Aplicatii JAVA

6

JAVA
Interfata grafica AWT (partea III)

Adrian Runceanu
www.runceanu.ro/adrian

Curs 6

Interfata grafica AWT (Abstract Window Toolkit) (partea a III-a)

6. Interfața grafică AWT(continuare)

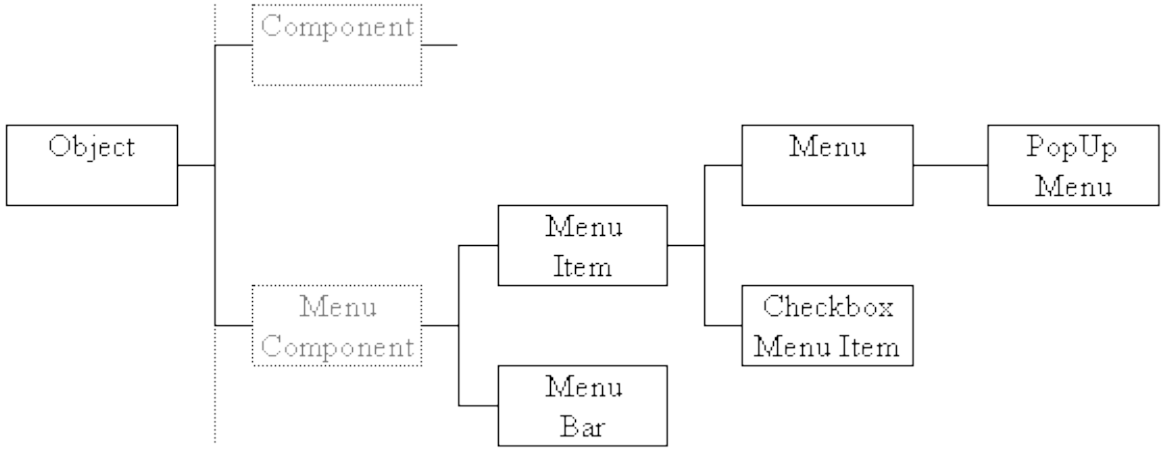
1. Meniuri în Java

2. Componente AWT(Abstract Window Toolkit)

Meniuri în Java

- Spre deosebire de celelalte obiecte grafice, care deriva din clasa Component, componentele unui meniu reprezinta instante ale unor clase derivate din superclasa abstracta MenuComponent.

- Aceasta exceptie este facuta deoarece multe platforme grafice limiteaza capabilitatile unui meniu.



Meniuri în Java

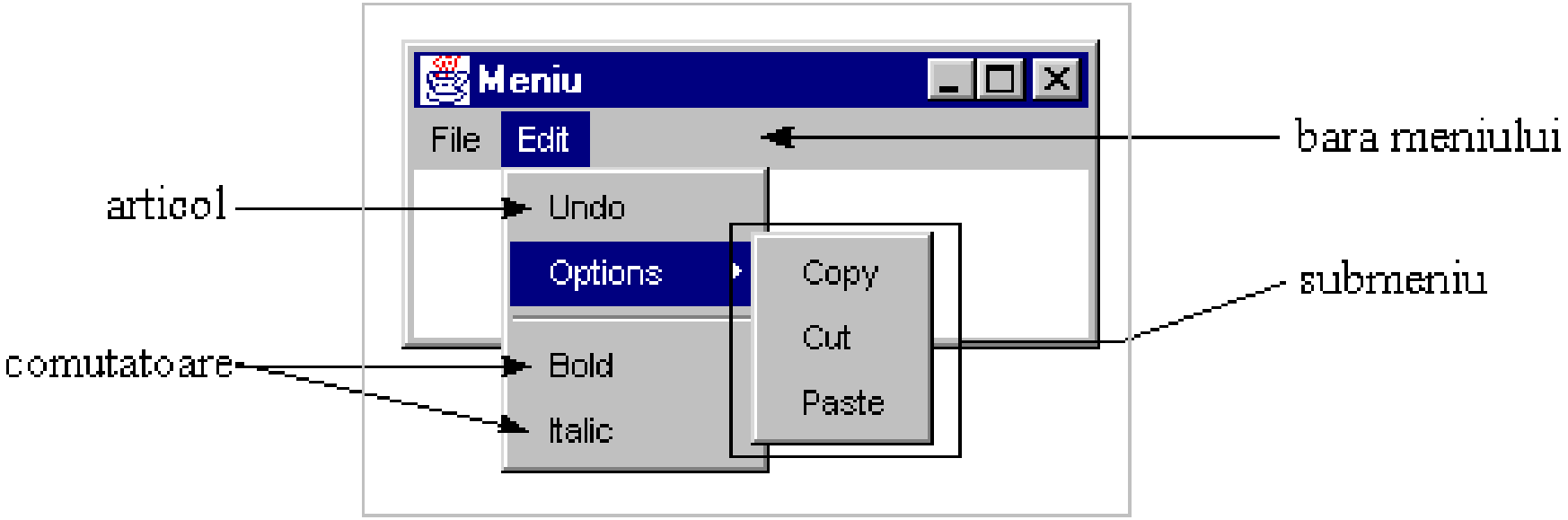
Meniurile sunt grupate în doua categorii:

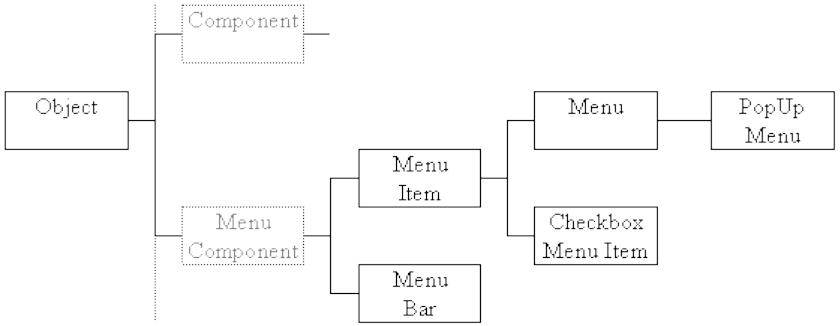
1. Meniuri fixe (vizibile permanent): sunt grupate într-o bara de meniuri ce contine câte un meniu pentru fiecare intrare a sa; la rândul lor aceste meniuri contin articole ce pot fi selectate, comutatoare - care au doua stari (checkbox) sau alte meniuri (submeniuri). O fereastra poate avea un singur meniu fix.

2. Meniuri de context (popup): sunt meniuri invizibile asociate unei ferestre si care se activeaza prin apasarea butonul drept al mouse-ului. Diferenta fata de meniurile fixe consta în faptul ca meniurile de context nu au bara de meniuri.

Meniuri în Java

- Exemplu de meniu fix care contine o bara de meniuri, doua meniuri principale File si Edit.
- Meniul Edit contine la rândul lui alt meniu (submeniu) Options, articolul Undo si doua comutatoare Bold si Italic.





Meniuri în Java

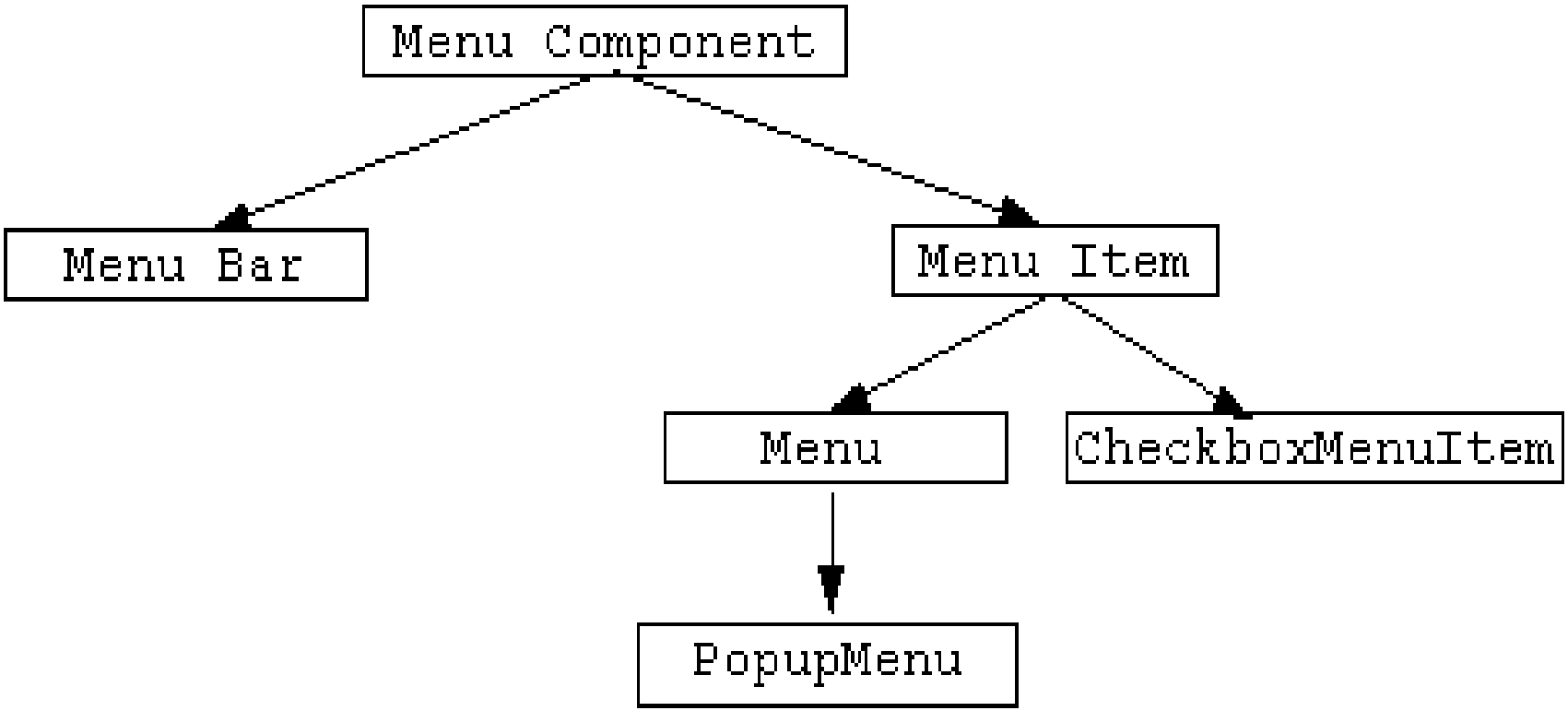
- In Java AWT meniurile sunt reprezentate ca instanțe al clasei **MenuBar**, aceasta fiind clasa care descrie barele de meniuri.

- Un obiect de tip **MenuBar** contine obiecte de tip Menu, care sunt de fapt meniurile derulante propriu-zise.

- La rândul lor acestea pot contine obiecte de tip MenuItem, CheckBoxMenuItem, dar si alte obiecte de tip Menu (submeniuri).

Meniuri în Java

Ierarhia claselor folosite în lucrul cu meniuri:



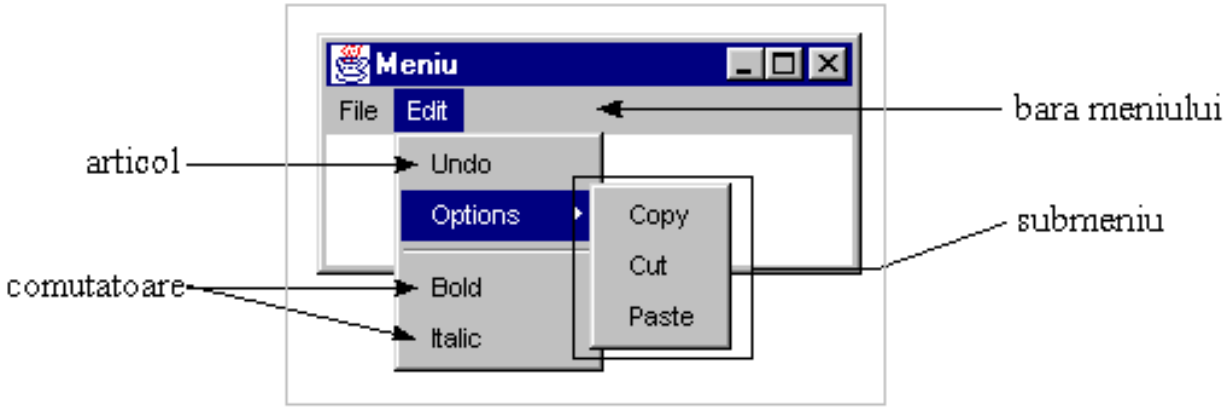
Meniuri în Java

- Orice componenta care ar trebui sa contina un meniu trebuie sa aiba implementata interfata **MenuContainer**.

- Cel mai adesea meniurile sunt atasate ferestrelor, mai precis obiectelor de tip Frame, aceste implementând interfata **MenuContainer**.

- Atasarea unei bare de meniuri la o fereastra se face prin metoda addMenuBar a clasei Frame.

Meniuri în Java



Codul sursa pentru meniul din imagine:

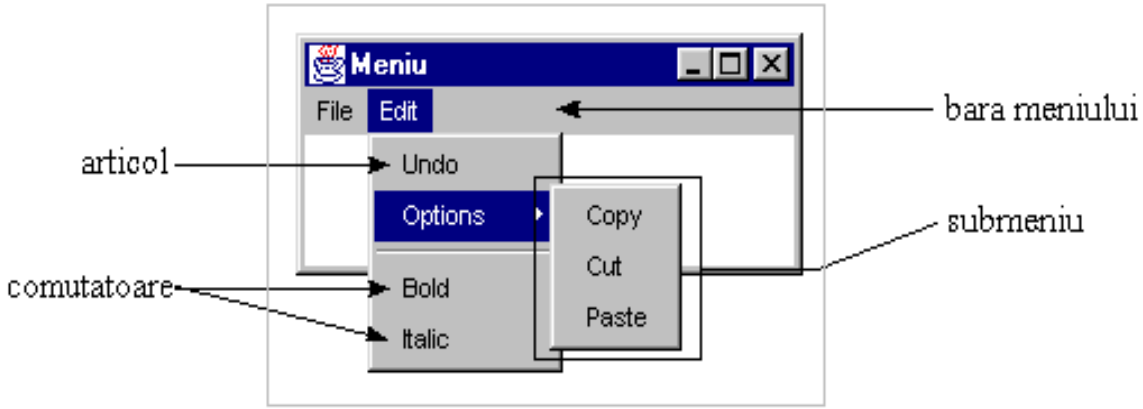
```
import java.awt.*;
import java.awt.event.*;
public class TestMenu {
public static void main(String args[]) {
        Frame f = new Frame("Meniu");
        MenuBar mb = new MenuBar();
```

Meniuri în Java

```
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);
```

Meniuri în Java

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

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

        f.setMenuBar(mb);
        f.show();
}
}
```



Meniuri în Java

Clasa **MenuComponent**

- Este o clasa abstracta, din care sunt extinse toate celelalte clase folosite pentru lucrul cu meniuri, fiind analoaga celeilalte superclase abstracte Component.
- Clasa MenuComponent contine metode de ordin general:
  - getName
  - setName
  - getFont
  - setFont

Meniuri în Java

Clasa **MenuBar**

- Permite crearea barelor de meniuri asociate unei ferestre cadru (Frame).
- Aceasta clasa adapteaza conceptul de bara de meniuri la platforma curenta de lucru.
- Pentru a lega bara de meniuri la o anumita fereastra trebuie apelata metoda setMenuBar din clasa Frame.

Meniuri în Java

Crearea unei bare de meniuri si legarea ei de o fereastra se realizeaza astfel:

```
// se creeaza bara de meniuri
MenuBar mb = new MenuBar();
// se adauga meniurile derulante la bara de meniuri
. . .
// se ataseaza unei ferestre bara de meniuri
Frame f = new Frame("Fereastra cu meniu");
f.addMenuBar(mb);
```

Meniuri în Java

Clasa **MenuItem**

- Orice articol al unui meniu trebuie sa fie o instanta a clasei MenuItem.
- Instantele acestei clase descriu asadar articolele (optiunile individuale) ale meniurilor derulante, cum sunt "Open", "Close", "Exit", etc.
- O instanta a clasei MenuItem reprezinta de fapt o eticheta ce descrie numele cu care va aparea articolul în meniu, însotita eventual de un *accelerator* (obiect de tip MenuShortcut) ce reprezinta combinatia de taste cu care articolul poate fi apelat rapid

Meniuri în Java

Acceleratori (Clasa MenuShortcut)

- Incepând cu Java AWT 1.1 este posibila specificarea unor combinatii de taste (acceleratori - shortcuts) pentru accesarea directa, prin intermediul tastaturii, a optiunilor dintr-un meniu.
- Astfel, oricarui obiect de tip MenuItem îi poate fi asociat un obiect de tip accelerator, definit prin intermediul clasei MenuShortcut.
- Singurele combinatii de taste care pot juca rolul acceleratorilor sunt: + sau ++ .

Atribuirea unui accelerator la un articol al unui meniu poate fi realizata prin constructorul obiectelor de tip MenuItem în forma:

```
MenuItem(String eticheta, MenuShortcut accelerator)
// Exemplu
new MenuItem("Open", new MenuShortcut(KeyEvent.VK_O));
```

Meniuri în Java

Clasa Menu

- *Este clasa care permite crearea unui meniu derulant într-o bara de meniuri.*
- Optional, un meniu poate fi declarat ca fiind tear-off, ceea ce înseamna ca poate fi deschis si deplasat cu mouse-ul (dragged) într-o alta pozitie decât cea originala ("rupt" din pozitia sa).
- Acest mecanism este dependent de platforma si poate fi ignorat pe unele dintre ele.
- *Fiecare meniu are o eticheta, care este de fapt numele sau ce va fi afisat pe bara de meniuri.*

Meniuri în Java

- Articolele dintr-un meniu trebuie sa apartina clasei MenuItem, ceea ce înseamna ca pot fi instante ale uneia din clasele MenuItem, Menu sau CheckboxMenuItem.

```
MenuBar mb = new MenuBar();                    // se creeaza bara de meniuri
Menu optiuni = new Menu("Options");            // se creeaza un meniu
optiuni.add(new MenuItem("Copy"));
optiuni.add("Cut");                            // se adauga articole
optiuni.add("Paste);
optiuni.addSeparator();
optiuni.add("Help");
mb.add(optiuni);                               // se adauga meniul la bara
Frame f = new Frame("Fereastra cu meniu");
f.addMenuBar(mb);                              // se ataseaza bara unei ferestre
```

Meniuri în Java

Clasa CheckboxMenuItem

- *Implementeaza într-un meniu articole de tip comutator - articole care au doua stari logice (validat/nevalidat), actionarea asupra articolului determinând trecerea sa dintr-o stare în alta.*
- La validarea unui comutator în dreptul etichetei sale va aparea un simbol grafic care indica acest lucru; la invalidarea sa, simbolul grafic va disparea.
- Clasa CheckboxMenuItem are aceeasi functionalitate cu cea a casetelor de validare, implementând interfata ItemSelectable.

Meniuri în Java

Tratarea evenimentelor generate de meniuri

- La alegerea unei optiuni dintr-un meniu se genereaza un eveniment de tip **ActionEvent** si comanda este reprezentata de numele optiunii alese.
- Asadar pentru a activa optiunile unui meniu *trebuie implementat un obiect receptor* care sa implementeze interfata **ActionListener** si care în metoda **actionPerformed** sa specifice *codul ce trebuie executat la alegerea unei optiuni*.

Meniuri în Java

- Fiecarui meniu îi putem asocia un obiect receptor diferit, ceea ce usureaza munca în cazul în care ierarhia de meniuri este complexa.
- Pentru a realiza *legatura între obiectul meniu si obiectul receptor* trebuie sa adaugam receptorul în lista de ascultatori a meniului respectiv prin comanda: *meniu.addActionListener(listener)*
- Asadar, *tratarea evenimentelor unui meniu este asemanatoare cu tratarea butoanelor*, ceea ce face posibil ca unui buton de pe suprafata de afisare sa îi corespunda o optiune într-un meniu, ambele cu acelasi nume, tratarea evenimentului corespunzator apasarii butonului, sau alegerii optiunii facându-se o singura data într-o clasa care este înregistrata ca receptor atât la buton cât si la meniu.

Meniuri în Java

- Un caz special îl constituie optiunile de tip CheckboxMenuItem.

- Obiectele de acest tip se gasesc într-o categorie comuna cu List, Choice, CheckBox, implementeaza o interfata comuna ItemSelectable si toate genereaza evenimente de tip ItemEvent.
- Din aceasta cauza actionarea unei optiuni de tip CheckboxMenuItem nu va determina generarea unui eveniment de tip ActionEvent de catre meniul din care face parte, ci va genera un eveniment ItemEvent chiar de catre articolul respectiv.

Meniuri în Java

- Pentru a intercepta un asemenea eveniment avem nevoie de un obiect receptor care sa implementeze interfata **ItemListener** si sa specifice în metoda acesteia itemStateChanged codul ce trebuie executat la validarea/invalidarea optiunii din meniu.
- De asemenea receptorul trebuie înregistrat cu metoda addItemListener.
- Tipul de operatie selectare / deselectare este codificat de câmpurile statice ItemEvent.SELECTED si ItemEvent.DESELECTED.

Meniuri în Java

Exemplu de tratare a evenimentelor unui meniu

```
import java.awt.*;
import java.awt.event.*;
public class TestMenuEvent extends Frame
implements ActionListener, ItemListener{
    public TestMenuEvent(String titlu) {
    super(titlu);
}
```

Meniuri în Java

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

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

Meniuri în Java

```
public static void main(String args[]) {
        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);
        TestMenuEvent f = new TestMenuEvent("Test Meniu");
        Button btnExit = new Button("Exit");
```

Meniuri în Java

```
        f.setMenuBar(mb);
        f.add(btnExit, BorderLayout.SOUTH);
        f.setSize(300, 200);
        f.show();

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

Meniuri în Java

Meniuri de context (popup)

- Sunt implementate prin intermediul clasei **PopupMenu**, subclasa directa a clasei Menu.
- Sunt meniuri invizibile care *sunt activate uzual prin apasarea butonului drept al mouse-ului*, fiind afisate la pozitia la care se gasea mouse-ul în momentul apasarii butonului sau drept.
- *Metodele de adaugare a articolelor unui meniu popup sunt identice cu cele de la meniurile fixe*, PopupMenu fiind subclasa directa a clasei Menu.

Meniuri în Java

```
popup = new PopupMenu("Options");
popup.add(new MenuItem("New"));
popup.add(new MenuItem("Edit"));
popup.addSeparator();
popup.add(new MenuItem("Exit"));
```

- Afisarea meniului de context se face prin metoda show:

```
popup.show(Component origin, int x, int y)
```

si este, de obicei, rezultatul apasarii unui buton al mouse-ului, pentru a avea acces rapid la meniu.

- Argumentul "origin" reprezinta componenta fata de originile careia se va calcula pozitia de afisare a meniului popup.
- De obicei, reprezinta instanta ferestrei în care se va afisa meniul.

Meniuri în Java

- *Meniurile de context nu se adauga la un alt meniu (bara sau sub-meniu)* ci se ataseaza la o componenta (de obicei la o fereastra) prin metoda add: fereastra.add(pm).

- In cazul când avem mai multe meniuri popup pe care vrem sa le folosim într-o fereastra, trebuie sa le definim pe toate si, la un moment dat, vom adauga ferestrei meniul corespunzator.

Meniuri în Java

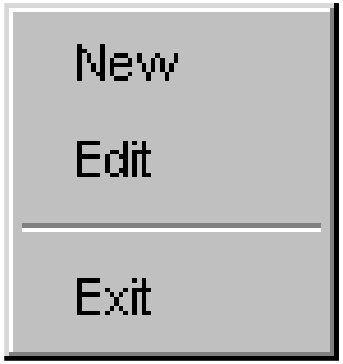
Dupa închiderea acestuia vom "rupe" legatura între fereastra si meniu prin instructiunea remove:

```
fereastra.add(popup1);
. . .
fereastra.remove(popup1);
fereastra.add(popup2);
```

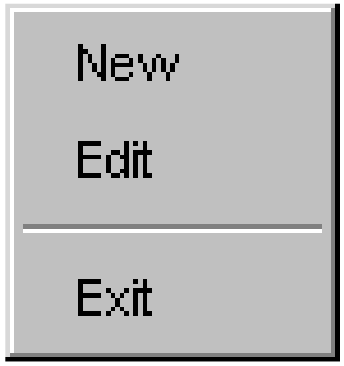
Meniuri în Java

Un meniu de contex ca în imaginea de mai jos, care se va activa la apasarea butonului drept al mouse-ului pe suprafata ferestrei principale:

La alegerea optiunii "Exit" din meniu se va termina programul.

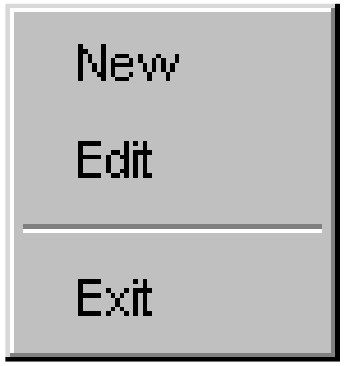


Meniuri în Java



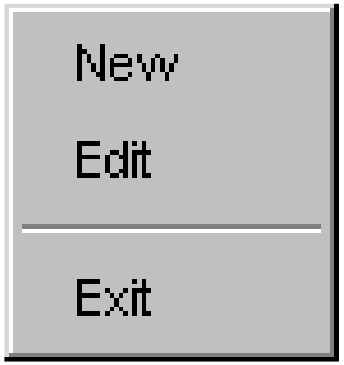

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements ActionListener{
        private PopupMenu popup; //definim meniul popup
al ferestrei
        private Component origin; //pozitia meniului va fi
relativa la pozitia ferestrei
        public Fereastra(String titlu) {
                super(titlu);
                origin = this;
```

Meniuri în Java



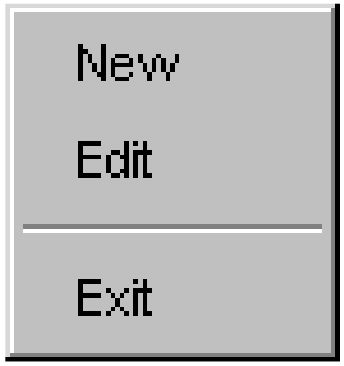

```
this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
    });
this.addMouseListener(new MouseAdapter() {
public void mouseClicked(MouseEvent e) {
if ( (e.getModifiers() & InputEvent.BUTTON3_MASK)
== InputEvent.BUTTON3_MASK )
popup.show(origin, e.getX(), e.getY());
// BUTTON3 reprezinta butonul din dreapta mouse-ului
}
});
setSize(300, 300);
```

Meniuri în Java




```
        // se creaza 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);          // se ataseaza meniul popup ferestrei
        popup.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
          String command = e.getActionCommand();
          // La alegerea optiunii "Exit" din meniu se paraseste aplicatia
          if (command.equals("Exit")) System.exit(0);
          }
}
```

Meniuri în Java




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

6. Interfața grafică AWT(continuare)

1. Meniuri în Java

2. Componente AWT(Abstract Window Toolkit)

Componente AWT

Folosirea componentelor

1. Label
2. Button
3. Checkbox
4. CheckboxGroup
5. Choice
6. List
7. Scrollbar
8. ScrollPane
9. TextField
10. TextArea

2.1 Clasa Label

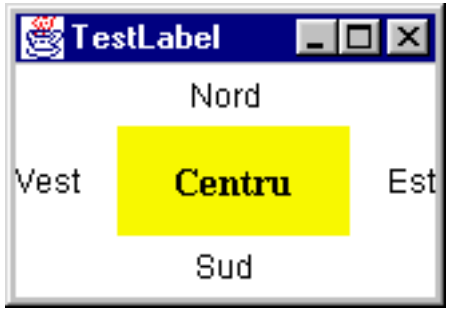
2.1. Clasa **Label**

- Un obiect de tip Label (eticheta) reprezinta o *componenta pentru plasarea unui text pe o suprafata de afisare.*

- O eticheta este formata dintr-o singura linie de text static ce nu poate fi modificat de catre utilizator, dar poate fi modificat din program.

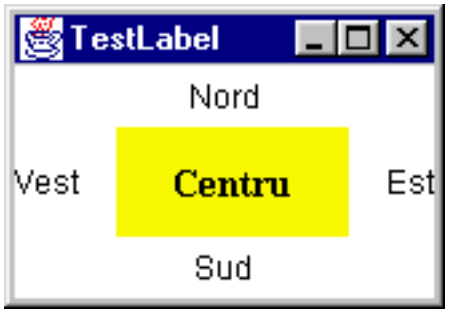
2.1 Clasa Label

Definirea a cinci etichete si plasarea lor într-un container:



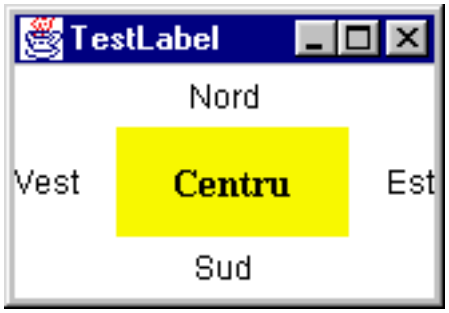

```
import java.awt.*;

public class TestLabel {
public static void main(String args[]) {
        Frame f = new Frame("TestLabel");
        f.setLayout(new BorderLayout());
```

2.1 Clasa Label




```
Label nord, sud, est, vest, centru;
nord = new Label("Nord", Label.CENTER);
sud = new Label("Sud", Label.CENTER);
est = new Label("Est", Label.RIGHT);
vest = new Label("Vest", Label.LEFT);
centru = new Label("Centru", Label.CENTER);
centru.setBackground(Color.yellow);
centru.setFont(new Font("Arial", Font.BOLD, 14));
```

2.1 Clasa Label



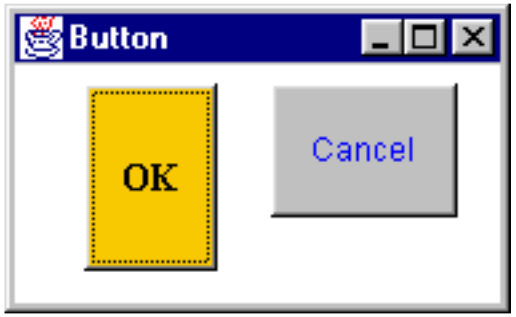

```
f.add(nord, BorderLayout.NORTH);
f.add(sud, BorderLayout.SOUTH);
f.add(est, BorderLayout.EAST);
f.add(vest, BorderLayout.WEST);
f.add(centru, BorderLayout.CENTER);
f.pack();
f.show();
}
}
```

2.2 Clasa Button

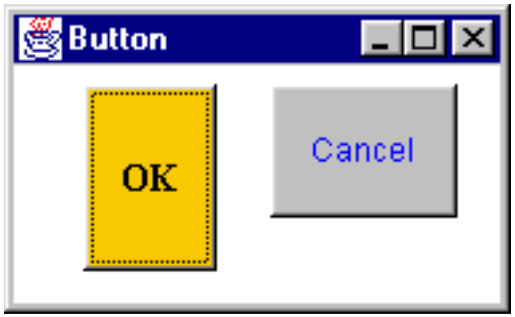
2.2 Clasa **Button**

Un obiect de tip **Button** (buton) reprezinta *o componenta pentru plasarea unui buton etichetat pe o suprafata de afisare.*

Exemplu: definim doua butoane si le plasam pe o fereastra; la apasarea butonului "OK" titlul ferestrei va fi "Confirmare", iar la apasarea butonului "Cancel" titlul ferestrei va fi "Renuntare".

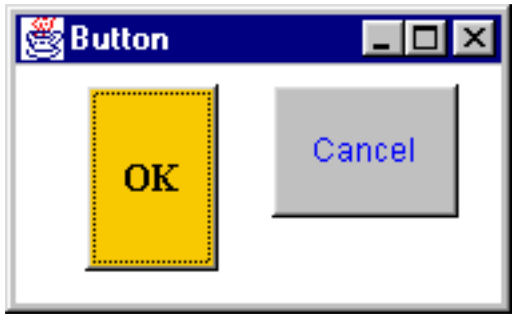


2.2 Clasa Button



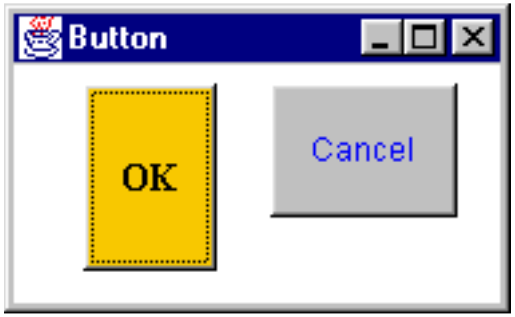

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements ActionListener{
public Fereastra(String titlu) {
   super(titlu);
   this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
   }
  });
}
```

2.2 Clasa Button




```
public void initializare() {
    setLayout(null);
    setSize(200, 200);
    Button b1 = new Button("OK");
    b1.setBounds(30, 30, 50, 70);
    b1.setFont(new Font("Arial", Font.BOLD, 14));
    b1.setBackground(java.awt.Color.orange);
    add(b1);
    Button b2 = new Button("Cancel");
    b2.setBounds(100, 30, 70, 50);
    b2.setForeground(java.awt.Color.blue);
    add(b2);
    b1.addActionListener(this);
    b2.addActionListener(this);
}
```

2.2 Clasa Button




```
// metoda interfetei ActionListener
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    System.out.println(e.toString());
    if (command.equals("OK")) setTitle("Confirmare!");
    if (command.equals("Cancel")) setTitle("Anulare!");
    }
}
public class TestButton {
public static void main(String args[]) {
    Fereastra f = new Fereastra("Button");
    f.initializare();
    f.show();
    }
}
```

2.3 Clasa Checkbox

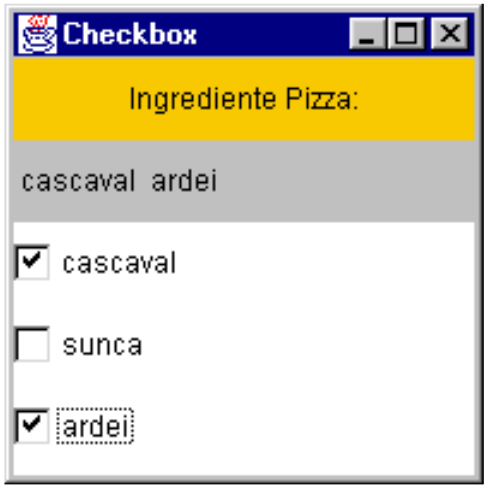
2.3 Clasa **Checkbox**

Un obiect de tip **Checkbox** (comutator) *reprezinta o componenta care se poate gasi în doua stari: "selectata" sau "neselectata" (on/off).*

- *Actiunea utilizatorului asupra unui comutator îl trece pe acesta în starea complementara celei în care se gasea.*
- Este folosit pentru a prelua o anumita optiune de la utilizator.

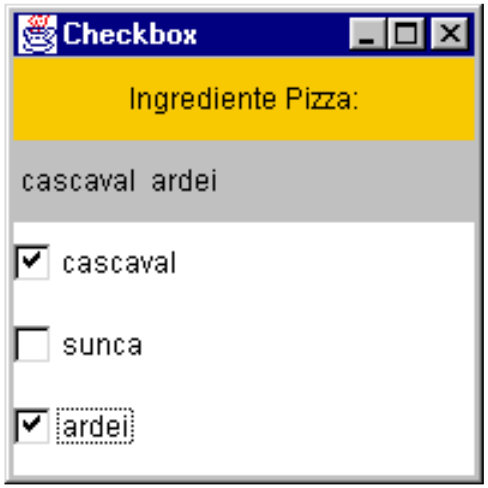
2.3 Clasa Checkbox

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements ItemListener {
        private Label label1, label2;
        private Checkbox cbx1, cbx2, cbx3;
        public Fereastra(String titlu) {
                super(titlu);
this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
  });
}
```



2.3 Clasa Checkbox

```
public void initializare() {
        setLayout(new GridLayout(5, 1));
        label1 = new Label("Ingrediente Pizza:", Label.CENTER);
        label1.setBackground(Color.orange);
        label2 = new Label("");
        label2.setBackground(Color.lightGray);
        cbx1 = new Checkbox("cascaval");
        cbx2 = new Checkbox("sunca");
        cbx3 = new Checkbox("ardei");
        add(label1);
        add(label2);
        add(cbx1);
        add(cbx2);
        add(cbx3);
        pack();
        setSize(200, 200);
        cbx1.addItemListener(this);
        cbx2.addItemListener(this);
        cbx3.addItemListener(this);
}
```

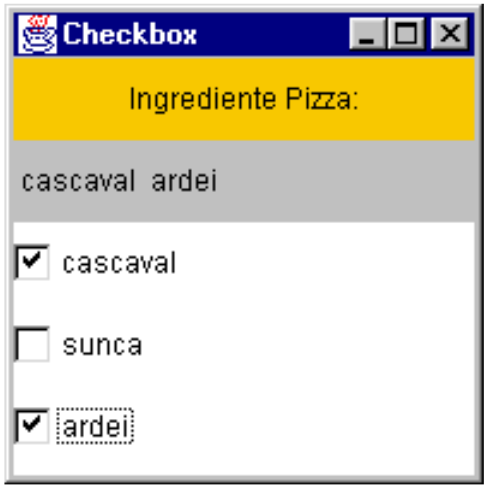


2.3 Clasa Checkbox

```
// metoda interfetei ItemListener
public void itemStateChanged(ItemEvent e) {
    StringBuffer ingrediente = new StringBuffer();
    if (cbx1.getState() == true) ingrediente.append(" cascaval ");
    if (cbx2.getState() == true) ingrediente.append(" sunca ");
    if (cbx3.getState() == true) ingrediente.append(" ardei ");
    label2.setText(ingrediente.toString());
    }
}
```

2.3 Clasa Checkbox

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



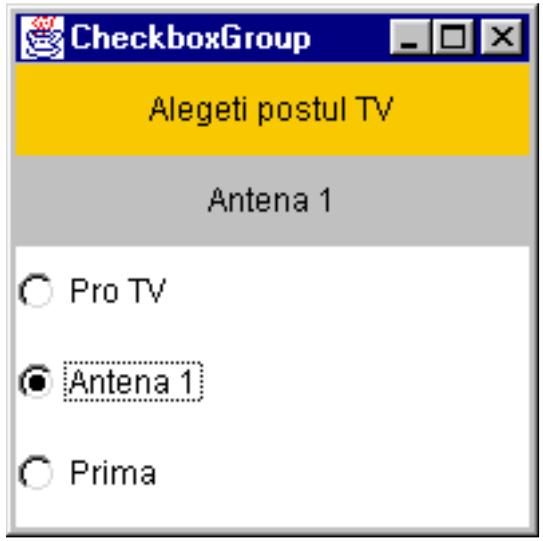
2.4 Clasa CheckboxGroup

2.4 Clasa **CheckboxGroup**

Un obiect de tip **CheckboxGroup** *defineste un grup de comutatoare din care doar unul poate fi selectat*.
Uzual, aceste componente se mai numesc butoane radio.

2.4 Clasa CheckboxGroup

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements ItemListener {
private Label label1, label2;
private Checkbox cbx1, cbx2, cbx3;
private CheckboxGroup cbg;
public Fereastra(String titlu) {
      super(titlu);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
      });
}
```



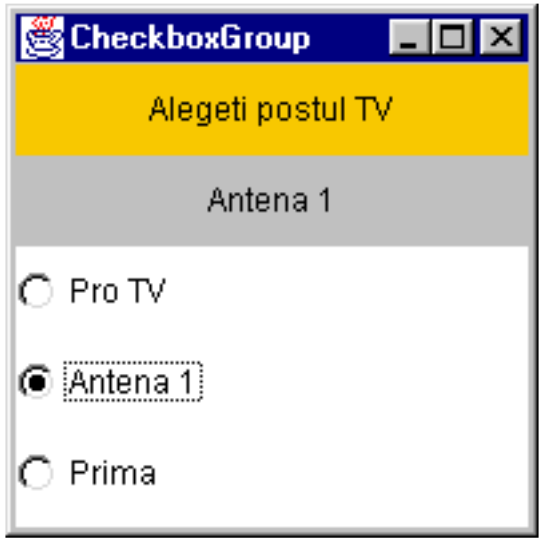
2.4 Clasa CheckboxGroup


```
public void initializare() {
setLayout(new GridLayout(5, 1));

label1 = new Label("Alegeti postul TV",
Label.CENTER);
label1.setBackground(Color.orange);
label2 = new Label("", Label.CENTER);
label2.setBackground(Color.lightGray);

cbg = new CheckboxGroup();
cbx1 = new Checkbox("Pro TV", cbg, false);
cbx2 = new Checkbox("Antena 1", cbg,
false);
cbx3 = new Checkbox("Prima", cbg, false);
```

```
add(label1);
add(label2);
add(cbx1);
add(cbx2);
add(cbx3);
pack();

setSize(200, 200);
cbx1.addItemListener(this);
cbx2.addItemListener(this);
cbx3.addItemListener(this);

}
```

2.4 Clasa CheckboxGroup

```
//metoda interfetei ItemListener
public void itemStateChanged(ItemEvent e) {
Checkbox cbx = cbg.getSelectedCheckbox();
        if (cbx != null)
                label2.setText(cbx.getLabel());
        }
}
public class TestCheckboxGroup {
        public static void main(String args[]) {
        Fereastra f = new Fereastra("CheckboxGroup");
        f.initializare();
        f.show();
}
}
```



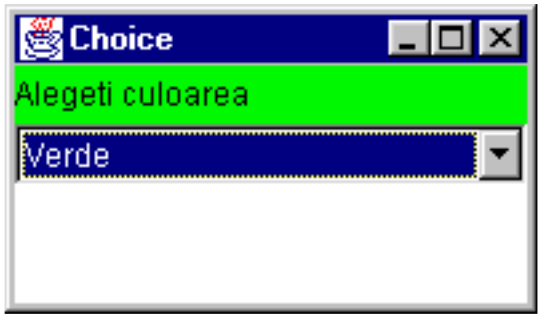
2.5 Clasa Choice

2.5 Clasa **Choice**

Un obiect de tip **Choice** *defineste o lista de optiuni din care utilizatorul poate selecta una singura*.
La un moment dat, din întreaga lista doar o singura optiune este vizibila, cea selectata în momentul curent.
O componenta **Choice** este însotita de un buton etichetat cu o sageata verticala la apasarea caruia este afisata întreaga sa lista, pentru ca utilizatorul
sa poata selecta o anumita optiune.

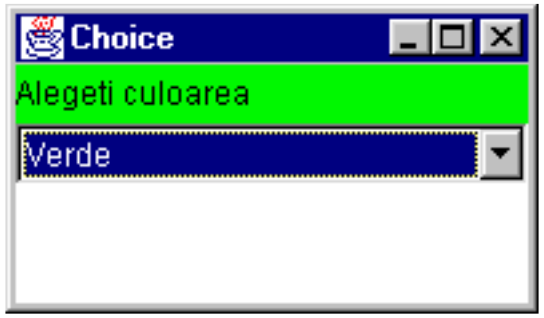
2.5 Clasa Choice

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements ItemListener {
private Label label;
private Choice culori;
public Fereastra(String titlu) {
      super(titlu);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
      });
}
```



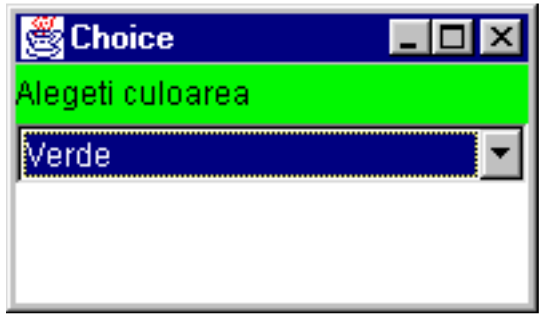
2.5 Clasa Choice

```
public void initializare() {
      setLayout(new GridLayout(4, 1));
      label = new Label("Alegeti culoarea");
      label.setBackground(Color.red);
      culori = new Choice();
      culori.add("Rosu");
      culori.add("Verde");
      culori.add("Albastru");
      culori.select("Rosu");
      add(label);
      add(culori);
      pack();
      setSize(200, 100);
      culori.addItemListener(this);
}
```



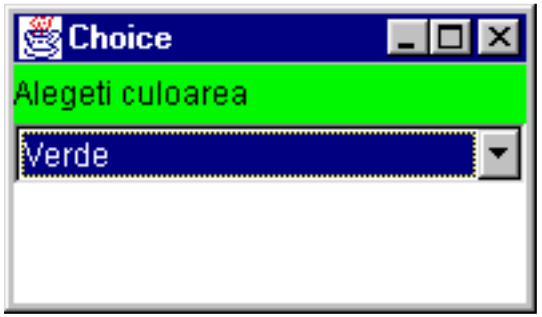
2.5 Clasa Choice

```
//metoda interfetei ItemListener
public void itemStateChanged(ItemEvent e) {
    switch (culori.getSelectedIndex()) {
    case 0: label.setBackground(Color.red);
    break;
    case 1: label.setBackground(Color.green);
    break;
    case 2: label.setBackground(Color.blue);
    }
}
}
```



2.5 Clasa Choice

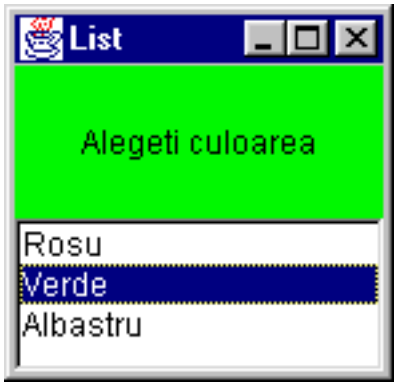
```
public class TestChoice {
public static void main(String args[]) {
        Fereastra f = new Fereastra("Choice");
        f.initializare();
        f.show();
        }
}
```



2.6 Clasa List

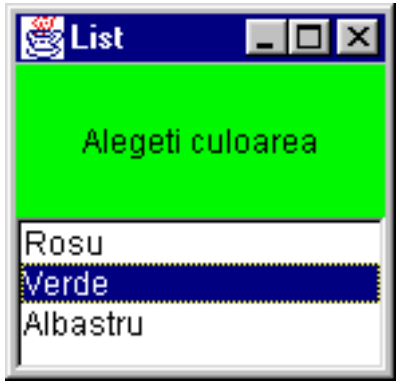
2.6 Clasa **List**

Un obiect de tip **List** *defineste o lista de optiuni care poate fi setata astfel încât utilizatorul sa poata selecta o singura optiune sau mai multe*.
Toate optiunile listei sunt vizibile în limita dimensiunilor grafice ale componentei.



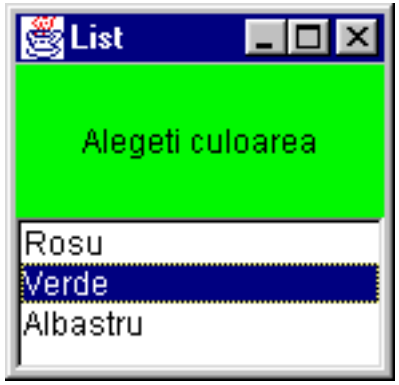
2.6 Clasa List

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements ItemListener {
private Label label;
private List culori;
public Fereastra(String titlu) {
      super(titlu);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
      });
}
```



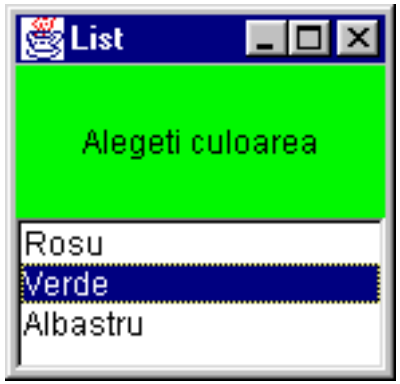
2.6 Clasa List

```
public void initializare() {
      setLayout(new GridLayout(2, 1));
      label = new Label("Alegeti culoarea", Label.CENTER);
      label.setBackground(Color.red);
      culori = new List(3);
      culori.add("Rosu");
      culori.add("Verde");
      culori.add("Albastru");
      culori.select(3);
      add(label);
      add(culori);
      pack();
      setSize(200, 200);
      culori.addItemListener(this);
}
```



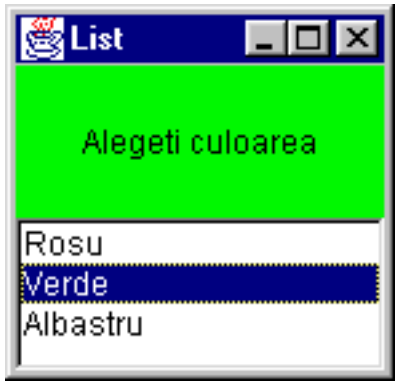
2.6 Clasa List

```
//metoda interfetei ItemListener
public void itemStateChanged(ItemEvent e) {
    switch (culori.getSelectedIndex()) {
    case 0: label.setBackground(Color.red);
    break;
    case 1: label.setBackground(Color.green);
    break;
    case 2: label.setBackground(Color.blue);
    }
    }
}
```



2.6 Clasa List

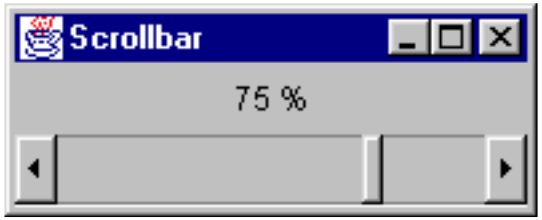
```
public class TestList {
public static void main(String args[]) {
        Fereastra f = new Fereastra("List");
        f.initializare();
        f.show();
        }
}
```



2.7 Clasa Scrollbar

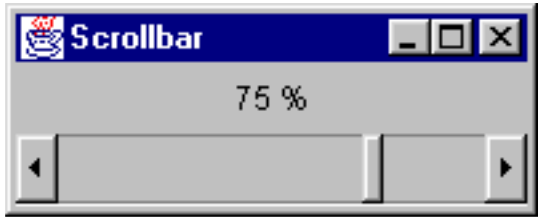
2.7 Clasa **Scrollbar**

Un obiect de tip **Scrollbar** *defineste o bara de defilare verticala sau orizontala.*
Este utila pentru punerea la dispozitie a utilizatorului a unei modalitati sugestive de a alege o anumita valoare dintr-un interval.



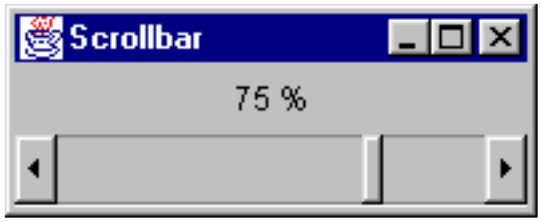
2.7 Clasa Scrollbar

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements AdjustmentListener {
private Scrollbar scroll;
private Label valoare;
public Fereastra(String titlu) {
      super(titlu);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
      });
}
```



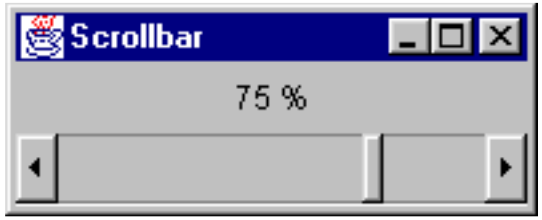
2.7 Clasa Scrollbar

```
public void initializare() {
    setLayout(new GridLayout(2, 1));
    valoare = new Label("", Label.CENTER);
    valoare.setBackground(Color.lightGray);
    scroll = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 101);
    add(valoare);
    add(scroll);
    pack();
    setSize(200, 80);
    scroll.addAdjustmentListener(this);
}
```

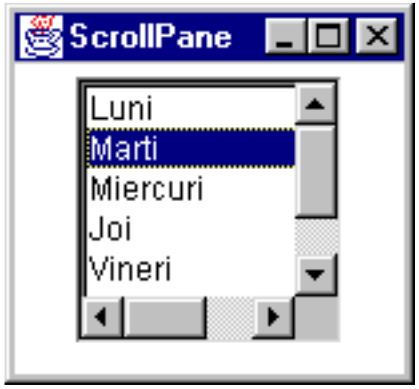


2.7 Clasa Scrollbar

```
//metoda interfetei ItemListener
public void adjustmentValueChanged(AdjustmentEvent e) {
        valoare.setText(scroll.getValue() + " %");
        }
}
public class TestScrollbar {
public static void main(String args[]) {
    Fereastra f = new Fereastra("Scrollbar");
    f.initializare();
    f.show();
    }
}
```



2.8 Clasa ScrollPane

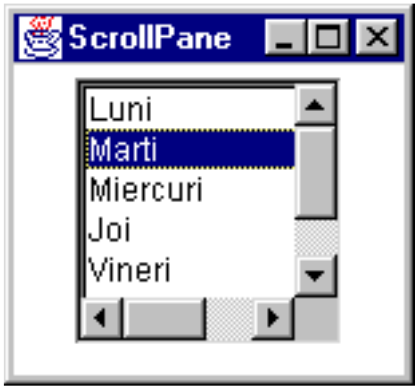


2.8 Clasa **ScrollPane**

Un obiect de tip **ScrollPane** *permite atasarea unor bare de defilare (orizontala si/sau verticala) oricarei componente grafice.*

Acest lucru este util pentru acele componente care nu au implementata functionalitatea de defilare automata, cum ar fi listele (obiecte din clasa List).

2.8 Clasa ScrollPane

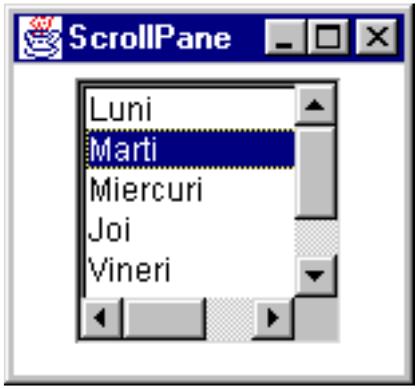



```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame {
private ScrollPane sp;
private List list;
public Fereastra(String titlu) {
    super(titlu);
    this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
    System.exit(0);
    }
    });
}
```

2.8 Clasa ScrollPane


```
public void initializare() {
      setLayout(new FlowLayout());
      list = new List(7);
      list.add("Luni");
      list.add("Marti");
      list.add("Miercuri");
      list.add("Joi");
      list.add("Vineri");
      list.add("Sambata");
      list.add("Duminica");
      list.select(1);
      sp = new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
      sp.add(list);
      add(sp);
      pack();
      setSize(200, 200);
      }
}
```

2.8 Clasa ScrollPane




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

2.9 Clasa TextField

2.9 Clasa **TextField**

Un obiect de tip **TextField** *defineste un control de editare a textului pe o singura linie*.

Este util pentru interogarea utilizatorului asupra unor valori.

2.9 Clasa TextField

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements TextListener {
private TextField nume, parola;
private Label acces;
private static final String UID="Ion", PWD="java" ;
public Fereastra(String titlu) {
      super(titlu);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
      });
}
```

2.9 Clasa TextField


```
public void initializare() {
    setLayout(new GridLayout(3, 1));
    setBackground(Color.lightGray);
    nume = new TextField("", 30);
    parola = new TextField("", 10);
    parola.setEchoChar('*');
    Panel p1 = new Panel();
    p1.setLayout(new FlowLayout(FlowLayout.LEFT));
    p1.add(new Label("Nume:"));
    p1.add(nume);
    Panel p2 = new Panel();
    p2.setLayout(new FlowLayout(FlowLayout.LEFT));
    p2.add(new Label("Parola:"));
    p2.add(parola);
```

2.9 Clasa TextField

```
    acces = new Label("Introduceti numele si parola!",
    add(p1);
    add(p2);
    add(acces);
    pack();
    setSize(350, 100);
    nume.addTextListener(this);
    parola.addTextListener(this);
}
```

2.9 Clasa TextField

```
//metoda interfetei TextListener
public void textValueChanged(TextEvent e) {
    if ((nume.getText().length() == 0) || acces.setText("");
            return;
    }
    if (nume.getText().equals(UID) && acces.setText("Acces
    permis!");
    else
        acces.setText("Acces interzis!");
    }
}
```

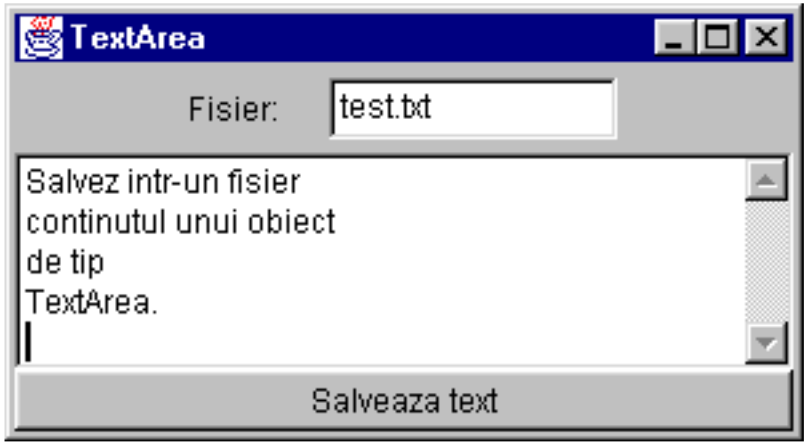
2.9 Clasa TextField

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

2.10 Clasa TextArea

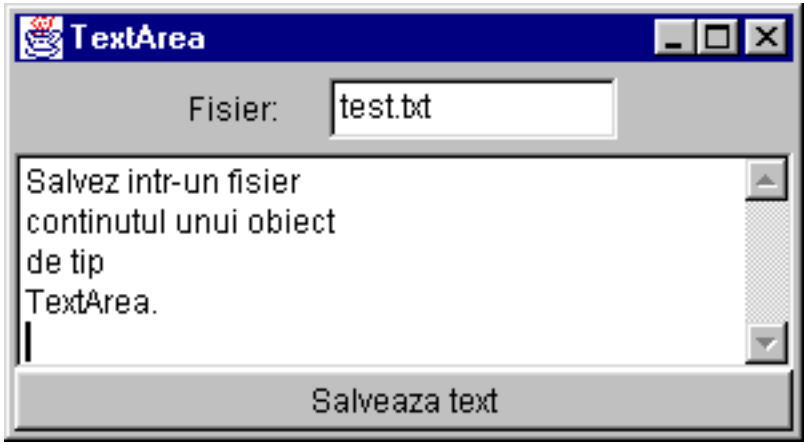
2.10 Clasa **TextArea**

Un obiect de tip **TextArea** *defineste un control de editare a textului pe mai multe linii*.
Este util pentru editarea de texte, introducerea unor comentarii, etc.



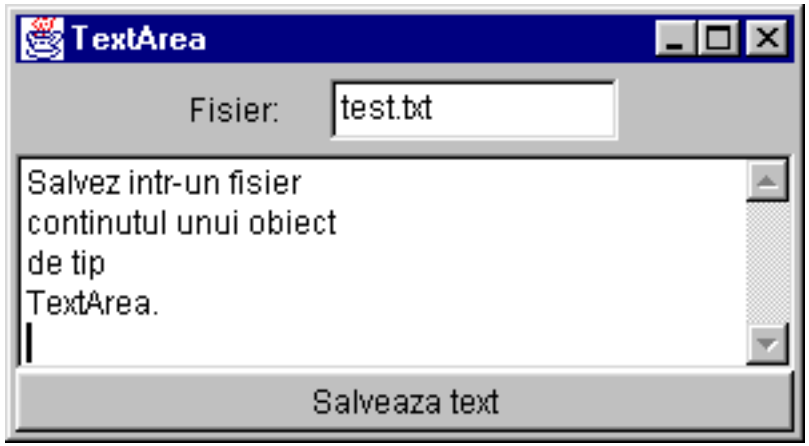
2.10 Clasa TextArea

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
class Fereastra extends Frame implements TextListener, ActionListener {
private TextArea text;
private TextField nume;
private Button save;
public Fereastra(String titlu) {
      super(titlu);
      this.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
      System.exit(0);
      }
      });
}
```



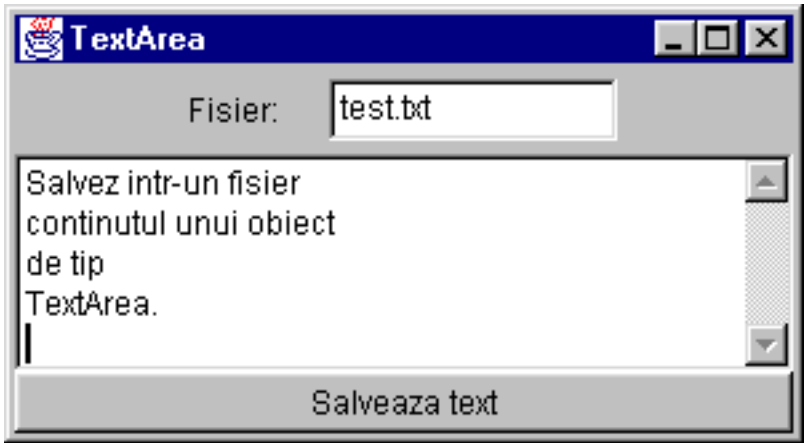
2.10 Clasa TextArea

```
public void initializare() {
        setBackground(Color.lightGray);
        text = new TextArea("", 30, 10,
        nume = new TextField("", 12);
        save = new Button("Salveaza text");
        save.setActionCommand("save");
        save.setEnabled(false);
        Panel fisier = new Panel();
        fisier.add(new Label("Fisier:"));
        fisier.add(nume);
        add(fisier, BorderLayout.NORTH);
        add(text, BorderLayout.CENTER);
        add(save, BorderLayout.SOUTH);
        pack();
        setSize(300, 200);
        text.addTextListener(this);
        save.addActionListener(this);
}
```



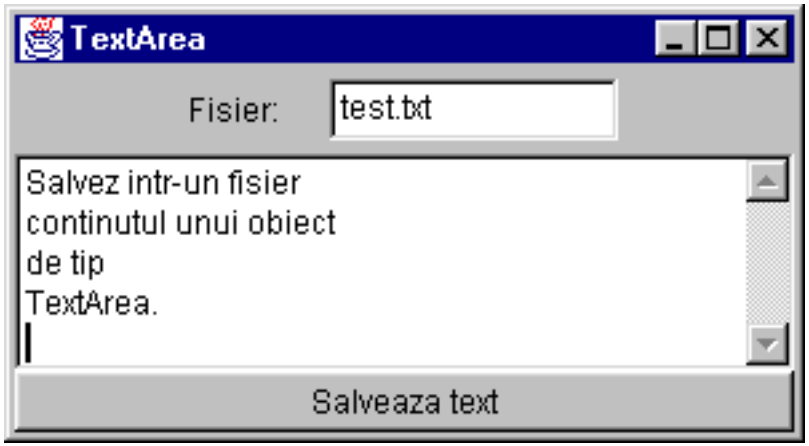
2.10 Clasa TextArea

```
//metoda interfetei TextListener
public void
textValueChanged(TextEvent e) {
if ((text.getText().length() == 0)
        save.setEnabled(false);
else save.setEnabled(true);
}
//metoda interfetei ActionListener
public void
actionPerformed(ActionEvent e) {
String continut = text.getText();
int len = continut.length();
char buffer[] = new char[len];
try {
FileWriter out = new
FileWriter(nume.getText());
continut.getChars(0, len-1, buffer, 0);
out.write(buffer);
out.close();
text.requestFocus();
}
catch(IOException ex) {
ex.printStackTrace();
}
}
}
```



2.10 Clasa TextArea

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



Referinte

- Curs practic de Java, Cristian Frasinaru – capitolul Interfata grafica cu utilizatorul
- http://docs.oracle.com/javase/6/docs/technotes/guides/awt/
- http://www.tutorialspoint.com//awk/index.htm
- http://archive.oreilly.com/oreillyschool/courses/java3/archive/java3.2/java306.html

Întrebări?