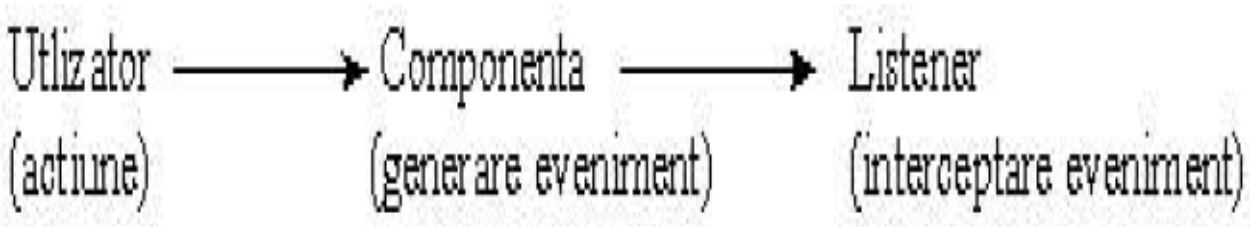
Interfața grafică cu utilizatorul - AWT

Programare Orientată pe Obiecte

Tratarea evenimentelor

- Eveniment: apăsarea unui buton, modificarea textului, închiderea unei ferestre,etc.
- Sursă: componentă care generează un eveniment.
- Interceptarea evenimentelor generate de componentele unui program se realizează prin intermediul unor clase de tip listener
- Evenimentele - AWTEvent:

  ActionEvent, TextEvent.



Ascultătorii - EventListener:

```
ActionListener, TextListener.
class AscultaButoane implements ActionListener {
    public void actionPerformed(ActionEvent e) {
    // Metoda interfetei ActionListener
    ...
    }
}
class AscultaTexte implements TextListener {
    public void textValueChanged(TextEvent e) {
    // Metoda interfetei TextListener
    ...
    }
}
class Ascultator implements ActionListener, TextListener {
    public void actionPerformed(ActionEvent e) {
    ...
    }
    public void textValueChanged(TextEvent e) {
    ...
    }
}
```

Componentele AWT şi tipurile de evenimente generate

Component	ComponentListener FocusListener KeyListener MouseListener MouseMotionListener
Container	ContainerListener
Window	WindowListener
Button List MenuItem TextField	ActionListener
Choice Checkbox List CheckboxMenuItem	ItemListener
Scrollbar	AdjustmentListener
TextField TextArea	TextListener

Interfețe ascultător

Interfaţă	Metode
ActionListener	actionPerformed(ActionEvent e)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent e)
ComponentListener	componentHidden(ComponentEvent e) componentMoved(ComponentEvent e) componentResized(ComponentEvent e) componentShown(ComponentEvent e)
ContainerListener	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)
FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
ItemListener	itemStateChanged(ItemEvent e)
KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)
MouseListener	mouseClicked(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mousePressed(MouseEvent e) mouseReleased(MouseEvent e)
MouseMotionListener	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)
TextListener	textValueChanged(TextEvent e)
WindowListener	windowActivated(WindowEvent e) windowClosed(WindowEvent e) windowClosing(WindowEvent e) windowDeactivated(WindowEvent e) windowDeiconified(WindowEvent e) windowIconified(WindowEvent e) windowOpened(WindowEvent e)

Evenimente de la surse multiple

- metoda getSource returnează obiectul responsabil cu generarea evenimentului.

```
public void actionPerformed(ActionEvent e) {
    Object sursa = e.getSource();
    if (sursa instanceof Button) {
        // A fost apasat un buton
        Button btn = (Button) sursa;
        if (btn == ok) {// A fost apasat butonul ’ok’
        }
        ...
    }
    if (sursa instanceof TextField) {
        // S-a apasat Enter dupa editarea textului
        TextField tf = (TextField) sursa;
        if (tf == nume) { // A fost editata componenta ’nume’
        }
        ...
    }
}
```

- ActionEvent conține metoda getActionCommand care, implicit, returnează eticheta butonului care a fost apăsat.

Adaptori şi a clase anonime

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

class Fereastra extends Frame implements WindowListener {
  public Fereastra(String titlu) {
    super(titlu);
    this.addWindowListener(this);
  }

  // Metodele interfetei WindowListener
  public void windowOpened(WindowEvent e) {}
  public void windowClosing(WindowEvent e) {
    // Terminare program
    System.exit(0);
  }
  public void windowClosed(WindowEvent e) {}
  public void windowIconified(WindowEvent e) {}
  public void windowDeiconified(WindowEvent e) {}
  public void windowActivated(WindowEvent e) {}
  public void windowDeactivated(WindowEvent e) {}
}

public class TestWindowListener {
  public static void main(String args[]) {
    Fereastra f = new Fereastra("Test WindowListener");
    f.show();
  }
}
```

Folosirea unui adaptor

```
this.addWindowListener(new Ascultator());
...
class Ascultator extends WindowAdapter {
    // Supradefinim metodele care ne intereseaza
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
```

- TipXListener - TipXAdapter
- Folosirea unei clasa anonime:

```
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
    // Terminam aplicatia
        System.exit(0);
    }
});
```

Structura generală a unei ferestre

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

class Fereastra extends Frame implements ActionListener {

  // Constructorul
  public Fereastra(String titlu) {
    super(titlu);

    // Tratam evenimentul de inchidere a ferestrei
    this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        dispose(); // inchidem fereastra
        // sau terminam aplicatia
        System.exit(0);
      }
    });

    // Eventual, schimbam gestionarul de pozitionare
    setLayout(new FlowLayout());

    // Adaugam componentele pe suprafata ferestrei
    Button exit = new Button("Exit");
    add(exit);

    // Facem inregistrarea claselor listener
    exit.addActionListener(this);

    // Stabilim dimensiunile
    pack(); // implicit

    //sau explicit
    // setSize(200, 200);
  }

  // Implementam metodele interfetelor de tip listener
  public void actionPerformed(ActionEvent e) {
```

Structura generală a unei ferestre

```
      System.exit(0);
   }
}

public class TestFrame {
   public static void main(String args[]) {
      // Cream fereastra
      Fereastra f = new Fereastra("O fereastra");

      // O facem vizibila
      f.show();
   }
}
```

Metode

- getFrames
- setIconImage
- setMenuBar
- setTitle
- setResizable

Tratarea evenimentelor - Exemplu

1. Program pentru afisarea unui buton cu inscriptia "Click Me" si afisarea unei casete de dialog cu titlul "Event Fired" la fiecare clic pe buton (cu mouse). Afisarea casetei de dialog se face astfel: JOptionPane.showMessageDialog(new JFrame(), "“, "Event Fired !", JOptionPane.PLAIN_MESSAGE);

Se vor examina pe rand urmatoarele variante de definire a clasei ascultator la evenimente generate de buton:

a) Cu trei clase separate: clasa ascultator, clasa derivata din "JFrame“ care contine si un buton, clasa cu "main" (care afiseaza fereastra).

b) Cu doua clase: clasa ascultator si clasa derivata din "JFrame" si care contine metoda "main".

c) Cu o singura clasa: clasa ascultator cu nume inclusa in clasa ce contine metoda "main"

d) Cu o singura clasa : clasa ascultator anonima, inclusa intr-un bloc (metoda "addActionListener") din clasa ce contine metoda "main".

e) Cu doua clase: O subclasa a clasei "JButton" care contine si metoda "actionPerformed" si clasa care contine metoda "main".

f) Cu doua clase : clasa ascultator inclusa intr-o subclasa a clasei "JFrame“ (separata de clasa ce contine metoda "main")

g) O singura clasa care extinde pe "JFrame" si implementeaza "ActionListener“ (clasa este si generator si ascultator la evenimente).

a. Cu trei clase separate: clasa ascultator, clasa derivata din "JFrame“ care contine si un buton, clasa cu "main" (care afiseaza fereastra).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class ascultator implements ActionListener{
    public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog(new JFrame(),"","Event
    Fired !",JOptionPane.PLAIN_MESSAGE);
    }
}
class jframe extends JFrame{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(new ascultator());
        setSize(250,250);
        setVisible(true);
    }
}
class test{
    public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

b. Cu doua clase: clasa ascultator si clasa derivata din "JFrame" si care contine metoda "main".

```
class ascultator implements ActionListener{
    public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog(new
                    JFrame(),"","Event Fired!",
                    JOptionPane.PLAIN_MESSAGE);
    }
}
class jframe extends JFrame{
    public jframe(){
            JButton jb=new JButton("Click me!");
            getContentPane().setLayout(new FlowLayout());
            getContentPane().add(jb);
            jb.addActionListener(new ascultator());
            setSize(250,250);
            setVisible(true);
    }
    public static void main(String arg[]){
            JFrame jf=new jframe();
    }
}
```

c. Cu o singura clasa: clasa ascultator cu nume inclusa in clasa ce contine metoda "main"

```
class jframe extends JFrame{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(new ascultator());
        setSize(250,250);
        setVisible(true);
    }
    class ascultator implements ActionListener{
        public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog(new
    JFrame(),"","Event Fired !",JOptionPane.PLAIN_MESSAGE);
        }
    }
    public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

d. Cu o singura clasa : clasa ascultator anonima, inclusa intr-un bloc (metoda "addActionListener") din clasa ce contine metoda "main".

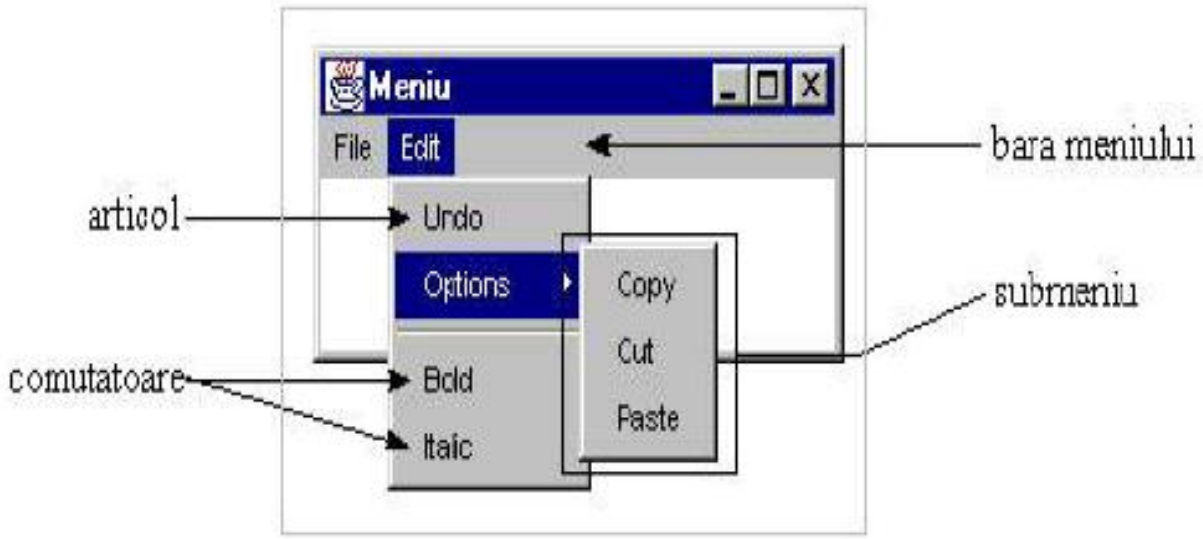
```
class jframe extends JFrame{
    public jframe(){
            JButton jb=new JButton("Click me!");
            getContentPane().setLayout(new FlowLayout());
            getContentPane().add(jb);
            jb.addActionListener(new ActionListener(){
               public void actionPerformed(ActionEvent e){
                        JOptionPane.showMessageDialog(new
                                JFrame(),"","Event Fired!",
                                JOptionPane.PLAIN_MESSAGE);
                    }
            });
            setSize(250,250);
            setVisible(true);
    }
  public static void main(String arg[]){
            JFrame jf=new jframe();
    }
}
```

g. O singura clasa care extinde pe "JFrame" si implementeaza "ActionListener“ (clasa este si generator si ascultator la evenimente).

```
class jframe extends JFrame implements ActionListener{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(this);
        setSize(250,250);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
         JOptionPane.showMessageDialog(new
                JFrame(),"","Event Fired !",
                JOptionPane.PLAIN_MESSAGE);
    }
  public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

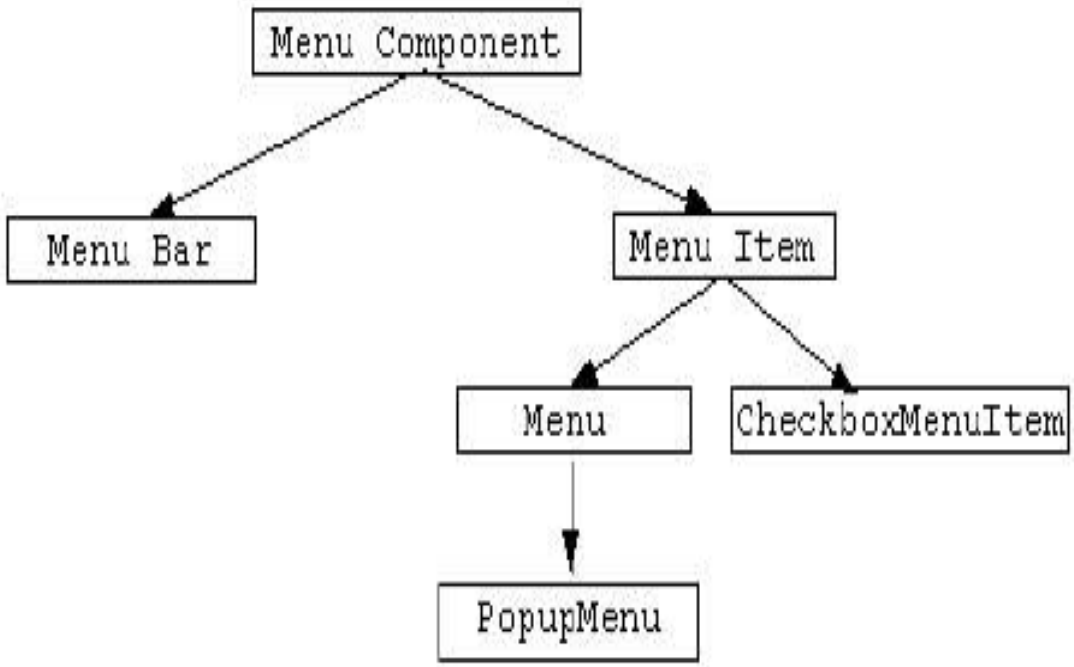
Folosirea meniurilor

- Componentele unui meniu sunt derivate din MenuComponent.
  - Meniuri fixe (vizibile permanent)
  - Meniuri de context (popup)



addMenuBar

Ierarhia claselor ce descriu meniuri



Exemplu

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

public class TestMenu {
  public static void main(String args[]) {
    Frame f = new Frame("Test Menu");

    MenuBar mb = new MenuBar();

    Menu fisier = new Menu("File");
    fisier.add(new MenuItem("Open"));
    fisier.add(new MenuItem("Close"));
    fisier.addSeparator();
    fisier.add(new MenuItem("Exit"));

    Menu optiuni = new Menu("Options");
    optiuni.add(new MenuItem("Copy"));
    optiuni.add(new MenuItem("Cut"));
    optiuni.add(new MenuItem("Paste"));

    Menu editare = new Menu("Edit");
    editare.add(new MenuItem("Undo"));
    editare.add(optiuni);

    editare.addSeparator();
    editare.add(new CheckboxMenuItem("Bold"));
    editare.add(new CheckboxMenuItem("Italic"));

    mb.add(fisier);
    mb.add(editare);

    f.setMenuBar(mb);
    f.setSize(200, 100);
    f.show();
  }
}
```

Tratarea evenimentelor generate de meniuri

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

public class TestMenuEvent extends Frame
     implements ActionListener, ItemListener {

  public TestMenuEvent(String titlu) {
    super(titlu);

    MenuBar mb = new MenuBar();
    Menu test = new Menu("Test");
    CheckboxMenuItem check = new CheckboxMenuItem("Check me")
        ;
    test.add(check);
    test.addSeparator();
    test.add(new MenuItem("Exit"));

    mb.add(test);
    setMenuBar(mb);

    Button btnExit = new Button("Exit");
    add(btnExit, BorderLayout.SOUTH);
    setSize(300, 200);
    show();
```

Tratarea evenimentelor generate de meniuri

```
    test.addActionListener(this);
    check.addItemListener(this);
    btnExit.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    // Valabila si pentru meniu si pentru buton
    String command = e.getActionCommand();
    if (command.equals("Exit"))
      System.exit(0);
  }

  public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED)
      setTitle("Checked!");
    else
      setTitle("Not checked!");
  }

  public static void main(String args[]) {
    TestMenuEvent f = new TestMenuEvent("Tratare evenimente
        meniuri");
    f.show();
  }
}
```

Meniuri de context (popup)

```
PopupMenu popup = new PopupMenu("Options");
popup.add(new MenuItem("New"));
popup.add(new MenuItem("Edit"));
popup.addSeparator();
popup.add(new MenuItem("Exit"));
...
popup.show(Component origine, int x, int y)
fereastra.add(popup1);
...
fereastra.remove(popup1);
fereastra.add(popup2);
```

isPopupTrigger() :

- dacă acest eveniment de mouse este evenimentul care poate afişa un meniu popup
- trebuie testat în ambele metode mousePressed şi mouseReleased pentru că depinde de platformă

Folosirea unui meniu de context (popup)

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

class Fereastra extends Frame implements ActionListener{
  // Definim meniul popup al ferestrei
  private PopupMenu popup;
  // Pozitia meniului va fi relativa la fereastra
  private Component origin;
  public Fereastra(String titlu) {
    super(titlu);
    origin = this;

    this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });

    this.addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
        if (e.isPopupTrigger())
          popup.show(origin, e.getX(), e.getY());
      }
      public void mouseReleased(MouseEvent e) {
        if (e.isPopupTrigger())
          popup.show(origin, e.getX(), e.getY());
      }
    });
    setSize(300, 300);
```

Folosirea unui meniu de context (popup) (2)

```
    // Cream meniul popup
    popup = new PopupMenu("Options");
    popup.add(new MenuItem("New"));
    popup.add(new MenuItem("Edit"));
    popup.addSeparator();
    popup.add(new MenuItem("Exit"));
    add(popup); //atasam meniul popup ferestrei
    popup.addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.equals("Exit"))
          System.exit(0);
    }
  }

  public class TestPopupMenu {
    public static void main(String args[]) {
      Fereastra f = new Fereastra("PopupMenu");
      f.show();
    }
  }
```

Acceleratori

Clasa MenuShortcut

Ctrl + Tasta sau Ctrl + Shift + Tasta

```
// Ctrl+O
new MenuItem("Open",new MenuShortcut(
    KeyEvent.VK_O));

// Ctrl+P
new MenuItem("Print",new MenuShortcut('p'));

// Ctrl+Shift+P
new MenuItem("Preview",new MenuShortcut('p'),
    true);
```

Interfața grafică cu utilizatorul - Swing

Programare Orientată pe Obiecte

Swing

- JFC (Java Foundation Classes)
- Componentele Swing
- Asemănări şi deosebiri cu AWT
- Folosirea ferestrelor
- Ferestre interne
- Folosirea componentelor
- Containere
- Look and Feel
- Arhitectura modelului Swing
- Folosirea modelelor
- Tratarea evenimentelor

JFC (Java Foundation Classes)

- Componente Swing

  Sunt componente ce înlocuiesc şi în acelaşi timp extind vechiul set oferit de modelul AWT.
- Look-and-Feel

  Permite schimbarea înfăţişării şi a modului de interacţiune cu aplicaţia în funcţie de preferinţe
- Accessibility API

  Permite dezvoltarea de aplicaţii care să comunice cu dispozitive utilizate de către persoane cu diverse tipuri de handicap.
- Java 2D API

  Permite crearea de aplicaţii care utilizează grafică la un nivel avansat. Clasele puse la dispoziţie permit crearea de desene complexe, efectuarea de operaţii geometrice (rotiri, scalări, translaţii, etc.), prelucrarea de imagini, tipărire, etc.
- Drag-and-Drop

  Oferă posibilitatea de a efectua operaţii drag-and-drop între aplicaţii Java şi aplicaţii native.
- Internaţionalizare

  Permite dezvoltarea de aplicaţii care să poată fi configurate pentru exploatarea lor în diverse zone ale globului, utilizând limba şi particularităţile legate de formatarea datei, numerelor sau a monedei din zona respectivă.

Swing API

- **javax.accessibility**
- **javax.swing.plaf**
- **javax.swing.text.html**
- **javax.swing**
- **javax.swing.plaf.basic**
- **javax.swing.text.parser**
- **javax.swing.border**
- **javax.swing.plaf.metal**
- **javax.swing.text.rtf**
- **javax.swing.colorchooser**
- **javax.swing.plaf.multi**
- **javax.swing.tree**
- **javax.swing.event**
- **javax.swing.table**
- **javax.swing.undo**
- **javax.swing.filechooser**
- **javax.swing.text**

Cel mai important: **javax.swing**

Componentele Swing

- **Componente atomice**
  JLabel, JButton, JToggleButton (JCheckBox, JRadioButton), JScrollBar, JSlider, JProgressBar, JSeparator
- **Componente complexe**
  JTable, JTree, JComboBox, JSpinner, JList, JFileChooser, JColorChooser, JOptionPane
- **Componente pentru editare de text**
  JTextField, JFormattedTextField, JPasswordField, JTextArea, JEditorPane, JTextPane
- **Meniuri**
  JMenuBar, JMenu, JPopupMenu, JMenuItem, JCheckboxMenuItem, JRadioButtonMenuItem
- **Containere intermediare**
  JPanel, JScrollPane, JSplitPane, JTabbedPane, JDesktopPane, JToolBar
- **Containere de nivel înalt**
  JFrame, JDialog, JWindow, JInternalFrame, JApplet

Asemănări şi deosebiri cu AWT

- Tehnologia Swing extinde AWT.

```
import javax.swing.*;
import java.awt.*; //Font, Color, ...
import java.awt.event.*;
```

- Convenția ”J”

java.awt.Button - javax.swing.JButton

java.awt.Label - javax.swing.JLabel,etc.

- Noi gestionari de poziționare: BoxLayout, SpringLayout
- Folosirea HTML

```
JButton simplu = new JButton("Text simplu");
JButton html = new JButton(
"<html><u>Text</u> <i>formatat</i></html>");
```

O aplicație simplă AWT

```
import java . awt .*;
import java . awt. event .*;
public class ExempluAWT extends Frame implements
                                                    ActionListener {
    public ExempluAWT ( String titlu ) {
            super ( titlu );
            setLayout (new FlowLayout ());
            add (new Label (" Hello AWT "));
            Button b = new Button (" Close ");
            b. addActionListener ( this );
            add (b);
            pack ();
            show ();
    }
    public void actionPerformed ( ActionEvent e) {
            System . exit (0);
    }
    public static void main ( String args []) {
            new ExempluAWT (" Hello ");
    }
}
```

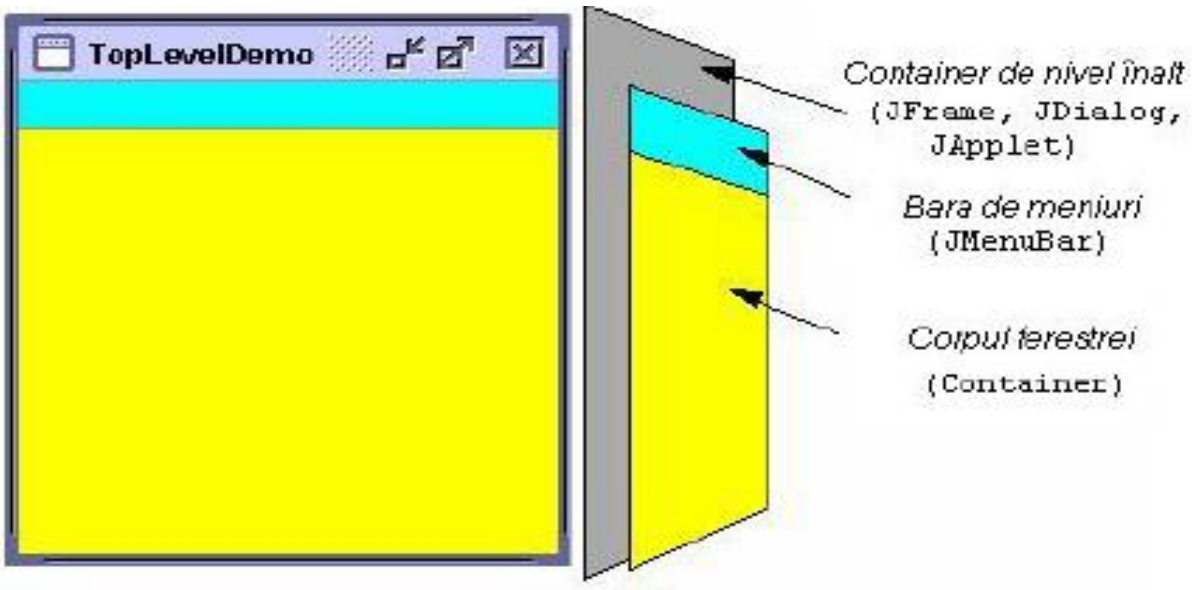
Aplicația rescrisă folosind Swing–1.4

```
import javax . swing .*;
import java . awt .*;
import java . awt. event .*;
class ExempluSwing extends JFrame implements ActionListener {
        public ExempluSwing ( String titlu ) {
                super ( titlu );
                // Metoda setLayout nu se aplica direct ferestrei
                getContentPane (). setLayout (new FlowLayout ());
                // Componentele au denumiri ce incep cu litera J
                getContentPane ().add( new JLabel ("Swing"));
                JButton b = new JButton ("Close");
                b. addActionListener ( this );
                // Metoda add nu se aplica direct ferestrei
                getContentPane ().add(b);
                pack ();
                show ();
        }
        public void actionPerformed ( ActionEvent e) {
                // Tratarea evenimentelor se face ca in AWT
                System . exit (0);
        }
        public static void main ( String args []) {
                new ExempluSwing (" Hello ");
        }
}
```

Aplicația rescrisă folosind Swing–1.5

```
import javax . swing .*;
import java . awt .*;
import java . awt. event .*;
class ExempluSwing extends JFrame implements ActionListener
    {
    public ExempluSwing ( String titlu ) {
            super ( titlu );
            setLayout (new FlowLayout ());
            add( new JLabel ("Swing"));
            JButton b = new JButton ("Close");
            b. addActionListener ( this );
            add(b);
            pack ();
            show ();
    }
    public void actionPerformed ( ActionEvent e) {
            System . exit (0);
    }
    public static void main ( String args []) {
            new ExempluSwing (" Hello ");
    }
}
```

Folosirea ferestrelor




```
Frame f = new Frame();
f.setLayout(new FlowLayout());
f.add(new Button("OK"));
```

```
JFrame jf = new JFrame();
jf.getContentPane().setLayout(new FlowLayout());
jf.getContentPane().add(new JButton("OK"));
jf.setDefaultCloseOperation(WindowConstants.HIDE_ON_CLOSE);
```

- WindowConstants.DO_NOTHING_ON_CLOSE
- JFrame.EXIT_ON_CLOSE

Look and Feel

- Modul în care sunt desenate componentele Swing şi felul în care acestea interacționează cu utilizatorul
- Acelaşi program poate utiliza diverse moduri Look-and-Feel, cum ar fi cele standard Windows, Mac, Java, Motif sau altele oferite de diverşi dezvoltatori

• javax.swing.plaf.metal.MetalLookAndFeel

Varianta implicită de L&F şi are un aspect specific Java.

• com.sun.java.swing.plaf.windows.WindowsLookAndFeel

Varianta specifică sistemelor de operare Windows. Incepând cu versiunea 1.4.2 există şi implementarea pentru Windows XP .

• com.sun.java.swing.plaf.mac.MacLookAndFeel

Varianta specifică sistemelor de operare Mac.

• com.sun.java.swing.plaf.motif.MotifLookAndFeel

Specifică interfața CDE/Motif.

• com.sun.java.swing.plaf.gtk.GTKLookAndFeel

GTK+ reprezintă un standard de creare a interfețelor grafice dezvoltat independent de limbajul Java. (GTK este acronimul de la GNU ImageManipulation Program Toolkit).

Specificare unei anumite interfețe L&F

- **Folosirea clasei UIManager (metode statice):**

- getLookAndFeel - Obține varianta curentă, returnând un obiect de tip LookAndFeel.
- setLookAndFeel - Setează modul curet L&F. Metoda primeşte ca argument un obiect dintr-o clasă derivată din LookAndFeel, fie un şir de caractere cu numele complet al clasei L&F.
- getSystemLookAndFeelClassName - Obține variantă specifică sistemului de operare folosit. In cazul în care nu există nici o astfel de clasă, returnează varianta standard.
- getCrossPlatformLookAndFeelClassName - Returnează interfața grafică standard Java (JLF).

// Exemple:

```
UIManager.setLookAndFeel(
    "com.sun.java.swing.plaf.motif.MotifLookAndFeel“ );
UIManager.setLookAndFeel (
    UIManager.getSystemLookAndFeelClassName() );
```

Specificare unei anumite interfețe L&F

- **Setarea proprietății swing.defaultlaf**

1. direct de la linia de comandă prin setarea proprietății swing.defaultlaf:

   java -Dswing.defaultlaf = com.sun.java.swing.plaf.gtk.GTKLookAndFeel App

2. În lib/swing.properties:

   # Swing properties

   swing.defaultlaf = com.sun.java.swing.plaf.windows.WindowsLookAndFeel

- Există posibilitatea de a schimba varianta de L&F chiar şi după afişarea componentelor. Acesta este un proces care trebuie să actualizeze ierarhiile de componente:

```
UIManager.setLookAndFeel(numeClasaLF);
SwingUtilities.updateComponentTreeUI(f);
f.pack();
```