Interfața grafică cu utilizatorul - AWT

Programare Orientată pe Obiecte

Interfața grafică cu utilizatorul

- Modelul AWT
- Componentele AWT
- Gestionarea poziționării
- Gruparea componentelor
- Tratarea evenimentelor
- Tipuri de evenimente
- Adaptori şi clase anonime
- Folosirea ferestrelor

GUI – Interfața grafică cu utilizatorul

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 aplicații:

• Design

– Crearea unei suprafete de afişare

– Crearea şi asezarea componentelor

• Funcționalitate

– Definirea unor acțiuni

– ”Ascultarea” evenimentelor

Modelul AWT

- Pachete:
  java.awt, java.awt.event
- Obiectele grafice:
  Component, MenuComponent.
- Suprafețe de afişare:
  Container
- Gestionari de poziționare:
  LayoutManager
- Evenimente:
  AWTEvent

Exemplu: 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

- orice obiect care poate avea o reprezentare grafică şi care poate interacționa cu utilizatorul.
- Button
- Canvas
- Checkbox, CheckBoxGroup;
- Choice
- Container
- Label
- List
- Scrollbar
- TextComponent
  - TextField
  - TextArea

Componente grafice

Clasa Label

Clasa Button

Componente grafice

Clasa Checkbox

Clasa CheckboxGroup

Componente grafice

Clasa Choice

Clasa List

Componente grafice

Clasa ScrollBar

Clasa ScrollPane

Componente grafice

Clasa TextField

Clasa TextArea

Proprietăți comune

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

Suprafețe de afişare

- un container este folosit pentru a adăuga componente pe suprafața lui
- Container – superclasa pentru suprafețe de afişare
  - 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 poziționării

Exemplu: Poziționarea 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 ();
        }
}
```

Gestionarea poziționării


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

Gestionarea poziționării (2)

- Un gestionar de poziționare (layout manager) este un obiect care controlează dimensiunea şi aranjarea (poziția) componentelor unui container.
- Fiecare obiect de tip Container are asociat un gestionar de poziționare.
- Toate clasele care instanțiază obiecte pentru gestionarea poziționării implementează interfața LayoutManager.
- La instanțierea unui container se creează implicit un gestionar de poziționare asociat acestuia. (ferestre: BorderLayout, panel-uri: FlowLayout).

Folosirea gestionarilor de poziționare

- 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
- Poziționarea absolută

  container.setLayout(null);

  Button b = new Button("Buton");

  b.setSize(10, 10); b.setLocation (0, 0);

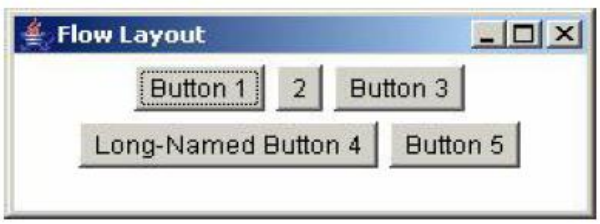
  b.add();

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

Implicit pentru Window

```
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();
  }
}
```

Cele cinci butoane ale ferestrei vor fi afişate astfel:

Gestionarul GridLayout

Organizare tabelară

```
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 cărți

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. . .şi apoi
gridBag.setConstraints(componenta, c);
container.add(componenta);
```

Gestionarul GridBagLayout (2)

- Cele mai utilizate tipuri de constrângeri pot fi specificate prin intermediul următoarelor variabile din clasa GridBagConstraints:

- gridx, gridy - celula ce reprezintă colțul stânga sus al componentei;
- gridwidth, gridheight - numărul de celule pe linie şi coloană pe care va fi afişată componenta;
- fill - folosită pentru a specifica dacă o componentă va ocupa întreg spațiul pe care îl are destinat; valorile posibile sunt HORIZONTAL, VERTICAL, BOTH, NONE;
- insets - distanțele dintre componentă şi marginile suprafeței sale de afişare;
- anchor - folosită atunci când componenta este mai mică decât suprafața sa de afişare pentru a forța o anumită dispunere a sa: nord, sud, est, vest, etc.
- weigthx, weighty - folosite pentru distribuția spațiului liber; uzual au valoarea 1;

Gruparea componentelor

- Gruparea componentelor se face folosind clasa Panel.
- Un panel este cel mai simplu model de container.
- Nu are o reprezentare vizibilă, rolul său fiind de a oferi o suprafață de afişare pentru componente grafice, inclusiv pentru alte panel-uri.
- Aranjare eficientă a componentelor unei ferestre presupune:
  - gruparea componentelor ”înfrățite”;
  - aranjarea componentelor unui panel;
  - aranjarea panel-urilor pe suprafața ferestrei.
- Gestionarul implicit este FlowLayout.

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();
  }
}
```

Folosirea ferestrelor

- Window: Frame, Dialog
- Gestionar de poziționare: BorderLayout.
- Window: crearea de ferestre care nu au chenar şi nici bară de meniuri; nu interacționează cu utilizatorul ci doar oferă anumite informații.
- Metode:

• show - face vizibilă fereastra. Implicit, o fereastră nou creată nu este vizibilă;
• hide - face fereastra invizibilă fără a o distruge însă; pentru a redeveni vizibila se poate apela metoda show;
• isShowing -testează dacă fereastra este vizibilă sau nu;
• dispose - închide fereastra şi şi eliberează toate resursele acesteia;
• pack - redimensionează automat fereastra la o suprafața optimă care să cuprindă toate componentele sale; trebuie apelată în general după adăugarea tuturor componentelor pe suprafața ferestrei.

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();
```

Clasa Dialog

- o fereastră de dialog este dependentă de o altă fereastră (normală sau tot fereastră de dialog), numită şi fereastră părinte
- ferestrele de dialog pot fi:
  - modale: care blochează accesul la fereastra părinte în momentul deschiderii lor,
  - nemodale: care nu blochează fluxul de intrare către fereastra părinte

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

- fereastră de dialog folosită pentru selectarea unui nume de fişier în vederea încărcării sau salvării unui 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",

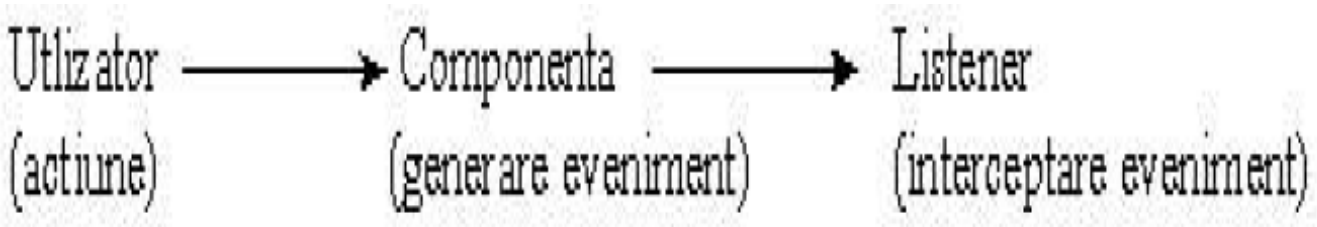
FileDialog.SAVE);

- Swing: JFileChooser

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:

- O clasă ascultător poate fi orice clasă care specifică în declarația sa că doreşte să asculte evenimente de un anumit tip.
- Acest lucru se realizează prin implementarea unei interfețe specifice fiecărui tip de eveniment.
- Pentru ascultarea evenimentelor de tip ActionEvent clasa respectivă trebuie să implementeze interfața ActionListener, pentru TextEvent interfață care trebuie implementată este TextListener,etc.
- Toate aceste interfețe sunt derivate din EventListener.
- Fiecare interfață defineşte una sau mai multe metode care vor fi apelate automat la apariția unui eveniment

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) {
    ...
    }
}
```

Ascultătorii (2):

- Inregistrarea/eliminare unei clase în lista ascultătorilor unei componente:
  - addTipEvenimentListener,
  - removeTipEvenimentListener.
- 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 addTIPListener;
• O clasă care ascultă evenimente trebuie să implementeze interfețe specifice fiecărui tip de eveniment – acestea descriu metode ce vor fi apelate automat la apariția evenimentelor.

Exemplu: 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 liste
   // 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();
  }
}
```

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

- interacțiunea 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

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.