TEHNOLOGII JAVA
PENTRU DEZVOLTAREA APLICAȚIILOR
LUCRARE DE LABORATOR 14

JList. JComboBox. JSpinner.
Componente text.

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. JList

Vezi noțiunile teoretice prezentate la curs.

Observații:

Clasa **JList** nu furnizează mecanismul de "scrolling" (bare de defilare). Dacă lista conține mai multe informații decât pot fi afişate în spațiul disponibil, trebuie adăugată lista la o instanță **JScrollPane**.

Nu există o metodă *add( )* în clasa **JList**. **JList** utilizează un model de date pentru a-şi stoca informațiile din listă. Avantajul este următorul. Dacă se doreşte afişarea într-o listă a unor date dinamice dintr-un vector, tot ceea ce trebuie făcut este identificarea vectorului ca fiind modelul de date corespunzător instanței **JList** (se scrie mai puțin cod decât dacă s-ar lucra cu o listă AWT).

Vom prezenta un exemplu, în care o listă este dinamic actualizată de către utilizator. Se furnizează posibilitatea de a adăuga text nou la listă: se creează un buton **Add** şi un câmp de editare în care utilizatorul să tipărească noua informație. Pentru a se putea şterge din listă, se creează un buton **Remove**, căruia i se ataşează cod pentru ştergerea informațiilor din listă. De asemenea, se interceptează selecțiile efectuate în listă şi se transferă informația în câmpul de editare.

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

```
class Lista extends JFrame
            implements ActionListener,ListSelectionListener
{
private JPanel topPanel;
private JList listbox;
private Vector listData;
private JButton addButton;
private JButton removeButton;
private JTextField dataField;
private JScrollPane scrollPane;
public Lista()
{
setTitle( "Advanced List Box 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 un model de date
listData = new Vector();
// Creeaza o lista
listbox = new JList( listData );
listbox.addListSelectionListener( this );
// Adauga lista la un “scrolling pane”
scrollPane = new JScrollPane();
scrollPane.getViewport().add( listbox );
topPanel.add( scrollPane, BorderLayout.CENTER );
CreateDataEntryPanel();
}
public void CreateDataEntryPanel()
{
// Creeaza un panou pentru a stoca celelalte componente
JPanel dataPanel = new JPanel();
dataPanel.setLayout( new BorderLayout() );
topPanel.add( dataPanel, BorderLayout.SOUTH );
// Creeaza butoane
addButton = new JButton( "Add" );
dataPanel.add( addButton, BorderLayout.WEST );
addButton.addActionListener( this );
removeButton = new JButton( "Delete" );
dataPanel.add( removeButton, BorderLayout.EAST );
removeButton.addActionListener( this );
// Creeaza un camp de editare
dataField = new JTextField();
dataPanel.add( dataField, BorderLayout.CENTER );
}
public void valueChanged( ListSelectionEvent event )
{
  if( event.getSource() == listbox && !event.getValueIsAdjusting())
  {
   String stringValue = (String)listbox.getSelectedValue();
```

```
   if( stringValue != null )
        dataField.setText( stringValue );
  }
}
public void actionPerformed( ActionEvent event )
{
 if( event.getSource() == addButton )
 {
  String stringValue = dataField.getText();
  dataField.setText( "" );
  if( stringValue != null )
  {
   listData.addElement( stringValue );
   listbox.setListData( listData );
   scrollPane.revalidate();
   scrollPane.repaint();
  }
 }
 if( event.getSource() == removeButton )
 {
  int selection = listbox.getSelectedIndex();
  if( selection >= 0 )
  {
   listData.removeElementAt( selection );
   listbox.setListData( listData );
   scrollPane.revalidate();
   scrollPane.repaint();
// Selecteaza urmatoarea intrare din lista
   if( selection >= listData.size() )
       selection = listData.size() - 1;
   listbox.setSelectedIndex( selection );
  }
 }
}
public static void main( String args[] )
{
  Lista mainFrame = new Lista();
  mainFrame.setVisible( true );
}
}
```

JList acceptă înlocuirea modelului său de date cu o versiune "custom" implementată de programator. Un model de date "custom" definit de către programator pentru o listă extinde clasa **AbstractListModel**, şi deci trebuie să implementeze 2 metode. Prima este *getSize()*, care returnează numărul de intrări din listă. A doua este *getElementAt()*, care returnează opțiunea din listă cu indicele specificat (începând de la 0). Exemplu:

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

```
class CLista extends JFrame
{
private JPanel topPanel;
private JList listbox;
private CustomListModel listData;
private JScrollPane scrollPane;
public CLista()
{
setTitle( "Custom Data Model List Application" );
setSize( 300, 100 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
listData = new CustomListModel();// creeaza modelul de date
listbox = new JList( listData );
scrollPane = new JScrollPane();
scrollPane.getViewport().add( listbox );
topPanel.add( scrollPane, BorderLayout.CENTER );
}
public static void main( String args[] )
{
CLista mainFrame = new CLista();
mainFrame.setVisible( true );
}
}
class CustomListModel extends AbstractListModel
{
public int getSize()
{
return 300;
}
public Object getElementAt( int index )
{
return "Item " + index;
}
}
```

Pentru a îmbunătăți aspectul vizual al aplicațiilor Java, se pot adăuga imagini grafice (de dimensiuni mici) opțiunilor unei liste, şi de asemenea, se pot modifica font-ul şi culorile utilizate pentru afişarea elementelor listei. În acest scop, se defineşte o clasă responsabilă pentru desenarea elementelor listei. Iată un exemplu:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class ImgLista extends JFrame
{
private JPanel topPanel;
private JList listbox;
public ImgLista()
```

```
{
setTitle( "Rendered ListBox Application" );
setSize( 300, 160 );
setBackground( Color.gray );
topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza datele ce vor fi adaugate in lista
String listData[] ={"Smile","Naughty","Amazed","Angry"};
listbox = new JList( listData );
listbox.setCellRenderer( new CustomCellRenderer() );
topPanel.add( listbox, BorderLayout.CENTER );
}
public static void main( String args[] )
{
ImgLista mainFrame = new ImgLista ();
mainFrame.setVisible( true );
}
}
class CustomCellRenderer extends JLabel implements ListCellRenderer
{
private ImageIcon image[];
public CustomCellRenderer()
{
setOpaque(true);
// Se pre-incarca imaginile pentru a economisi timp
image = new ImageIcon[4];
image[0] = new ImageIcon( "pic1.gif" );
image[1] = new ImageIcon( "pic2.gif" );
image[2] = new ImageIcon( "pic3.gif" );
image[3] = new ImageIcon( "pic4.gif" );
}
public Component getListCellRendererComponent(JList list, Object value,
                    int index,boolean isSelected, boolean cellHasFocus)
{

// afiseaza textul pentru aceasta intrare  a listei
setText(value.toString());
// Seteaza imaginea corecta
setIcon( image[index] );

// Deseneaza fontul si culorile corecte
if( isSelected )
{
// Seteaza culoarea si fontul pentru o optiune selectata
setBackground( Color.red );
setForeground( Color.white );
setFont( new Font( "Roman", Font.BOLD, 24 ) );
}
else
{
// Seteaza culoarea si fontul pentru o optiune neselectata
```

```
setBackground( Color.white );
setForeground( Color.black );
setFont( new Font( "Roman", Font.PLAIN, 12 ) );
}
return this;
}
}
```

Observație:

Se observă că pentru clasa "custom" care se ocupă de desenarea elementelor listei, s-a extins clasa **JLabel**, desenându-se de fapt etichete pentru intrările listei. Se pot de asemenea utiliza şi alte clase: **JButton, JTextField, JCheckBox,** etc., chiar şi **JTextArea**.

2. JComboBox

Vezi noțiunile teoretice prezentate la curs.

Să considerăm următorul exemplu:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
class Combo extends JFrame implements ItemListener
{
private JComboBox combo;
final String[] sList ={"Canada","USA","Australia","Bolivia",
                       "Denmark","Japan"};
public Combo()
{
setTitle( "ComboBox Application" );
setSize( 300, 80 );
setBackground( Color.gray );
JPanel topPanel = new JPanel();
topPanel.setLayout( null );
getContentPane().add( topPanel );
combo = new JComboBox();
combo.setBounds( 20, 15, 260, 20 );
topPanel.add( combo );
// Populeaza lista ComboBox
for( int iCtr = 0; iCtr < sList.length; iCtr++ )
combo.addItem( sList[iCtr] );
// Permite editarea
combo.setEditable( true );
combo.addItemListener( this );
}
public void itemStateChanged( ItemEvent event )
{
if( event.getSource() == combo &&
    event.getStateChange() == ItemEvent.SELECTED )
```

```
{
System.out.println( "Change:"+ combo.getSelectedItem() );
}
}
public static void main( String args[] )
{
Combo mainFrame = new Combo();
mainFrame.setVisible( true );
}
}
```

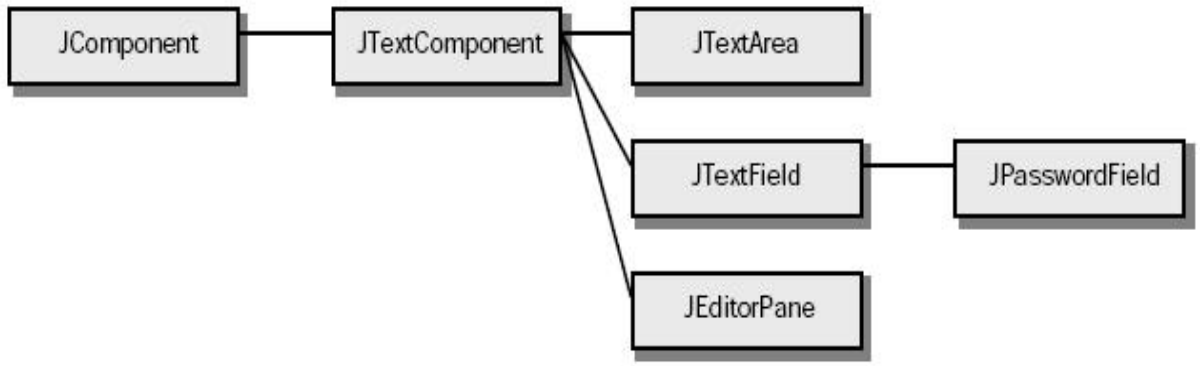
3. JSpinner

Vezi noțiunile teoretice prezentate la curs.
Să considerăm următorul exemplu:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MySpinner extends JFrame
{
   public MySpinner()
   {
     setSize(400,400);
     JPanel content=(JPanel)getContentPane();
     content.setLayout(new BorderLayout());
     ImageIcon image=new ImageIcon("Bart.gif");
     final JLabel label=new JLabel("This man is poor", image,
                                                  SwingConstants.CENTER);
     content.add(label,BorderLayout.CENTER);
     JPanel panel=new JPanel();
     final JSpinner spin=new JSpinner();
     panel.add(spin);
     JButton but=new JButton("Ok");
     panel.add(but);
     but.addActionListener(new ActionListener()
     {
          public void actionPerformed(ActionEvent e)
          {
             label.setText("This man has "+spin.getValue()+"$!");
          }
     });
     content.add(panel,BorderLayout.SOUTH);
     setVisible(true);
   }
   public static void main(String args[])
   {
     new MySpinner();
   }
}
```

4. Componente Text

Vezi noțiunile teoretice prezentate la curs.
Să observăm ierarhia de clase corespunzătoare componentelor text din Java Swing:



JTextComponent suportă manipulări clipboard de genul "cut", "copy" şi "paste". Metodele pentru acestea sunt:

```
textComponent.copy();
textComponent.cut();
textComponent.paste();
```

Aceste operații lucrează cu orice componentă text Swing, fie că este TextField, TextArea, sau o componentă text definită de către programator ce suportă un format specific.

JTextComponent poate salva sau încărca orice tip de flux
(un URL, de exemplu):

```
textComponent.read( xcReadStream, "http://www.mysite.com" );
textComponent.read( xcStreamStream );
```

În Swing, pentru manipularea modificărilor şi formatărilor textului se utilizează o clasă numită **Document**. Clasa **Document** serveşte drept model. Nu conține nici o capabilitatea pentru interfață utilizator. Clasa **Document** este un container Swing utilizat pentru a stoca text şi pentru a furniza notificări ale modificărilor textului. Implementează, de asemenea, suport pentru selectarea textului (mark-up).

4.1. JTextField şi JPasswordField

Să considerăm următorul exemplu:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.event.*;
class CompText extends JFrame
               implements DocumentListener,ActionListener
{
```

```
private JTextField field1;
private JTextField field2;
private JButton button1;
private JLabel label1;
private JLabel label2;
private JLabel label3;
private JPasswordField password;

public CompText()
{
setTitle( "TextHandling Application" );
setSize( 300, 300 );
setBackground( Color.gray );
JPanel topPanel = new JPanel();
topPanel.setLayout(null );
getContentPane().add( topPanel );

field1 = new JTextField();
field1.setBounds( 20, 40, 260, 25 );
field1.setFocusAccelerator( 'v' );
topPanel.add( field1 );
label1 = new JLabel( "Value 1:");
label1.setBounds( 20, 15, 260, 20 );
label1.setLabelFor(field1 );
label1.setDisplayedMnemonic( 'V' );
topPanel.add( label1 );

field2 = new JTextField();
field2.setBounds( 20, 90, 260, 25 );
field2.setFocusAccelerator( 'a' );
topPanel.add( field2 );
label2 = new JLabel( "Value 2:" );
label2.setDisplayedMnemonic( 'a' );
label2.setBounds( 20, 65, 260, 20 );
label2.setLabelFor(field2 );
topPanel.add( label2 );

password=new JPasswordField();
password.setBounds( 20, 140, 260, 25 );
password.setFocusAccelerator( 'p' );
topPanel.add(password);
label3 = new JLabel( "Password:" );
label3.setDisplayedMnemonic( 'P' );
label3.setBounds( 20, 115, 260, 20 );
label3.setLabelFor(password );
topPanel.add( label3 );

button1 = new JButton( "OK" );
button1.setBounds( 100, 180, 100, 25 );
button1.setEnabled(false );
topPanel.add( button1 );
// adauga un document listener la primul camp de editare
```

```
Document document = field1.getDocument();
document.addDocumentListener( this );
}

// manipuleaza acceleratorii
public void actionPerformed( ActionEvent e )
{
// obtine sursa care a generat evenimentul
JLabel label = (JLabel)e.getSource();
// ofera componentei associate focus-ul
Component fieldComponent = label.getLabelFor();
fieldComponent.requestFocus();
}

// manipuleaza inserarile in campul de editare
public void insertUpdate( DocumentEvent event )
{
String sString = field1.getText();
try {
int iValue = Integer.parseInt( sString );
button1.setEnabled( true );
}
catch( NumberFormatException e )
{
button1.setEnabled( false );
}
}

// manipuleaza stergerile din campul de editare
public void removeUpdate( DocumentEvent event )
{
// nu permite utilizatorului sa introduca un camp vid
if( field1.getText().length() == 0 )
button1.setEnabled( false );
else
{
// efectueaza aceeasi verificare a erorilor ca si insertUpdate()
insertUpdate( event );
}
}

// manipuleaza modificarile in campul de editare
public void changedUpdate( DocumentEvent event )
{}

public static void main( String args[] )
{
CompText mainFrame = new CompText();
mainFrame.setVisible( true );
}
}
```

Observații:

Pentru a asocia o instanță **JLabel** cu o componentă text, de exemplu o instanță **JTextField**, se procedează astfel:

```
JTextField field = new JTextField();
JLabel label = new JLabel( “My Label:” );
label.setLabelFor( field );
```

Apoi pentru a adăuga un accelerator etichetei şi câmpului de editare utilizăm:

```
label.setDisplayedMnemonic( ‘A’ );
```

In final, trebuie să informăm componenta text că acum are ataşată o combinație de taste care generează o acțiune atunci când utilizatorul o selectează:

```
field1.setFocusAccelerator( 'a' );
```

Acest cod generează un eveniment (“action event”) când combinația de taste ALT+A este apăsată de către utilizator, iar evenimentul generat necesită o implementare a metodei **actionPerformed()**. Ca rezultat al acțiunii, codul trebuie să schimbe focusul către componenta corectă. Pentru acest lucru se scrie codul:

```
Component fieldComponent = label.getLabelFor();
fieldComponent.requestFocus();
```

Acum, tasta A este ataşată drept accelerator, iar asocierea dintre etichetă şi câmpul de editare setează automat focusul pe câmpul de editare când acceleratorul este selectat de utilizator.

Un alt punct de interes al exemplului de mai sus, îl constituie următoarele linii de cod.

```
Document document = field1.getDocument();
document.addDocumentListener( this );
```

Se obține întâi instanța **Document** conectată la câmpul de editare **field1** şi se adaugă un “document listener” la aceasta. Din acest moment, aplicația va detecta orice modificare a textului din câmpul **field1**. Acest lucru se efectuează prin implementarea următoarelor metode abstracte din clasa **DocumentListener**:

```
public void insertUpdate( DocumentEvent event )
public void removeUpdate( DocumentEvent event )
public void changedUpdate( DocumentEvent event )
```

4.2. JTextArea

Să considerăm următorul exemplu:

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class TextA extends JFrame
{
public TextA()
{
setTitle( "Text Area Application" );
setSize( 310, 230 );
setBackground( Color.gray );
getContentPane().setLayout( new BorderLayout() );
```

```
JPanel topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel, BorderLayout.CENTER );
JTextArea area = new JTextArea();
JScrollPane scrollPane = new JScrollPane();
scrollPane.getViewport().add( area );
topPanel.add( scrollPane, BorderLayout.CENTER );
// Incarca un fisier in TextArea, trateaza eventualele exceptii
try {
FileReader fileStream = new FileReader("TextA.java" );
area.read( fileStream, "TextA.java" );
}
catch( FileNotFoundException e )
{
System.out.println( "File not found" );
}
catch( IOException e )
{
System.out.println( "IOException occurred" );
}
}
public static void main( String args[] )
{
TextA mainFrame = new TextA();
mainFrame.setVisible( true );
}
}
```

4.3. JEditorPane

Putem utiliza clasa **JEditorPane** pentru a afişa text HTML. În exemplul următor, viewer-ul HTML încarcă o pagină dintr-un fişier, dar la fel de uşor se pot extrage pagini web de pe site-urile din Internet. Dar **JEditorPane** nu se limitează numai la afişarea de conţinut HTML. Fiecărei instanţe **JEditorPane** i se asignează un **EditorKit** care controlează politica unui tip de conţinut MIME specific. Dacă se încarcă în editor un URL, instanţa **JEditorPane** va determina automat tipul conţinutului, şi va afişa datele în formatul corect.

JEditorPane suportă un obiect ascultător ("listener") care poate fi utilizat pentru a detecta modificările hyperlink.

La fel ca **JTextArea**, **JEditorPane** trebuie plasat într-un panou de defilare "scrolling pane" pentru a putea beneficia de barele de defilare în cazul în care ferasstra aplicaţiei este prea mică pentru a afişa toate informaţiile.

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
class HTMLViewer extends JFrame implements HyperlinkListener
```

```
{
private JEditorPane html;
private String sPath = System.getProperty( "user.dir" ) + "/";
public HTMLViewer ()
{
setTitle( "HTML Application" );
setSize( 400, 300 );
setBackground( Color.gray );
getContentPane().setLayout( new BorderLayout() );
JPanel topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel, BorderLayout.CENTER );
try {
// incarca URL pe care vrem sa-l afisam
URL url = new URL( "file:///" + sPath + "Main.htm" );
// Creeaza un HTML viewer pentru a afisa URL-ul
html = new JEditorPane( url );
html.setEditable( false );
JScrollPane scrollPane = new JScrollPane();
scrollPane.getViewport().add( html, BorderLayout.CENTER );
topPanel.add( scrollPane, BorderLayout.CENTER );
html.addHyperlinkListener( this );
}
catch( MalformedURLException e )
{
System.out.println( "Malformed URL: " + e );
}
catch( IOException e )
{
System.out.println( "IOException: " + e );
}
}
public void hyperlinkUpdate( HyperlinkEvent event )
{
if( event.getEventType() == HyperlinkEvent.EventType.ACTIVATED )
{
// incarca niste cursoare
Cursor cursor = html.getCursor();
Cursor waitCursor = Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR );
html.setCursor( waitCursor );
// manipuleaza modificarile hyperlink
// ...
}
}
public static void main( String args[] )
{
HTMLViewer mainFrame = new HTMLViewer ();
mainFrame.setVisible( true );
}
}
```

Swing furnizează şi un al doilea editor kit, numit **RTFEditorKit**. Clasa **RTFEditorKit** manipulează citirea fişierelor RTF, incluzând toate atributele caracterelor şi stilurile de text. Fişierele RTF pot fi produse de majoritatea procesoarelor de text, sau de aplicația WORDPAD din Microsoft Windows. Iată un exemplu:

```
import java.awt.*;
import java.io.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.text.rtf.*;
class RTFViewer extends JFrame
{
public RTFViewer ()
{
setTitle( "RTF Text Application" );
setSize( 400, 240 );
setBackground( Color.gray );
getContentPane().setLayout( new BorderLayout() );
JPanel topPanel = new JPanel();
topPanel.setLayout( new BorderLayout() );
getContentPane().add( topPanel, BorderLayout.CENTER );
// Creeaza o fereastra editor RTF
RTFEditorKit rtf = new RTFEditorKit();
JEditorPane editor = new JEditorPane();
editor.setEditorKit( rtf );
editor.setBackground( Color.white );
// textul poate fi mare, deci adaugam un “scroll pane”
JScrollPane scroller = new JScrollPane();
scroller.getViewport().add( editor );
topPanel.add( scroller, BorderLayout.CENTER );
// incarca un fisier RTF in editor
try {
FileInputStream fi = new FileInputStream( "test.rtf" );
rtf.read( fi, editor.getDocument(), 0 );
}
catch( FileNotFoundException e )
{
System.out.println( "File not found" );
}
catch( IOException e )
{
System.out.println( "I/O error" );
}
catch( BadLocationException e )
{}
}
public static void main( String args[] )
{
RTFViewer mainFrame = new RTFViewer ();
mainFrame.setVisible( true );
}
}
```

4.4. JTextPane

Swing oferă o clasă numită **JTextPane**, care extinde **JEditorPane** şi care conţine suport pentru atributele caracterelor şi textului. La fel ca **JTextArea**, **JTextPane** permite utilizatorului să introducă şi să editeze text de la tastatură. Dar, **JTextPane** permite în plus afişarea unui număr infinit de stiluri configurabile de text. De asemenea, **JTextPane** suportă şi conţinut grafic.

Să observăm exemplul următor.

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;
class TextPane extends JFrame implements ActionListener
{
private Hashtable attributes;
private JComboBox styleCombo;
private DefaultStyledDocument doc;
private JTextPane textComponent;
public TextPane ()
{
setTitle( "Document Handling Application" );
setSize( 300, 190 );
setBackground( Color.gray );
JPanel topPanel = new JPanel( new BorderLayout() );
getContentPane().add( topPanel );
// Creeaza stiluri pentru document
StyleContext sc = new StyleContext();
doc = new DefaultStyledDocument( sc );
createStyles( sc );
// Creeaza un “text pane” pentru a afisa text
textComponent = new JTextPane( doc );
textComponent.setBackground( Color.white );
topPanel.add( textComponent, BorderLayout.CENTER );
// Creeaza un “toolbar” pentru a manipula modificarile de stil
topPanel.add( createToolBar(), BorderLayout.NORTH );
}
// Creeaza un “toolbar panel” foarte simplu
public JPanel createToolBar()
{
JPanel panel = new JPanel( new FlowLayout() );
styleCombo = new JComboBox();
styleCombo.addActionListener( this );
panel.add( styleCombo );
// Adauga fiecare stil la “combo box”
for( Enumeration e = attributes.keys(); e.hasMoreElements(); )
styleCombo.addItem( e.nextElement().toString() );
return panel;
}
```

```
//Manipuleaza modificarile efectuate in “combo box”
public void actionPerformed( ActionEvent e )
{
if( e.getSource() == styleCombo )
{
try {
// Determina noul stil
Style s = (Style)attributes.get(
styleCombo.getSelectedItem() );
// Seteaza stilul
doc.insertString( textComponent.getCaret().getDot()," ", s );
// se intoarce la fereastra editor
textComponent.grabFocus();
}
catch( BadLocationException exception )
{}
}
}
// Creeaza stiluri diferite de font
public void createStyles( StyleContext sc )
{
Style myStyle;
// Aloca o tabela de dispersie pentru stiluri
attributes = new Hashtable();
// Nici un stil
myStyle = sc.addStyle( null, null );
attributes.put( "none", myStyle );
// Normal
myStyle = sc.addStyle( null, null );
StyleConstants.setLeftIndent( myStyle, 10 );
StyleConstants.setRightIndent( myStyle, 10 );
StyleConstants.setFontFamily( myStyle, "Helvetica" );
StyleConstants.setFontSize( myStyle, 14 );
StyleConstants.setSpaceAbove( myStyle, 4 );
StyleConstants.setSpaceBelow( myStyle, 4 );
attributes.put( "normal", myStyle );
// Big
myStyle = sc.addStyle( null, null );
StyleConstants.setFontFamily( myStyle, "Dialog" );
StyleConstants.setFontSize( myStyle, 28 );
attributes.put( "big", myStyle );
// Bold
myStyle = sc.addStyle( null, null );
StyleConstants.setBold( myStyle, true );
attributes.put( "bold", myStyle );
}
public static void main( String args[] )
{
TextPane mainFrame = new TextPane ();
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 aplicaţii Java stand-alone, se lansează interpretorul Java. Se tipăreşte la prompter următoarea comandă **java nume_clasă_care_conţine_main** şi se apasă Enter. De ex., dacă clasa din program care conţine 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 aplicaţiei 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.