king.gif" );
}
public Component getTreeCellRendererComponent( JTree tree,
Object value, boolean bSelected, boolean bExpanded,
boolean bLeaf, int iRow, boolean bHasFocus )
{
// Afla care nod este curent desenat si se obtine textul atasat nodului
DefaultMutableTreeNode node = (DefaultMutableTreeNode)value;
```

```
String labelText = (String)node.getUserObject();
this.bSelected = bSelected;
// Seteaza culoarea textului
if( !bSelected )
setForeground( Color.black );
else
setForeground( Color.white );
// Determina imaginea corecta
if( labelText.equals( "Deck" ) )
setIcon( deckImage );
else if( labelText.equals( "Clubs" ) )
setIcon( suitImages[0] );
else if( labelText.equals( "Diamonds" ) )
setIcon( suitImages[1] );
else if( labelText.equals( "Spades" ) )
setIcon( suitImages[2] );
else if( labelText.equals( "Hearts" ) )
setIcon( suitImages[3] );
else if( labelText.equals( "Ace" ) )
setIcon( cardImages[0] );
else if( labelText.equals( "Two" ) )
setIcon( cardImages[1] );
else if( labelText.equals( "Three" ) )
setIcon( cardImages[2] );
else if( labelText.equals( "Four" ) )
setIcon( cardImages[3] );
else if( labelText.equals( "Five" ) )
setIcon( cardImages[4] );
else if( labelText.equals( "Six" ) )
setIcon( cardImages[5] );
else if( labelText.equals( "Seven" ) )
setIcon( cardImages[6] );
else if( labelText.equals( "Eight" ) )
setIcon( cardImages[7] );
else if( labelText.equals( "Nine" ) )
setIcon( cardImages[8] );
else if( labelText.equals( "Ten" ) )
setIcon( cardImages[9] );
else if( labelText.equals( "Jack" ) )
setIcon( cardImages[10] );
else if( labelText.equals( "Queen" ) )
setIcon( cardImages[11] );
else if( labelText.equals( "King" ) )
setIcon( cardImages[12] );
// Add the text to the cell
setText( labelText );
return this;
}
public void paint( Graphics g )
{
Color bColor;
Icon currentI = getIcon();
```

```
// Seteaza culoarea de fond corecta
bColor = bSelected ? SystemColor.textHighlight : Color.white;
g.setColor( bColor );
// Deseneaza un dreptunghi pe fundalul celulei (nodului)
g.fillRect( 0, 0, getWidth() - 1, getHeight() - 1 );
super.paint( g );
}
}
```

Nodurile arborelui se pot edita. Iată un exemplu.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Tree2 extends JFrame
{
private JPanel topPanel;
private JTree tree;
private JScrollPane scrollPane;
public Tree2()
{
setTitle( "Editable Tree Application" );
setSize( 300, 100 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
tree = new JTree();
tree.setEditable( true );
// Creeaza un ComboBox pentru editarea optiunilor
JComboBox box = new JComboBox();
box.addItem( "Swing" );
box.addItem( "Java" );
box.addItem( "neat" );
box.addItem( "funky" );
box.addItem( "life" );
box.addItem( "awesome" );
box.addItem( "cool!" );
// Adauga arborelui un  „cell editor”
tree.setCellEditor( new DefaultCellEditor( box ) );
scrollPane = new JScrollPane();
scrollPane.getViewport().add( tree );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
public static void main( String args[] )
{
Tree2 mainFrame = new Tree2();
mainFrame.setVisible( true );
}
}
```

În continuare vom prezenta două exemple simple, care arată cum se poate proceda pentru tratarea evenimentelor generate de selectarea nodurilor arborelui şi respectiv, tratarea evenimentelor generate de expandarea/comprimarea structurii arborelui.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;
class Tree3 extends JFrame implements TreeSelectionListener
{
private JPanel topPanel;
private JTree tree;
private JScrollPane scrollPane;
public Tree3()
{
setTitle( "TreeSelectionListener Application" );
setSize( 300, 100 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
tree = new JTree();
// Adauga un obiect ascultator pentru selectiile nodurilor arborelui
tree.addTreeSelectionListener( this );
scrollPane = new JScrollPane();
scrollPane.getViewport().add( tree );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
// Manipuleaza selectiile nodurilor
public void valueChanged( TreeSelectionEvent event )
{
if( event.getSource() == tree )
{
// Afiseaza calea completa catre nodul curent selectat
TreePath path = tree.getSelectionPath();
System.out.println( "Selection path="+ path.toString() );
// Obtine textul atasat ultimului nod selectat
System.out.println( "Selection="+ path.getLastPathComponent() );
}
}
public static void main( String args[] )
{
Tree3 mainFrame = new Tree3();
mainFrame.setVisible( true );
}
}

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

```
import javax.swing.event.*;
import javax.swing.tree.*;
class Tree4 extends JFrame implements TreeExpansionListener
{
private JPanel topPanel;
private JTree tree;
private JScrollPane scrollPane;
public Tree4()
{
setTitle( "TreeExpansionListener Application" );
setSize( 300, 300 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
tree = new JTree();
// Adauga un obiect ascultator
tree.addTreeExpansionListener( this );
scrollPane = new JScrollPane();
scrollPane.getViewport().add( tree );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
// Manipuleaza expansiunea nodurilor arborelui
public void treeExpanded( TreeExpansionEvent event)
{
if( event.getSource() == tree )
{
// Afiseaza calea completa catre nodul expandat
TreePath path = event.getPath();
System.out.println( "Node Expanded=" + path.toString() );
}
}
// Manipuleaza comprimarea nodurilor arborelui
public void treeCollapsed( TreeExpansionEvent event )
{
if( event.getSource() == tree )
{
TreePath path = event.getPath();
System.out.println( "Node Collapsed=" + path.toString() );
}
}
public static void main( String args[] )
{
Tree4 mainFrame = new Tree4();
mainFrame.setVisible( true );
}
}
```

2. JTable

JTable este o componentă Java care permite afişarea unor cantităţi mari de date într-un mod simplu bidimensional. Are o înfăţişare similară unei foi de calcul ("spreadsheet"). Nu se poate utiliza întocmai ca o foaie de calcul (datorită unor limitări), dar suportă multe alte caracteristici. Să considerăm următorul exemplu:

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
class Table1 extends JFrame
{
private JPanel topPanel;
private JTable table;
private JScrollPane scrollPane;
private String columnNames[];
private String dataValues[][];
public Table1()
{
setTitle( "Advanced Table Application" );
setSize( 300, 200 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza coloane
CreateColumns();
CreateData();
// Creeaza o instanta tabel
table = new JTable( dataValues, columnNames );
// Configureaza cativa din parametrii JTable
table.setShowHorizontalLines( false );
table.setRowSelectionAllowed( true );
table.setColumnSelectionAllowed( true );
// Modifica culoarea de selectie
table.setSelectionForeground( Color.white );
table.setSelectionBackground( Color.red );
// Adauga tabelul la un "scrolling pane"
scrollPane = table.createScrollPaneForTable( table );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
public void CreateColumns()
{
// Creeaza etichetele coloanelor
columnNames = new String[8];
for( int iCtr = 0; iCtr < 8; iCtr++ )
columnNames[iCtr] = "Col:" + iCtr;
}
public void CreateData()
{
```

```
// Creeaza fiecare element
dataValues = new String[100][8];
for( int iY = 0; iY < 100; iY++ )
{
for( int iX = 0; iX < 8; iX++ )
{
dataValues[iY][iX] = "" + iX + "," + iY;
}
}
}
public static void main( String args[] )
{
Table1 mainFrame = new Table1();
mainFrame.setVisible( true );
}
}
```

La fel ca multe alte componente Swing, **JTable** suportă înlocuirea modelului său de date implicit cu unul definit de programator, şi de asemenea, permite utilizarea unei clase (definită de programator) responsabilă pentru desenarea fiecărei celule a tabelului.

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.border.*;
class Table2 extends JFrame
{
private JPanel topPanel;
private JTable table;
private JScrollPane scrollPane;
private String columnNames[];
private String dataValues[][];
public Table2()
{
setTitle( "Custom Header Rendering Application" );
setSize( 300, 200 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza un model de date
CustomDataModel customDataModel = new CustomDataModel();
table = new JTable( customDataModel );
CreateColumns();
table.setShowHorizontalLines( false );
table.setRowSelectionAllowed( true );
table.setColumnSelectionAllowed( true );
table.setSelectionForeground( Color.white );
table.setSelectionBackground( Color.red );
scrollPane = table.createScrollPaneForTable( table );
```

```
topPanel.add( scrollPane, BorderLayout.CENTER );
}
public void CreateColumns()
{
// Spunem ca vom crea manual coloanele
table.setAutoCreateColumnsFromModel( false );
for( int iCtr = 0; iCtr < 4; iCtr++ )
{
// Manual cream o noua coloana
TableColumn column = new TableColumn( iCtr );
column.setHeaderValue( (Object)("Col:" + iCtr) );
// Adaugam un obiect care se va ocupa de desenarea unei celule header
column.setHeaderRenderer( new CustomHeaderRenderer() );
// Adaugam coloana la tabel
table.addColumn( column );
}
}
public static void main( String args[] )
{
Table2 mainFrame = new Table2();
mainFrame.setVisible( true );
}
}

class CustomDataModel extends AbstractTableModel
{
public Object getValueAt( int iRowIndex, int iColumnIndex )
{
return "" + iColumnIndex + "," + iRowIndex;
}
public void setValueAt( Object aValue, int iRowIndex,int iColumnIndex )
{
}
public int getColumnCount()
{
return 0;
}
public int getRowCount()
{
return 500;
}
}

class CustomHeaderRenderer extends JLabel implements TableCellRenderer
{
private boolean isSelected;
private boolean hasFocus;
private ImageIcon[] suitImages;
public CustomHeaderRenderer()
{
suitImages = new ImageIcon[4];
suitImages[0] = new ImageIcon( "clubs.gif" );
```

```
suitImages[1] = new ImageIcon( "diamonds.gif" );
suitImages[2] = new ImageIcon( "spades.gif" );
suitImages[3] = new ImageIcon( "hearts.gif" );
}
public Component getTableCellRendererComponent( JTable table,
Object value, boolean isSelected, boolean hasFocus,int row, int column
)
{
// Obtine textul care se va afisa
String sText = (String)value;
// Seteaza optiuni de aliniament
setVerticalAlignment( SwingConstants.CENTER );
setHorizontalAlignment( SwingConstants.CENTER );
setHorizontalTextPosition( SwingConstants.CENTER );
setVerticalTextPosition( SwingConstants.BOTTOM );
// Asigneaza o margine
setBorder( new TitledBorder( new EtchedBorder(), sText ) );
// Populeaza cu imagine si text
setIcon( suitImages[column] );
// Set the text to the correct suit
switch( column )
{
case 0:
setText( "Clubs" );
break;
case 1:
setText( "Diamonds" );
break;
case 2:
setText( "Hearts" );
break;
case 3:
setText( "Spades" );
break;
}
return this;
}
}
```

În unele situații, va fi nevoie ca aplicația să efectueze sarcini speciale atunci când mouse-ul este poziționat pe o anume celulă a tabelului. Implicit, clasa **JTable** nu oferă această capabilitate. Deci, pentru a manipula evenimentele generate de mouse sau de tastatura, se procedează ca mai jos – se derivează o nouă clasă din clasa **JTable**:

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
class Table3 extends JFrame
{
```

```
private JPanel topPanel;
private JTable table;
private JScrollPane scrollPane;
private String columnNames[];
private String dataValues[][];
public Table3()
{
setTitle( "Custom Table Data Model Application" );
setSize( 300, 200 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza modelul de date
CustomDataModel customDataModel = new CustomDataModel();
table = new MyTable( customDataModel );
CreateColumns();
table.setShowHorizontalLines( false );
table.setRowSelectionAllowed( true );
table.setColumnSelectionAllowed( true );
table.setSelectionForeground( Color.white );
table.setSelectionBackground( Color.red );
scrollPane = table.createScrollPaneForTable( table );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
public void CreateColumns()
{
// Spunem ca vom crea coloanele manual
table.setAutoCreateColumnsFromModel( false );
for( int iCtr = 0; iCtr < 8; iCtr++ )
{
// Manual cream a noua coloana
TableColumn column = new TableColumn( iCtr );
column.setHeaderValue( (Object)("Col:" + iCtr) );
// Adaugam coloana la tabel
table.addColumn( column );
}
}
public static void main( String args[] )
{
Table3 mainFrame = new Table3();
mainFrame.setVisible( true );
}
}

class CustomDataModel extends AbstractTableModel
{
public Object getValueAt( int iRowIndex, int iColumnIndex )
{
return "" + iColumnIndex + "," + iRowIndex;
}
```

```
public void setValueAt( Object aValue, int iRowIndex,int iColumnIndex )
{ }
public int getColumnCount()
{
return 0;
}
public int getRowCount()
{
return 500;
}
}

class MyTable extends JTable implements MouseListener
{
public MyTable( CustomDataModel model )
{
super( model );
// Configure the table
setFont( new Font( "Helvetica", Font.PLAIN, 12 ) );
setColumnSelectionAllowed( false );
setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
setShowGrid( false );
setIntercellSpacing( new Dimension( 0, 1 ) );
setAutoCreateColumnsFromModel( false );
sizeColumnsToFit( true );
// Prevent table column reordering
JTableHeader header = getTableHeader();
header.setUpdateTableInRealTime( false );
header.setReorderingAllowed( false );
// Attach a mouse listener
addMouseListener( this );
}
public void mouseClicked( MouseEvent e )
{
int iMouseX = e.getX();
int iMouseY = e.getY();
int iSelectedColumn = columnAtPoint(new Point( iMouseX, iMouseY ) );
int iSelectedRow = rowAtPoint( new Point( iMouseX, iMouseY ) );
System.out.println("S-a selectat celula de la pozitia "+iSelectedRow+"
, "+iSelectedColumn);
}
public void mouseEntered( MouseEvent e )
{}
public void mouseExited( MouseEvent e )
{}
public void mousePressed( MouseEvent e )
{}
public void mouseReleased( MouseEvent e )
{}
}
```

3. Meniuri

Swing implementează o clasă **JMenuBar** care, pentru compatibilitate, suportă API-ul AWT pentru barele de meniu. Dar, oferă şi facilităţi în plus, cum ar fi faptul că barele de meniu Swing pot fi plasate oriunde în ferastra aplicaţiei, iar instanţele **JMenuBar** pot fi aplicate şi applet-urilor.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Meniu1 extends JFrame
{
private JPanel topPanel;
private JMenuBar menuBar;
private JMenu menuFile;
private JMenu menuEdit;
private JMenu menuProperty;
public Meniu1()
{
setTitle( "Menu Application" );
setSize( 310, 130 );
topPanel = new JPanel();topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza  bara de meniu
menuBar = new JMenuBar();
// Seteaza aceasta instanta ca fiind bara de meniu a aplicatiei
setJMenuBar( menuBar );
// creeaza un submeniu "Properties"
menuProperty = new JMenu( "Properties" );
// Creeaza un meniu "File"
menuFile = new JMenu( "File" );
menuBar.add( menuFile );
// Adauga submeniul "Properties" la meniul "File"
menuFile.addSeparator();
menuFile.add( menuProperty );
menuFile.addSeparator();
// Creeaza  meniul "Edit"
menuEdit = new JMenu( "Edit" );
menuBar.add( menuEdit );
}
public static void main( String args[] )
{
Meniu1 mainFrame = new Meniu1();
mainFrame.setVisible( true );
}
}
```

Să considerăm un exemplu mai complex.

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

```
import javax.swing.*;
class Meniu2 extends JFrame implements ActionListener
{
private final int ITEM_PLAIN = 0; // Item types
private final int ITEM_CHECK = 1;
private final int ITEM_RADIO = 2;
private JPanel topPanel;
private JMenuBar menuBar;
private JMenu menuFile;
private JMenu menuEdit;
private JMenu menuProperty;
private JMenuItem menuPropertySystem;
private JMenuItem menuPropertyEditor;
private JMenuItem menuPropertyDisplay;
private JMenuItem menuFileNew;
private JMenuItem menuFileOpen;
private JMenuItem menuFileSave;
private JMenuItem menuFileSaveAs;
private JMenuItem menuFileExit;
private JMenuItem menuEditCopy;
private JMenuItem menuEditCut;
private JMenuItem menuEditPaste;
public Meniu2()
{
setTitle( "Complete Menu Application" );
setSize( 310, 130 );topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza bara de meniu
menuBar = new JMenuBar();
// Seteaza aceasta instanta ca fiind bara de meniu a aplicatiei
setJMenuBar( menuBar );
// creeaza un submeniu "Properties"
menuProperty = new JMenu( "Properties" );
menuProperty.setMnemonic( 'P' );
// Creeaza optiunile din submeniul "Properties"
menuPropertySystem =
CreateMenuItem( menuProperty,ITEM_PLAIN,"System...", null, 'S', null );
menuPropertyEditor =
CreateMenuItem(menuProperty, ITEM_PLAIN,"Editor...", null, 'E', null );
menuPropertyDisplay =
CreateMenuItem(menuProperty,ITEM_PLAIN,"Display...", null, 'D', null );
// Creeaza un meniu "File"
menuFile = new JMenu( "File" );
menuFile.setMnemonic( 'F' );
menuBar.add( menuFile );
// Construieste optiunile meniului "File"
menuFileNew=CreateMenuItem( menuFile,ITEM_PLAIN,"New",null,'N', null );
menuFileOpen =
CreateMenuItem(menuFile,ITEM_PLAIN,"Open...",new ImageIcon("open.gif"),
'O',"Open a new file" );
```

```
menuFileSave = CreateMenuItem( menuFile, ITEM_PLAIN, "Save",
new ImageIcon( "save.gif" ), 'S'," Save this file" );
menuFileSaveAs = CreateMenuItem( menuFile, ITEM_PLAIN,"Save As...",
null, 'A',"Save this data to a new file" );
// Adauga submeniul "Properties" la meniul "File"
menuFile.addSeparator();
menuFile.add( menuProperty );
menuFile.addSeparator();
menuFileExit = CreateMenuItem( menuFile, ITEM_PLAIN,"Exit",
null, 'x',"Exit the program" );
// Creeaza meniul "Edit"
menuEdit = new JMenu( "Edit" );
menuEdit.setMnemonic( 'E' );
menuBar.add( menuEdit );
// Creeaza optiunile meniului "Edit"
menuEditCut = CreateMenuItem( menuEdit, ITEM_PLAIN,"Cut", null, 't',
"Cut data to the clipboard" );
menuEditCopy = CreateMenuItem( menuEdit, ITEM_PLAIN,"Copy", null, 'C',
"Copy data to the clipboard" );
menuEditPaste = CreateMenuItem( menuEdit, ITEM_PLAIN,"Paste", null,
'P',"Paste data from the clipboard" );
}

public JMenuItem CreateMenuItem( JMenu menu, int iType,String sText,
                  ImageIcon image,int acceleratorKey, String sToolTip )
{
// Creeaza optiunea meniului
JMenuItem menuItem;
switch( iType )
{
case ITEM_RADIO:
menuItem = new JRadioButtonMenuItem();
break;
case ITEM_CHECK:
menuItem = new JCheckBoxMenuItem();
break;
default:
menuItem = new JMenuItem();
break;
}
// Adauga textul optiunii
menuItem.setText( sText );
// Adauga imaginea (optional)
if( image != null )
menuItem.setIcon( image );
// Adauga acceleratorul (combinatia de taste)
if( acceleratorKey > 0 )
menuItem.setMnemonic( acceleratorKey );
// Adauga textul pentru "tool tip" (optional)
if( sToolTip != null )
menuItem.setToolTipText( sToolTip );
```

```
//Adauga un obiect ascultator de evenimente pentru aceasta optiune a
// meniului
menuItem.addActionListener( this );
menu.add( menuItem );
return menuItem;
}
public void actionPerformed( ActionEvent event )
{
System.out.println( event );
}
public static void main( String args[] )
{
Meniu2 mainFrame = new Meniu2();
mainFrame.setVisible( true );
}
}
```

În Swing, opțiunea din meniu de tipul "check box" este implementată într-o clasă **JCheckBoxMenuItem**. Ca orice componentă **JMenuItem**, suportă text sau imagine.

```
import java.awt.*;
import javax.swing.*;
class Meniu3 extends JFrame
{
private JPanel topPanel;
public Meniu3()
{
setTitle( "Menu Application #2" );
setSize( 310, 130 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
JMenuBar menuBar = new JMenuBar();
setJMenuBar( menuBar );
JMenu optionMenu = new JMenu( "Menu" );
menuBar.add( optionMenu );
// Creeaza optiunile meniului "check box"
JCheckBoxMenuItem menuEditInsert= new JCheckBoxMenuItem( "Insert" );
optionMenu.add( menuEditInsert );
JCheckBoxMenuItem menuEditWrap= new JCheckBoxMenuItem( "Wrap lines" );
optionMenu.add( menuEditWrap );
JCheckBoxMenuItem menuEditCaps= new JCheckBoxMenuItem( "Caps Lock" );
optionMenu.add( menuEditCaps );
}
public static void main( String args[] )
{
Meniu3 mainFrame = new Meniu3();
mainFrame.setVisible( true );
}
}
```

Swing furnizează şi o altă clasă numită **JRadioButtonMenuItem**. Să urmărim exemplul:

```
import java.awt.*;
import javax.swing.*;
class Meniu4 extends JFrame
{
private JPanel topPanel;
public Meniu4()
{
setTitle( "Menu Application #3" );
setSize( 310, 130 );
topPanel = new JPanel();
getContentPane().add( topPanel );
JMenuBar menuBar = new JMenuBar();
setJMenuBar( menuBar );
JMenu optionMenu = new JMenu( "Menu" );
menuBar.add( optionMenu );
JRadioButtonMenuItem menuCursorSmall =
                      new JRadioButtonMenuItem( "Small Cursor" );
optionMenu.add( menuCursorSmall );
JRadioButtonMenuItem menuCursorMedium =
                      new JRadioButtonMenuItem( "Medium Cursor" );
optionMenu.add( menuCursorMedium );
JRadioButtonMenuItem menuCursorLarge =
                      new JRadioButtonMenuItem( "Large Cursor" );
optionMenu.add( menuCursorLarge );
ButtonGroup cursorGroup = new ButtonGroup();
cursorGroup.add( menuCursorSmall );
cursorGroup.add( menuCursorMedium );
cursorGroup.add( menuCursorLarge );
}
public static void main( String args[] )
{
Meniu4 mainFrame = new Meniu4();
mainFrame.setVisible( true );
}
}
```

În final vom examina clasa **JPopupMenu** (pentru a apare meniul, se va apăsa butonul dreapta al mouse-ului).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class Meniu5 extends JFrame implements ActionListener
{
private JPanel topPanel;
private JPopupMenu popupMenu;
public Meniu5()
{
```

```
setTitle( "Popup Menu Application" );
setSize( 310, 130 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( null );
getContentPane().add( topPanel );
// Creeaza optiunile pentru meniul pop-up
JMenuItem menuFileNew = new JMenuItem( "New" );
JMenuItem menuFileOpen = new JMenuItem( "Open..." );
JMenuItem menuFileSave = new JMenuItem( "Save" );
JMenuItem menuFileSaveAs = new JMenuItem( "Save As..." );
JMenuItem menuFileExit = new JMenuItem( "Exit" );
// Creeaza un meniu pop-up
popupMenu = new JPopupMenu( "Menu" );
popupMenu.add( menuFileNew );
popupMenu.add( menuFileOpen );
popupMenu.add( menuFileSave );
popupMenu.add( menuFileSaveAs );
popupMenu.add( menuFileExit );
topPanel.add( popupMenu );
//Suport pentru ascultarea evenimentelor generate de actiunile cu
// mouse-ul
enableEvents( AWTEvent.MOUSE_EVENT_MASK );
menuFileNew.addActionListener( this );
menuFileOpen.addActionListener( this );
menuFileSave.addActionListener( this );
menuFileSaveAs.addActionListener( this );
menuFileExit.addActionListener( this );
}
public void processMouseEvent( MouseEvent event )
{
if( event.isPopupTrigger() )
{
popupMenu.show( event.getComponent(),
event.getX(), event.getY() );
}
super.processMouseEvent( event );
}
public void actionPerformed( ActionEvent event )
{
System.out.println( event );
}
public static void main( String args[] )
{
Meniu5 mainFrame = new Meniu5();
mainFrame.setVisible( true );
}
}
```

4. ToolBar

Să considerăm următorul exemplu.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MyToolBar extends JFrame implements ActionListener
{
private JPanel topPanel;
private JButton buttonNew;
private JButton buttonOpen;
private JButton buttonSave;
private JButton buttonCopy;
private JButton buttonCut;
private JButton buttonPaste;
public MyToolBar()
{
setTitle( "Basic Toolbar Application" );
setSize( 310, 130 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeza un toolbar
JToolBar myToolbar = new JToolBar();
topPanel.add( myToolbar, BorderLayout.NORTH );
// Adauga butoane la toolbar
buttonNew = addToolbarButton( myToolbar, false, "New",
                              "new", "Create a new document" );
buttonOpen = addToolbarButton( myToolbar, true, "Open","open",
                              "Open an existing document" );
buttonSave = addToolbarButton( myToolbar, true, "Save","save",
                              "Open an existing document" );
myToolbar.addSeparator();
buttonCopy = addToolbarButton( myToolbar, true, null,"copy",
                               "Copy selection to the clipboard" );
buttonCut = addToolbarButton( myToolbar, true, null,"cut",
                               "Cut selection to the clipboard" );
buttonPaste = addToolbarButton( myToolbar, true, null,"paste",
                                "Paste selection from the clipboard" );
// Adauga o zona de editare pentru a umple spatiul
JTextArea textArea = new JTextArea();
topPanel.add( textArea, BorderLayout.CENTER );
}
// Metoda care creeaza butoanele din toolbar
public JButton addToolbarButton( JToolBar toolBar,boolean bUseImage,
                  String sButtonText,String sButton, String sToolHelp )
{
JButton b;
// Creeaza un nou buton
```

```
if( bUseImage )
b = new JButton( new ImageIcon( sButton + ".gif" ) );
else
b = (JButton)toolBar.add( new JButton() );
// Adauga butonul la toolbar
toolBar.add( b );
// Adauga text la buton (optional)
if( sButtonText != null )
b.setText( sButtonText );
else
{
b.setMargin( new Insets( 0, 0, 0, 0 ) );
}
// Adauga "tool tip" (optional)
if( sToolHelp != null )
b.setToolTipText( sToolHelp );
// Ne asiguram ca butonul trimite un mesaj cand utilizatorul face click
// pe el
b.setActionCommand( "Toolbar:" + sButton );
b.addActionListener( this );
return b;
}
public void actionPerformed( ActionEvent event )
{
System.out.println( event );
}
public static void main( String args[] )
{
MyToolBar mainFrame = new MyToolBar();
mainFrame.setVisible( true );
}
}
```

III. MODUL DE LUCRU

1. Se editează codul sursă al programului Java folosind un editor de text disponibil(de ex., se poate utiliza Notepad).
2. Se salvează fişierul cu extensia **.java**. Fişierul trebuie salvat la următoarea locaţie: **c:\JBulider7\jdk1.3.1\bin**
3. Compilarea aplicaţiei Java se va face din linia de comandă. Se poate proceda astfel. Se deschide o fereastră MS-Dos: Start ->Run, se tipăreşte **command** în căsuţa de text şi se apasă butonul OK. Printr-o schimbare de directoare şi subdirectoare se trece la locaţia: **c:\JBulider7\jdk1.3.1\bin.** Sau, se lansează WindowsCommander. Se trece la locaţia **c:\JBulider7\jdk1.3.1\bin.** Se deschide o fereastră MS-Dos: Commander ->Run Dos.
4. Pentru compilare, se tipăreşte la prompter **javac nume_fişier_sursă.java** şi se apasă Enter. De ex., dacă fişierul se numeşte Test.java, se va scrie **javac Test.java**. În cazul în care programul conţine erori acestea vor fi semnalate şi afişate în fereastra MS-Dos, după care va apare iar prompter-ul. Dacă programul nu conţine erori şi compilarea se face cu succes, atunci va apare numai prompter-ul.

5. Pentru rularea unei aplicații Java stand-alone, se lansează interpretorul Java. Se tipăreşte la prompter următoarea comandă **java nume_clasă_care_conține_main** şi se apasă Enter. De ex., dacă clasa din program care conține metoda *main( )* se numeşte Test, se va scrie **java Test**.
6. Dacă programul Java este un applet, se editează fişierul .**html**. Se salvează în acelaşi director cu fişierul **.class** rezultat în urma compilării cu succes a fişierului sursă java. Apoi pentru rulare se poate utiliza **appletviewer nume.html.**
   Alternativ, după compilarea aplicației Java, fişierul **.class** împreună cu fişierul **.html** pot fi mutate în orice alt director (nu trebuie neapărat să fie în **c:\JBulider7\jdk1.3.1\bin**). Se încarcă fişierul **.html** într-un browser Web (ex., Internet Explorer).

IV. TEMĂ

1. Se vor parcurge toate exemplele prezentate în platforma de laborator testându-se practic.