Aplicații Integrate pentru Întreprinderi

Laborator 4

25.10.2011

Dezvoltarea unei interfețe grafice cu utilizatorul folosind Swing

Scopul laboratorului îl reprezintă familiarizarea cu clasele puse la dispoziție de pachetul `javax.swing.*` pentru dezvoltarea de interfețe grafice precum şi înțelegerea modelului pe care este construit acest pachet.

1. Ce este Swing ?

Swing este o parte a JFC (*Java Foundation Classes*) care cuprinde o serie de facilități destinate dezvoltării de interfețe grafice cu utilizatorul (GUI). Acestea cuprind o parte propriu-zisă de componente grafice, foarte diversificată precum şi o parte de interacțiune.

Funcționalitățile puse la dispoziția utilizatorului în JFC sunt următoare:

- **componente grafice** Swing: ferestre, butoane, câmpuri text, liste, tabele, dispunând de funcționalități extinse ca: sortare, tipărire, drag and drop;
- **look and feel** adaptabil, putându-se stabili ca aspectul interfeței grafice să fie specific Windows sau specific Java (prin suportul oferit pentru GTK+ există pentru componentele Swing posibilități nenumărate de afişare);
- **accesibilitate**, interfața putând interacționa cu tehnologii de asistare ca ecrane tactile / dispozitive Braille spre a prelua informația de la utilizator;
- **Java2D** – include metode pentru integrarea de conținut grafic de calitate în cadrul aplicațiilor, fie că este vorba de text, imagini sau grafică 2D.
- **localizarea conținutului** permite dezoltarea de aplicații care fie capabile să interacționeze cu utilizatori din regiuni diferite, respectând limba şi convențiile culturale proprii.

2. Arhitectura claselor din pachetul javax.swing.*

Clasele din pachetul javax.swing.* respectă paradigma de proiectare Model-View-Controller, arhitectură dezvoltată pe trei niveluri care cuprinde:

- modelul – folosit pentru gestiunea informațiilor, cât şi pentru notificarea observatorilor atunci când se produc modificări;
- vizualizarea – utilizată pentru prezentarea modelului într-o formă care poate fi folosită în interacțiune (elemente de interfață);
- observatorul – responsabil pentru primirea unor cereri (din partea utilizatorului) şi pentru oferirea unor răspunsuri prin apelarea metodelor obiectelor model şi vizualizare care trebuie să execute acțiunile corespunzătoare parametrilor specificați;

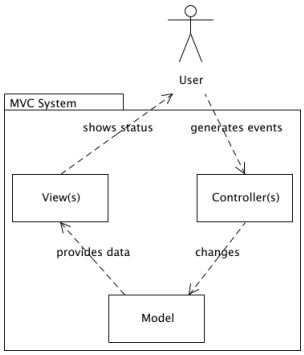


Figura 1 – Arhitectura de proiectare *Model-View-Controller*

Multe din clasele JFC extind sau folosesc clase AWT, incluzând față de acestea componente noi şi îmbunătățite care optimizează modul de afişare precum şi funcționalitatea intefețelor grafice. Clasele Swing sunt mai flexibile decât clasele AWT, permițând uşor particularizarea aspectului şi funcționalității. Există posibilitatea combinării de componente AWT şi componente Swing[1].

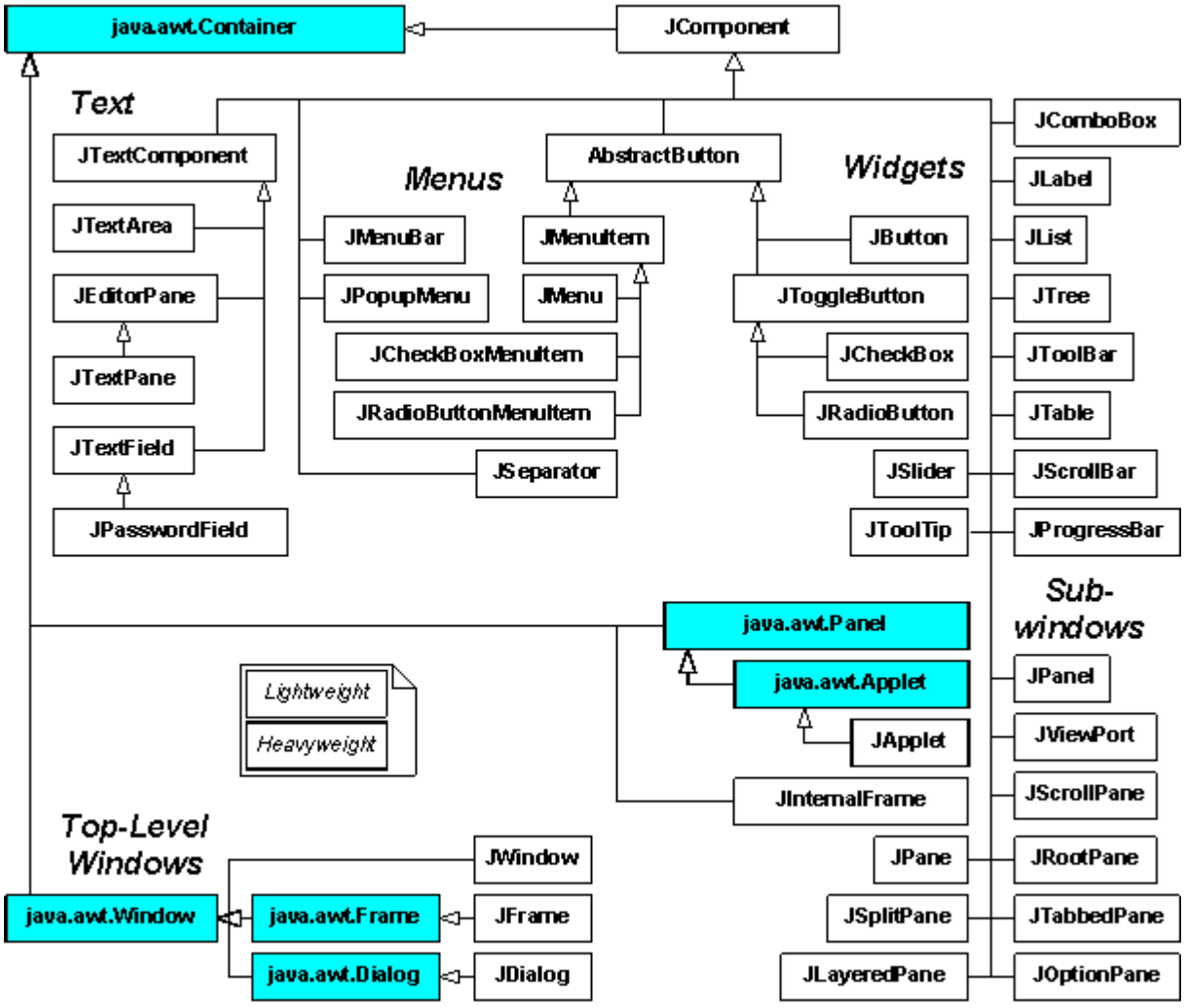


Figura 2 – Arhitectura *Swing*

[1] În acest mod, se permite adăugarea de funcționalitate Swing la aplicații AWT.

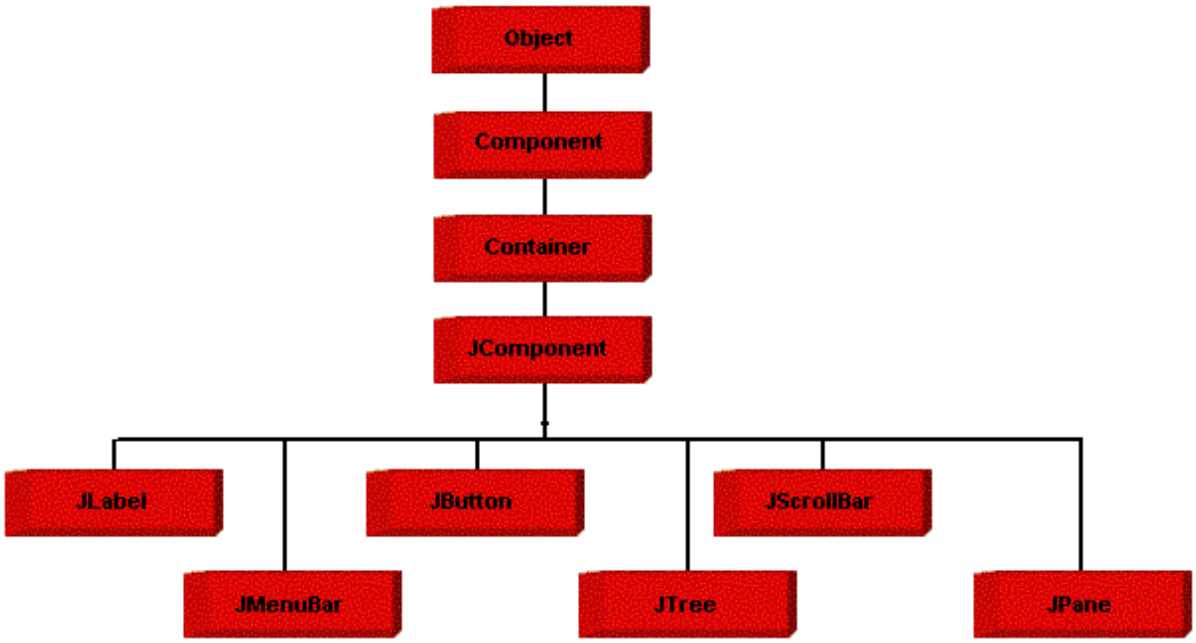


Figura 3 – Arhitectura *Swing* (simplificată)

O clasificare[2] pe baza arhitecturii Model-View-Controller a claselor din pachetul `javax.swing.*` ar fi următoarea:

Clase de tip *model* din arhitectura MVC sunt: DefaultButtonModel, DefaultListSelectionModel, DefaultTreeModel, ce conțin implementarea standard pentru modelele de date asociate componentelor respective.

Clase de tip *view* sunt reprezentate de componentele vizuale Swing care pot fi de două categorii:

- componente grafice care nu pot conține alte obiecte Swing, cum ar fi JButton, JLabel, JSlider;
- componente *container* care grupează mai multe componente grafice precum şi alte containere, clasificarea lor făcându-se în subcategorii:
  - componente de tipul *container top-level* (conțin fereastra principală a aplicației): JFrame, JDialog şi JApplet;
  - componente intermediare, incluse în alte containere, utilizate doar pentru operații de grup: poziționare, adăugare/ştergere la fereastră, cum ar fi JPanel;

Pentru a putea fi afişată, o componentă trebuie să fie parte a unei ierarhii (arbore de componente care are drept rădăcină un container top-level), putând fi conținută de un singur container[3]. Toate containerele de tip top-level conțin un panou de conținuturi (eng. content pane) care conține toate componentele grafice vizibile sau invizibile care îi sunt asociate. Pentru un container de tip top-level poate fi adăugat, opțional, un obiect de tip bară de meniu (JMenuBar), aşa cum se poate observa din exemplul de mai jos unde este redată şi ierarhia de clase:

[2] Mai există o clasificare care împarte clasele Swing în clase care au corespondent AWT (denumite la fel ca cele din AWT, la care se adaugă prefixul 'J': JButton, JLabel, JscrollBar) şi clase JFC extinse: JSlider, JTable, JTree;

[3] În momentul în care o componentă care se găseşte într-un container e adăugată unui container nou, ea va fi asociată containerului corespunzător celei mai recente operații.

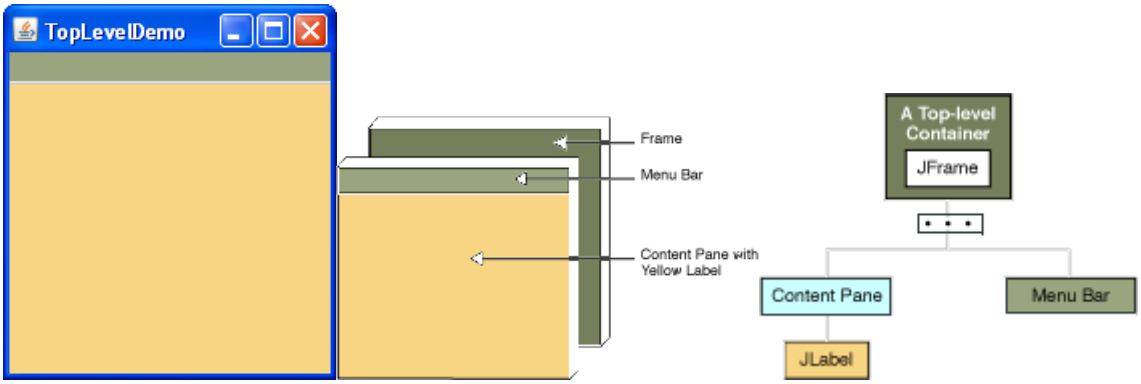


Figura 4 – Ierarhia de clase în dezvoltarea unei interfete grafice simple Swing

Clase de tip *controller* sunt reprezentate de modulele care se ocupă de interacțiunea cu utilizatorul şi care generează acțiunile care trebuie realizate în momentul în care se produce un eveniment: ActionListener. Ele se asociază componentelor de tip grafic prin intermediul metodei *addActionListener()*.

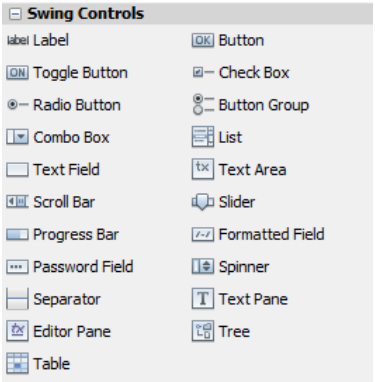
Utilizatorul Swing poate stabili modul de afişare a componentelor grafice într-o fereastră:

- prin folosirea metodelor interfeței LayoutManager[4];
- prin utilizarea de containere (suprafețe delimitate în cadrul ferestrei care pot fi delimitate prin chenare) ca sub-ferestre intermediare;

3. Componente Swing

O altă clasificare a componentelor Swing ar putea fi următoarea:

- componente grafice de bază
  - JButton
  - JCheckBox
  - JComboBox
  - JList
  - JMenu
  - JRadioButton
  - JSlider
  - JSpinner
  - JTextField
  - JPasswordField



[4] Clase ce implementează această interfață sunt: BasicScrollBarUI, BorderLayout, BoxLayout, CardLayout, DefaultMenuLayout, FlowLayout, GridBagLayout, GroupLayout, MetaScrollBarUI, OverlayLayout, ScrollPaneLayout, SpringLayout, ViewPortLayout.

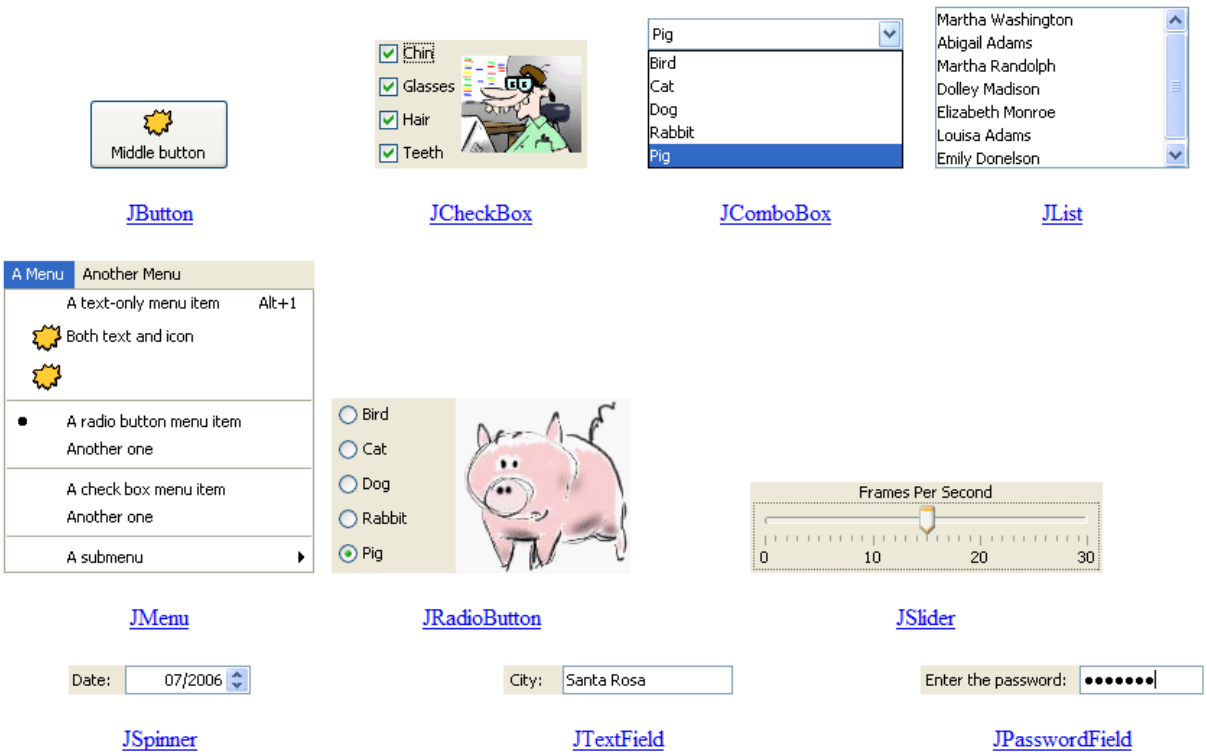


Figura 5 – Componente Swing de bază (în NetBeans Designer şi JavaTutorial)

- componente vizuale interactive cu informații formatate
  - JcolorChooser
  - JEditorPane
  - JTextPane
  - JFileChooser
  - JTable
  - JTextArea
  - JTree

- componente grafice ne-editabile pentru afişarea de informații
  - JLabel
  - JProgressBar
  - JSeparator
  - JToolTip

- containere de tip top-level
  - JApplet
  - JDialog
  - JFrame

- containere cu scop general
  - JPanel
  - JScrollPane
  - JSplitPane
  - JTabPane
  - JToolBar

- containere cu scop specific
  - JInternalFrame
  - JLayeredPane

Clasa JComponent este moştenită de toate compnentele grafice din pachetul Swing care sunt prefixate de litere „J", extinzând clasa Container care extinde la rândul ei clasa Component.

Clasa Container conține toate metodele necesare pentru adăugarea de componente vizuale la container-ul curent.

Clasa Component conține toate metodele necesare pentru desenarea de componente vizuale şi tratarea evenimentelor.

Funcționalitatea pe care clasa JComponent o pune la dispoziția tuturor componentelor vizuale care o extind sunt:

- afişarea de mesaje explicative (eng. tool tips), prin apelarea metodei setToolTipText();
- desenare şi trasarea marginilor (prin metoda setBorder);
- stabilirea unui şablon pentru aspect (eng. Look and Feel) prin metoda ComponentUI.setLookandFeel() corespunzătoare pentru fiecare JComponent;
- impunerea unor proprietați specifice, necesare pentru specificarea unor constrângeri, prin metodele: put/get ClientProperty();
- suport pentru dispunerea componentelor vizuale (prin metodele setMinimumSize, setMaximumSize, setAlignementX, set AlignementY);
- asigurare de suport pentru accesibilitate;
- asigurare de suport pentru drag and drop;
- folosirea modalității de desenare double buffering pentru afişarea pe ecran mai fină a componentelor;
- stabilirea de asocieri între evenimente (apăsări de taste) cu anumite comportamente predefinite;

Particularizarea aspectului componentei se face prin metodele:

get/setBorder	obținere sau stabilire margine
setForeground/Background	stabilirea culorii pentru partea din fața componentei sau pentru partea din spatele componentei
getForeground/Background	obținerea culorii pentru partea din fața componentei sau pentru partea din spatele componentei
setOpaque/isOpaque	stabileşte sau verifică dacă componenta este opacă[5]
setFont/getFont	stabilirea sau obținerea tipului de caractere care este atribuit pentru componentă[6]
setCursor/getCursor	stabilirea sau obținerea unui cursor afişat deasupra componentei cât şi asupra tuturor componentelor conținute[7]

[5] O componentă opacă stabileşte fundalul din spatele ei cu aceeaşi culoare cu care este desenată fațada componentei.

[6] Dacă un tip de caractere nu a fost atribuit pentru componentă se va întoarce tipul de caractere corespunzător obiectului părinte al componentei.

[7] Excepție fac obiectele copil ale componentei pentru care s-a stabilit deja un alt cursor.

Gestiunea stării componentelor poate fi realizată prin metodele:

setComponentPopupMenu	stabilirea obiectului de tip JPopupMenu asociat componentei, procesul de stabilire a legăturii precum şi procesul de asociere a obiectelor responsabile de interacțiune cade în sarcina interfeței cu utilizatorul
set/getTransferHandler	adaugă sau elimină proprietatea transferHandler[8]
setToolTipText	stabileşte textul care se afişează în cadrul mesajului explicativ (eng. tooltip)
set/getName	stabileşte sau obține numele componentei[9]
isShowing	verifică dacă componenta este vizibilă pe ecran (componenta este vizibilă şi face parte dintr-un container care este vizibil)
setEnabled/isEnabled	stabileşte sau verifică dacă o componentă e activată[10]
setVisible/isVisible	stabileşte sau verifică dacă o componentă e vizibilă[11]

Tratarea evenimentelor se realizează prin intermediul funcțiilor:

add/removeHierarchyListener	adaugă sau elimină un obiect „ascultător” către evenimente generate de modificări ale ierarhiei la nivelul componentei[12]
add/removeMouseListener	adaugă sau elimină un obiect „ascultător” către evenimentele generate de operații cu mouse-ul care interacționează cu componenta
add/removeMouseMotionListener	adaugă sau elimină un obiect „ascultător” către evenimentele generate de operații de mişcare a mouse-ului deasupra componentei
add/removeKeyListener	adaugă sau elimină un obiect „ascultător” către evenimentele generate de operații cu tastatura care interacționează cu componenta
add/removeComponentListener	adaugă sau elimină un obiect „ascultător” către evenimentele generate de operații de tip afişare/neafişare, mutare, redimensionare
contains	precizează dacă un punct (specificat ca parametru) face parte din componentă[13]
getComponent	obține componenta care se găseşte la coordonatele specificate ca parametru, aflată pe poziția cea mai înaltă, în ca că mai multe componente se suprapun
set/getComponentZOrder	mută/obține componenta specificată la indexul dat ca pozitie în ordinea de desenare

[8] Proprietatea oferă suport pentru operații de cut, copy, paste realizate in/din clipboard precum şi operații de tip drag and drop.
[9] Metoda este utilă când se doreşte asocierea unui text cu o componentă care nu afişează nici un şir de caractere.
[10] O componentă care este activată interacționează cu utilizatorul şi generează evenimente.
[11] Componentele sunt inițial vizibile (excepție fac componentele de tip top-level).
[12] Evenimentele sunt generate atunci când se modifică ierarhia containerului din care componenta face parte.
[13] Trebuie precizate valorile x şi y pentru punct relativ la sistemul de coordonate al componentei.

Desenarea de componente se realizează folosind metodele:

repaint	determină o parte din componentă sau componenta în totalitate să fie redesenată (dreptunghiul care va fi redesenat este precizat ca parametru prin coordonatele x, y, lățime, înălțime)
revalidate	determină componenta şi containerele afectate să fie reaşezate[14]
paintComponent	desenează componenta (se implementează pentru componente personalizate, se urmăreşte suprascrierea metodei de desenare implicită)

Interacțiunea cu ierarhia se face prin metodele puse la dispoziție:

add	adaugă componenta specificată ca parametru la container
remove/removeAll	elimină unul sau toate componentele din cadrul container-ului
getRootPane	obține rădăcina panoului care conține componenta
getTopLevelAncestor	obține container-ul de nivelul cel mai înalt (Window, Applet)
getParent	obține container-ul imediat pentru componentă
getComponentCount	obține numărul de componente în container
getComponent/getComponents	obține unul sau toate componentele din container
getComponentZOrder	obține ordinea de desenare ale componentei în container

Metodele apelate pentru **dispunerea componentelor** sunt enumerate mai jos:

setPreferred/Minimum/MaximumSize	stabileşte dimensiunile preferate[15], minime sau maxime pentru componentă, exprimate în pixeli[16]
getPreferred/Minimum/MaximumSize	obține dimensiunile preferate, minime sau maxime pentru componentă, exprimate în pixeli
setAlignementX/Y	stabileşte alinierea pe axa Ox, şi pe Oy, indicând poziția relativă a componentei față de alte componente[17]
getAlignementX/Y	obține alinierea pe axa Ox şi pe Oy
set/getLayout	stabileşte sau obține obiectul responsabil cu gestiunea modului de dispunere a conținutului în componentă (LayoutManager)
Apply/setComponentOrientation	stabileşte proprietatea ComponentOrientation a containerului

[14] Nu se impune apelarea unei astfel de metode decât în cazul în care dimensiunile unei componente din container sunt modificate.
[15] Dimensiunea preferată indică cea mai potrivită dimensiune pentru componentă.
[16] Componenta nu ar trebui să fie mai mică decât dimensiunea minimă şi nici mai mare decât dimensiunea maximă.
[17] Valorile sunt cuprinse între 0 şi 1 unde 0 este originea, 1 distanța cea mai depărtată față de origine, iar 0.5 reprezintă o aliniere centrată.

Obținerea informațiilor despre dimensiune şi poziție poate fi făcută prin metodele:

getWidth/Height	obține lățimea respectiv înălțimea componentei exprimată în pixeli
getDimension	obține dimensiunea componentei măsurată în pixeli[18]
getX/Y	obține coordonata pe axa Ox sau Oy relativ la poziția stânga-sus a părintelui, exprimată în pixeli
getBounds	obține limitele componentei exprimate în pixeli[19]
getLocation	obține locația curentă a componentei relativ la poziția stânga-sus a părintelui, exprimată în pixeli
getLocationOnScreen	obține locația curentă a componentei relativ la poziția stânga-sus a ecranului, exprimată în pixeli
getInsets	obține dimensiunea marginii componentei

Specificarea dimensiunii şi poziției absolute se poate realiza doar prin intermediul următoarelor funcții:

setLocation	stabileşte locația componentei exprimată în pixeli, relativ la colțul stânga-sus al părintelui[20]
setSize	stabileşte dimensiunea componentei, exprimată în pixeli
setBounds	stabileşte dimensiunea şi locația relativă a componentei față de colțul stânga-sus al părintelui (se specifica x, y, lungime, înălțime)

3.1 Componente de tip text

Componentele de tip text din pachetul Swing sunt responsabile de afişarea şirurilor de caractere care, în anumite condiții, pot fi şi editate.

Sunt oferite şase componente de tip text ce pun la dispoziția utilizatorului metode complexe de gestiune a textului. Toate componentele moştenesc clasa JTextComponent care dispune de metode configurabile de manipulare a textelor.

Clasificarea componentelor de tip text cuprinde următoarele categorii:

- controale (câmpuri) de text – afişează doar o linie de text editabil, generând evenimente: JTextField, JFormattedTextField, JPasswordField;
- suprafețe de text simplu – afişează mai multe linii de text editabil, întregul şir de caractere folosind acelaşi tip de caractere: JTextArea;
- text stilizat – poate afişa mai multe linii de text editabil folosind mai mult de un tip de caractere, permițând includerea de imagini şi de componente: JEditorPane, JTextPane;

[18] Programatorul este responsabil de creearea obiectului de tip Dimension în care este întors rezultatul.
[19] Prin limite se înțeleg lungimea, lățimea şi originea componentei relative la părinte.
[20] Metoda se foloseşte pentru poziționarea componentei atunci când nu se foloseşte un obiect responsabil cu gestiunea modului de dispunere a conținutului în componentă: `setLayoutManager(null)`.

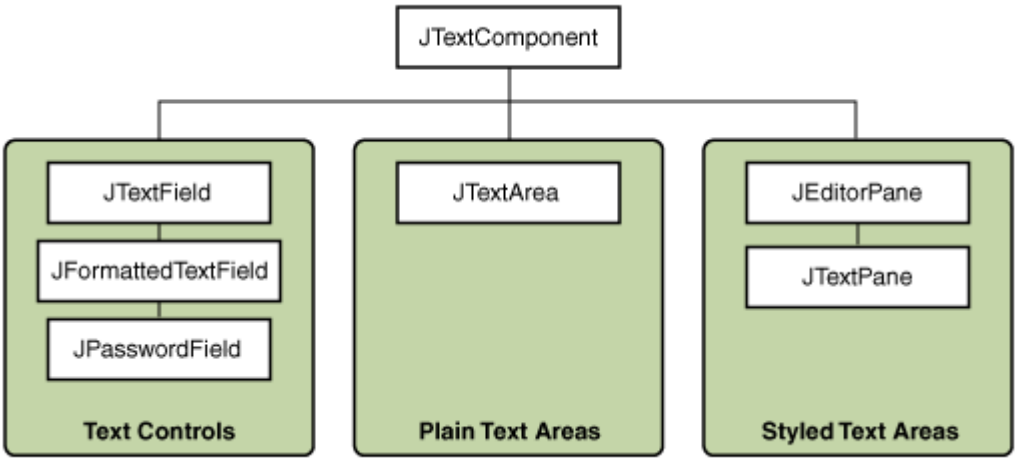


Figura 6 – Ierarhia JTextComponent

3.2 Componente de tip buton

În Swing, componentele de tip buton extind o clasă abstractă denumită AbstractButton:

- JButton – un buton obişnuit;
- JCheckBox – un buton de tip checkbox;
- JRadioButton – un buton dintr-un grup de butoane radio;
- JMenuItem – un element dintr-un meniu;
- JCheckBoxMenuItem – un element într-un meniu care are asociat în plus un buton de tip checkbox;
- JRadioButtonMenuItem – un element într-un meniu ce are asociat în plus un buton de tip radio;
- JToggleButton – implementează un mecanism de tip toggle, moştenit de JCheckBox şi JRadioButton, putând fi instanțiat pentru a crea butoane care pot avea două stări;

Exemplu. Să se implementeze o interfață grafică pentru autentificarea în sistem pe baza unui nume de utilizator şi a unei parole.

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

public class InterfataGrafica
{
    JFrame fereastra;
    Container continut_fereastra;
    JPanel panou;
    JLabel eticheta_nume_utilizator, eticheta_parola;
    JTextField camp_nume_utilizator;
    JPasswordField camp_parola;
    JButton buton_acord, buton_revocare;

    public InterfataGrafica ()
    {
        initializare();
    }
```

```
    public void initializare ()
    {
        fereastra = new JFrame ();

        fereastra.setTitle(Configurare.DENUMIRE_FEREASTRA);
        fereastra.setSize(Configurare.LATIME_FEREASTRA,
                          Configurare.INALTIME_FEREASTRA);
        fereastra.setResizable(false);
        fereastra.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        adauga_componente();

        fereastra.setVisible(true);
    }

    public void adauga_componente ()
    {
        continut_fereastra = fereastra.getContentPane();
        continut_fereastra.setLayout(null);

        panou = new JPanel (null);
        panou.setBorder(new TitledBorder(Configurare.DENUMIRE_PANOU));
        panou.setBounds (Configurare.PANOU_X,
                         Configurare.PANOU_Y,
                         Configurare.PANOU_W,
                         Configurare.PANOU_H);

        eticheta_nume_utilizator =
        new JLabel (Configurare.ETICHETA_NUME_UTILIZATOR);
        eticheta_nume_utilizator.setBounds(...);
        panou.add(eticheta_nume_utilizator);

        camp_nume_utilizator =
        new JTextField(Configurare.DIMENSIUNE_CAMP_TEXT);
        camp_nume_utilizator.setBounds(...);
        panou.add(camp_nume_utilizator);

        eticheta_parola = new JLabel (Configurare.ETICHETA_PAROLA);
        eticheta_parola.setBounds(...);
        panou.add(eticheta_parola);

        camp_parola = new JPasswordField(Configurare.DIMENSIUNE_CAMP_TEXT);
        camp_parola.setBounds(...);
        panou.add(camp_parola);

        continut_fereastra.add(panou);

        buton_acord = new JButton (Configurare.BUTON_ACORD);
        buton_acord.setBounds(...);
        continut_fereastra.add(buton_acord);

        buton_revocare = new JButton (Configurare.BUTON_REVOCARE);
        buton_revocare.setBounds(...);
        continut_fereastra.add(buton_revocare);
    }

    public static void main(String[] args)
    {
        new InterfataGrafica();
    }
}
```

Rezultatul rulării aplicației este redat în figura de mai jos:

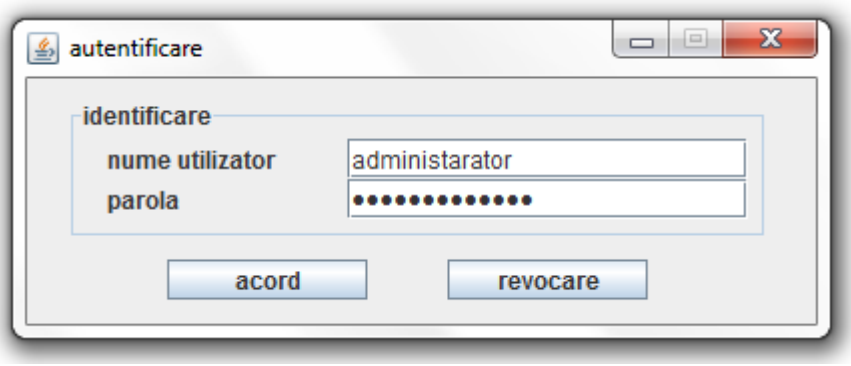


Figura 7 – Interfața autentificare

Am creat o fereastră având două câmpuri text, ambele editabile, între care unul are textul afişat în clar, iar altul are textul ascuns.

Am preferat să nu folosim nici un obiect responsabil cu gestiunea modului de dispunere a conținutului (LayoutManager), specificând manual dimensiunile pentru fiecare componentă în parte, astfel încât să avem control maxim asupra felului în care sunt aşezate în fereastră elementele grafice.

4. Tratarea evenimentelor asociate obiectelor Swing

Obiectele Swing, în special cele editabile sau cele care sunt caracterizate prin interacțiunea cu utilizatorul, generează evenimente asincrone, al căror moment de producere nu poate fi prevăzut de execuția programului.

Spre exemplu, evenimente tipice care pot genera evenimente sunt apăsarea unei taste în cadrul unei componente de tip text, apăsarea pe mouse deasupra unei componente de tip buton, etc.

Aşa-zisa programare condusă de evenimente (*event driven programming*) trebuie să reacționeze la orice eveniment produs de o acțiune a utilizatorului, apelând metode exclusiv ca urmare a producerii unor evenimente.

Arhitectura unor astfel de aplicații este de tip **observer-listener** în care există două tipuri de evenimente:

- obiecte **producătoare** de evenimente (obiecte observate) – sunt de obicei instanțe ale unor clase JFC (sau AWT) creând obiecte de tip eveniment ca urmare a acțiunii operatorului asupra componentei în cauză;
- obiecte **consumatoare** de evenimente (obiecte observator sau ascultător) – reprezintă instanțe ale unor clase dezvoltate în particular pentru fiecare aplicație, în funcție de specificul pe care îl prezintă.

Evenimentul este echivalent prin urmare cu apelarea unei metode din obiectul observator sau ascultător, primind ca argument un obiect eveniment, având tipul componentei JFC (sau AWT) care l-a generat.

```
public void actionPerformed (ActionEvent e);
```

Evenimentele JFC se pot clasifica astfel:

- evenimente specifice componentelor Swing;
- evenimente specifice dispozitivelor de intrare/ieşire;

Pentru fiecare obiect producător de evenimente se pot înregistra mai multe obiecte observator (ascultător), interesate de declanşarea evenimentelor generate fapt realizat printr-o metodă de forma addXXXListener, unde XXX reprezintă tipul evenimentului (şi care dă şi numele interfeței corespunzătoare);

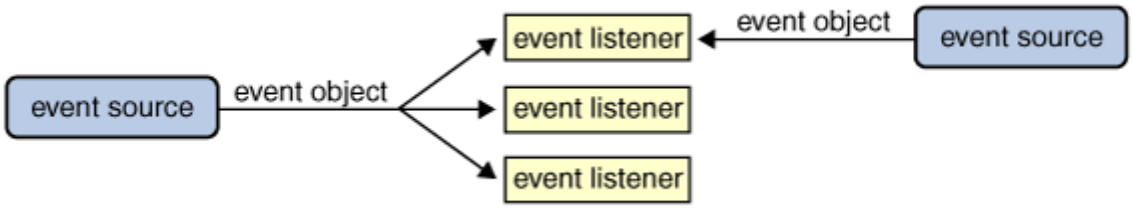


Figura 8 – Modelul observer-listener

Exemple pentru astfel de funcții ar fi:

- addActionListener;
- addHierarchyListener;
- addKeyListener;
- addMouseListener;
- addMouseMotionListener;

Etapele dezvoltării unui sistem sensibil la acțiunile utilizatorului ar fi:

1. **definirea clasei observator (ascultător)** ca implementând o interfață JFC (AWT), de tipul ActionListener, HierarchyListener, KeyListener, MouseListener, MouseMotionListener prin stabilirea comportamentului metodei(lor) ce specifică comportamentul la producerea unui eveniment;
2. **legarea obiectelor observator (ascultător) la componentele Swing** susceptibile de a genera evenimente;

Exemplu (cont'd). Dacă se doreşte ca interfața grafică dezvoltată să reacționeze corespunzător la acțiunile utilizatorului, vom adăuga următoarele instrucțiuni:

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

public class InterfataGrafica implements ActionListener
{
    ...

    public void adauga_componente ()
    {

        buton_acord = new JButton (Configurare.BUTON_ACORD);
        buton_acord.setBounds(Configurare.BUTON_ACORD_X,
                              Configurare.BUTON_ACORD_Y,
                              Configurare.BUTON_ACORD_W,
                              Configurare.BUTON_ACORD_H);
        buton_acord.addActionListener(this);
        continut_fereastra.add(buton_acord);
```

```
        buton_revocare = new JButton (Configurare.BUTON_REVOCARE);
        buton_revocare.setBounds(Configurare.BUTON_REVOCARE_X,
                                 Configurare.BUTON_REVOCARE_Y,
                                 Configurare.BUTON REVOCARE W,
                                 Configurare.BUTON_REVOCARE_H);
        buton_revocare.addActionListener(this);
        continut_fereastra.add(buton_revocare);
    }

    public boolean verificare_identitate
    (String nume_utilizator, String parola)
    {
        return nume_utilizator.equals(parola);
    }

    public void actionPerformed (ActionEvent e)
    {
        if (buton_acord == (JButton)e.getSource())
        {
            if (verificare_identitate
                (camp_nume_utilizator.getText(),camp_parola.getText()))

                JOptionPane.showMessageDialog
                (fereastra,
                 Configurare.AUTENTIFICARE_REUSITA,
                 Configurare.AUTENTIFICARE_REUSITA,
                 JOptionPane.INFORMATION_MESSAGE);

            else

                JOptionPane.showMessageDialog
                (fereastra,
                 Configurare.AUTENTIFICARE_ESUATA,
                 Configurare.AUTENTIFICARE_ESUATA,
                 JOptionPane.ERROR_MESSAGE);
        }

        if (buton_revocare == (JButton)e.getSource())
        {
            System.exit(0);
        }
    }

    public static void main(String[] args)
    {
        ...
    }
}
```

Rezultatul rulării aplicației este redat în figura de mai jos:

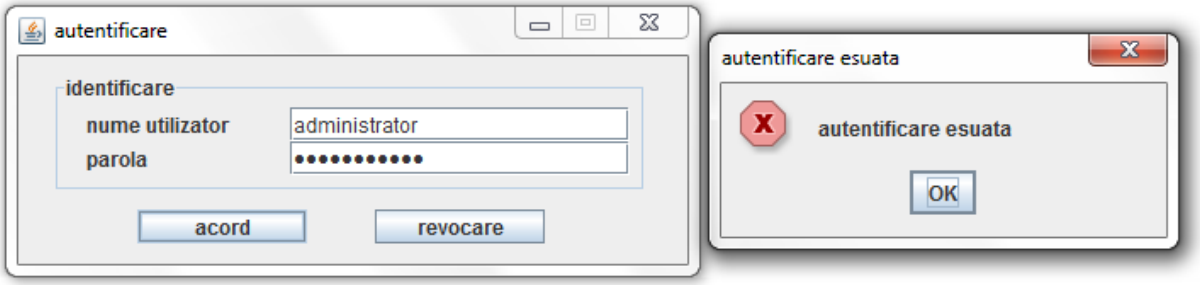


Figura 9 – Rezultatul unei autentificări eşuate

Aşadar, pentru simplitate (este vorba despre uşurința accesului la componentele Swing din metoda actionPerformed) clasa observator a fost definită în clasa care dezvoltă interfața grafică, fiind implementată metoda *actionPerformed* unde evenimentul este tratat în funcție de sursa care l-a generat

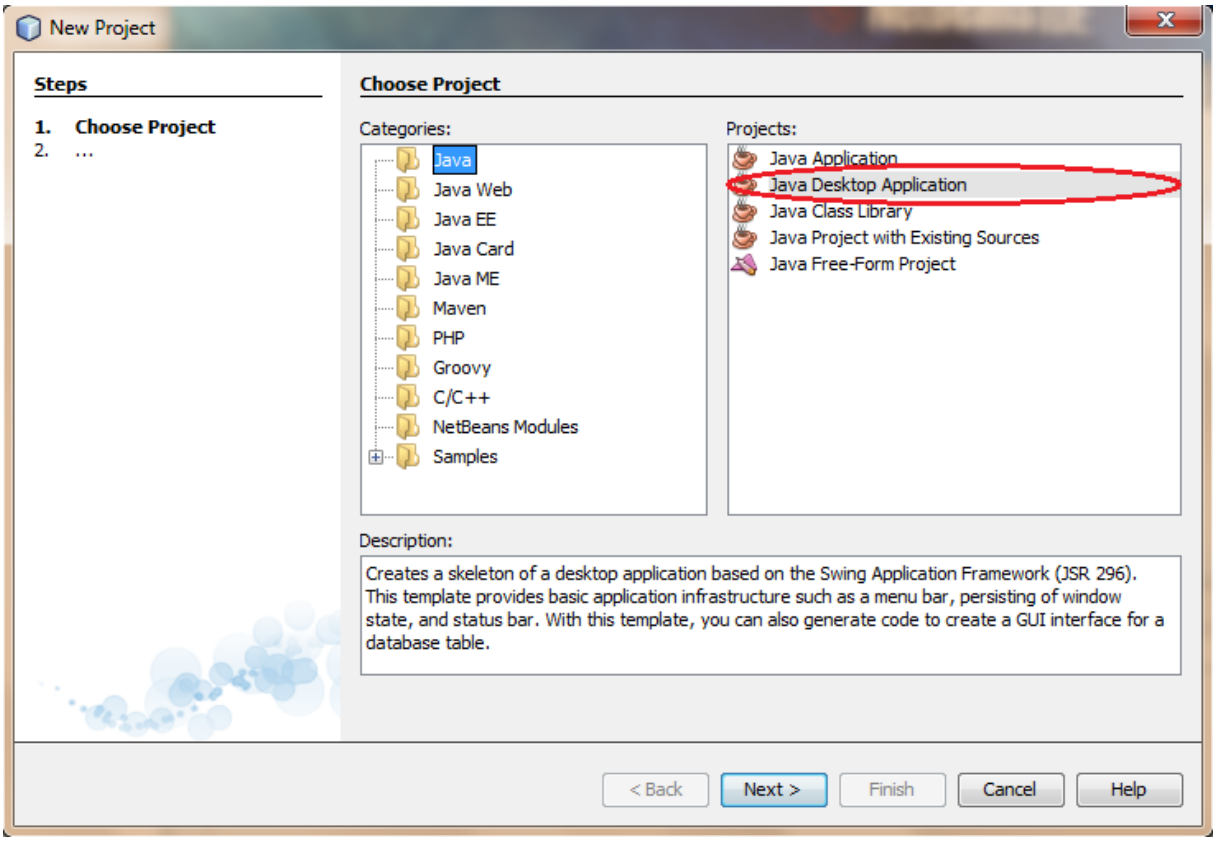
Dezvoltarea unei interfețe grafice presupune, aşadar, etapele:

- crearea unui **obiect de tip fereastră** JFrame sau un subtip al acesteia prin stabilirea unor proprietăților ferestrei (titlu, dimensiuni, etc.);
- crearea **componentelor grafice Swing** (de tip text, de tip buton, etc.) şi stabilirea proprietăților (chenare, localizare, etc.);
- **asocierea** componentelor grafice **în grupuri** (JPanel sau clasă derivată din aceasta) – o astfel de etapă nu este obligatorie;
- **integrarea conținuturilor intermediare** (în cazul în care au fost definite, altfel se vor integra componentele grafice) la fereastra principală şi stabilirea **modului de dispunere** (layout) al acestora;
- **tratarea evenimentelor** asociate **ferestrei principale** şi **componentelor** prin definirea unor clase de tip observator (ascultător) la evenimente generate de acestea;

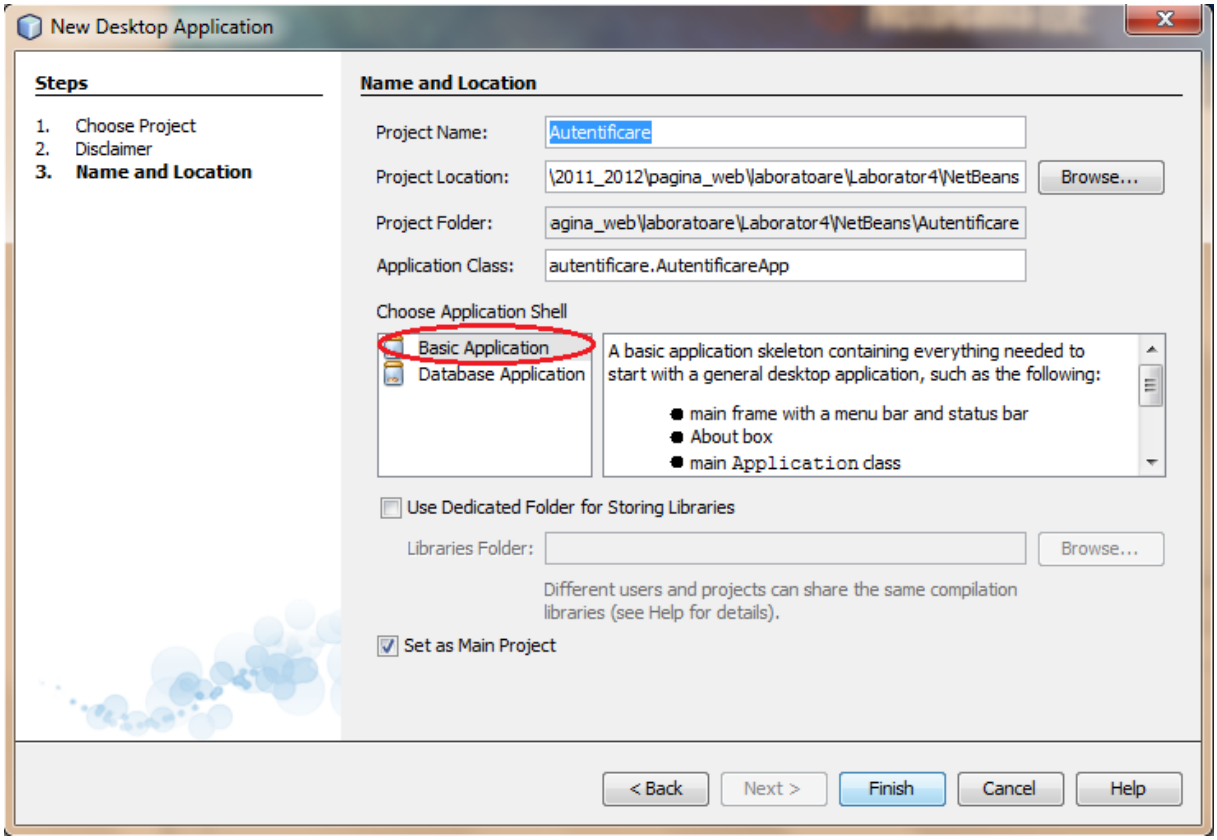
5. Dezvoltarea de interfețe grafice Swing în mod vizual

5.1 NetBeans 7.0.1

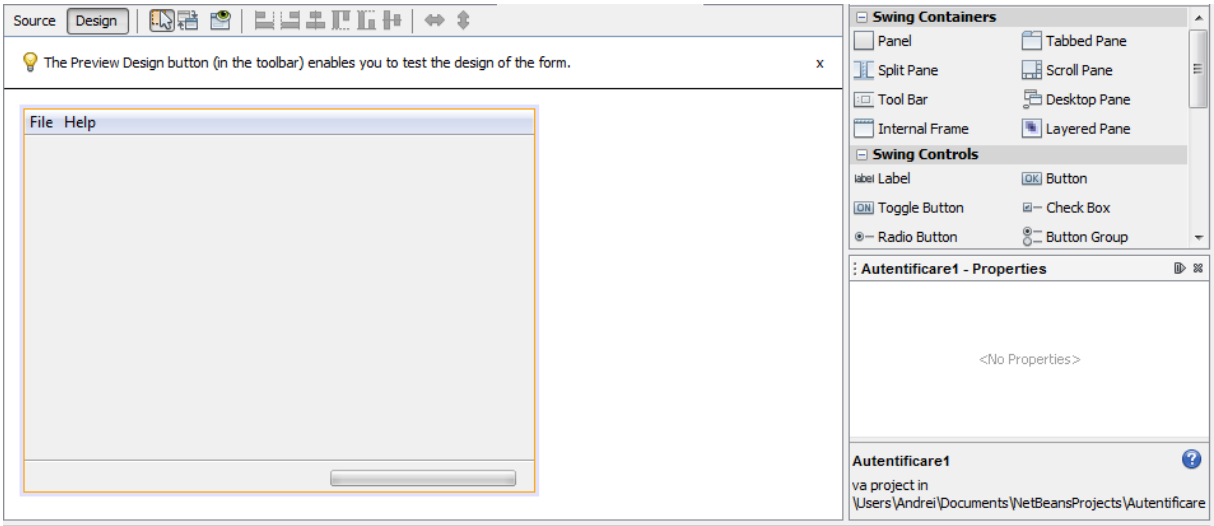
După selectarea opțiunii New Project din meniul Files, se va opta pentru tipul de aplicație „Java Desktop Application”:



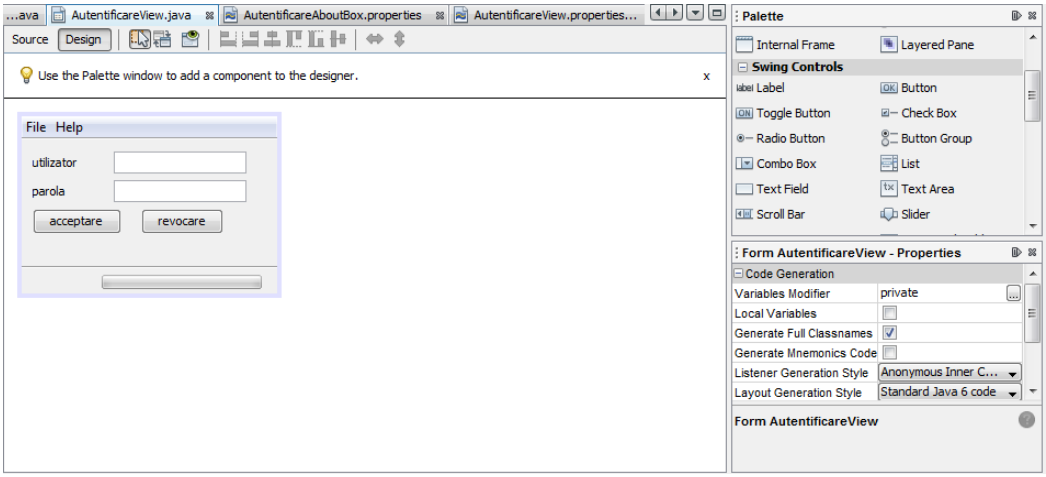
Se va selecta opțiunea „Basic Application”. Prin alegerea opțiunii „Database Application”, se poate lega aplicația la o bază de date pentru care trebuie specificați parametrii de conectare, oferindu-se o funcționalitate de tip CRUD (Create, Read, Update, Delete):

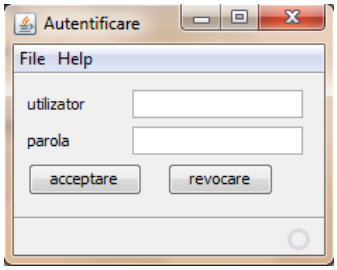


Vom obține un mediu de lucru în care va fi activat modulul Designer unde se pot crea componente grafice Swing, codul sursă fiind generat automat, şi putând fi vizualizat în modul Source:

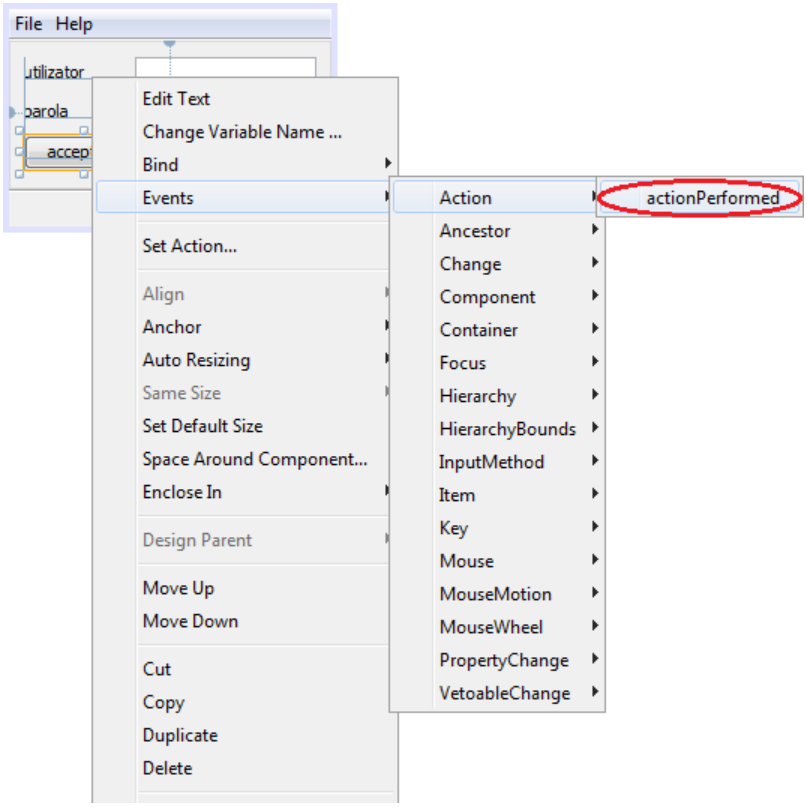


În continuare, dezvoltarea aplicației presupune exclusiv operații de tipul drag-and-drop, ajungându-se la un rezultat asemănător cu cel de mai jos:





Integrarea de acțiuni asociate evenimentelor ce pot fi generate de către componentele Swing se face din meniul care se deschide când face click dreapta pe obiectul pentru care vrem să adăugăm acțiunea:

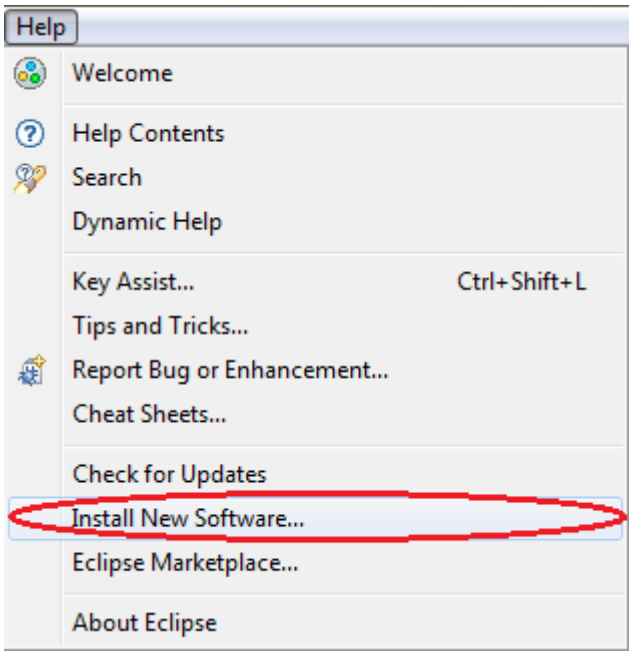


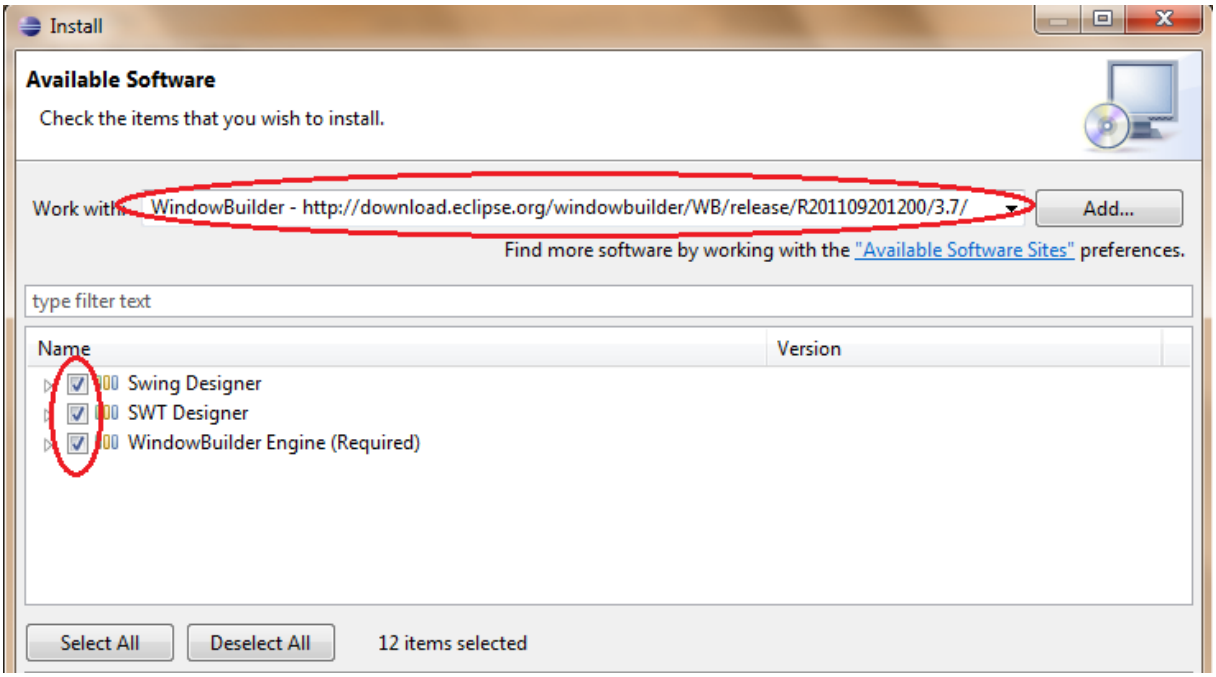
Se va adăuga în mod automat codul corespunzător evenimentului dorit, urmând să fie completat cu acțiunea care se impune a fi luată:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

5.2 Eclipse Indigo

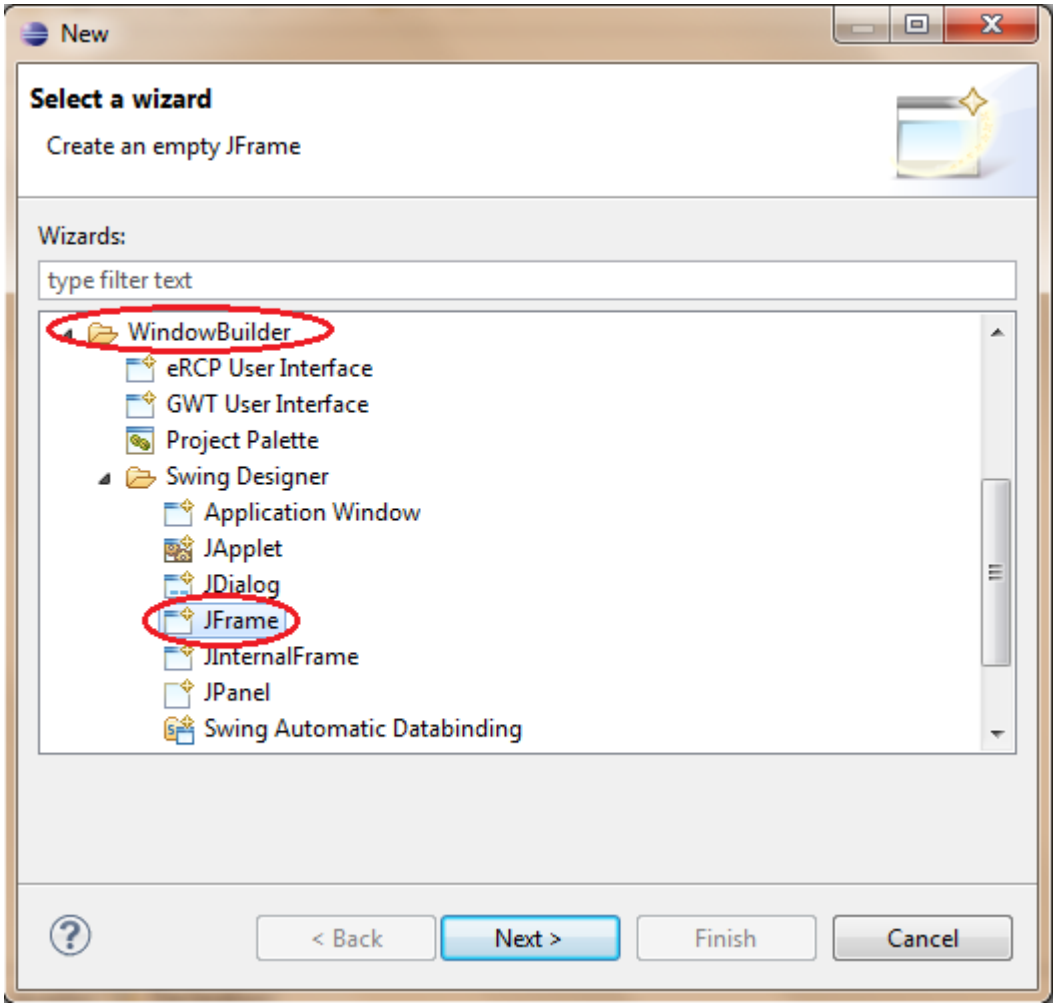
Pentru Eclipse există un utilitar denumit *WindowBuilder Pro* care poate fi instalat de la http://eclipse.org/windowbuilder/download.php.



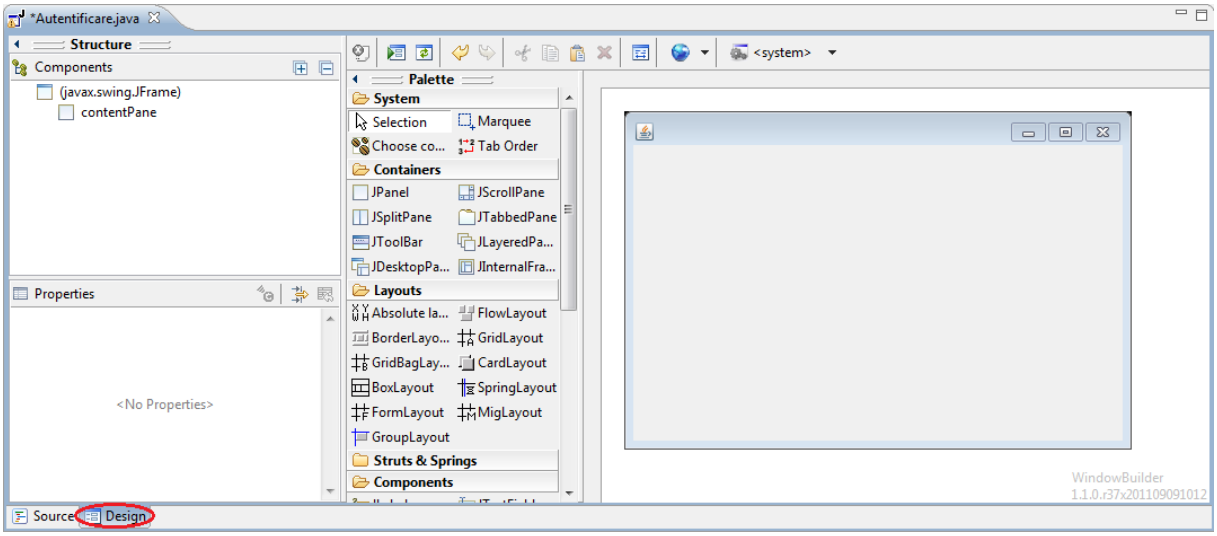


Eclipse va trebui repornit înainte de a putea folosi *WindowBuilder Pro*.

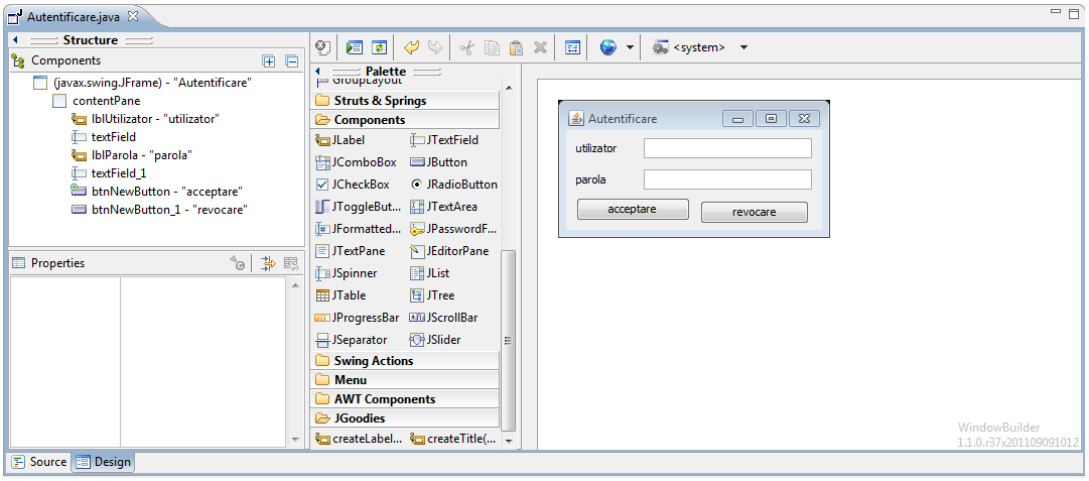
După crearea unui proiect nou, se va adăuga o nouă resursă la proiect (prin click dreapta pe numele proiectului → New → Other... → WindowBuilder).

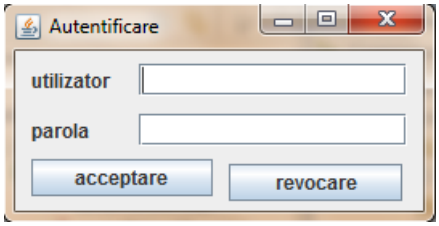


Vom obține un mediu de lucru cu modul ***Design*** activat unde pot fi create componente grafice Swing, codul sursă fiind generat automat (vizualizarea lui este disponibilă în modul Source):



În continuare, dezvoltarea aplicației presupune exclusiv operații de tipul selectare componentă grafică şi plasare la coordonatele dorite[21], ajungându-se la un rezultat asemănător cu cel de mai jos:





Aplicație de Laborator

Se doreşte ca pentru aplicația `TelecomunicaţiiMobile` dezvoltată anterior să se adauge o interfață grafică pentru a permite următoarele operații: vizualizare, adăugare, modificare, ştergere şi căutare de informații în cadrul bazei de date.

S-a realizat deja vizualizarea informațiilor într-o componentă de tip `JTable`.

(0p) 0. Creați structura bazei de date şi populați tabelele conform script-ului `Laborator4.sql`

(0p) 0. Modificați parametrii de conectare la baza de date din clasa `Constante`.

(3p) 1. Să se creeze în clasa `Componenta` un buton care să permită adăugarea de înregistrări în toate tabelele bazei de date.

(3p) 2. Să se creeze în clasa `Componenta` un buton care să permită modificarea de înregistrări în toate tabelele bazei de date.

(3p) 3. Să se creeze în clasa `Componenta` un buton care să permită ştergerea de înregistrări în toate tabelele bazei de date.

(1p) 4. În proiectul `Autentificare` (dezvoltat folosind medii vizuale), verificați existența unui client în baza de date telecomunicatii_mobile, stiind că utilizator este `CNP`, iar parola are forma `nume.prenume`.

BONUS

(3p) 5. Pentru fiecare atribut reținut într-o tabelă să se creeze un câmp text precum şi un buton care să permită căutarea după valoarea câmpului respectiv.

[21] Înainte de această operație, trebuie selectat un mod de gestiune al conținutului (`LayoutManager`). În cazul de față a fost ales Absolute layout.