Interfaţa grafică cu utilizatorul

- **Modelul AWT**
- **Componentele AWT**
- **Gestionarea poziţionării**
- **Gruparea componentelor**
- **Tratarea evenimentelor**
- **Tipuri de evenimente**
- **Adaptori şi clase anonime**
- **Folosirea ferestrelor**
- **Folosirea meniurilor**
- **Tratarea evenimentelor generate de meniuri**

Interfaţa cu utilizatorul

Human Machine Interface = Modalităţile prin care un sistem (software) interacţionează cu utilizatorii săi (umani).

- **Linia de comandă**
- **Grafice**
  (Graphical User Interface - GUI)
- **Tactile**
  (Touch User Interface - TUI)
- **Multimedia** (voce, animaţie, multi-screen, zooming)
- **Inteligente** (recunoaşterea gesturilor, conversaţionale, etc)
- etc.

GUI

Comunicarea *vizuală* **între un program şi utilizatori**.

- **AWT** (Abstract Windowing Toolkit)
- **Swing** - parte din **JFC** (Java Foundation Classes) Sun, Netscape şi IBM

Etapele creării unei aplicaţii:

- **Design**
  - Crearea unei suprafete de afişare
  - Crearea şi asezarea componentelor
- **Funcţionalitate**
  - Definirea unor acţiuni
  - "Ascultarea" evenimentelor

Modelul AWT

Pachete:
java.awt, **java.awt.event**

Obiectele grafice:
Component, **MenuComponent**.

Suprafeţe de afişare:
Container

Gestionari de poziţionare:
LayoutManager

Evenimente:
AWTEvent

Listing 1: O fereastră cu două butoane

```
import java.awt.*;
public class ExempluAWT1 {
  public static void main(String args[]) {

    // Crearea ferestrei - un obiect de tip Frame
    Frame f = new Frame("O fereastra");

    // Setarea modului de dipunere a componentelor
    f.setLayout(new FlowLayout());

    // Crearea celor doua butoane
    Button b1 = new Button("OK");
    Button b2 = new Button("Cancel");

    // Adaugarea butoanelor
    f.add(b1);
    f.add(b2);
    f.pack();

    // Afisarea fereastrei
    f.show();
  }
}
```

Componentele AWT

- **Button**
- **Canvas**
- **Checkbox, CheckBoxGroup;**
- **Choice**
- **Container**
- **Label**
- **List**
- **Scrollbar**
- **TextComponent**
  TextField
  TextArea

Proprietăţi comune

- **Poziţie**
  `getLocation, getX, getY, getLocationOnScreen`
  `setLocation, setX, setY`
- **Dimensiuni**
  `getSize, getHeight, getWidth`
  `setSize, setHeight, setWidth`
- **Dimensiuni şi poziţie**
  `getBounds`
  `setBounds`
- **Culoare** (text şi fundal)
  `getForeground, getBackground`
  `setForeground, setBackground`
- **Font**
  `getFont`
  `setFont`
- **Vizibilitate**
  `setVisible`
  `isVisible`
- **Interactivitate**
  `setEnabled`
  `isEnabled`

Suprafeţe de afişare

Container

- **Window**
  - **Frame** - ferestre standard
  - **Dialog** - ferestre de dialog
- **Panel**, **Applet**
- **ScrollPane** - derulare

Metode comune

- **add**
- **remove**
- **setLayout**
- **getInsets**
- **validate**

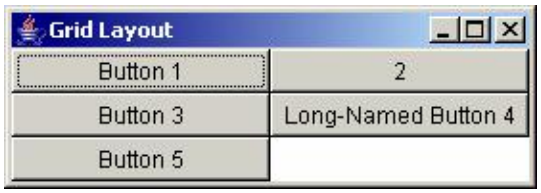
Exemplu

```
Frame f = new Frame("O fereastra");

// Adaugam un buton direct pe fereastra
Button b = new Button("Hello");
f.add(b);

// Adaugam doua componente pe un panel
Label et = new Label("Nume:");
TextField text = new TextField();

Panel panel = new Panel();
panel.add(et);
panel.add(text);

// Adaugam panel-ul pe fereastra
// si, indirect, cele doua componente
f.add(panel);
```

Gestionarea poziţionării

Listing 2: Poziţionarea a 5 butoane

```
import java.awt.*;
public class TestLayout {
  public static void main(String args[]) {
    Frame f = new Frame("Grid Layout");
    f.setLayout(new GridLayout(3, 2));  //*

    Button b1 = new Button("Button 1");
    Button b2 = new Button("2");
    Button b3 = new Button("Button 3");
    Button b4 = new Button("Long-Named Button 4");
    Button b5 = new Button("Button 5");

    f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
    f.pack();
    f.show();
  }
}
```




```
Frame f = new Frame("Flow Layout");
f.setLayout(new FlowLayout());
```

Un **gestionar de poziţionare (layout manager)** este un obiect care controlează dimensiunea şi aranjarea (poziţia) componentelor unui container.

Fiecare obiect de tip `Container` are asociat un gestionar de poziţionare.

Toate clasele care instanţiaza obiecte pentru gestionarea poziţionării implementează interfaţă **LayoutManager**.

La instanţierea unui container se creează implicit un gestionar de poziţionare asociat acestuia. (ferestre: `BorderLayout`, panel-uri: `FlowLayout`).

Folosirea gestionarilor de poziţionare

Gestionari AWT

`FlowLayout`, `BorderLayout`, `GridLayout`, `CardLayout`, `GridBagLayout`

Metoda **setLayout**

```
FlowLayout gestionar = new FlowLayout();
container.setLayout(gestionar); // sau:
container.setLayout(new FlowLayout());
```

Dimensionarea

`getPreferredSize, getMinimumSize, getMaximumSize`

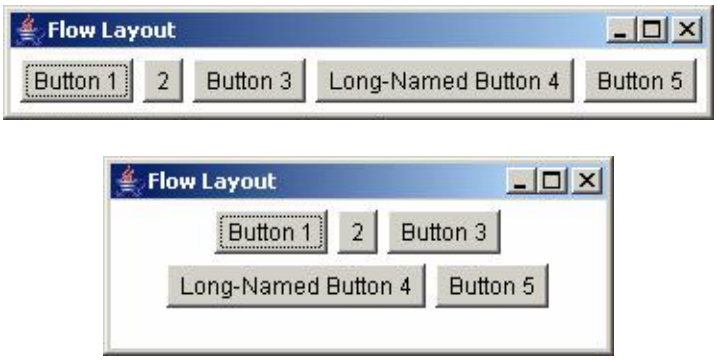
Poziţionarea absolută

```
container.setLayout(null);
Button b = new Button("Buton");
b.setSize(10, 10); b.setLocation (0, 0);
b.add();
```

Gestionarul FlowLayout

Listing 3: Gestionarul `FlowLayout`

```
import java.awt.*;
public class TestFlowLayout {
  public static void main(String args[]) {
    Frame f = new Frame("Flow Layout");
    f.setLayout(new FlowLayout());

    Button b1 = new Button("Button 1");
    Button b2 = new Button("2");
    Button b3 = new Button("Button 3");
    Button b4 = new Button("Long-Named Button 4");
    Button b5 = new Button("Button 5");

    f.add(b1); f.add(b2); f.add(b3); f.add(b4); f.add(b5);
    f.pack();
    f.show();
  }
}
```



Gestionar implicit pentru `Panel`.

Gestionarul BorderLayout

Cinci regiuni: `NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`

Listing 4: Gestionarul `BorderLayout`

```
import java.awt.*;
public class TestBorderLayout {
  public static void main(String args[]) {
    Frame f = new Frame("Border Layout");
    // Apelul de mai jos poate sa lipseasca
    f.setLayout(new BorderLayout());

    f.add(new Button("Nord"), BorderLayout.NORTH);
    f.add(new Button("Sud"), BorderLayout.SOUTH);
    f.add(new Button("Est"), BorderLayout.EAST);
    f.add(new Button("Vest"), BorderLayout.WEST);
    f.add(new Button("Centru"), BorderLayout.CENTER);
    f.pack();

    f.show();
  }
}
```

Gestionar implicit pentru `Window`.

Gestionarul GridLayout

Organizare tabelară

Listing 5: Gestionarul `GridLayout`

```
import java.awt.*;
public class TestGridLayout {
  public static void main(String args[]) {
    Frame f = new Frame("Grid Layout");
    f.setLayout(new GridLayout(3, 2));

    f.add(new Button("1"));
    f.add(new Button("2"));
    f.add(new Button("3"));
    f.add(new Button("4"));
    f.add(new Button("5"));
    f.add(new Button("6"));

    f.pack();
    f.show();
  }
}
```

Gestionarul CardLayout

Pachet de cărţi

Prima ”carte” este vizibilă

A doua ”carte” este vizibilă

Swing: `JTabbedPane`

Gestionarul GridBagLayout

```
GridBagLayout gridBag = new GridBagLayout();
container.setLayout(gridBag);
GridBagConstraints c = new GridBagConstraints();
//Specificam restrictiile
. . .
gridBag.setConstraints(componenta, c);
container.add(componenta);
```

- `gridx, gridy`
- `gridwidth, gridheight`
- `fill`
- `insets`
- `anchor`
- `weigthx, weighty`

Gruparea componentelor

Gruparea componentelor se face folosind clasa **Panel**.

Aranjare eficientă a componentelor unei ferestre presupune:

- gruparea componentelor "înfrăţite";
- aranjarea componentelor unui panel;
- aranjarea panel-urilor pe suprafaţa ferestre.

Gestionarul implicit este `FlowLayout`.

Listing 6: Gruparea componentelor

```
import java.awt.*;
public class TestPanel {
  public static void main(String args[]) {
    Frame f = new Frame("Test Panel");

    Panel intro = new Panel();
    intro.setLayout(new GridLayout(1, 3));
    intro.add(new Label("Text:"));
    intro.add(new TextField("", 20));
    intro.add(new Button("Adaugare"));

    Panel lista = new Panel();
    lista.setLayout(new FlowLayout());
    lista.add(new List(10));
    lista.add(new Button("Stergere"));

    Panel control = new Panel();
    control.add(new Button("Salvare"));
    control.add(new Button("Iesire"));

    f.add(intro, BorderLayout.NORTH);
    f.add(lista, BorderLayout.CENTER);
    f.add(control, BorderLayout.SOUTH);

    f.pack();
    f.show();
  }
}
```

Interactţiunea cu utilizatorul

Event-Driven Programming

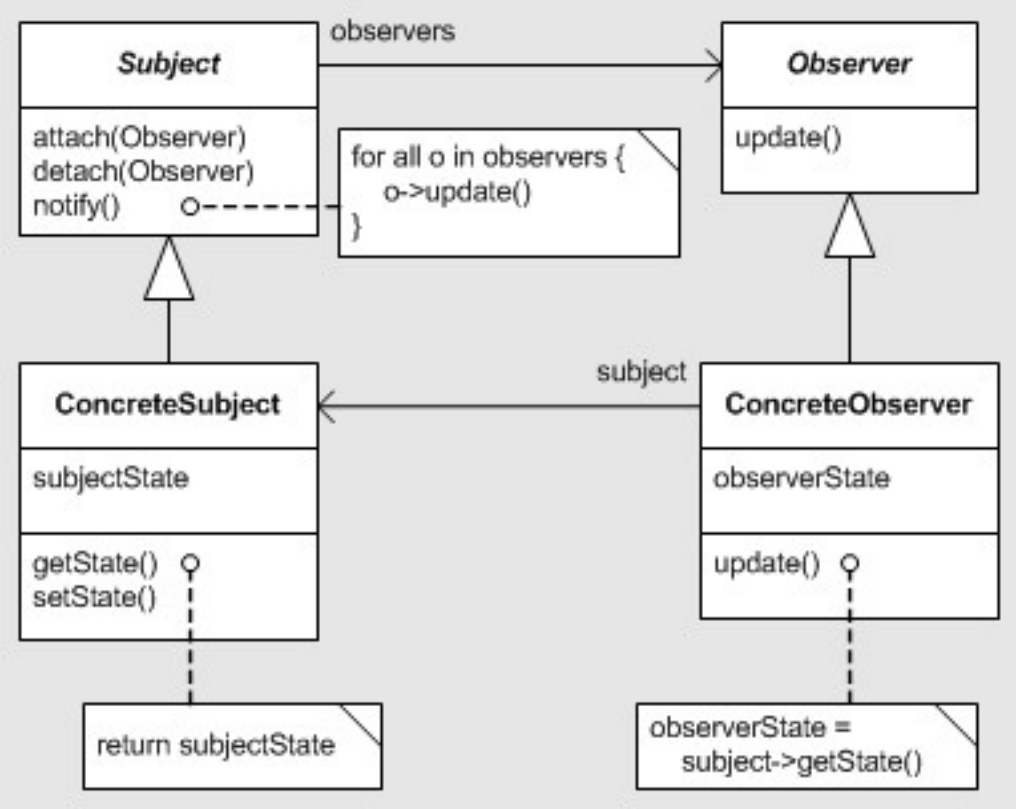
Eveniment: apăsarea unui buton, modificarea textului, închiderea unei ferestre, etc.

Sursă: componentă care generează un eveniment.

Listener: interceptarea evenimentelor (ascultător, consumator de evenimente).

Observer Design Pattern

Observarea stării unei entităţi în cadrul unui sistem (*Publish-Subscribe*)



Evenimente - **AWTEvent**:
`ActionEvent`, `TextEvent`

Observatori - **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
    ...
  }
}
```

Inregistrarea/eliminare unei clase în lista ascultătorilor unei componente: **add***TipEveniment***Listener**, **remove***TipEveniment***Listener**.

Tratarea evenimentelor se desfăşoară astfel:

- Componentele generează evenimente când ceva "interesant" se întâmplă;
- Sursele evenimentelor permit oricărei clase să "asculte" evenimentele sale prin metode de tip `addXXXListener`;
- O clasă care ascultă evenimente trebuie să implementeze interfeţe specifice fiecărui tip de eveniment - acestea descriu metode ce vor fi apelate automat la apariţia evenimentelor.

Exemplu

Listing 7: Ascultarea evenimentelor a două butoane

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

class Fereastra extends Frame {
  public Fereastra(String titlu) {
    super(titlu);
    setLayout(new FlowLayout());
    setSize(200, 100);
    Button b1 = new Button("OK");
    Button b2 = new Button("Cancel");
    add(b1);
    add(b2);

    Ascultator listener = new Ascultator(this);
    b1.addActionListener(listener);
    b2.addActionListener(listener);
   // Ambele butoane sunt ascultate de obiectul listener,
   // instanta a clasei Ascultator, definita mai jos
  }
}

class Ascultator implements ActionListener {
  private Fereastra f;
  public Ascultator(Fereastra f) {
    this.f = f;
  }

  // Metoda interfetei ActionListener
  public void actionPerformed(ActionEvent e) {
   f.setTitle("Ati apasat " + e.getActionCommand());
  }
}

public class TestEvent1 {
  public static void main(String args[]) {
    Fereastra f = new Fereastra("Test Event");
    f.show();
  }
}
```

Listing 8: Tratarea evenimentelor în fereastră

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

class Fereastra extends Frame implements ActionListener {
  Button ok = new Button("OK");
  Button exit = new Button("Exit");
  int n=0;

  public Fereastra(String titlu) {
    super(titlu);
    setLayout(new FlowLayout());
    setSize(200, 100);
    add(ok);
    add(exit);

    ok.addActionListener(this);
    exit.addActionListener(this);
    // Ambele butoane sunt ascultate in clasa Fereastra
    // deci ascultatorul este instanta curenta: this
  }

  // Metoda interfetei ActionListener
  public void actionPerformed(ActionEvent e) {

    if (e.getSource() == exit)
        System.exit(0); // Terminam aplicatia

    if (e.getSource() == ok) {
        n ++;
        this.setTitle("Ati apasat OK de " + n + " ori");
    }
  }
}

public class TestEvent2 {
  public static void main(String args[]) {
    Fereastra f = new Fereastra("Test Event");
    f.show();
  }
}
```

Tipuri de evenimente

Evenimente de nivel jos - apăsare de tastă, mişcarea mouse-ului, etc.

`ComponentEvent`	Ascundere, deplasare, redimensionare, afişare
`ContainerEvent`	Adăugare pe container, eliminare
`FocusEvent`	Obţinere, pierdere foucs
`KeyEvent`	Apăsare, eliberare taste, tastare
`MouseEvent`	Operaţiuni cu mouse-ul: click, drag, etc.
`WindowEvent`	Operaţiuni asupra ferestrelor: minimizare, maximizare,etc.

Evenimente semantice - interacţiunea cu o componentă GUI: apăsarea unui buton, selectarea unui articol dintr-o listă, etc.

`ActionEvent`	Acţionare
`AdjustmentEvent`	Ajustarea unei valori
`ItemEvent`	Schimbarea stării
`TextEvent`	Schimbarea textului

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

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

```
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'
    }
    ...
  }
}
```

Adaptori şi a clase anonime

Listing 9: Implementarea interfeţei `WindowListener`

```
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 {
  // Suprdefinim metodele care ne intereseaza
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}
```

XXXListener - XXXAdapter

Folosirea unei clasa anonime

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

Folosirea ferestrelor

Window: Frame, Dialog

Clasa Window

Crearea de ferestre care nu au chenar
şi nici bară de meniuri; nu interacţionează
cu utilizatorul ci doar oferă anumite informaţii.

Metode

- `show`
- `hide`
- `isShowing`
- `dispose`
- `pack`

Clasa Frame

```
import java.awt.*;
public class TestFrame {
  public static void main(String args[]) {
    Frame f = new Frame("Titlul ferestrei");
    f.show();
  }
}
```

```
import java.awt.*;
class Fereastra extends Frame{
  // Constructorul
  public Fereastra(String titlu) {
    super(titlu);
    ...
  }
}
```

```
    ...
    Fereastra f = new Fereastra("Titlul ferestrei");
    f.show();
```

Gestionar de poziţionare: `BorderLayout`.

Structura generală a unei ferestre

Listing 10: 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) {
```

```
    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

Clasa Dialog

Uzual, ferestrele de dialog au un părinte.

Ferestrele de dialog pot fi:

- **modale**
- **nemodale**

Constructori

```
Dialog(Frame parinte)
Dialog(Frame parinte, String titlu)
Dialog(Frame parinte, String titlu, boolean modala)
Dialog(Frame parinte, boolean modala)
Dialog(Dialog parinte)
Dialog(Dialog parinte, String titlu)
Dialog(Dialog parinte, String titlu, boolean modala)
```

Clasa FileDialog

Selectarea unui nume de fişier

Constructori

```
FileDialog(Frame parinte)
FileDialog(Frame parinte, String titlu)
FileDialog(Frame parinte, String titlu, boolean mod)
```

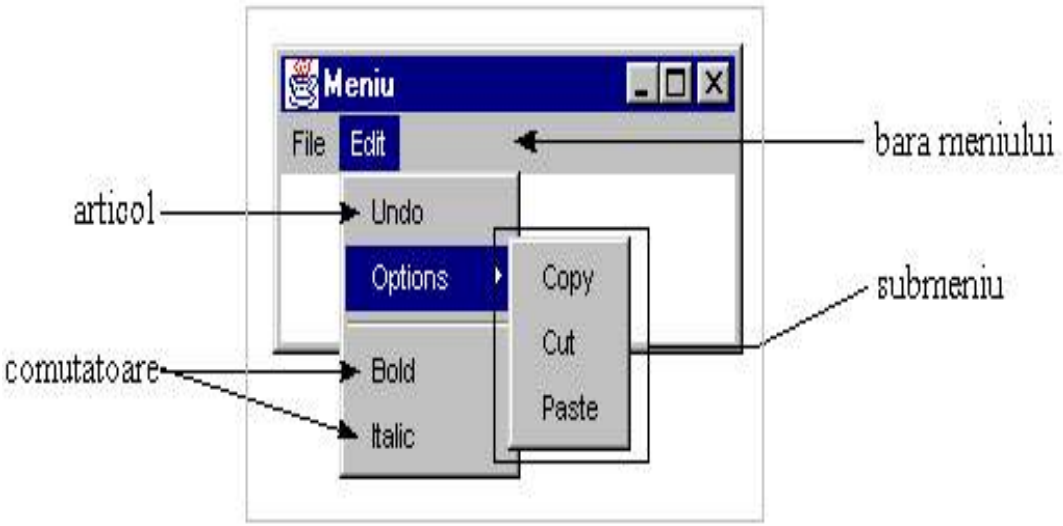
```
// Dialog pentru incarcarea unui fisier
new FileDialog(parinte, "Alegere fisier",
  FileDialog.LOAD);

// Dialog pentru salvarea unui fisier
new FileDialog(parinte, "Salvare fisier",
  FileDialog.SAVE);
```

Swing: `JFileChooser`

Folosirea meniurilor

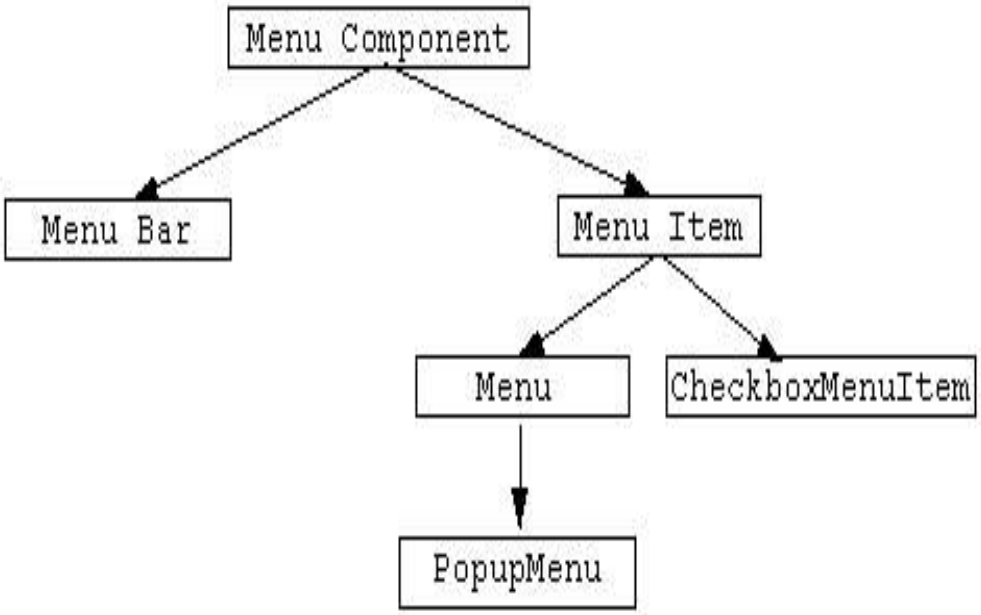
Componentele unui meniu subt derivate din **MenuComponent**.

- **Meniuri fixe** (vizibile permanent)
- **Meniuri de context** (popup)



addMenuBar

Ierarhia claselor ce descriu meniuri



Exemplu

Listing 11: Crearea unui meniu

```
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

Listing 12: Tratarea evenimentelor unui meniu

```
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();

    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);
```

Listing 13: 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);

    // 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();
  }
}
```

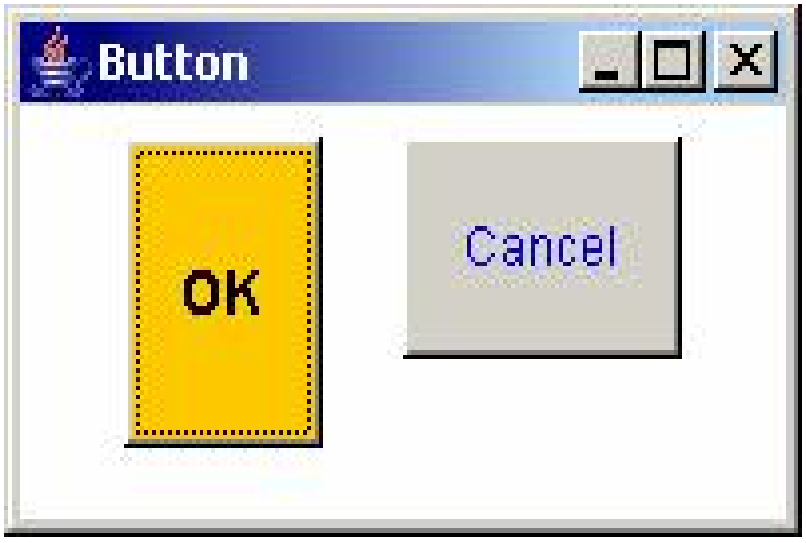
Acceleratori (Clasa MenuShortcut)

`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);
```

Clasa Label

Clasa Button



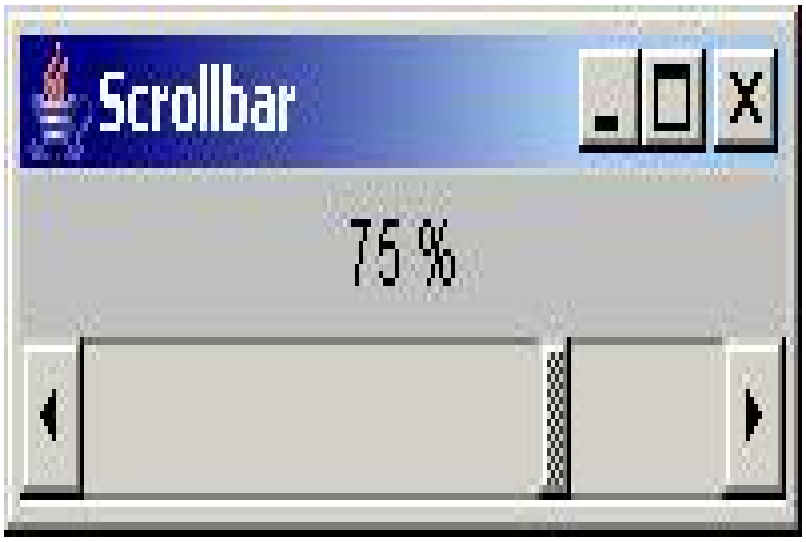
Clasa Checkbox

Clasa CheckboxGroup

Clasa Choice

Clasa List

Clasa ScrollBar



Clasa ScrollPane

Clasa TextField

Clasa TextArea