Advanced Programming Methods

Lecture 9

Content

- GUI in Java and C#

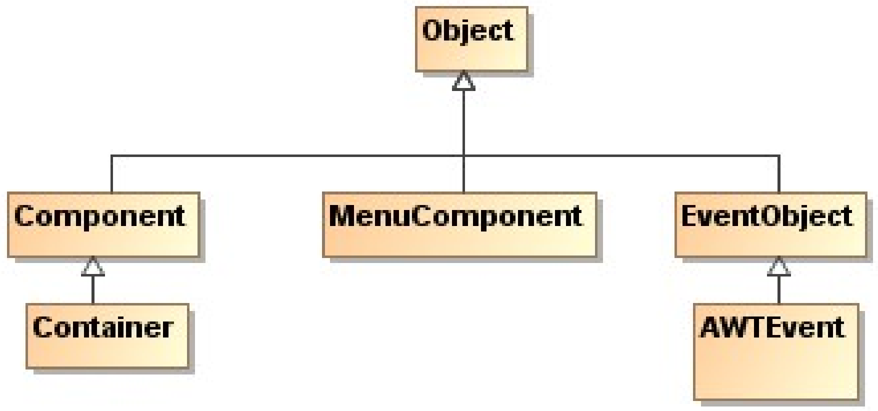
Java GUI

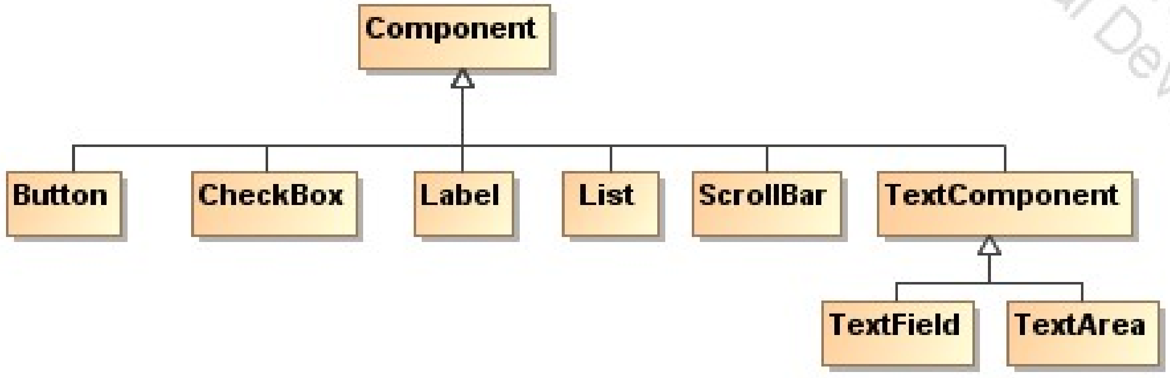
Graphical User Interface

- Java contains a set of components for writing GUI. The same set is used for each OS on which the application is running.
- The same components can be used for desktop and web(applet) applications.
- Java programmers use the *Java Foundation Classes (JFC)* to create GUI applications.
- The two sets of JFC classes are:
  - AWT classes
  - Swing classes.

Abstract Window Toolkit (AWT), JDK 1.0

- The components are written in native code.
- Packages: java.awt, java.awt.event, ...



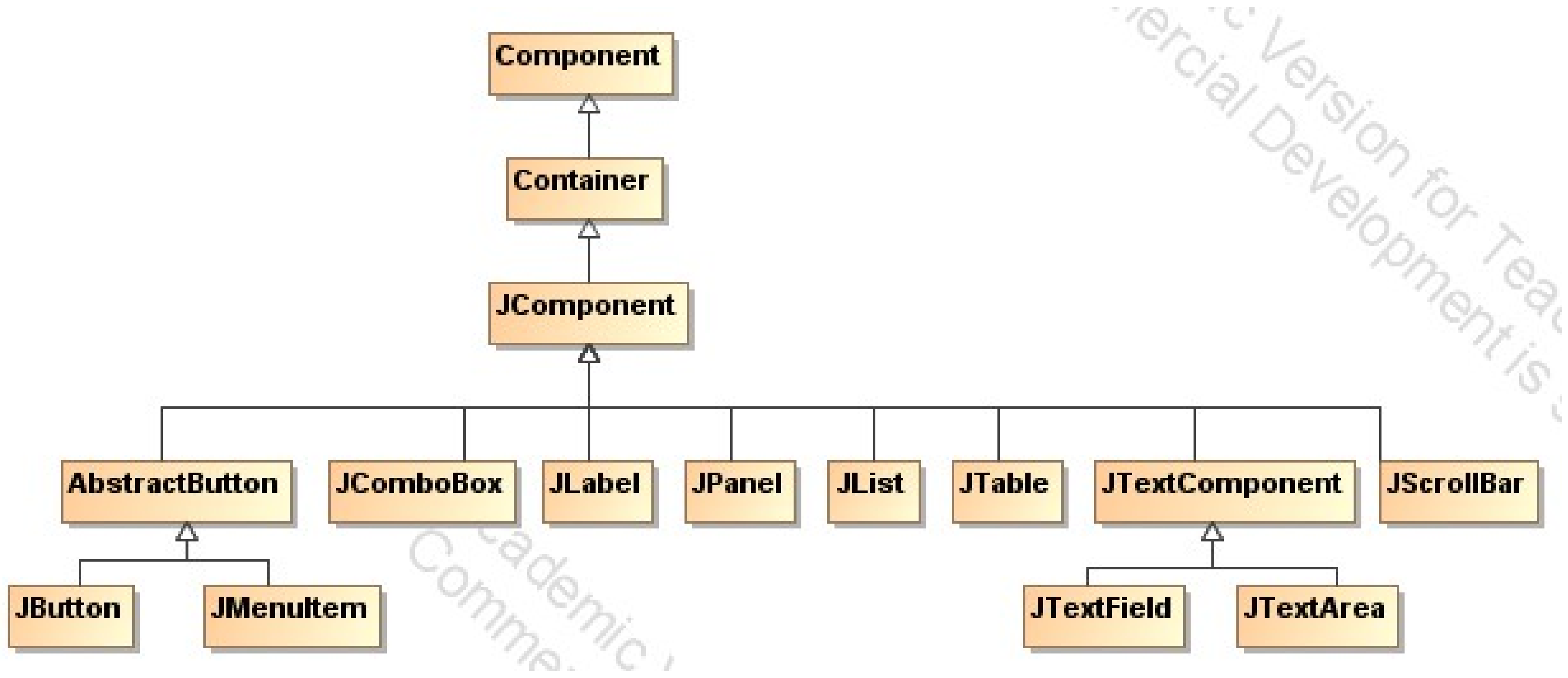


AWT

- The AWT does not actually draw user interface components on the screen. It communicates with a layer of software, *peer classes*. Each version of Java for a particular operating system has its own set of peer classes.
- The behavior of components across various operating systems can differ.
- Programmers cannot easily extend the AWT components.
- AWT components are commonly called *heavyweight components*.

Swing -JFC , JDK1.2

- The components are written in Java and have the same aspect on all platforms.
- The components aspect can be adapted to a given style.
- Packages: javax.swing, javax.swing.event, javax.swing.table,....



Swing

- provide an improved alternative for creating GUI applications and applets.
- Very few Swing classes rely on peer classes, so they are referred to called *lightweight components.*
- Swing components have a consistent look and predictable behavior on any operating system.
- Swing components can be easily extended.

Graphical Components – javax.swing

Windows – JFrame, JDialog, JWindow

Containers: JPanel

Buttons – JButton, JRadioButton, JCheckBox

Labels - JLabel

Text fields – JTextField, JTextArea, JPasswordField

Lists - JList

Tables - JTable

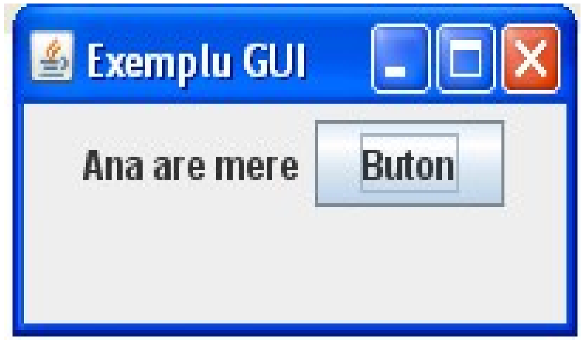
Menus – JMenuBar, JMenu, JMenuItem, JSeparator

JOptionPane, JFileChooser, JScrollPane

...

Simple Example

```
import javax.swing.*;
public class ExempluSimplu extends JFrame {
    public ExempluSimplu(String title) {
        super(title);
        JPanel pan=new JPanel();
        pan.add(new JLabel("Ana are mere"));
        pan.add(new JButton("Buton"));
        getContentPane().add(pan);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable(){
            public void run() {
                ExempluSimplu es=new ExempluSimplu("Exemplu GUI");
                es.setSize(200,100);
                es.setLocation(150,150);
                es.setVisible(true);
            }
        });
    }}
```



Swing Event Dispatch Thread

- it’s really not a good idea for the **main( )** thread to write directly to the GUI components

- Swing has its own thread dedicated to receiving UI events and updating the screen

- GUI tasks must be submitted to Swing event dispatch thread ---- by handing a task to **SwingUtilities.invokeLater( )**, which puts it on the *event queue* to be (eventually) executed by the event dispatch thread

- If you start manipulating the screen with other threads, you can have the collisions and deadlock

Layout Managers

- An important part of designing a GUI application is determining the layout of the components.
- The term *layout* refers to the positioning and sizing of components. You do not normally specify the exact location of a component within a window.
- A *layout manager* is an object that:
  - controls the positions and sizes of components, and
  - makes adjustments when necessary.
- The layout manager object and the container work together.

Layout Manager

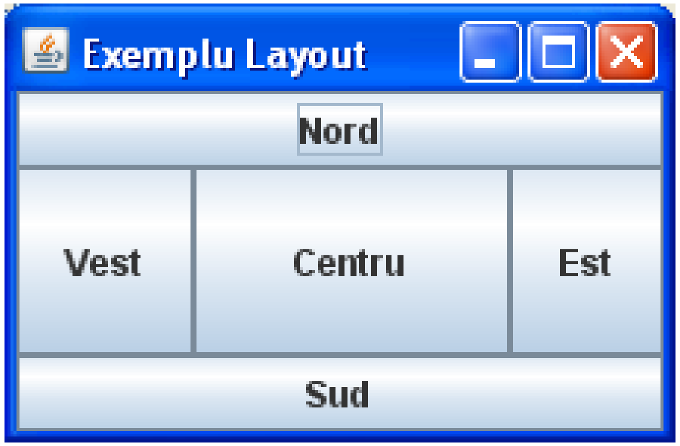
Interface `LayoutManager` (package `java.awt`):

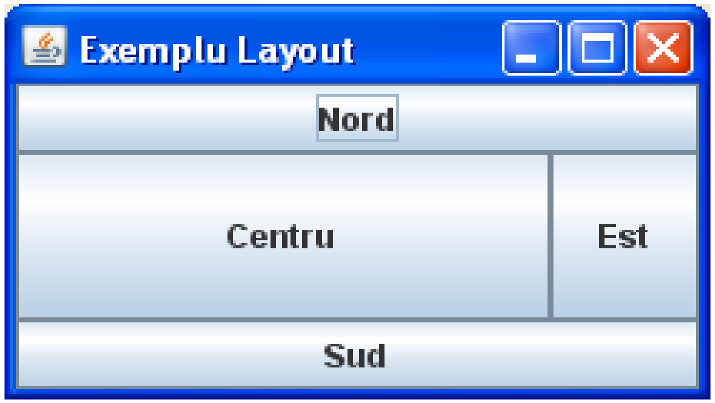
- `BorderLayout`
- `FlowLayout`
- `GridLayout`
- `GridBagLayout`
- `BoxLayout`
- `GroupLayout`
- `CardLayout`
- etc.

`BorderLayout` (package `java.awt`):

- Divides the container in 5 regions: north, south, east, west and center.
- If we do not add components into a region, that region space is occupied by the components of a neighbor region.
- is the default manager for `JFrame` objects.

```
JPanel pan=new JPanel();
pan.setLayout(new BorderLayout());
pan.add(new JButton("Nord"),BorderLayout.NORTH);
pan.add(new JButton("Sud"), BorderLayout.SOUTH);
pan.add(new JButton("Est"), BorderLayout.EAST);
pan.add(new JButton("Vest"), BorderLayout.WEST);
pan.add(new JButton("Centru"), BorderLayout.CENTER);   //default
```

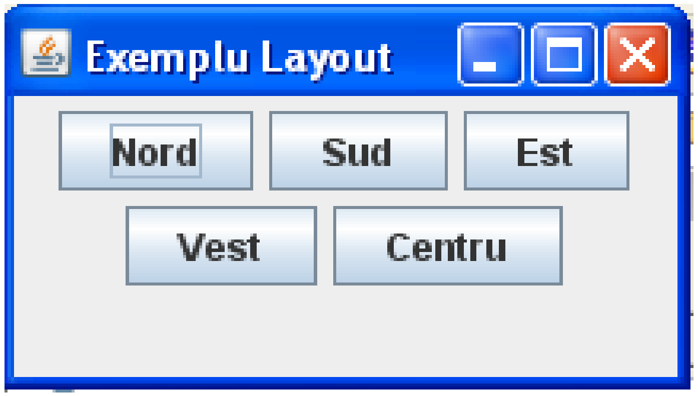


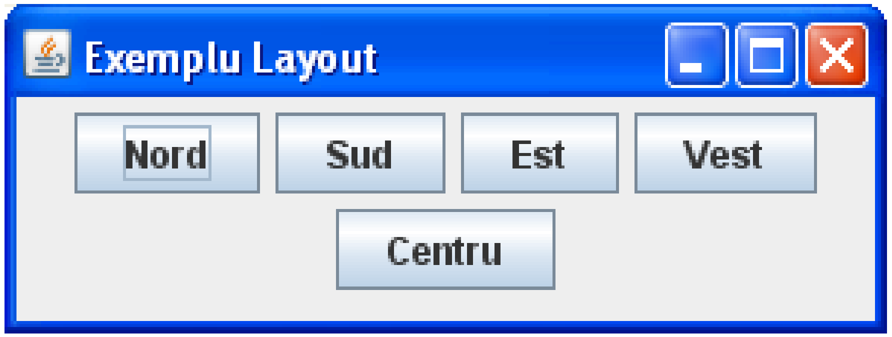


`FlowLayout` (package `java.awt`):

- Components appear horizontally, from left to right, in the order that they were added. When there is no more room in a row, the next components “flow” to the next row.
- is the default layout manager for `JPanel` objects.

```
JPanel pan=new JPanel();
pan.add(new JButton("Nord"));
pan.add(new JButton("Sud"));
pan.add(new JButton("Est"));
pan.add(new JButton("Vest"));
pan.add(new JButton("Centru"));
```

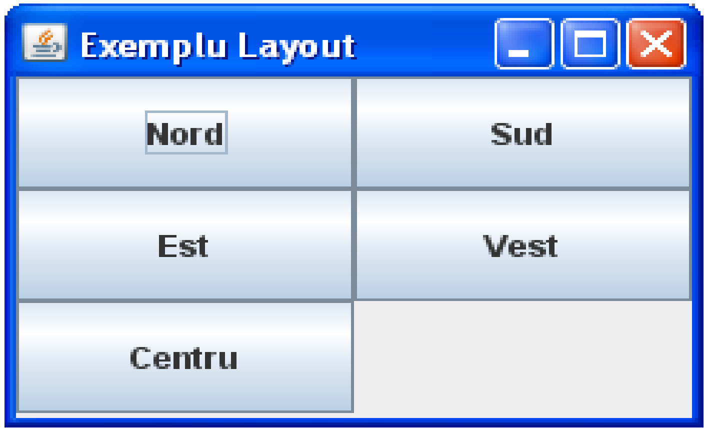


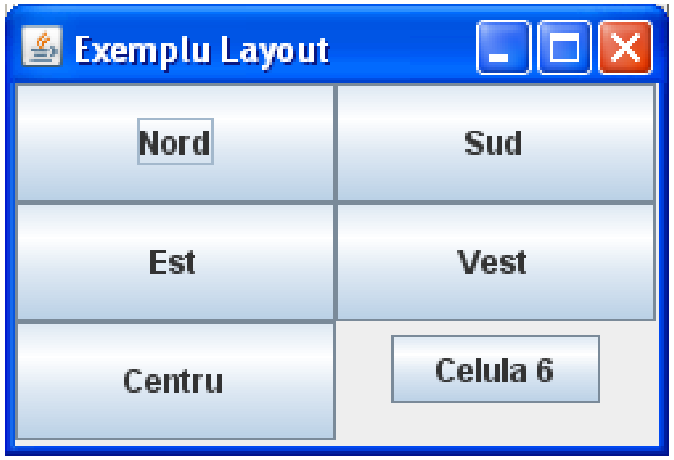


GridLayout (package java.awt):

- Divides the container space into n rows and m columns . The components are added from left to right and from top to down.
- A component from a cell fills the entire cell space.
- A cell without nothing remains with empty space.

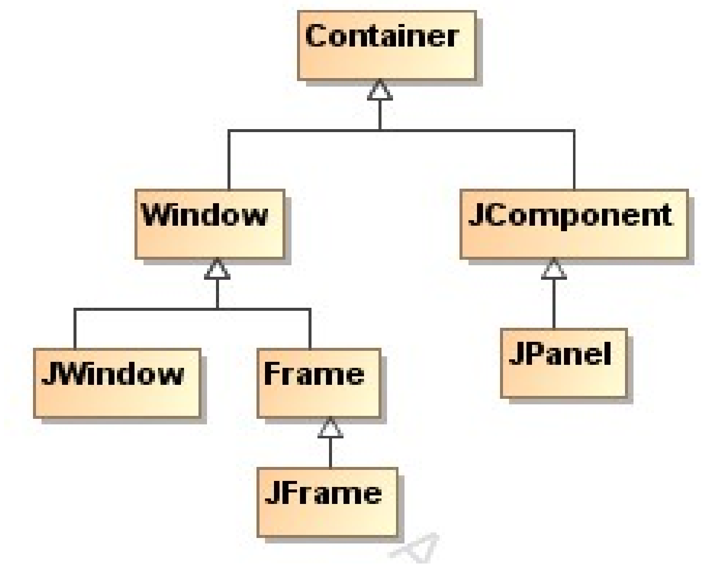
```
JPanel pan=new JPanel();
pan.setLayout(new GridLayout(3,2));
pan.add(new JButton("Nord"));
pan.add(new JButton("Sud"));
pan.add(new JButton("Est"));
pan.add(new JButton("Vest"));
pan.add(new JButton("Centru"));

//JPanel pan2=new JPanel();
//pan2.add(new JButton("Celula 6"));
//pan.add(pan2);
```

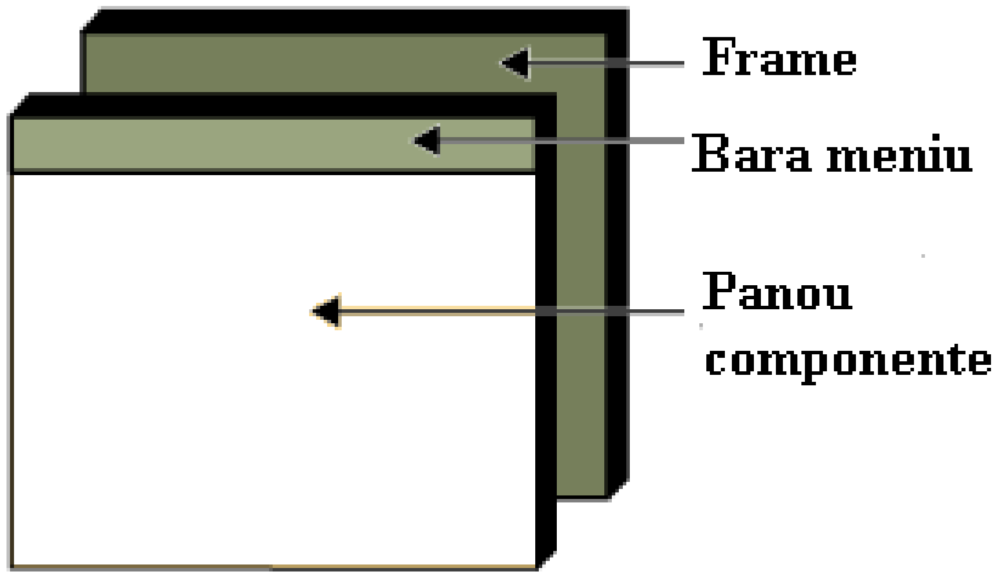




Containers

- **`JFrame`** Any desktop application with GUI uses this class to define the main window. Its default layout is BorderLayout.
- **`JDialog`** It is the main class for the dialog windows .
- **`JPanel`** It is used to add and arrange the components. This container must be added to a top-level container (**`JFrame, JDialog, or JApplet`**).





Swing Top-level Containers

There are three generally useful top-level container classes:

- **JFrame,**
- JDialog, and
- JApplet.

Some useful rules

- to appear onscreen, **every GUI component must be part of a containment hierarchy**. A containment hierarchy is a tree of components that has a top-level container as its root.

- **each GUI component can be contained only once**. If a component is already in a container and you try to add it to another container, the component will be removed from the first container and then added to the second.

Some useful rules

- each top-level container has a **content pane** that contains (directly or indirectly) **the visible components** in that top-level container's GUI

- a **menu bar** can be optionally added to a top-level container. The menu bar is by convention positioned within the top-level container, but outside the content pane.

Top-Level Containers and Containment Hierarchies

- a standalone application with a Swing-based GUI has at least one containment hierarchy with a JFrame as its root.

-For example, if an application has one main window and two dialogs, then the application has three containment hierarchies:

  - One containment hierarchy has a JFrame as its root, and
  - each of the other two has a JDialog object as its root.

Adding Components to the Content Pane

- the default content pane is a simple intermediate container that inherits from JComponent, and that uses a BorderLayout as its layout manager.

- the content pane of a top-level container can be found by calling the getContentPane method

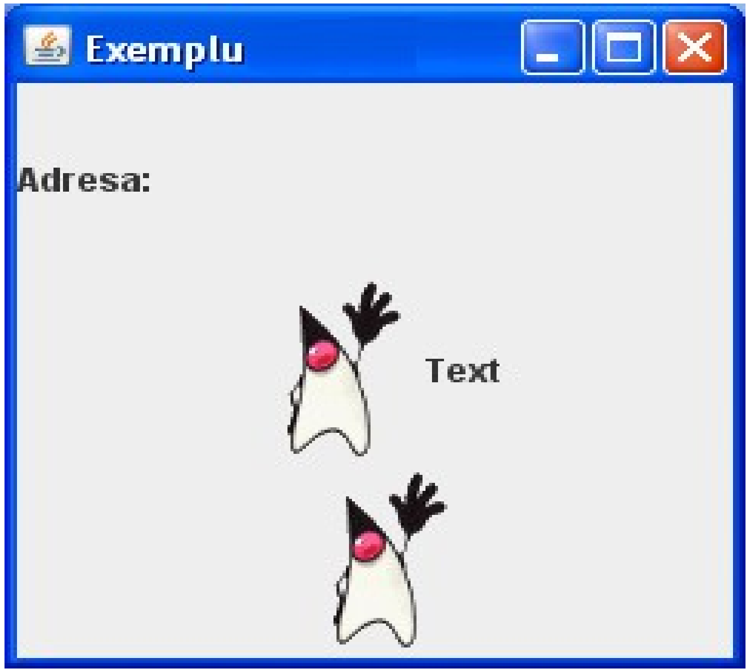
```
frame.getContentPane().add(yellowLabel,
 BorderLayout.CENTER);
```

Setting a Content Pane

```
//Create a panel and add components to it.
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent,
   BorderLayout.CENTER);
contentPane.add(anotherComponent,
   BorderLayout.PAGE_END);


topLevelContainer.setContentPane(contentPane);
```

Labels - JLabel

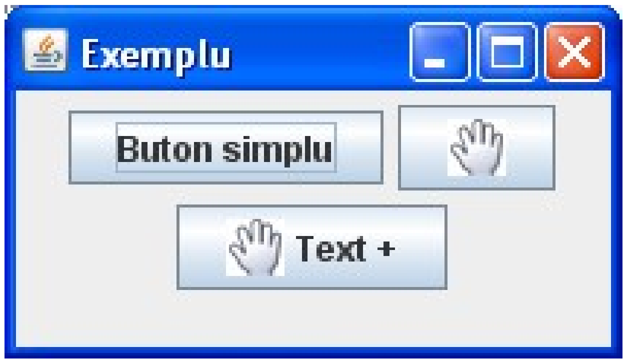
- `JLabel` – Displays text or image which cannot be selected by the user.
  - `JLabel(Icon)`
  - `JLabel(String)`
  - `JLabel(String, Icon, int)`
  - `setText(String)`
  - `getText(): String`
  - `setIcon(Icon)`
  - `getIcon(): Icon`



```
JPanel pan=new JPanel();
pan.setLayout(new GridLayout(3,1));
pan.add(new JLabel("Adresa: "));
ImageIcon imagine=new ImageIcon("img/duke.gif");
pan.add(new JLabel("Text",imagine,JLabel.CENTER));
pan.add(new JLabel(imagine));
```

Buttons - JButton

- JButton :
  - JButton(String, Icon)
  - JButton(String)
  - JButton(Icon)
  - JButton()
  - setText(String)
  - getText(): String
  - setIcon(Icon)
  - getIcon(): Icon

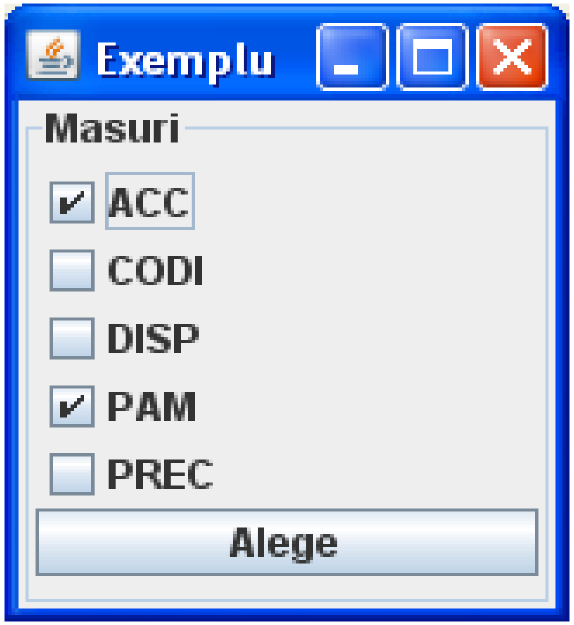


```
JButton butText,butImagine, butImagText;
JPanel panou=new JPanel();
panou.add(butText=new JButton("Buton simplu"));
ImageIcon imagine=new ImageIcon("img/handtool.gif");
panou.add(butImagine=new JButton(imagine));
panou.add(butImagText=new JButton("Text +", imagine));
```

Buttons - JCheckBox

- **JCheckBox**
  - JCheckBox(String)
  - JCheckBox(String, boolean)
  - JCheckBox(Icon)
  - isSelected():boolean

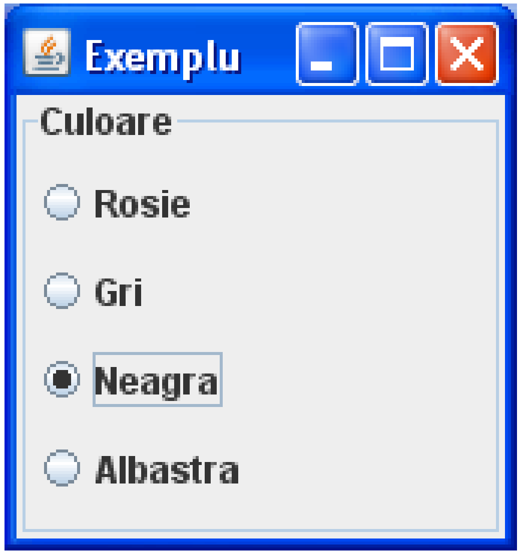
```
 JButton alege;
 JCheckBox[] mas=new JCheckBox[5];
 JPanel panou=new JPanel();
 panou.setLayout(new GridLayout(6,1));
 panou.add(mas[0]=new JCheckBox("ACC",true));
 panou.add(mas[1]=new JCheckBox("CODI"));
 panou.add(mas[2]=new JCheckBox("DISP"));
 panou.add(mas[3]=new JCheckBox("PAM",true));
 panou.add(mas[4]=new JCheckBox("PREC"));
 panou.add(alege=new Jbutton("Alege"));
panou.setBorder(BorderFactory.createTitledBorder("Masuri"));
```



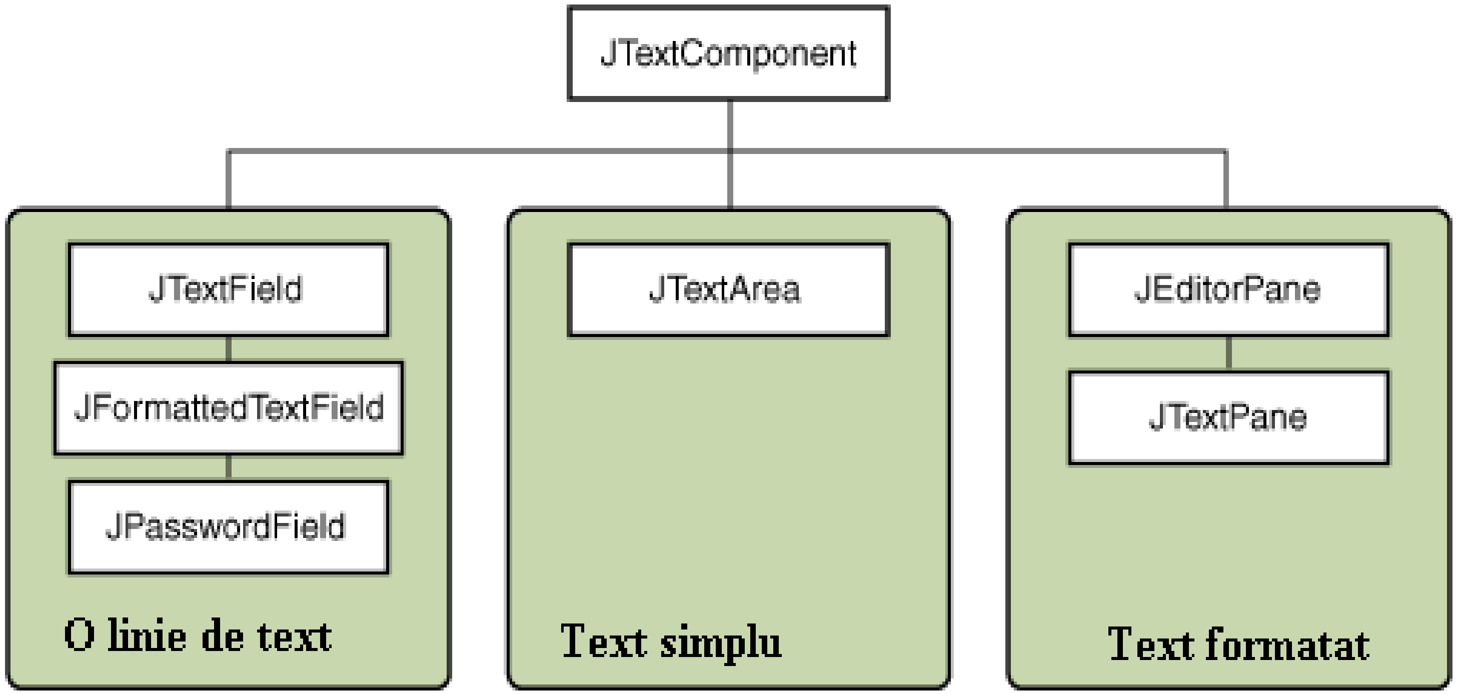
Buttons - JRadioButton

- `JRadioButton`
  - `JRadioButton(String)`
  - `JRadioButton(String, boolean)`
  - `JRadioButton(Icon)`
  - `JRadioButton(Icon, boolean)`
  - `isSelected():boolean`
  - `setActionCommand(String)`
  - `getActionCommand():String`

```
JPanel panou=new JPanel();
JRadioButton[] cul=new JRadioButton[4];
panou.setLayout(new GridLayout(4,1));
panou.add(cul[0]=new JRadioButton("Rosie"));
panou.add(cul[1]=new JRadioButton("Gri"));
panou.add(cul[2]=new JRadioButton("Neagra", true));
panou.add(cul[3]=new JRadioButton("Albastra"));
panou.setBorder(BorderFactory.createTitledBorder("Culoare"));
ButtonGroup group = new ButtonGroup();
for(JRadioButton but:cul)
   group.add(but);
```



Components with text

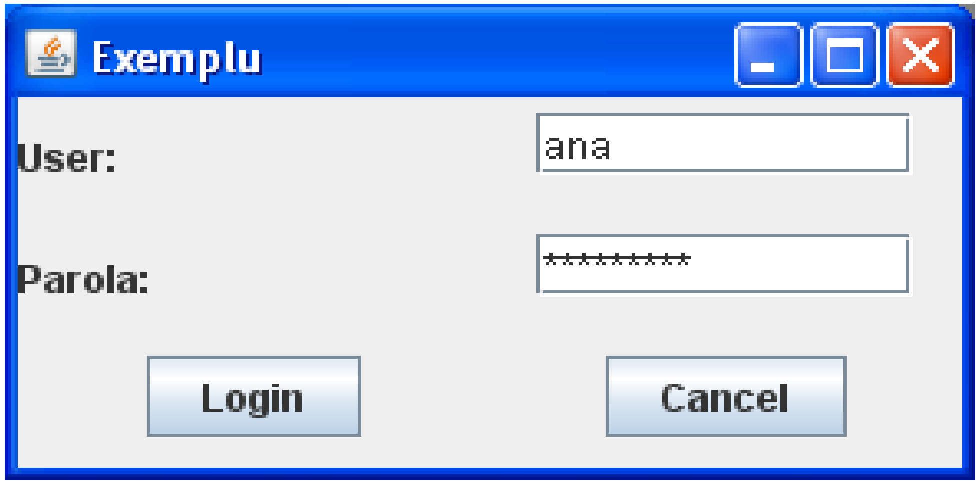


Text – JTextField, JPasswordField, JTextArea

- `JTextField`
  - `JTextField(String, int)`
  - `JTextField(int)`
  - `setText(String)`
  - `getText(): String`
- `JPasswordField`
  - `JPasswordField(String, int)`
  - `JPasswordField(int)`
  - `getPassword(): char[]`
  - `setEchoChar(char)`
  - `getEchoChar(): char`
- `JTextArea`
  - `JTextArea(String, int, int)`
  - `JTextArea(int, int)`
  - `setLineWrap(boolean):int`
  - `setText(String)`
  - `getText(): String`
  - `append(String)`
  - `insert(String, int)`

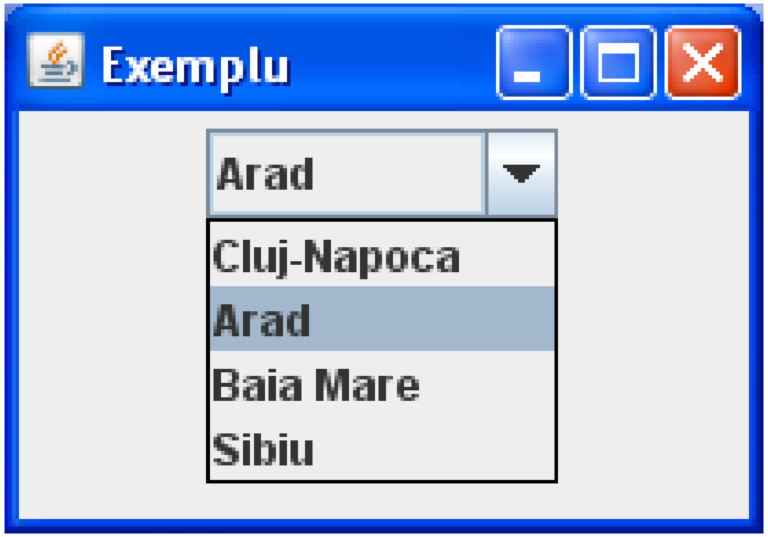
Text – JTextField, JPasswordField

```
JPasswordField passwd;
JTextField user;
JPanel panou=new JPanel();
panou.setLayout(new GridLayout(3,2));
panou.add(new JLabel("User:"));
panou.add(UtileGUI.putInPanel(user=new JTextField(10)));
panou.add(new JLabel("Parola:"));
panou.add(UtileGUI.putInPanel(passwd=new JPasswordField(10)));
passwd.setEchoChar('*');
panou.add(UtileGUI.putInPanel(new JButton("Login")));
panou.add(UtileGUI.putInPanel(new JButton("Cancel")));
```



JComboBox

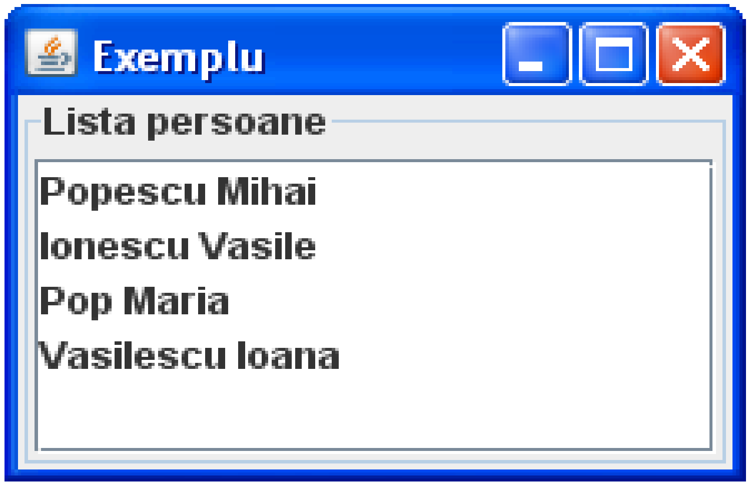
- **JComboBox:**
  - JComboBox()
  - JComboBox(ComboBoxModel)
  - JComboBox(Object[])
  - addItem(Object)
  - insertItemAt(Object, int)
  - getItemAt(int): Object
  - getSelectedItem(): Object
  - setSelectedIndex(int anIndex)



```
String[] orase={"Cluj-Napoca", "Arad", "Baia Mare", "Sibiu"};
JPanel panou=new JPanel();
JComboBox combo=new JComboBox(orase);
combo.setSelectedIndex(1);
panou.add(combo);
```

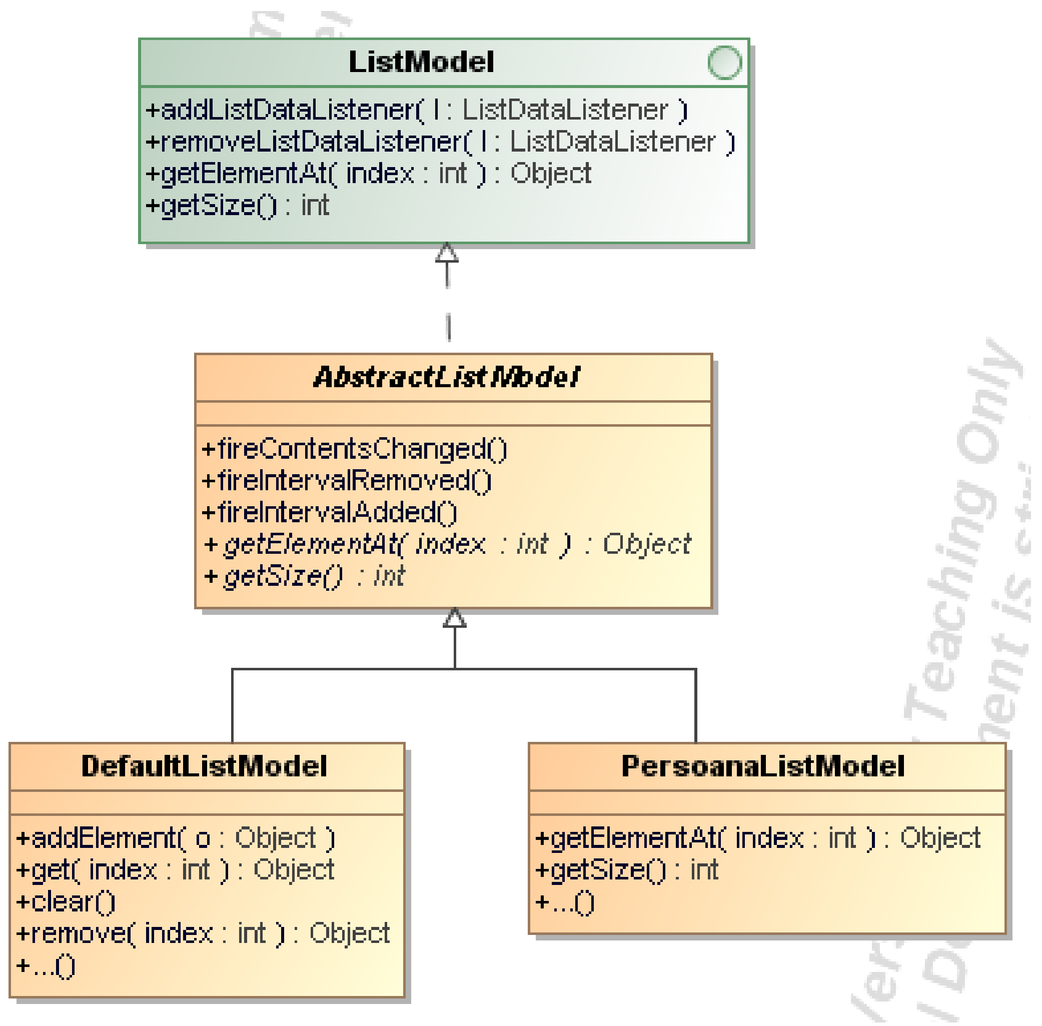
List JList

- JList:
  - JList(ListModel)
  - JList(Object[])
  - setSelectionMode(int)
  - getSelectionMode(): int
  - getSelectedIndex() : int
  - clearSelection()
  - isSelectionEmpty(): boolean



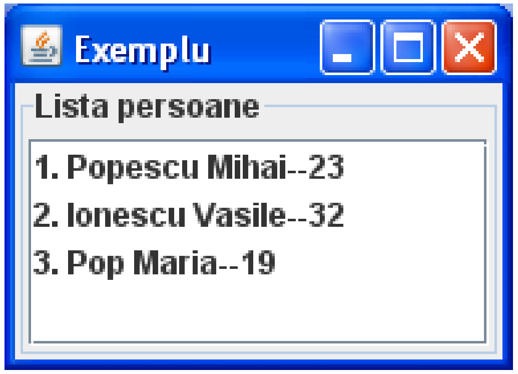
```
JPanel panou=new JPanel(new GridLayout(1,1));
DefaultListModel dlm=new DefaultListModel();  //store the data
dlm.addElement("Popescu Mihai");
dlm.addElement("Ionescu Vasile");
dlm.addElement("Pop Maria");
dlm.addElement("Vasilescu Ioana");
JList lista=new JList(dlm);
lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
JScrollPane pane=new JScrollPane(lista);  //scroll bar
panou.add(pane);
panou.setBorder(new TitledBorder("Lista persoane"));
```

List -ListModel, AbstractListModel


ListModel
+addListDataListener(l : ListDataListener)
+removeListDataListener(l : ListDataListener)
+getElementAt(index : int) : Object
+getSize() : int
AbstractListModel
+fireContentsChanged()
+fireIntervalRemoved()
+fireIntervalAdded()
+getElementAt(index : int) : Object
+getSize() : int
DefaultListModel
+addElement(o : Object)
+get(index : int) : Object
+clear()
+remove(index : int) : Object
+...()
PersoanaListModel
+getElementAt(index : int) : Object
+getSize() : int
+...()

Example ListModel

```
public class PersoaneListModel extends AbstractListModel {
    private List<Persoana> persoane;
    public PersoaneListModel() {
        persoane=new ArrayList<Persoana>();
    }
    public int getSize() {
        return persoane.size();
    }
    public Object getElementAt(int index) {
        Persoana pers=persoane.get(index);
        return ""+(index+1)+". "+pers.getNume()+" "+pers.getPrenume()
  +"--"+pers.getVarsta();
    }
    public void adaugaPersoana(Persoana p){
        persoane.add(p);
        fireContentsChanged(this,persoane.size()-1,persoane.size());
    }
}
```



```
1. Popescu Mihai--23
2. Ionescu Vasile--32
3. Pop Maria--19
```

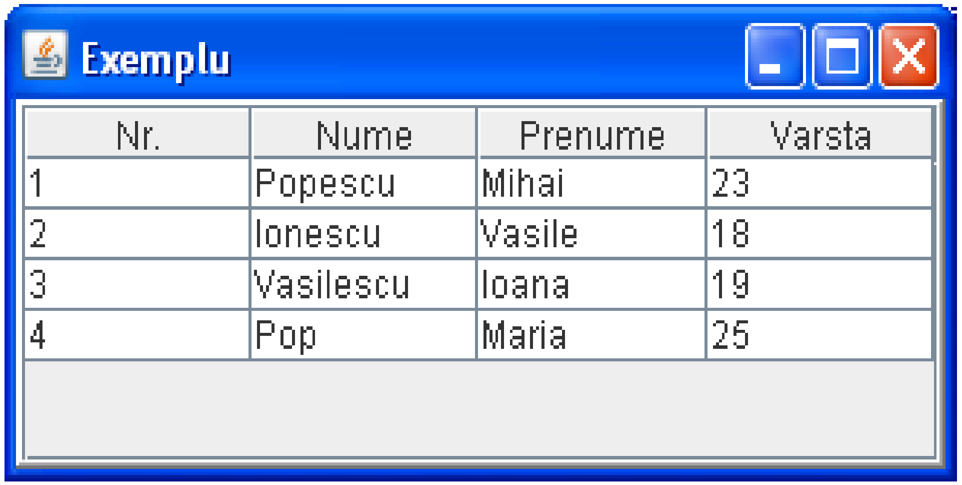
Models

- models automatically propagate changes to all interested listeners, making it easy for the GUI to stay in sync with the data.

- For example, to add items to a list you can invoke methods on the list model. When the model's data changes, the model fires events to the JList and any other registered listeners, and the GUI is updated accordingly.

Table - JTable

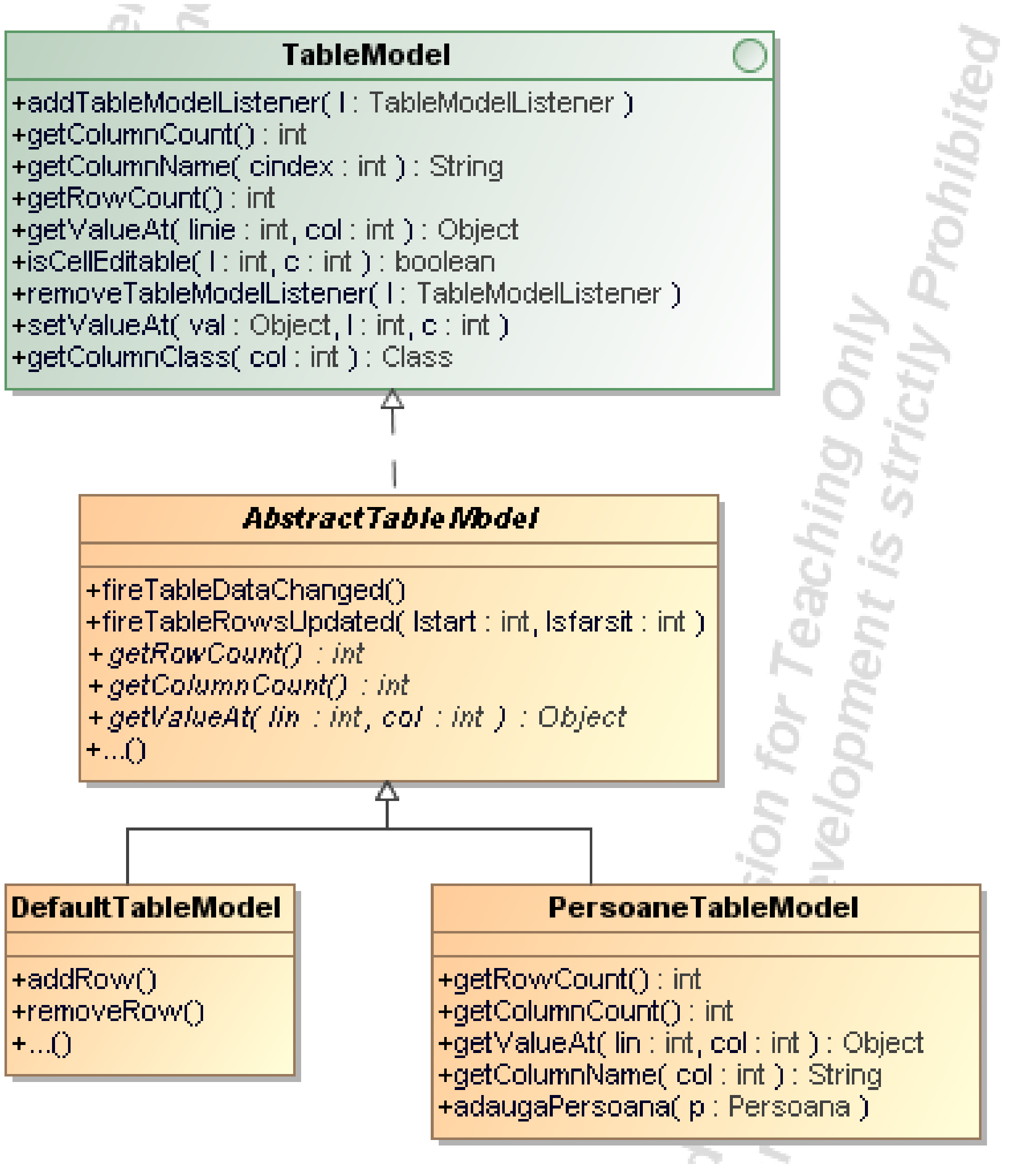
- **JTable:**
  - `JTable(TableModel)`
  - `JTable(Object[][],Object[])`
  - `JTable(int,int)`
  - `getValueAt(int,int): Object`
  - `setSelectionMode(int)`
  - `getSelectedRow():int`
  - ...



Nr.	Nume	Prenume	Varsta
1	Popescu	Mihai	23
2	Ionescu	Vasile	18
3	Vasilescu	Ioana	19
4	Pop	Maria	25

```
JPanel panou=new JPanel(new GridLayout(1,1));
PersoaneTabelModel pers=new PersoaneTabelModel();
pers.adaugaPersoana(new Persoana("Popescu","Mihai",23));
pers.adaugaPersoana(new Persoana("Ionescu", "Vasile", 18));
pers.adaugaPersoana(new Persoana("Vasilescu", "Ioana", 19));
pers.adaugaPersoana(new Persoana("Pop", "Maria", 25));
JTable tabel=new JTable(pers);
JScrollPane pane=new JScrollPane(tabel);
panou.add(pane);
panou.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
```

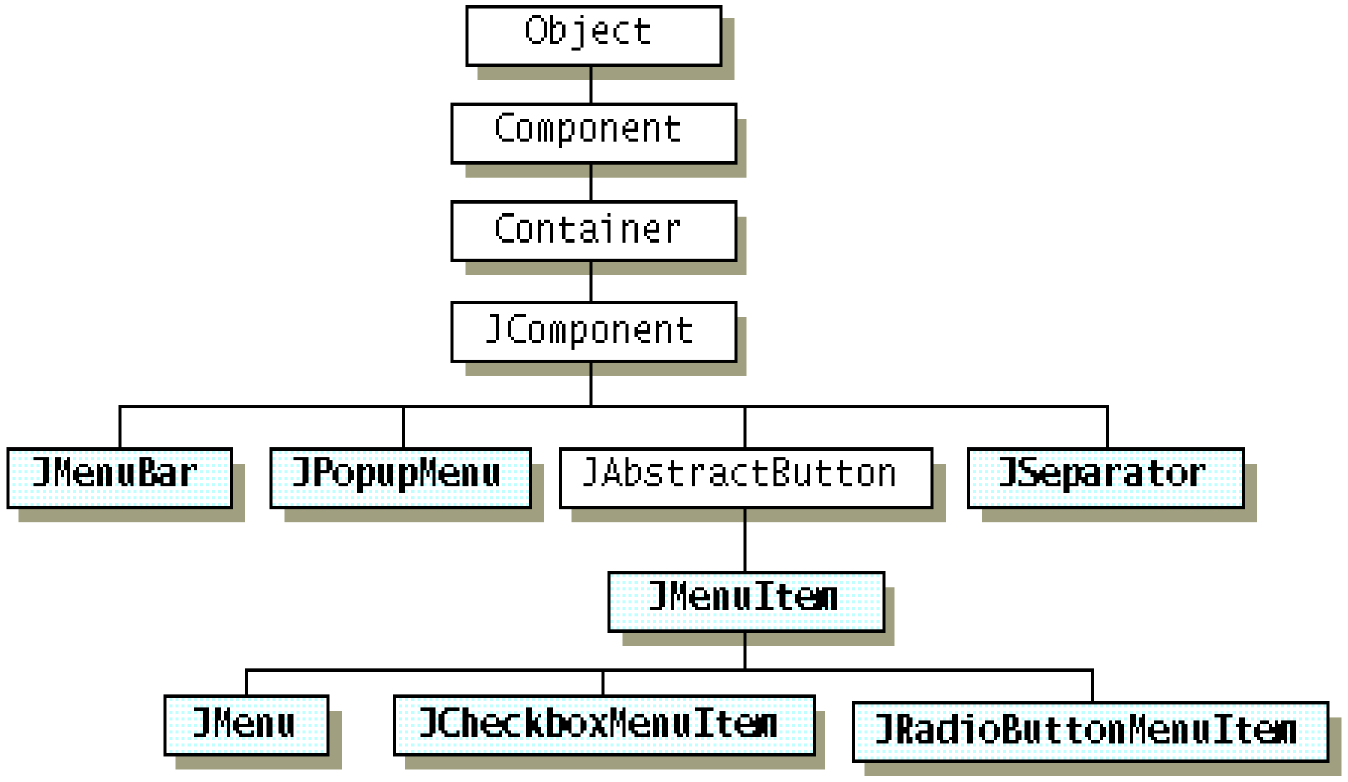
Table – TableModel, AbstractTableModel


TableModel
+addTableModelListener(l : TableModelListener)
+getColumnCount() : int
+getColumnName(cindex : int) : String
+getRowCount() : int
+getValueAt(linie : int, col : int) : Object
+isCellEditable(l : int, c : int) : boolean
+removeTableModelListener(l : TableModelListener)
+setValueAt(val : Object, l : int, c : int)
+getColumnClass(col : int) : Class
AbstractTableModel
+fireTableDataChanged()
+fireTableRowsUpdated(lstart : int, lsfarsit : int)
+getRowCount() : int
+getColumnCount() : int
+getValueAt(lin : int, col : int) : Object
+...()
DefaultTableModel
+addRow()
+removeRow()
+...()
PersoaneTableModel
+getRowCount() : int
+getColumnCount() : int
+getValueAt(lin : int, col : int) : Object
+getColumnName(col : int) : String
+adaugaPersoana(p : Persoana)

Example TableModel

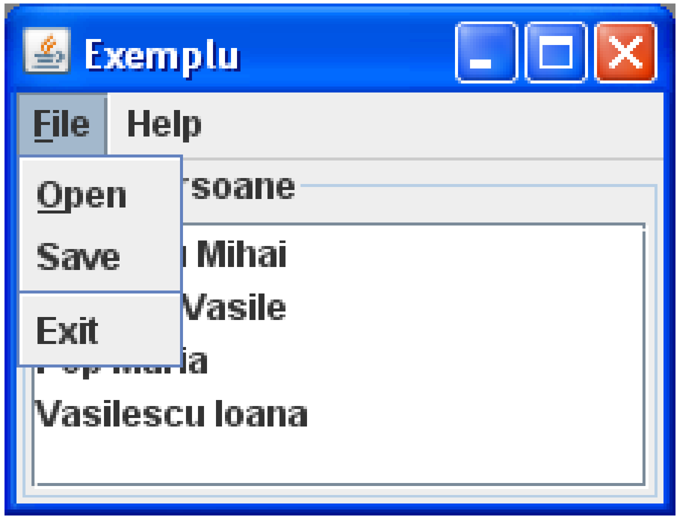
```
public class PersoaneTabelModel extends AbstractTableModel {
    private String[] numeColoane={"Nr.", "Nume", "Prenume", "Varsta"};
    private List<Persoana> persoane;
    public PersoaneTabelModel(){ persoane=new ArrayList<Persoana>();}
    public int getRowCount() { return persoane.size(); }
    public int getColumnCount() {return numeColoane.length;}
    public boolean isCellEditable(int row, int column) {  return false;  }
    public Object getValueAt(int rowIndex, int columnIndex) {
        if (columnIndex==0)  return rowIndex+1;
        Persoana pers=persoane.get(rowIndex);
        switch (columnIndex){
            case 1: return pers.getNume();
            case 2: return pers.getPrenume();
            case 3: return pers.getVarsta();
        }
        return null; }
    public void adaugaPersoana(Persoana pers){
        persoane.add(pers);  fireTableDataChanged();    }
    public String getColumnName(int column) { return numeColoane[column]; }
}
```

Menu


Object
Component
Container
JComponent
JMenuBar
JPopupMenu
JAbstractButton
JSeparator
JMenuItem
JMenu
JCheckboxMenuItem
JRadioButtonMenuItem

Menu – JMenuBar, JMenu, JMenuItem

- `JMenuBar:`
  - `JMenuBar()`
  - `add(JMenu):JMenu`
- `JFrame, JDialog`
  - `setJMenuBar(JMenuBar)`
- `JMenu`
  - `JMenu(String)`
  - `add(JMenuItem):JMenuItem`
  - `addSeparator()`
  - `setMnemonic(int)`
- `JMenuItem`
  - `JMenuItem(String)`
  - `JMenuItem(String, int)`
  - `setMnemonic(int)`
  - `setEnabled(boolean)`
  - `setActionCommand(String)`



Adding a Menu Bar

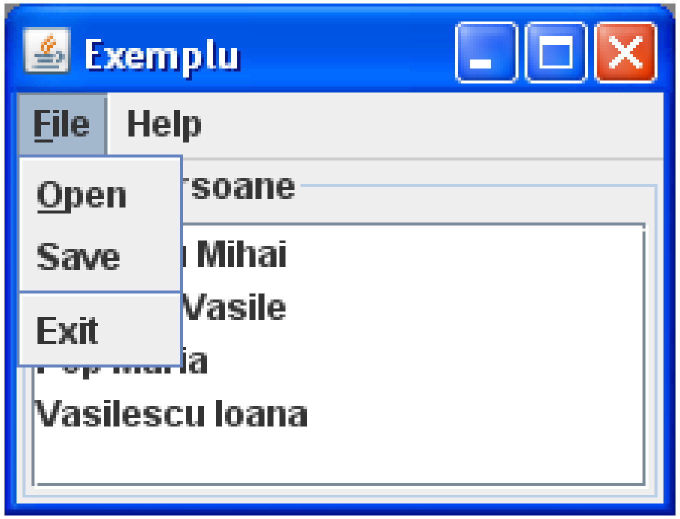
- all top-level containers can hold a menu bar

- to add a menu bar to a top-level container, create a JMenuBar object, populate it with menus, and then call setJMenuBar

```
frame.setJMenuBar(greenMenuBar);
```

Menu – JMenuBar, JMenu, JMenuItem

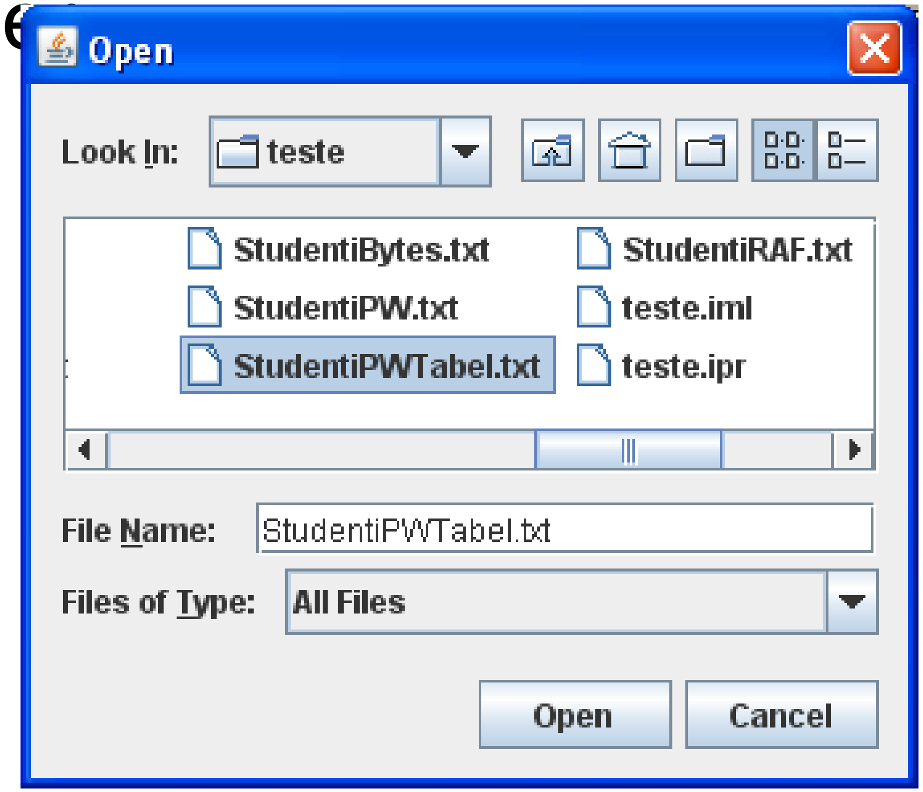
```
public class Exemplu extends JFrame {
  public Exemplu()  {
    super("Exemplu ");
    setJMenuBar(creeazaMeniu());
    add(creeazaLista());
    setDefaultCloseOperation(EXIT_ON_CLOSE); }
private JMenuBar creeazaMeniu(){
  JMenuBar mb=new JMenuBar();
  JMenu file=new JMenu("File");
  file.setMnemonic(KeyEvent.VK_F);
  JMenuItem open=new JMenuItem("Open ", KeyEvent.VK_O);
  JMenuItem save=new JMenuItem("Save ");
  JMenuItem exit=new JMenuItem("Exit");
  file.add(open);  file.add(save); file.addSeparator();
  file.add(exit);
  mb.add(file);
  JMenuItem help=new JMenuItem("Help");
  mb.add(help);
  return mb;
}}
```



JFileChoose

- JFileChooser:
  - JFileChooser()
  - JFileChooser(File)
  - JFileChooser(String)
  - showOpenDialog(Component):int
  - showSaveDialog(Component):int
  - getSelectedFile():File
  - setFileFilter(FileFilter)

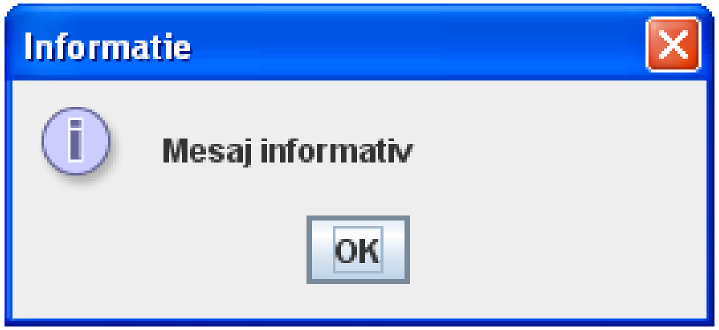
```
JFileChooser jf=new JFileChooser(".");
int rezultat=jf.showOpenDialog(Exemplu.this);
if (rezultat==jf.APPROVE_OPTION){
     File fisier=jf.getSelectedFile();
     String numefisier=fisier.getAbsolutePath();
     System.out.println(" S-a selectat fisierul "+numefisier);
}
```



JOptionPane

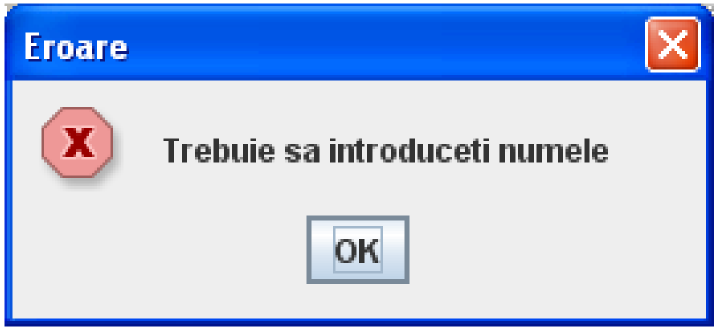
- JOptionPane:
  - showMessageDialog(Component, Object, String, int)
  - showConfirmDialog(Component, Object, String, int):int
  - showInputDialog(Component, Object, String, int) :String
  - showOptionDialog(...):int

```
JOptionPane.showMessageDialog(Exemplu.this,
    "Mesaj informativ", "Informatie",
     JOptionPane.INFORMATION_MESSAGE);
```



Mesaj informativ

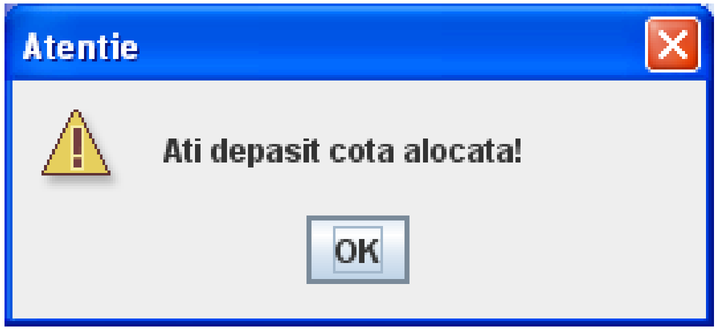
```
JOptionPane.showMessageDialog(Exemplu.this,
   "Trebuie sa introduceti numele", "Eroare",
      JOptionPane.ERROR_MESSAGE);
```



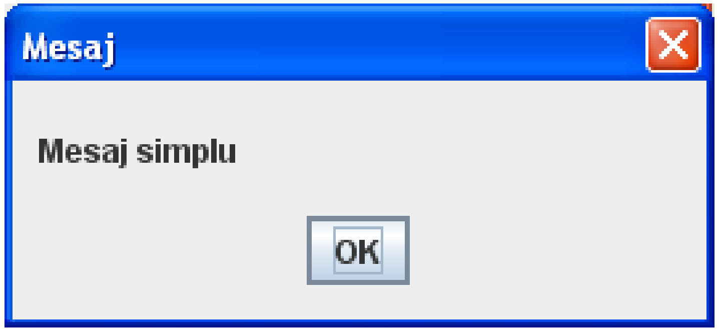
Trebuie sa introduceti numele

JOptionPane

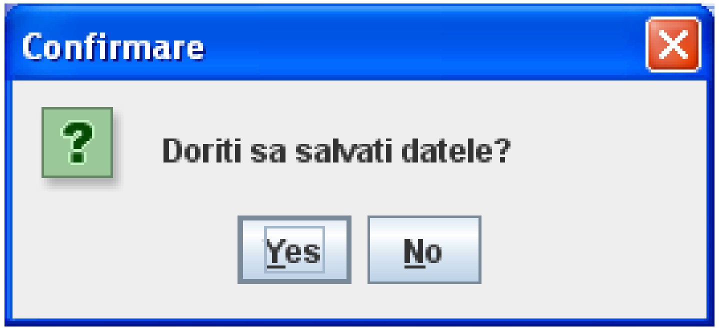
```
JOptionPane.showMessageDialog(Exemplu.this,
    "Ati depasit cota alocata!", "Atentie",
     JOptionPane.WARNING_MESSAGE);
```



```
JOptionPane.showMessageDialog(Exemplu.this,
      "Mesaj simplu", "Mesaj",
       JOptionPane.PLAIN_MESSAGE);
```

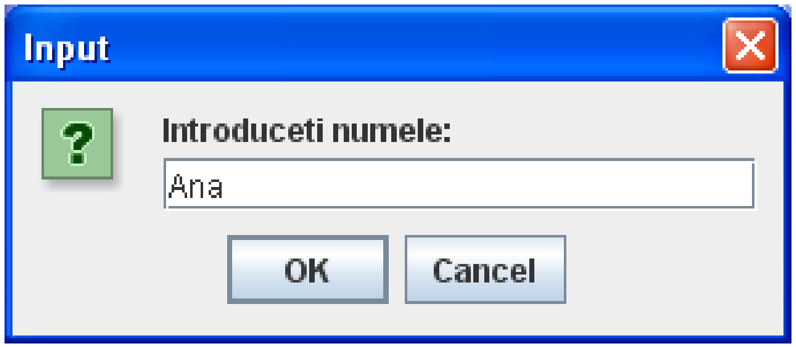


```
int rez=JOptionPane.showConfirmDialog(Exemplu.this,
     "Doriti sa salvati datele?","Confirmare",
         JOptionPane.YES_NO_OPTION,
       JOptionPane.QUESTION_MESSAGE);
if (rez==JOptionPane.YES_OPTION)
     System.out.println("S-a apasat yes");
```



JOptionPane

```
String nume=JOptionPane.showInputDialog(Exemplu.this,"Introduceti numele:");
 if ((nume!=null)&&(nume.length()>0) )
           System.out.println("Date valide "+nume);
```

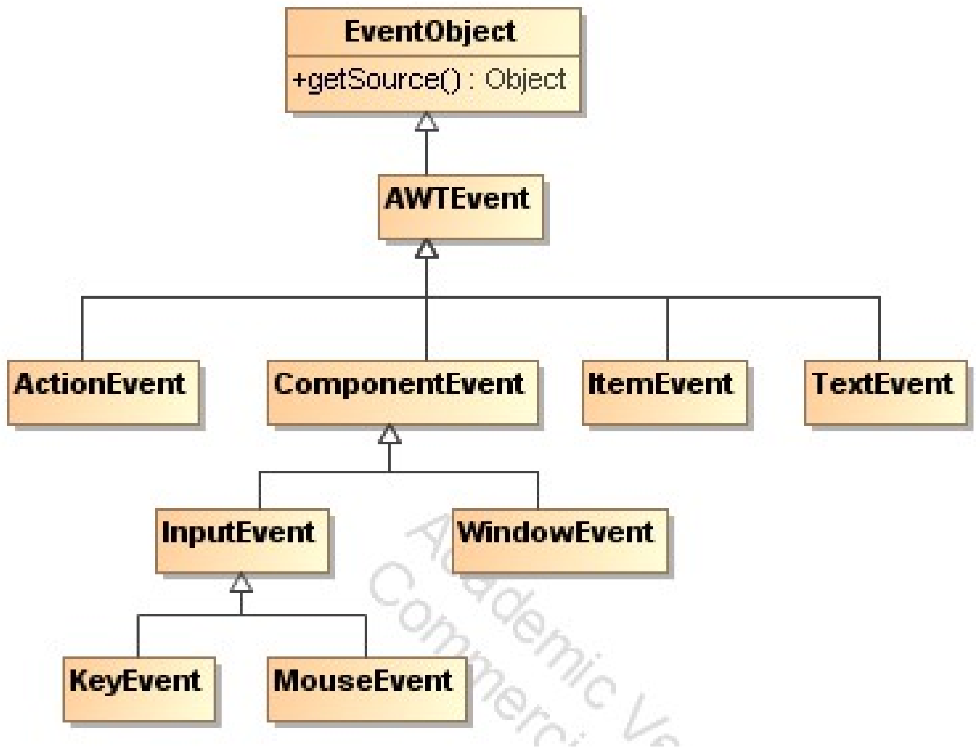


Introduceti numele:

Ana

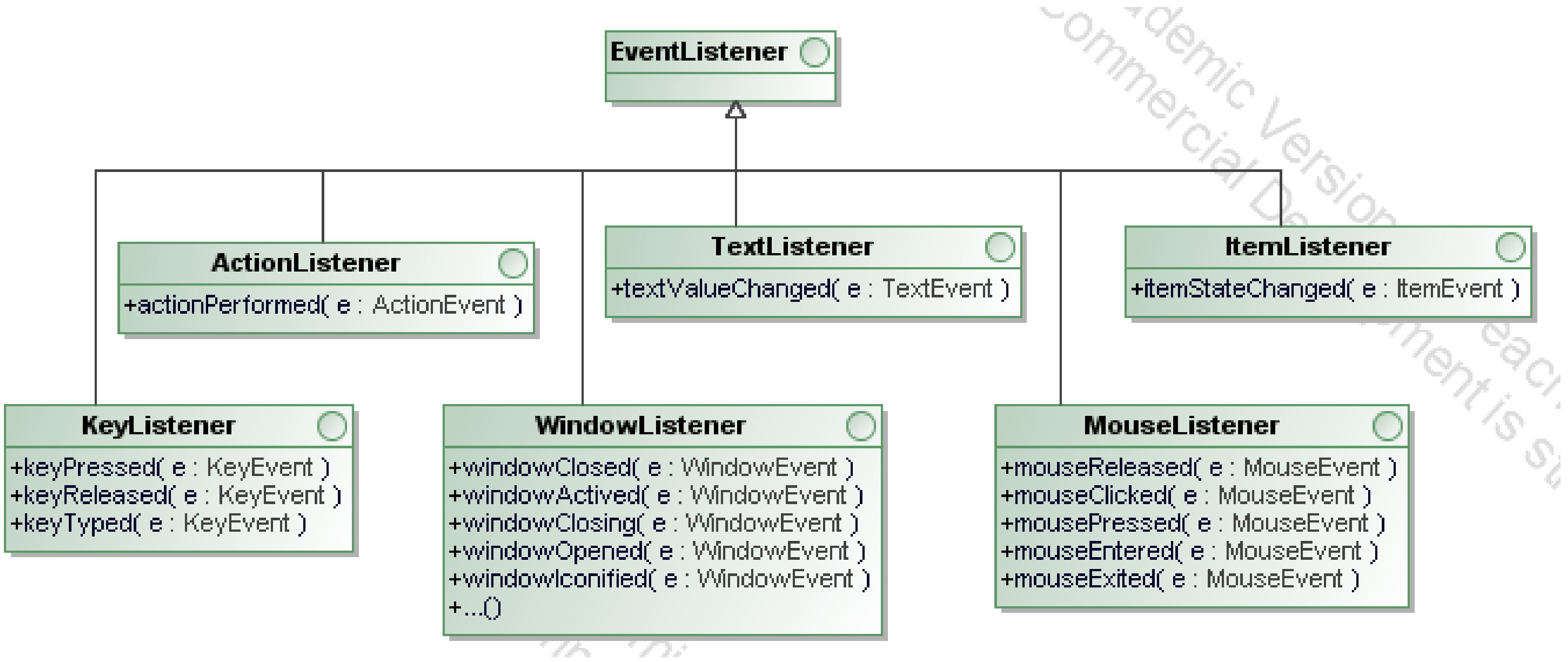
Events treatment

- Events are used to interact with the user
- An event is generated when the user press a key, input a char or press a button.
- Main class is `EventObject` (package `java.util`)



Events treatment

- Each graphical component has listeners which are announced when the events occur.
- A graphical component can answer to an event if it has a listener for that event.


EventListener
ActionListener
+actionPerformed(e : ActionEvent)
TextListener
+textValueChanged(e : TextEvent)
ItemListener
+itemStateChanged(e : ItemEvent)
KeyListener
+keyPressed(e : KeyEvent)
+keyReleased(e : KeyEvent)
+keyTyped(e : KeyEvent)
WindowListener
+windowClosed(e : WindowEvent)
+windowActived(e : WindowEvent)
+windowClosing(e : WindowEvent)
+windowOpened(e : WindowEvent)
+windowIconified(e : WindowEvent)
+...()
MouseListener
+mouseReleased(e : MouseEvent)
+mouseClicked(e : MouseEvent)
+mousePressed(e : MouseEvent)
+mouseEntered(e : MouseEvent)
+mouseExited(e : MouseEvent)

Listeners

Component	**Listener**
JButton	ActionListener
JTextField	ActionListener KeyListener
JList	ListSelectionListener
JTable	ListSelectionListener
JComboBox	ItemListener
JFrame, JDialog	WindowListener
JMenuItem	ActionListener

Action Listener

- the implementation of an action listener has to define what should be done when an user performs certain operation

- an action event occurs, whenever an action is performed by the user

- the result is that an actionPerformed message is sent to all action listeners that are registered on the relevant component.

To write an Action Listener

1. Declare an event handler class and specify that the class either implements an ActionListener interface or extends a class that implements an ActionListener interface:

```
public class MyClass implements ActionListener {
```

2. Register an instance of the event handler class as a listener on one or more components:

```
someComponent.addActionListener(instanceOfMyClass);
```

To write an Action Listener

3. Include code that implements the methods of the listener interface:

```
public void actionPerformed(ActionEvent e) {
    ...//code that reacts to the action...
}
```

JButton- ActionListener

```
private JButton logBut, cancelBut;
private JPanel createLogin(){
        JPanel res=new JPanel(new GridLayout(3,2));
        res.add(new JLabel("User id:")); res.add(userId=new JTextField(15));
        res.add(new JLabel("Password:")); res.add(passwd=new JPasswordField(15));
        res.add(logBut=new JButton("Login")); res.add(cancelBut=new
  JButton("Clear"));
        ActionListener al=new ButListener();
        logBut.addActionListener(al); cancelBut.addActionListener(al);
        return res;
}
private class ButListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (e.getSource()==logBut){
                String user=userId.getText();
                String pass=new String(passwd.getPassword());
                return;
            }
            if (e.getSource()==cancelBut){
                userId.setText("");
                passwd.setText("");
            }
        }
    }
```

JMenuItem - ActionListener

```
 JMenuItem save=new JMenuItem("Save ");
save.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e) {
      System.out.println("S-a apasat Save");
      JFileChooser jf=new JFileChooser(".");
      int rezultat=jf.showSaveDialog(Exemplu.this);
      if (rezultat==jf.APPROVE_OPTION){
            File fisier=jf.getSelectedFile();
            String numefisier=fisier.getAbsolutePath();
            System.out.println("[Save ...] S-a selectat fisierul "+numefisier);
   }}
});
```

JMenuItem - ActionListener

```
 private JMenuBar creazaMeniu(){
        JMenuBar meniu=new JMenuBar();
        JMenu prods=new JMenu("Produse");
        JMenuItem adauga=new JMenuItem("Adauga produs");
        JMenuItem exit=new JMenuItem("Exit");
        prods.add(adauga);  prods.addSeparator();  prods.add(exit);
        meniu.add(prods);
        adauga.setActionCommand("adauga");   exit.setActionCommand("exit");
        ActionListener al=new MeniuListener();
        adauga.addActionListener(al); exit.addActionListener(al);
        return meniu;
}
private class MeniuListener implements ActionListener{
        public void actionPerformed(ActionEvent e) {
            String cmd=e.getActionCommand();
            if ("adauga".equals(cmd)){
                System.out.println("Adauga");
            }
            if ("exit".equals(cmd)){
                 //...
                  System.exit(0);
            }   }
    }
```

JList - ListSelectionListener

```
  private JList lista;
public JPanel creeazaLista(){
   JPanel panou=new JPanel(new GridLayout(1,1));
   plm=new PersoaneListModel();
   plm.adaugaPersoana(new Persoana("Popescu", "Mihai", 23));
   lista=new JList(plm);
   lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   lista.getSelectionModel().addListSelectionListener(new ListaPersoaneListener());
  //...
}
private class ListaPersoaneListener implements ListSelectionListener {
   public void valueChanged(ListSelectionEvent e) {
     System.out.println("valueChanged...");
     if (!e.getValueIsAdjusting()){
       int index=lista.getSelectedIndex();
       if (index<0)
           System.out.println("Nu s-a selectat nimic");
       else
           System.out.println("S-a selectat linia "+index );
       }
   }
 }
```

JTable - ListSelectionListener

```
  private JTable tabel_produse;
private JPanel createTabel(){
   JPanel rezultat=new JPanel();
    rezultat.setLayout(new GridLayout(1,1));
    produseTM=new ProduseTableModel();
    tabel_produse=new JTable(produseTM);
  tabel_produse.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
 tabel_produse.getSelectionModel().addListSelectionListener(new ProduseTabelListener());
        JScrollPane pane=new JScrollPane(tabel_produse);
    //...
    return rezultat;
}
 private class ProduseTabelListener implements ListSelectionListener {
        public void valueChanged(ListSelectionEvent e) {
            if (!e.getValueIsAdjusting()){          //a line from tabel is selected
                int index=tabel_produse.getSelectedRow();
                if (index>=0){
                   //...
                }
            } }
    }
```

JFrame - WindowListener

```
  public class Exemplu extends JFrame {
    public Exemplu()  {
        super("Exemplu ");
        setJMenuBar(createMeniu());
        add(createLista());
        addWindowListener(new WindowAdapter(){
            @Override
            public void windowClosing(WindowEvent e) {
                System.out.println("Se inchide fereastra");
                close();
            }
        });
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
   //...
}
```

JComboBox - ItemListener

```
private JPanel creeazaComboBox(){
     String[] orase={"Cluj-Napoca", "Arad", "Baia Mare", "Sibiu"};
     JPanel panou=new JPanel();
      JComboBox combo=new JComboBox(orase);
     combo.setSelectedIndex(1);
     combo.addItemListener(new ItemListener(){
         public void itemStateChanged(ItemEvent e) {
             System.out.println("itemStateChanged...");
             if (e.getStateChange()==ItemEvent.SELECTED){
                 System.out.println("S-a selectat "+e.getItem());
             }
         }
     });
     panou.add(combo);
     return panou;
 }
```

Further Readings

- Java Swing tutorial from Oracle

  http://docs.oracle.com/javase/tutorial/uiswing/TOC.html