Interfețe grafice
(note de curs, vor fi consultate suplimentar față de tematica isciplinei)

Pentru comunicarea între utilizator şi program se foloseşte o *interfață grafică utilizator* (*Graphical User Interface* - GUI). Pe ecran apar mai multe componente sau controale, care conțin diferite obiecte grafice (deci care au o reprezentare pe ecran): etichete, meniuri, butoane, casete de dialog, imagini, etc. Pentru a efectua o acțiune în cadrul programului, utilizatorul foloseşte tastatura sau *mouse-ul*. Cele mai utilizate interfețe grafice utilizator sunt: **Windows** (a firmei Microsoft), **XWindow** (pentru sistemele de operare Unix sau Linux) şi **Macintosh** (a firmei Apple).

Limbajul Java permite realizarea interfețelor grafice independente de platforma pe care se execută aplicațiile. La dispoziția programatorilor există o mulțime de componente grafice, care sunt implemementate pe diferite sisteme de operare. Din această cauză este posibil ca un acelaşi obiect grafic să aibă aspecte diferite pe diverse sisteme de operare. Programatorul utilizează, în programe, aceste componente, fără să fie preocupat de implementarea acestora.

Incă de la primele versiuni ale limbajului Java există pachetul ***java.awt,*** care conține clase şi interfețe pentru diverse componente grafice (AWT - *Abstract Window Toolkit)*. A doua variantă de creare a interfețelor grafice este **Swing** - o parte dintr-o colecție foarte mare de clase, care formează ***JFC*** *(Java Foundation Classes)*. Swing s-a construit folosind foarte multe clase din AWT, dar apar clase noi cu facilități mai bogate.

In figurile următoare sunt date două ierarhii de clase pentru componentele grafice din **pachetul AWT**.

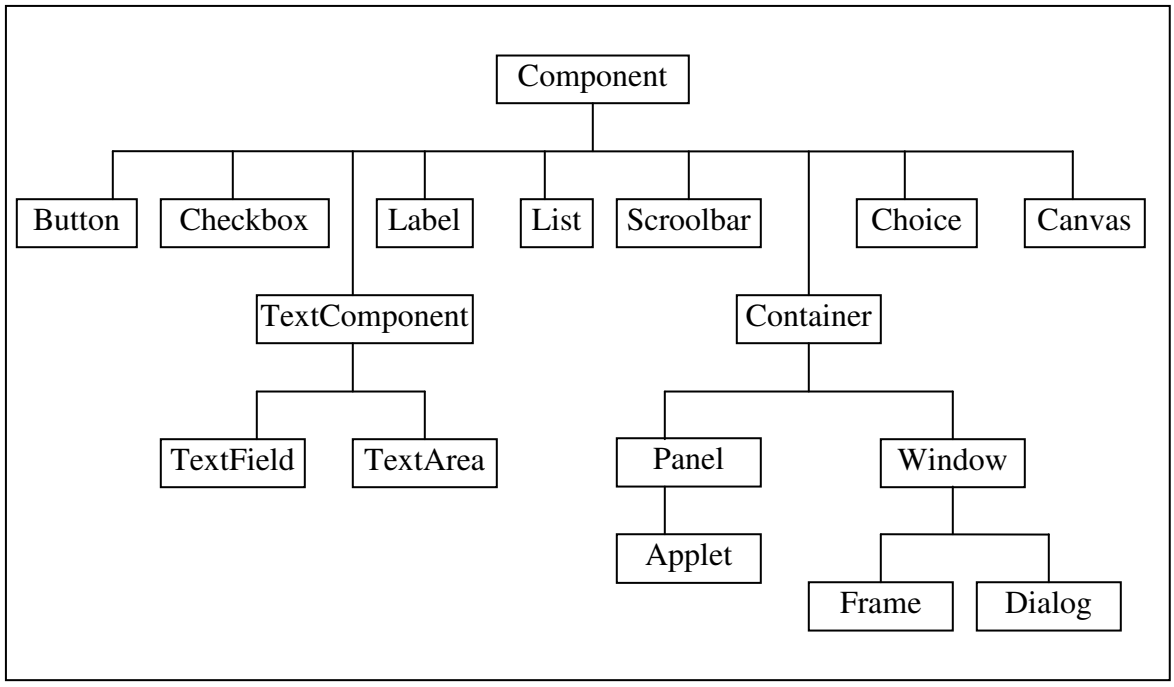


Fig. 1

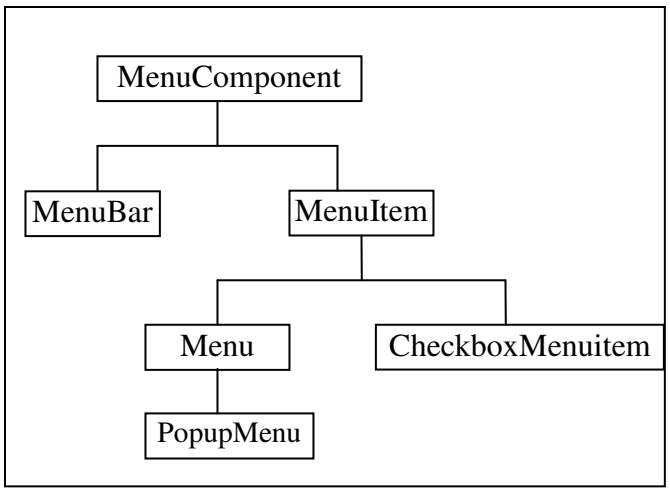


Fig. 2

Toate clasele date mai sus sunt cuprinse în pachetul ***java.awt***, cu o singură excepție, **clasa Applet**, care este conținută în pachetul ***java.applet***.

Rădăcina ierarhiei tuturor componentelor din AWT care nu sunt folosite pentru meniuri (precizate în fig.1) este **clasa Component**, care este o clasă abstractă.

O componentă (control) din interfață se obține prin instanțierea uneia din clasele din ierarhie. Un *obiect* are o reprezentare grafică în interfață şi poate să interacționeze cu utilizatorul. Un astfel de obiect trebuie:

- să fie *construit* (există mai mulți constructori pentru aceste clase),
- să fie *poziționat* pe o **suprafață de afişare** (o fereastră, un applet - subclase a clasei Container),
- să fie *dimensionat* şi eventual să i se atribuie anumite valori *proprietăților* pe care le are.

In continuare se vor aminti câteva dintre cele mai utilizate metode ale clasei **Component** (din cele peste 200 metode existente), moştenite de subclasele acesteia. In unele metode se folosesc anumite clase existente tot în pachetul *java.awt, şi* anume:

- clasa *Dimension* - are ca membri: *width* şi *height* (lățimea şi înălțimea componentei);
- clasa *Rectangle* - are ca membri: *width*, *height*, *x*, *y* (lățimea, înălțimea şi coordonatele x,y ale colțului stânga-sus al dreptunghiului care conține componenta);
- clasa *Color* - permite gestiunea culorilor;
- clasa *Font* - conține informații despre fontul utilizat.

Metode de **precizare** (setare) sau **determinare** a dimensiunilor şi a coordonatelor componentei:

- `public void` **`setSize`**`(int width, int height)` – setează dimensiunile (lățime, înălțime);
- `public void` **`setSize`**`(Dimension d)` – setează dimensiunile;
- `public void` **`setBounds`**`(int x, int y, int width, int height)` – setează coordonatele x,y ale originii componentei (colțul din stânga-sus) şi dimensiunile acesteia (lățime, înălțime);
- `public void` **`setBounds`**`(Rectangle r)` – setează dimensiunile şi coordonatele componentei;
- `public int` **`getX`**`()` – determină coordonata x a originii componentei;
- `public int` **`getY`**`()` – determină coordonata y a originii componentei;
- `public int` **`getWidth`** – determină lățimea componentei;
- `public int` **`getHeight`** – determină înălțimea componentei;
- `public Dimension getSize()` – determină dimensiunile componentei (lățime şi înălțime);

- `public Rectangle getBounds()` – determină dimensiunile şi coordonatele componentei (lățime, înălțime, x, y);

Pentru vizualizare există metodele:

- `public void` **`setVisible`**`(boolean visible)` – conform valorii argumentului, metoda face componenta vizibilă sau invizibilă;
- `public void` **`isVisible`**`()` – dă valoarea *true* în cazul în care componente este vizibilă;

Pentru precizarea/determinarea unor proprietăți **utile la desenare** avem metodele:

- `public void` **`setBackground`**`(Color c)` – setează culoarea de fond a componentei;
- `public void` **`setForeground`**`(Color c)` – setează culoarea de desenare a componentei (culoarea textului şi a desenului de pe componentă);
- `public Color` **`getBackground`**`()` – dă culoarea de fond a componentei;
- `public Color` **`getForeground`**`()` – dă culoarea de desenare a componentei;
- `public void` **`setFont`**`(Font f)` – precizează fontul componentei;
- `public Font` **`getFont`**`()` – dă fontul folosit de componentă;

Pentru desenarea/redesenarea componentei există metodele:

- `public void` **`paint`**`(Graphics g)` – desenează componenta folosind contextul grafic g;
- `public void` **`repaint`**`()` – redesenează componenta;

Pentru a se putea da unele exemple, vom descrie la început **clasa Container** şi subclasele acesteia. Cu ajutorul **clasei Container** se pot defini componente care conțin alte componente, inclusiv containere. Rolul acestor clase este de a construi diverse ferestre de afişare. Există un singur constructor pentru această clasă:

- `public` **`Container`**`()` – crează un nou container

Componentele care se adaugă la un container se păstrează într-o listă, iar poziția în această listă este folosită pentru extragerea pe ecran. O componentă se poate adăuga într-o anumită poziție, sau la sfârşitul listei.

Dintre metodele aceste clase amintim:

- `public Component` **`add`**`(Component c)` – componenta `c` este adăugată la sfârşitul listei de componente. Se întoarce o referință la componenta c;
- `public Component` **`add`**`(Component c, int index)` – componenta c este adăugată în listă pe pozitia index şi întoarce o referință la componenta adaugată;
- `public void` **`add`**`(Component c, Object constraints)` – se adaugă componenta `c` la sfârşitul listei, precizându-se că se foloseşte obiectul de restricții `constraints`;
- `public void` **`add`**`(Component c, Object constraints, int index)` – la fel ca metoda precedentă, dar se face înserarea pe poziția `index`;
- `public void` **`remove`**`(int index)` – se elimină componenta aflată pe pozitia `index`;
- `public void` **`remove`**`(Component c)` – se elimină componenta c;
- `public void` **`removeAll`**`()` – elimină toate componentele conținute în container;
- `public void` **`paintComponents`**`(Graphics g)` – afişează toate componentele containerului;
- `public Dimension` **`getPreferredSize`**`()` – întoarce dimensiunile preferate ale containerului;
- `public Dimension` **`getMinimumSize`**`()` – întoarce dimensiunile minime ale containerului;
- `public Dimension` **`getMaximumSize`**`()` – întoarce dimensiunile maxime ale containerului;

- `public Component` **`getComponentAt`**`(int x, int y)` – întoarce componenta care se află în punctul de coordonate (x,y);
- `public boolean` **`isAncestorOf`**`(Component c)` – întoarce *true* dacă componenta c există în ierarhia de componente a containerului.

Pentru pozitionarea componentelor în cadrul unui container se foloseşte un gestionar de ***pozitionare*** (*Layout Managers*), care implementează **interfata LayoutManager**. Cu ajutorul unor astfel de obiecte poziționarea componentelor se face în mod automat, după o anumită regulă (chiar dacă se modifică dimensiunile containerului, prin redimensionare). Dezavantajul constă în faptul că trebuie respectate configurațiile de amplasare a componentelor pentru care există astfel de gestionari de poziționare. Dintre clasele existente în pachetul **java.awt**, pentru crearea de gestionari de poziționare, amintim: *FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout*. Fiecare dintre aceste clase are diverşi constructori şi metode pentru gestiunea proprietăților. Asocierea unui gestionar la un container şi infromații despre gestionarul folosit se face cu metodele următoare:

- `public void` **`setLayout`**`(LayoutManager gestionar)` – setează gestionarul de poziționare al containerului;
- `public LayoutManager` **`getLayout`**`()` – dă o referință la gestionarul de poziționare al containerului;
- `public void` **`doLayout`**`()` – se face poziționarea componentelor din container;

Este posibil să se proiecteze interfața grafică astfel încât pozitionarea componentelor să se facă precizând efectiv coordonatele şi dimensiunile absolute ale componentelor. Pentru aceasta se pot folosi metodele:

- se precizează faptul că nu se foloseşte un gestionar de pozitionare: **`setLayout(null);`**
- se foloseşte metoda de stabilire a poziției şi a dimensiunilor:
  `public void` **`setBounds`**`(int x, int y, int width, int height);`

In continuare se vor descrie (parțial, un minim necesar pentru aplicații simple) subclasele clasei Container: *Panel, Window, Frame, Dialog*. Subclasa *Applet* se va descrie mai târziu.

Clasa Panel (inclusă în clasa Container) se foloseşte pentru a grupa diferite componente. Pentru gestiunea componentelor incluse, se folosesc metodele moştenite de la clase Container. Constructorii acestei clase sunt:

- `public` **`Panel`**`()` – crează un nou panou, folosind gestionarul de poziționare implicit (FlowLayout);
- `public` **`Panel`**`(LayoutManager layout)` – crează un nou panou, cu gestionarul de poziționare specificat prin argument.

Clasa Window (inclusă în clasa Container) generează ferestre de afişare fără margine (chenar) şi fără bara de titlu, deci apar pe ecran ca dreptunghiuri. Pe ecran se face o poziționare absolută, controlul (fereastra) nu se poate închide, deplasa sau redimensiona cu ajutorul mouse-ului. Aşa cum se vede din lista de constructori ai clasei, la creare trebuie să se precizeze un obiect părinte (obiect Frame sau alt obiect Window).

Constructorii clasei sunt:

- `public` **`Window`**`(Frame parinte)` – construieşte un obiect, invizibil, având ca părinte un obiect Frame;
- `public` **`Window`**`(Window parinte)` – construieşte un obiect, invizibil, având ca părinte un obiect Window.

Dintre metode amintim:

- `public void` **`toFront`**`()` – aduce fereastra în față;
- `public void` **`toBack`**`()` – duce fereastra în spatele componentelor existente;
- `public void` **`pack`**`()` – redimensionează fereastra astfel încât să includă toate componentele sale;
- `public void` **`dispose`**`()` – eliberează resursele folosite de fereastră, inclusiv de componentele ei;
- `public Window` **`getOwner`**`()` – dă părintele ferestrei;
- `public Window[]` **`getOwnedWindows`**`()` – dă tabelul ferestrelor incluse;

Clasa Frame este subclasă a clasei Window şi constituie **fereastra standard de implementare a dialogului cu utilizatorul**. La vizualizare apare o fereastră ce are chenar şi o bară (linie) de titlu. Ea nu poate fi inclusă în alta fereastră. In general este folosită ca fereastra principală a aplicației. Intr-o aplicație pot apare şi alte ferestre (secundare). Bara de titlu se află în partea superioara a ferestrei şi conține:

- în partea stângă este un meniu ascuns, cu opțiuni ce permit maximizarea, revenire la dimensiunile normale, iconificarea şi închiderea ferestrei;
- la centru zonei apare titlul ferestrei;
- în partea dreapta apar trei butoane care permit iconificarea, maximizarea/revenirea şi închiderea ferestrei. Acționarea pe butoanele de *maximizare/revenire* şi de *iconificare* are efect imediat (nu genereaza evenimente care trebuie tratate); dar acțiunile de *inchidere* (la fel ca opțiunea din meniul ascuns) generează un eveniment *WindowEvent* care se tratează prin metoda *windowClosing*() a interfeței *WindowListener*, deci această acțiune **trebuie tratată prin program**.

Instantele clasei Frame pot avea un meniu (instanță a clasei MenuBar). Constructorii clasei Frame sunt:

- `public` **`Frame`**`()` – construieşte un frame invizibil şi fără titlu;
- `public` **`Frame`**`(String titlu)` – construieşte un frame invizibil, cu titlul precizat prin argument.

Clasa Frame moşteneşte metodele claselor **Container** şi **Window**. Dintre metodele proprii amintim următoarele:

- `public String` **`getTitle`**`()` – furnizează titlul cadrului;
- `public void` **`setTitle`**`(String title)` – precizează un nou titlu;
- `public void` **`setIconImage`**`(Image image)` – setează pictograma cadrului;
- `public Image` **`getIconImage`**`()` – întoarce imaginea pictogramei cadrului (aplicatiei);
- `public void` **`setMenuBar`**`(MenuBar menu)` – precizează un meniu pentru frame;
- `public MenuBar` **`getMenuBar`**`()` – întoarce meniul asociat obiectului;
- `public void` **`setResizable`**`(boolean resizable)` - setează dacă acest cadru poate sau nu să fie redimensionat cu mouse-ul.
- `public boolean` **`isResizable`**`()` – întoarce *true* dacă acest obiect poate fi redimensionat cu mouse-ul;

In continuare se vor da exemple de controale Frame cu **diverse obiecte de gestionare** a repartizării componentelor pe suprafața de afişare.

Program **awt01a.java**

```
import java.awt.*;
public class awt01a {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
```

```
    Frame f = new Frame ("Prima fereastra");
    // Precizarea modului de pozitionare a componentelor
    f.setLayout(new FlowLayout());  // (1)
    // Creaza sase butoane
    Button b1 = new Button("Primul");
    Button b2 = new Button("Al doilea");
    Button b3 = new Button("Al treilea");
    Button b4 = new Button("Interfata");
    Button b5 = new Button("Java");
    Button b6 = new Button("AWT");
    // Adaugarea butoanelor create la forma
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b4);
    f.add(b6);
    // dimensionarea formei incat sa includa componentele
    // fara aceasta instructiune, nu se vede continutul
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

Dacă se execută acest program se obține fereastra din figura alăturată. După crearea butoanelor şi adăugarea lor la formă, nu s-au precizat adresele unde ele se vor poziționa în cadrul formei. Această aranjare în cadrul formei este făcută de un **gestionar de poziționare FlowLayout**, precizat cu instrucțiunea marcată cu (1) în textul sursă. Dacă se redimensionează fereastra, de exemplu cu mouse-ul, se poate obține vizualizarea din figura alăturată.

Dacă se acționează oricare dintre aceste butoane, sau dacă se apasă cu mouse-ul controlul marcat cu "X" din prima linie a ferestrei, sau se acționează ***Alt/F4*** pe fereastra afişată, atunci aceste acțiuni **nu au efect** deoarece nu s-a scris cod pentru aceste evenimente (**nu s-a precizat ce instrucțiuni se execută la un astfel de eveniment**). In această situație închiderea se poate face cu **Ctrl/C** în **fereastra Command** de unde s-a lansat în execuție acest program.

Dacă se foloseşte gestionarul **BorderLayout**, atunci zona de afişare este divizată în cinci subzone, corespunzătoare centrului şi celor patru puncte cardinale. Fiecare subzonă este identificată printr-o constantă din clasa **BorderLoayout**: NORTH, SOUTH, EAST, WEST, CENTER. La adăugarea unui control la container se precizează şi subzona unde se plasează, într-o subzonă **va apare o singură componentă**. Dacă într-o subzonă se plasează mai multe controale, atunci ele vor trebui grupate într-un panel (în acest nou container se foloseşte un gestionar propriu). Vom modifica programul precedent pentru a folosi gestionarul BorderLayout.

Program **awt01b.java**

```
import java.awt.*;
public class awt01 {
```

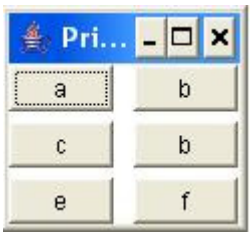
```
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Prima fereastra");
    // Precizarea modului de pozitionare a componentelor
    f.setLayout(new BorderLayout());
    // Crearea butoanelor
    Button b1 = new Button("Primul");
    Button b2 = new Button("Al doilea");
    Button b3 = new Button("Al treilea");
    Button b4 = new Button("Interfata");
    Button b5 = new Button("Java");
    Button b6 = new Button("AWT");
    Panel p = new Panel();
    p.setLayout(new FlowLayout());
    p.add(b2);
    p.add(b3);
    // Adaugarea componentelor la forma
    f.add(b1,BorderLayout.NORTH);
    f.add(p,BorderLayout.SOUTH);
    f.add(b4,BorderLayout.WEST);
    f.add(b5,BorderLayout.CENTER);
    f.add(b6,BorderLayout.EAST);
    // dimensionarea formei
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

La execuția programului, apare prima fereastra din figura alăturată. La o eventuală redimensionare poate apare a doua imagine. Unele subzone se redimensionează numai pe orizontală, altele pe verticală, iar centrul se redimensionează pe ambele direcții.

Cu ajutorul gestionarului **GridLayout** zona de afişare este divizată ca un tabel *cu celule de dimensiuni egale*, într-o celulă se poate repartiza un singur control (simplu sau container). Numărul de linii şi de coloane din tabel se precizează la construirea obiectului gestionar, dar se poate modifica ulterior cu metodele **setRows** şi **setCols**. Distanța dintre celule se poate stabili la construirea obiectului, sau ulterior. Redimensionarea ferestrei duce la redimensionarea celulelor tabelului amintit. In continuare se dă un exemplu pentru acest tip de gestionar şi rezultatul execuției programului.

Program **awt01c.java**

```
import java.awt.*;
public class awt01c {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Prima fereastra");
    // Precizarea modului de pozitionare a componentelor
    // Tabelul are 3 linii şi 2 coloane
    // Separarea pe vert.si oriz.se face cu 10, resp.5
```



```
pixeli
    f.setLayout (new GridLayout (3, 2, 10, 5));
    f.add(new Button("Buton a"));
    f.add(new Button("Buton b"));
    f.add(new Button("Buton c"));
    f.add(new Button("Buton b"));
    f.add(new Button("Buton e"));
    f.add(new Button("Buton f"));
    f.pack ();
    f.setVisible(true);
  }
}
```

Gestionarul GridBagLayout consideră că zona de afişare este divizată ca un tabel cu următoarele caracteristici:

- Numărul de linii şi numărul de coloane este determinat de acest gestionar în funcţie de caracteristicile componentelor care sunt incluse în container;
- Inălţimea liniilor poate diferi;
- Lăţimea coloanelor poate diferi;
- O componentă poate ocupa mai multe celule vecine (reunire de celule pe verticală şi pe orizontală).

Paşii care trebuie parcurşi sunt:

- Se construieşte un obiect gestionar, care se ataşează la container:

```
// Crearea unui obiect de tip Frame
Frame f = new Frame ("Fereastra");
//construirea unui gestionar de tip GridBagLayout
GridBagLayout g = new GridBagLayout();
// Precizarea modului de pozitionare a componentelor
f.setLayout (g);
```

- Se construieşte un obiect de tip **GridBagConstraints**, util la stabilirea unor informaţii despre repartizarea componentelor în container. Dintre proprietăţile clasei *GridBagConstraints* amintim:
  - *gridx*, *gridy* - care precizează celula din tabel de unde începe extragerea componentei (celula din stânga-sus are adresa: 0,0);
  - **gridwidth**, **gridheight** - numărul de linii şi coloane peste care se extrage componenta;
  - **fill** - pentru a preciza faptul că o componentă ocupă tot spaţiul rezervat (prin: gridwidth, gridheight), altfel foloseşte cât are nevoie pentru a desena. Valorile posibile: HORIZONTAL, VERTICAL, BOTH, NONE;
  - **insets** - de tip **Insets**, care precizează distanţele dintre componentă şi laturile suprafeţei de afişare. Există constructorul **Insets**(int top, int left, int bottom, int right).
- Se foloseşte metoda **setConstraints(componenta, c)** a gestionarului pentru a asocia obiectul c de tip **GridBagConstraints** la o componentă ce urmează să se adauge la container:

Program **awt01d.java**

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

public class awt01d {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Fereastra");
    //construirea unui gestionar de tip GridBagLayout
    GridBagLayout g = new GridBagLayout();
    // Precizarea modului de pozitionare a componentelor
```

```
f.setLayout (g);
//crearea unui obiect de tip GridBagConstraints
//pentru precizarea unor restrictii la componentele
//ce se adauga la frame f
GridBagConstraints c = new GridBagConstraints();
// Crearea controalelor si pozitionarea lor
Label c1 = new Label("Numele:");
c.gridx = 0;
c.gridy = 0;
c.gridwidth = 1;
c.gridheight = 1;
g.setConstraints (c1, c);
f.add(c1);

TextField c2 = new TextField("",20);
c.gridx = 1;
c.gridy = 0;
c.gridwidth = 1;
c.gridheight = 1;
g.setConstraints (c2, c);
f.add(c2);

Label c3 = new Label("Prenumele:");
c.gridx = 0;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 1;
g.setConstraints (c3, c);
f.add(c3);

TextField c4 = new TextField("",20);
c.gridx = 1;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 1;
g.setConstraints (c4, c);
f.add(c4);

Button c5 = new Button("Salvare");
c.gridx = 0;
c.gridy = 2;
c.gridwidth = 2;
c.gridheight = 1;
g.setConstraints (c5, c);
f.add(c5);

// crearea obiectului de tratare a evenimentului
f.addWindowListener(new WindowAdapter (){
  public void windowClosing(WindowEvent e) {
    // Terminare program
    System.exit(0);
  }
});
// dimensionarea formei
```

```
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

Pentru un program cu un text mai scurt, se poate construi o metodă ce precizează unele proprietăți ale componentelor. Prin rescrierea exemplului precedent se obține următorul text.

Program **awt01e.java**

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

public class awt01e {
  static Frame f;
  static GridBagLayout g;
  static GridBagConstraints c;

  static void adauga(Component comp,int x, int y, int w, int h) {
    c.gridx = x;
    c.gridy = y;
    c.gridwidth = w;
    c.gridheight = h;
    g.setConstraints(comp, c);
    f.add(comp);
  }

  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    f = new Frame ("Fereastra");
    //construirea unui gestionar de tip GridBagLayout
    g = new GridBagLayout();
    // Precizarea modului de pozitionare a componentelor
    f.setLayout (g);
    //crearea unui obiect de tip GridBagConstraints
    //pentru precizarea unor restrictii la componentele
    //ce se adauga la frame f
    c = new GridBagConstraints();
    c.insets = new Insets (1, 1, 1, 1);
    // Crearea controalelor
    Label c1 = new Label("Numele:");
    adauga(c1,0,0,1,1);

    TextField c2 = new TextField("",20);
    adauga(c2,1,0,1,1);

    Label c3 = new Label("Prenumele:");
    adauga(c3,0,1,1,1);

    TextField c4 = new TextField("",20);
    adauga(c4,1,1,1,1);

    Button c5 = new Button("Salvare");
    adauga(c5,0,2,2,1);
```

```
    // crearea obiectului de tratare a evenimentului
    f.addWindowListener(new WindowAdapter (){
      public void windowClosing(WindowEvent e) {
        // Terminare program
        System.exit(0);
      }
    });
    // dimensionarea formei
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

Următorul exemplu construieşte interfaţa pentru un program de desenare a suprafetelor. Controalele din interfaţă sunt **pozitionate absolut** (se foloseşte o metodă **adaugan**).

Program **awt01f.java**

```
import java.awt.*;
import java.awt.event.*;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.*;
import javax.swing.event.*;

public class awt01f extends Frame{
  JSpinner nrs;
  JSpinner unghi;
  Checkbox animatie;
  Checkbox polpline;
  Checkbox polf;
  Checkbox pols;
  Checkbox c1;
  Checkbox c2;
  Checkbox c3;
  Checkbox c4;
  Checkbox axe;
  JSpinner u1;
  JSpinner u2;
  JSpinner v1;
  JSpinner v2;
  TextField expr;
  Panel b1a;
  Panel b2a;
  Panel b3a;
  double fr=0,fg=0,fb=0;    //culoarea de fond
  double gr1=1,gg1=0,gb1=0; //culoarea grafic 1
  double gr2=0,gg2=0,gb2=1; //culoarea grafic 2

  public void adaugan(Container parinte, Component comp, double x, double
     y, double w, double h) {
    //containerul "parinte" este considerat ca un "grid" cu dimensiunile "zona"
    //se adauga o componeta "comp" cu adresa (x,y) in grid
    //dimensiunile componentei adaugate vor fi: w*zona.width, h*zona.height
    //se foloseste pentru un ontainer cu "setLayout(null)"
    Dimension zona = new Dimension(10,10);
    comp.setBounds((int)(x*zona.width),(int)(y*zona.height),(int)
       (w*zona.width),(int)(h*zona.height));
    parinte.add(comp);
```

```
}

public awt01f(){
  super("Desenarea suprafetelor");
  setLayout(null);

  Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
  setSize(dim);

  Panel p1 = new Panel();
  p1.setLayout(null);
  p1.setBackground(new Color(0x87,0xce,0xfa));
  Label L1 = new Label("Suprafata");
  L1.setFont(new Font("Arial", Font.BOLD, 13));
  //L1.setForeground(Color.red);
  adaugan(p1,L1,1,0,7,2);
  nrs = new JSpinner(new SpinnerNumberModel(1, 1, 47, 1));
  adaugan(p1,nrs,8,0,4,2);

  b1a = new Panel();
  b1a.setBackground(new Color(0x00,0x00,0x00));
  adaugan(p1,b1a,1,3,3,2);
  Button b1 = new Button("Culoare fond");
  adaugan(p1,b1,5,3,10,2);
  b2a = new Panel();
  b2a.setBackground(new Color(0xff,0x00,0x00));
  adaugan(p1,b2a,1,5,3,2);
  Button b2 = new Button("Culoare 1 grafic");
  adaugan(p1,b2,5,5,10,2);
  b3a = new Panel();
  b3a.setBackground(new Color(0x00,0x00,0xff));
  adaugan(p1,b3a,1,7,3,2);
  Button b3 = new Button("Culoare 2 grafic");
  adaugan(p1,b3,5,7,10,2);

  Label L2 = new Label("Ungh.pt.proiectie");
  //L2.setFont(new Font("Arial", Font.BOLD, 13));
  adaugan(p1,L2,1,10,10,2);
  unghi = new JSpinner(new SpinnerNumberModel(45, 30, 75, 1));
  adaugan(p1,unghi,11,10,4,2);

  animatie = new Checkbox("Animatie", true);
  adaugan(p1,animatie,1,13,8,2);

  polpline = new Checkbox("Poligoane pline", true);
  adaugan(p1,polpline,1,15,12,2);

  CheckboxGroup tippolig = new CheckboxGroup();
  polf = new Checkbox("Poligoane 'flat'",tippolig, true);
  adaugan(p1,polf,1,17,12,2);
  pols = new Checkbox("Poligoane 'smooth'",tippolig, true);
  adaugan(p1,pols,1,19,12,2);

  Panel p2 = new Panel();
  p2.setLayout(null);
  p2.setBackground(new Color(0x70,0xb0,0xd0));
  adaugan(p1,p2,1,21,14,10);
  Label L3 = new Label("Culorile poligoanelor:");
  L3.setFont(new Font("Arial", Font.BOLD, 13));
  adaugan(p2,L3,0,0,14,2);
```

```
CheckboxGroup culorip = new CheckboxGroup();
c1 = new Checkbox("Aleator",culorip, true);
adaugan(p2,c1,1,2,12,2);
c2 = new Checkbox("Val.Z,Cul1-Alb",culorip, false);
adaugan(p2,c2,1,4,12,2);
c3 = new Checkbox("Val.Z,Cul2-Alb",culorip, false);
adaugan(p2,c3,1,6,12,2);
c4 = new Checkbox("Val.Z,Cul1-Cul2",culorip, false);
adaugan(p2,c4,1,8,12,2);

axe = new Checkbox("Desenare axe", false);
adaugan(p1,axe,1,32,10,2);

Label a1 = new Label("X");
a1.setBackground(new Color(0xff,0x00,0x00));
adaugan(p1,a1,11,32,1,2);
Label a2 = new Label("Y");
a2.setBackground(new Color(0x00,0xff,0x00));
adaugan(p1,a2,12,32,1,2);
Label a3 = new Label("Z");
a3.setBackground(new Color(0x00,0x00,0xf0));
adaugan(p1,a3,13,32,1,2);

Label L4 = new Label("Intervale pt.parametri:");
L4.setFont(new Font("Arial", Font.BOLD, 13));
adaugan(p1,L4,1,35,14,2);

adaugan(p1,new Label("u:"),1,37,2,2);
u1 = new JSpinner(new SpinnerNumberModel(-10, -50, 50, 0.1));
adaugan(p1,u1,3,37,5,2);
u2 = new JSpinner(new SpinnerNumberModel(10, -50, 50, 0.1));
adaugan(p1,u2,8,37,5,2);

adaugan(p1,new Label("v:"),1,39,2,2);
v1 = new JSpinner(new SpinnerNumberModel(-10, -50, 50, 0.1));
adaugan(p1,v1,3,39,5,2);
v2 = new JSpinner(new SpinnerNumberModel(10, -50, 50, 0.1));
adaugan(p1,v2,8,39,5,2);

TextArea z1 = new TextArea("Daca nu se face animatie", 3,
     40,TextArea.SCROLLBARS_NONE);
z1.append(" se poate folosi mouse-ul");
z1.append(" pentru rotirea desenului");
z1.setFont(new Font("Arial", Font.PLAIN, 11));
z1.setEnabled(false);
adaugan(p1,z1,1,42,14,6);

p1.setBounds(0,35,160,500);
add(p1);

expr = new TextField("");
expr.setBounds(165,35,dim.width-170,25);
expr.setBackground(new Color(0xaa,0xaa,0xaa));
add(expr);

addWindowListener(new WindowAdapter (){
  public void windowClosing(WindowEvent e) {
    // Terminare program
    System.exit(0);
  }
});
```

```
    setVisible(true);
  }

  public static void main (String args[])
     {awt01f f =  new awt01f();
     }

}
```

Clasa Dialog este subclasă a clasei Window. Instanțele acestei clase se folosesc pentru dialog cu utilizatorul: pentru a transmite un mesaj, pentru a primi un răspuns. Aşa cum se vede din constructorii acestei clase, o astfel de fereastră este dependentă de o altă fereastră părinte. Când o fereastră este distrusă, atunci şi ferestrele sale fiu (inclusiv ferestrele de dialog) sunt distruse. Fereastra de dialog poate fi modală sau nemodală. Dacă este modală, ea blochează accesul la ferestrele părinte. Implicit, fereastra de dialog este nemodală. In momentul creării, fereastra este invizibilă.

Constructorii clasei sunt:

- `public` **`Dialog`**`(Dialog parinte)` – se crează o nouă fereastră de dialog fără titlu şi cu părintele precizat;
- `public` **`Dialog`**`(Dialog parinte, String title)` – se crează o nouă fereastră de dialog, cu părintele şi titlul specificate;
- `public` **`Dialog`**`(Dialog parinte, String title, boolean modal)` – se crează o nouă fereastră de dialog, cu părintele şi titlul din argumente, precum şi precizarea faptului că fereastra este modală sau nu;
- `public` **`Dialog`**`(Frame parinte)` – se crează o nouă fereastră de dialog fără titlu, având ca părinte un frame;
- `public` **`Dialog`**`(Frame parinte, boolean modal)` – se crează o nouă fereastră de dialog fără titlu, precizând frame-ul părinte şi dacă este sau nu fereastră modală;
- `public` **`Dialog`**`(Frame parinte, String title)` – se crează o nouă fereastră de dialog, cu frame-ul parinte şi titlul specificate;
- `public` **`Dialog`**`(Frame parinte, String title, boolean modal)` – se crează o nouă fereastră de dialog, cu frame-ul părinte şi titlul din argumente, precum şi precizarea faptului că fereastra este modală sau nu.

Clasa Dialog moşteneşte metodele claselor Container şi Window. Dintre metodele proprii amintim:

- `public void` **`setModal`**`(boolean b)` – setează dacă fereastra este sau nu fereastă modală;
- `public boolean` **`isModal`**`()` – întoarce *true* daca fereastra este modală, sau *fals* în caz contrar;
- `public void` **`setTitle`**`(String title)` – setează un titlu pentru fereastră;
- `public String` **`getTitle`**`()` – întoarce titlul ferestrei;
- `public void` **`setResizable`**`(boolean resizable)` – setează dacă fereastra poate sau nu să fie redimensionată.
- `public boolean` **`isResizable`**`()` – întoarce *true* dacă fereastra poate fi redimensionată cu mouse-ul;

In aplicații apare necesitatea de a selecta un fişier pentru deschidere sau alegerea unui fişier pentru salvare. In pachetul **java.awt** există **clasa FileDialog**, subclasă a clasei **Dialog**, pentru a deschide o fereastră de dialog în care se rezolvă această poroblemă. Constructorii clasei sunt:

- **`FileDialog`**`(Frame parinte)` - crează o fereastră cu titlul vid, pentru deschiderea unui fişier;

- **`FileDialog`**`(Frame parinte, String titlu)` - crează o fereastră cu titlul precizat, pentru deschiderea unui fişier;
- **`FileDialog`**`(Frame parinte, String titlu, boolean mod)` - crează o fereastră cu titlul precizat. Parametrul " mod" precizează, prin constantele LOAD şi SAVE (ale acestei clase) modul în care se foloseşte fereastra: pentru deschiderea sau salvarea unui fişier.

După creare, fereastra de dialog creată nu este vizibilă. Pentru vizualizare se poate folosi metoda **show**() - pentru o fereastră modală, sau **setVisible**(true) - pentru o fereastră nemodală. După alegerea unui fişier, fereastra nu este vizibilă.

Dintre metodele proprii ale acestei clase amintim:

- **`getDirectory`**`(), `**`getFile`**`()` - furnizează numele directorului şi a fişierului stabilit prin dialog;
- **`setFilenameFilter`**`(FilenameFilter filter)` - stabileşte un criteriu de filtrare;
- **`setMode`**`(int mode)` - stabileşte modul de folosire a ferestrei, prin constantele LOAD şi SAVE;
- **`setDirectory`**`(String dir), `**`setFile`**`(String dir)` - stabilirea numelui de director şi fişier cu care se începe dialogul.

In continuare se va face o scurtă prezentare a claselor ce generează obiecte într-o interfață cu utilizatorul.

Prin instanțierea **clasei Label** se obțin obiecte grafice care conțin o singură linie de text. In zona rezervată obiectului, textul poate fi aliniat la stânga, la dreapta sau pe centru. Pentru această aliniere, clasa contine trei campuri statice care indică modul de aliniere.

- `public static final int LEFT` – aliniere la stânga;
- `public static final int CENTER` – aliniere la centru;
- `public static final int RIGHT` – aliniere la dreapta.

Constructorii acestei clase sunt:

- `public` **`Label`**`()` - construieşte o etichetă vidă (fără text);
- `public` **`Label`**`(String text)` – construieşte o etichetă cu textul dat ca argument, aliniat la stânga;
- `public` **`Label`**`(String text, int alignment)` – construieşte o etichetă cu textul şi alinierea date ca argumente. Al doilea argument are una din valorile: `Label.LEFT, Label.CENTER` sau `Label.RIGHT`

Dintre metodele clasei Label amintim:

- `public void` **`setText`**`(String text)` – setează textul din obiect;
- `public String` **`getText`**`()` – dă textul din obiect;
- `public void` **`setAlignment`**`(int alignment)` – precizează alinierea textului etichetei.
- `public int` **`getAlignment`**`()` – dă alinierea textului etichetei;

Clasa TextComponent este superclasa claselor **TextField** şi **TextArea (**clase care permit introducerea/editarea unor texte de la tastatură). Pe lângă metodele moştenite de la clasa Component amintim şi următoarele metode proprii:

- `public synchronized String` **`getText`**`()` – întoarce textul conținut în componentă;
- `public synchronized void` **`setText`**`(String text)` – pune un text în componentă;
- `public synchronized String` **`getSelectedText`**`()` – întoarce subşirul selectat;
- `public synchronized void` **`setEditable`**`(boolean b)` – precizează dacă textul din obiect este editabil sau nu;

- `public boolean` **`isEditable`**`()` – întoarce *true* sau *false* după cum textul din obiect poate fi editat sau nu;
- `public synchronized void` **`setSelectionStart`**`(int selectionStart)` – setează poziția din text de la care începe subşirul selectat;
- `public synchronized int` **`getSelectionStart`**`()` – întoarce poziția de la care începe subşirul selectat;
- `public synchronized int` **`getSelectionEnd`**`()` – întoarce poziția din text unde se termină subşirul selectat;
- `public synchronized void` **`setSelectionEnd`**`(int selectionEnd)` – setează poziția din text unde se termină subşirul selectat;
- `public synchronized void` **`select`**`(int selectionStart, int selectionEnd)` – setează inceputul şi sfârsitul subâirului selectat;
- `public synchronized void` **`selectAll`**`()` – selectează întregul text din componentă.

Cu **clasa TextField** se poate edita un şir de caractere aflat pe o singură linie. In această componentă se pot introduce caractere de la tastatură şi se poate edita textul existent (ştergeri, adăugări, modificări). Dacă se doreşte ca textul să fie păstrat secret (de exemplu pentru o parolă), atunci în fereastra de pe ecran, în locul fiecărui caracter din text se afişează un caracter de ecou unic, stabilit prin program. Constructorii clasei sunt:

- `public` **`TextField`**`()` – construieşte o nouă componentă;
- `public` **`TextField`**`(String text)` – construieşte o nouă componentă, care conține şirul dat ca argument;
- `public` **`TextField`**`(int columns)` – construieşte o nouă componentă, având lungimea dată ca argument;
- `public` **`TextField`**`(String text, int columns)` – construieşte o nouă componentă, care conține şirul şi lungimea din argument.

Clasa TextField moşteneşte metodele clasei TextComponent. Dintre metodele proprii amintim:

- `public int` **`getColumns`**`()` – întoarce lungimea zonei de text (numărul de caractere);
- `public void` **`setColumns`**`(int columns)` – setează lungimea zonei de text;
- `public void` **`setEchoChar`**`(char c)` – setează caracterul care va fi folosit pentru ecou;
- `public char` **`getEchoChar`**`()` – întoarce caracterul folosit pentru ecou;

Pentru editarea unui text care se afişează în mai multe linii ecran se poate folosi **clasa TextArea**. Dacă textul este mai mare decât zona ecran alocată unei astfel de componente, atunci pot apare bare de defilare, pentru deplasarea pe verticală sau orizontală. Constructorii clasei sunt:

- `public` **`TextArea`**`()` – construieşte o zonă vidă;
- `public` **`TextArea`**`(String text)` – construieşte o zonă ce conține textul dat ca argument;
- `public` **`TextArea`**`(int rows, int columns)` – construieşte o zonă de text vidă, cu numărul de linii şi coloane date ca argumente;
- `public` **`TextArea`**`(String text, int rows, int columns)` – construieşte o zonă precizând conținutul şi dimensiunile;
- `public` **`TextArea`**`(String text, int rows, int columns, int scrollbars)` – construieşte o zonă precizând conținutul, dimensiunile şi barele de defilare. Ultimul parametru poate avea una din următoarele valori ale clasei TextArea: SCROLLBARS_BOTH, SCROLLBARS_VERTICAL_ONLY, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_NONE.

Dintre metodele propii acestei clase amintim:

- `public void` **`append`**`(String s)` – adaugă şirul *s* la sfârşitul textului existent;
- `public void` **`insert`**`(String s, int pos)` – inserează şirul *s* în textul existent, începând de la poziția *pos*;
- `public void` **`replaceRange`**`(String s, int start, int end)` – înlocuieşte prin şirul s subtextul din zona aflat între pozițiile start şi end;
- `public void` **`setRows`**`(int rows)` – setează înălțimea zonei de text, dată ca număr de linii;
- `public int` **`getRows`**`()` – întoarce înălțimea zonei, dată ca număr de linii;
- `public void` **`setColumns`**`(int columns)` – setează lățimea zonei de text, dată ca număr de coloane;
- `public int` **`getColumns`**`()` – întoarce lățimea zonei de text, dată ca număr de coloane.

Clasa Button permite definirea unui buton de comandă, pe care apare o etichetă (şir de caractere). Contructorii clasei sunt:

- `public` **`Button`**`()` – construieşte un buton fără etichetă;
- `public` **`Button`**`(String label)` – construieşte un buton având eticheta precizată ca argument;

Cu **clasa Checkbox** se poate construi un "comutator", sau "casetă de validare", adică un control grafic ce poate avea două stări. Mai multe obiecte de tip checkbox se pot grupa într-un container de tip **CheckboxGroup** şi vor forma butoanele radio (în acest caz numai un singur control din grup poate să fie marcat (are valoarea true), celelalte controale vor fi nemarcate (au valoarea *false*).
Există următorii constructori:

- `public` **`Checkbox`**`()` – crează o instanță Checkbox fără etichetă;
- `public` **`Checkbox`**`(String label)` – crează o instanță Checkbox cu eticheta dată ca argument;
- `public` **`Checkbox`**`(String label, boolean state)` – crează o instanță Checkbox cu eticheta şi starea date ca argumente;
- `public` **`CheckboxGroup`**`()` – construieşte un grup vid (fără controale);
- `public` **`Checkbox`**`(String label, boolean state, CheckboxGroup group)` – crează o instanță Checkbox cu eticheta, starea precizate şi care este inclus în grupul precizat ca al treilea argument.

Dintre metode claselor Checkbox şi CheckboxGroup amintim:

- `public void` **`setState`**`(boolean state)` – setează starea controlului;
- `public boolean` **`getState`**`()` – dă starea controlului;
- `public void` **`setLabel`**`(String label)` – setează eticheta;
- `public String` **`getLabel`**`()` – dă eticheta;
- `public void` **`setCheckboxGroup`**`(CheckboxGroup group)` – setează apartenența controlului la un grup;
- `public CheckboxGroup` **`getCheckboxGroup`**`()` – dă o referință la grupul de care aparține controlul;
- `public void` **`setSelectedCheckbox`**`(Checkbox box)` – setează controlul precizat.
- `public Checkbox` **`getSelectedCheckbox`**`()` – dă o referință la caseta selectată;

Pentru definirea "barelor de defilare (sau derulare)" se poate folosi **clasa Scrollbar**. Bara vizualizată de control are poziția orizontală sau verticală. Deplasarea cursorului din interiorul componentei se poate face cu ajutorul mouse-ului, între o valoare minimă şi o valoare maximă.
Clasa are două constante care precizează orientarea constrolului:

- `public static final int HORIZONTAL` – constanta precizează orientarea orizontală a barei de derulare;
- `public static final int VERTICAL` – constanta precizează orientarea verticală a barei.

Clasa are următorii constructori:

- `public` **`Scrollbar`**`()` – crează o bară de derulare cu următoarele caracteristici: orientarea verticală, valoarea minimă 0, valoarea maximă 100, valoarea curentă (poziția cursorului) 0, incrementul 1 (modificarea valorii cand se face click pe o săgeată), incrementul de bloc 10 (modificarea valorii când se face click pe porțiunea dintre cursor şi săgeată);
- `public` **`Scrollbar`**`(int orientation)` – la fel ca în cazul precedent, dar orientarea este precizată ca argument;
- `public` **`Scrollbar`**`(int orientation, int value, int visible, int minim, int maxim)` – crează o bară de defilare pentru care orientarea, valoarea curentă, vizibilitatea, valoarea minimă şi valoarea maximă sunt date ca argumente;

Dintre metodele clasei amintim:

- `public void` **`setOrientation`**`(int orientation)` – setează orientarea barei;
- `public int` **`getValue`**`()` – dă valoarea curentă;
- `public void` **`setValue`**`(int newValue)` – setează valoarea curentă;
- `public void` **`setMinimum`**`(int newMinimum)` – setează valoarea minimă;
- `public void` **`setMaximum`**`(int newMaximum)` – setează valoarea maximă;
- `public void` **`setUnitIncrement`**`(int newIncrement)` – setează incrementul unitar;
- `public void` **`setBlockIncrement`**`(int newIncrement)` – setează increnementul de bloc;
- `public int` **`getOrientation`**`()` – dă orientarea barei;
- `public int` **`getMinimum`**`()` – dă valoarea minimă;
- `public int` **`getMaximum`**`()` – dă valoarea maximă;
- `public int` **`getVisibleAmount`**`()` – dă lungimea cursorului barei;
- `public void` **`setVisibleAmount`**`(int newAmount)` – setează lungimea cursorului;
- `public int` **`getUnitIncrement`**`()` – dă incrementul unitar (când se face click pe o săgeată);
- `public int` **`getBlockIncrement`**`()` – dă incrementul de bloc (când se face click între cursor şi săgeată);
- `public void setValues(int value, int visible, int minimum, int maximum)` – setează valorile trecute ca **argumente**.

Clasa List permite gestiunea listelor, din care se poate selecta una sau mai multe opțiuni. Pentru componenta de acest tip este rezervată o zonă pe ecran. Dacă lista nu încape în această zonă, atunci poate apare o bară de defilare verticală. Constructorii clasei sunt:

- `public` **`List`**`()` – crează o nouă listă, ce are patru linii vizibile şi nu permite selecții multiple;
- `public` **`List`**`(int rows)` – crează o listă cu un număr de linii vizibile precizat ca argument şi care nu permite selecții multiple;
- `public` **`List`**`(int rows, boolean multipleMode)` – crează o listă derulantă cu un număr de linii vizibile precizat ca argument. Al doilea argument precizează dacă se admit selectii multiple.

Dintre metodele clasei amintim:

- `public int` **`getItemCount`**`()` – întoarce numărul de elemente din listă;
- `public String` **`getItem`**`(int index)` – întoarce elementul de pe poziția *index*;
- `public String[]` **`getItems`**`()` – întoarce un tablou ce conține toate elementele din listă;
- `public void` **`add`**`(String item)` – adaugă la listă elementul dat ca argument;

- `public void` **`add`**`(String item, int index)` – adaugă la listă elementul dat ca argument pe poziția precizată de al doilea argument;
- `public void` **`replaceItem`**`(String newValue, int index)` – înlocuieşte elementul de pe pozitia *index* cu noul element dat ca prim argument;
- `public void` **`remove`**`(String item)` – elimină din listă prima apariție a elementului *item*;
- `public void` **`remove`**`(int index)` – elimină din listă elementul de pe poziția *index*;
- `public void` **`removeAll`**`()` – elimină toate elementele din listă;
- `public int` **`getSelectedIndex`**`()` – întoarce indicele elementului selectat sau -1, dacă nu există element selectat;
- `public int[]` **`getSelectedIndexes`**`()` – întoarce un vector ce conține indicii elementelor selectate;
- `public String` **`getSelectedItem`**`()` – întoarce elementul selectat, sau *null* dacă nu există element selectat;
- `public String[]` **`getSelectedItems`**`[]` – întoarce tabloul elementelor selectate;
- `public void` **`select`**`(int index)` – selectează elementul de pe poziția *index*;
- `public void` **`deselect`**`(int index)` – deselectează elementul de pe poziția *index*;
- `public boolean` **`isIndexSelected`**`(int index)` - întoarce *true* dacă elementul de pe poziția *index* este selectat, şi *fals* în caz contrar;
- `public int` **`getRows`**`()` – întoarce numărul de linii vizibile din listă;
- `public boolean` **`isMultipleMode`**`()` – întoarce *true* dacă este permisă selecția multiplă;
- `public void` **`setMultipleMode`**`(boolean b)` – setează posibilitatea de selecție multiplă;

Clasa Choice permite gestiunea listelor de elemente din care numai una poate fi selectată, care este singura vizibilă, celelalte elemente sunt ascunse. Cu mouse-ul se poate vizualiza lista şi se poate selecta un nou element. Clasa are un singur constructor:

- `public` **`Choice`**`()` – crează o listă vidă de elemente;

Dintre metodele clasei amintim următoarele:

- `public int` **`getItemCount`**`()` – întoarce numărul de elemente din listă;
- `public String` **`getItem`**`(int index)` – întoarce elementul de pe poziția *index*;
- `public void` **`add`**`(String item)` – adaugă la listă elementul *item*;
- `public void` **`insert`**`(String item, int index)` – inserează elementul *item* pe poziția *index*;
- `public void` **`remove`**`(String item)` – elimină din listă elementul *item*;
- `public void` **`remove`**`(int position)` – elimină elementul de pe poziția *position*;
- `public void` **`removeAll`**`()` – elimină toate elementele din listă;
- `public String` **`getSelectedItem`**`()` – întoarce elementul selectat;
- `public int` **`getSelectedIndex`**`()` – întoarce indicele elementul selectat;
- `public Object[]` **`getSelectedObjects`**`()` – întoarce un tablou cu o singură poziție ce conține, elementul selectat;
- `public void` **`select`**`(int pos)` – selectează elementul de pe poziția *pos*;
- `public void` **`select`**`(String str)` – selectează elementul care este egal cu valoarea argumentului;

Dacă se foloseşte o interfață grafică pentru dialogul utilizatorului cu aplicația, atunci apare ***programarea orientată pe evenimente*** **(*Event-Oriented Programming*)**, sau ***programare ghidată de evenimente*** **(*Event Driven Programming*)**. Orice acțiune efectuată de operator generează un eveniment (apasarea sau eliberarea unei taste - de la tastatură, deplasarea mouse-ului, apăsarea sau eliberarea unui

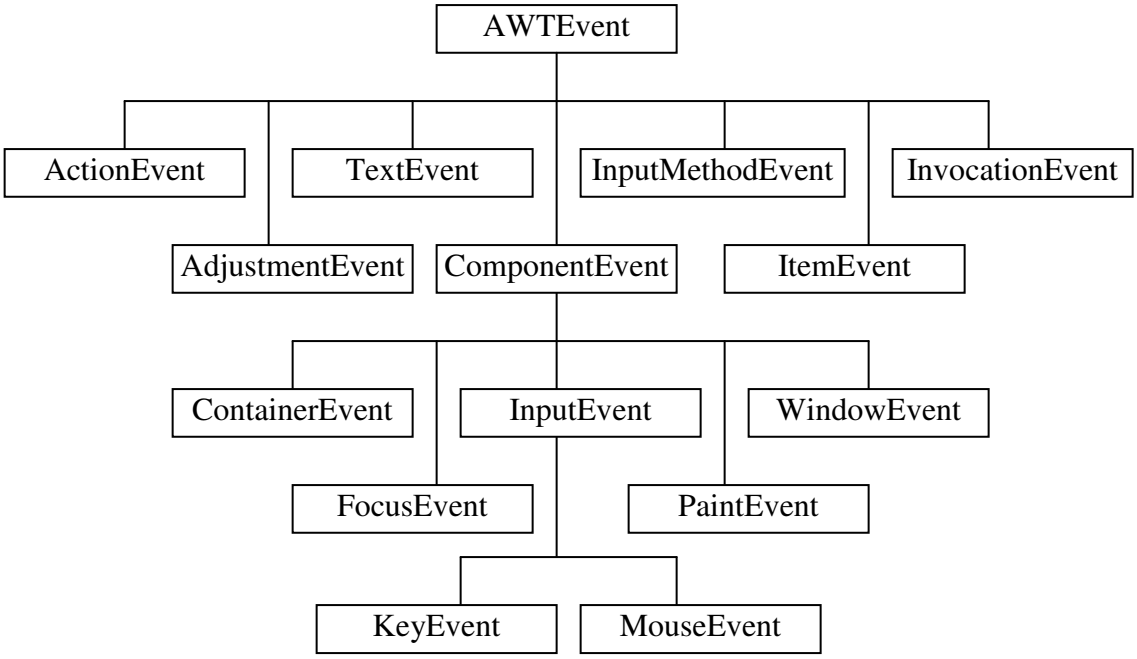
buton de mouse, deschiderea sau închiderea unei ferestre, efectuarea unui clic de mouse pe o componentă din interfață, intrarea/părăsirea cursorului de mouse în zona unei componente, etc.). Există şi evenimente care nu sunt generate de utilizatorul aplicației. Un eveniment **poate să fie tratat** prin execuția unui modul de program.

Pentru tratarea evenimentelor, începând cu JDK 1.1, se foloseşte un model bazat pe *delegare* (*Delegation Event Model*). In acest model se disting trei categorii de *obiecte* utilizate la tratarea *evenimentelor*:

- *surse de evenimente* (*Event Source*) - acele obiecte care generează evenimente;
- *evenimentele propriu-zise* (*Events*), care sunt tot *obiecte* (generate de *surse* şi recepționate de *consumatori).*
- *consumatori* sau *ascultători de evenimente* (*Event Listener*) - acele obiecte care recepționează şi tratează evenimentele. Fiecare consumator trebuie să fie *înregistrat* la sursa de eveniment. Prin acest procedeu se asigură că sursa cunoaşte toți consumatorii la care trebuie să transmită evenimentele pe care le generează.

Modelul "delegarii" presupune că sursa (un obiect) transmite evenimentele generate de ea **numai** către obiectele consumatori, care s-au înregistrat la sursa respectiva a evenimentului. Un obiect **consumator** recepționează evenimente numai de la obiectele sursă la care s-a înregistrat.

Evenimentele posibile sunt obiecte. Clasele din care se generează aceste obiecte depind de componenta care a generat evenimentul. Pentru fiecare tip de eveniment posibil *există o clasă* care instanțiază un obiect pentru acest eveniment, de exemplu, pentru acționarea unui buton de comandă există clasa *ActionEvent*. Toate aceste clase formează o *ierarhie de clase,* care are ca rădăcină clasa *AWTEvent*. Clasele şi interfețele acestei ierarhii sunt grupate în pachetul ***java.awt.event,*** *cu excepția clasei AWTEvent* (rădăcina acestei ierarhii), care se află în pachetul ***java.awt***. Câteva dintre clasele din aceasta ierarhie sunt date în figura următoare.



Semnificația tipurilor de evenimente se află în tabelul următor:

Eveniment	**Descriere**

ActionEvent	Acționarea unei componente (ex. buton de comandă)
AdjustmentEvent	Modificarea valorii într-un scroolbar
ComponentEvent	Deplasare, ascundere, vizualizare, redimensionare
ContainerEvent	Adăugarea sau eliminarea de componente (se breferă la un container)
FocusEvent	Primeşte/pierde focalizarea
ItemEvent	Schimbarea stării unei componente
KeyEvent	Apăsare, eliberare, apăsare tastă
MouseEvent	Click, mutare mouse
TextEvent	Schimbarea textului
WindowEvent	Minimizare, maximizare, redimensionare fereastră

Pentru a preciza consumatorii (receptorii, interceptorii) de evenimente se folosesc interfețele, incluse într-o ierarhie asemănătoare cu cea de mai sus, iar denumirea interfețelor seamănă cu denumirea claselor care definesc evenimente. Astfel, pentru a intercepta un **eveniment ActionEvent** se foloseşte **interfata ActionListener**, pentru un **eveniment ContainerEvent** se foloseste **interfața ContainerListener**, pentru un **eveniment ComponentEvent** se foloseste o **interfață ComponentListener**, etc. La rădăcina ierarhiei se află interfața **EventListener**. Fiecare dintre aceste interfețe are un număr de metode ce se vor apela la apariția unui eveniment. In tabelul următor se precizează metodele interfețelor.

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)

In tabelul următor sunt precizate evenimentele suportate de unele componente.

Componenta	Evenimente suportate de componentă
Button	ActionEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Checkbox	ItemEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
CheckboxMenuItem	ActionEvent, ItemEvent
Choice	ItemEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Component	FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Container	ContainerEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Dialog	ContainerEvent, WindowEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
FileDialog	ContainerEvent, WindowEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Frame	ContainerEvent, WindowEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Label	FocusEvent, KeyEvent, MouseEvent, ComponentEvent
List	ActionEvent, FocusEvent, KeyEvent, MouseEvent, ItemEvent, ComponentEvent
Menu	ActionEvent
MenuItem	ActionEvent
Panel	ContainerEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
PopupMenu	ActionEvent
Scrollbar	AdjustmentEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
TextArea	TextEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
TextComponent	TextEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
TextField	ActionEvent, TextEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Window	ContainerEvent, WindowEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent

Intr-o aplicație care se scrie este necesară implementarea interfețelor folosite prin definirea unor clase utilizator. In aceste clase trebuie să se descrie metodele din interfața pe care o implementează. De exemplu:

```
class ClickButoane implements ActionListener {
  public void actionPerformed(ActionEvent e){
  // continutul metodei
  }
}

class ModificContainer implements ContainerListener{
  public void componentAdded(ContainerEvent e) {
  }
  public void componentRemoved(ContainerEvent e) {
  }
}
```

Deoarece mai multe interfețe pot să fie implementate într-o singură clasă, putem avea situația următoare:

```
class Tratare implements ActionListener, ContainerListener {
  public void actionPerformed(ActionEvent e){
  }
  public void componentAdded(ContainerEvent e) {
  }
  public void componentRemoved(ContainerEvent e) {
  }
}
```

In tabelul următor se precizează tipurile de evenimente generate de unele componente şi interfețele care trebuie implementate pentru receptorii acestor evenimente. Evenimentele unei superclase (după cum se vede din figura 1) sunt şi pentru subclasele acesteia.

Componenta	**Evenimentul**	**Interfața**
Component	ComponentEvent FocusEvent KeyEvent MouseEvent	ComponentListener FocusListener KeyListener MouseListener, MouseMotionListener
Container	ContainerEvent	ContainerListener
Window	WindowEvent	WindowListener
Button List MenuItem TextField	ActionEvent	ActionListener
Choice Checkbox List CheckboxMenuItem	ItemEvent	ItemListener
Scrollbar	AdjustmentEvent	AdjustmentListener
TextField TextArea	TextEvent	TextListener

Un anumit eveniment poate apare pentru mai multe componente (de exemplu, *ActionEvent* apare pentru *Button, List, MenuItem, TextField*). La implementarea unei metode din interfața ce trateaza acest tip de eveniment (de exemplu "***actionPerformed(ActionEvent e)***"), trebuie să se ştie care este sursa evenimentului. Pentru aceasta se poate proceda astfel:

- Se foloseşte metoda "*public Object* ***getSource****()*" a evenimentului;
- Se folosesc metode ale evenimentului generat, de exemplu , evenimentul *ActionEvent* are metoda *GetActionCommand* care furnizează eticheta controlului pe care s-a actionat.

După cum am spus mai sus, "*fiecare consumator trebuie să fie înregistrat la sursa de eveniment*". După ce s-a descris clasa care implementează o interfață, prin instanțierea acesteia se obține un obiect "listener", iar acest obiect trebuie înregistrat în lista ascultătorilor. Acest lucru se realizează printr-o metodă de forma următoare:

- `numecontrol.addXXXListener(obiect_listener),`

unde "XXX" este tipul evenimentului.

Eliminarea unui obiect "listener" din lista ascultătorilor unui control se face astfel:

- `numecontrol.removeXXXListener(obiect_listener),`

Următorul program reia exemplul **awt01a.java** şi tratează evenimentul de click pe un buton de comandă. La acest eveniment se trece în titlul ferestrei (obiect Frame) textul de pe butonul apăsat.

Program **awt02a.java**

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

class ClickButoane implements ActionListener {
  Frame f;

  public ClickButoane(Frame f){
    this.f = f;
  }
  public void actionPerformed(ActionEvent e){
    String s=e.getActionCommand();
    f.setTitle(s);
  }
}

public class awt02a {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Fereastra");
    // Precizarea modului de pozitionare a componentelor
    f.setLayout (new FlowLayout());
    // Crearea butoanelor
    Button b1 = new Button("Primul");
    Button b2 = new Button("Al doilea");
    Button b3 = new Button("Al treilea");
    Button b4 = new Button("Interfata");
    Button b5 = new Button("Java");
    Button b6 = new Button("AWT");
    // Adaugarea butoanelor create la forma
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b4);
    f.add(b6);
    //crearea obiectului de ascultare a evenimentelor
    ClickButoane listener = new ClickButoane(f);
    // adaugarea obiectului la lista ascultatorilor
    b1.addActionListener(listener);
    b2.addActionListener(listener);
    b3.addActionListener(listener);
    b4.addActionListener(listener);
    b5.addActionListener(listener);
    b6.addActionListener(listener);
    // dimensionarea formei
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

Dacă o clasă utilizator trebuie să implementeze o interfață, atunci **trebuie descrise toate metodele din interfață**, chiar dacă unele nu se vor folosi. De exemplu, pentru interfața

WindowListener trebuie scris cod pentru fiecare din cele şapte metode existente în interfață. Următorul program tratează evenimentul de închidere a ferestrei.

Program **awt02b.java**

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

class Fereastra implements WindowListener {
  // 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 awt02b {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Fereastra");
    // Precizarea modului de pozitionare a componentelor
    f.setLayout (new FlowLayout());
    // Crearea butoanelor
    Button b1 = new Button("Primul");
    Button b2 = new Button("Al doilea");
    Button b3 = new Button("Al treilea");
    // Adaugarea butoanelor create la forma
    f.add(b1);
    f.add(b2);
    f.add(b3);
    //crearea obiectului de ascultare a evenimentelor
    Fereastra listener = new Fereastra();
    // adaugarea obiectului la lista ascultatorilor
    f.addWindowListener(listener);
    // dimensionarea formei
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

Aşa după cum se observă din tabelul precedent, dintre interfețele Listener amintite şapte interfețe au mai mult de o metodă. Pentru a nu fi nevoie să se descrie toate metodele pentru unele implementări al acestor interfețe, ca în exemplul precedent, se pot implementa anumite clase abstracte numite **adaptor**. Pentru o interfață **XXXListener** există un **adaptor** (o clasă) cu numele **XXXAdapter**, care implementează o interfață fără să scrie cod pentru metode. In clasa utilizator putem să extindem adaptorii şi să redefinim numai metodele care sunt utile în aplicație. In tabelul următor se precizează adaptorii existenți pentru interfețe.

ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter

MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

Programul următor este o modificare a programului precedent prin extinderea unui adaptor în loc de implementarea unei interfețe.

Program **awt02c.java**

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

class FereastraA extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    // Terminare program
    System.exit(0);
  }
}

public class awt02c {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Fereastra");
    // Precizarea modului de pozitionare a componentelor
    f.setLayout (new FlowLayout());
    // Crearea butoanelor
    Button b1 = new Button("Primul");
    Button b2 = new Button("Al doilea");
    Button b3 = new Button("Al treilea");
    // Adaugarea butoanelor create la forma
    f.add(b1);
    f.add(b2);
    f.add(b3);
    //crearea obiectului de ascultare a evenimentelor
    FereastraA listener = new FereastraA();
    // adaugarea obiectului la lista ascultatorilor
    f.addWindowListener(listener);
    // dimensionarea formei
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

In programul precedent am putea înlocui următoarele două instrucțiuni:

```
FereastraA listener = new FereastraA();
f.addWindowListener(listener);
```

prin un singură, deoarece obiectul listener se foloseşte o singură dată:

```
f.addWindowListener(new FereastraA());
```

Se poate simplifica şi mai mult, să nu definim clasa FereastraA, iar definiția acesteia să o mutăm în argumentul metodei addWindowListener (se obține o **clasă anonimă**). Cu această transformare se obține programul următor.

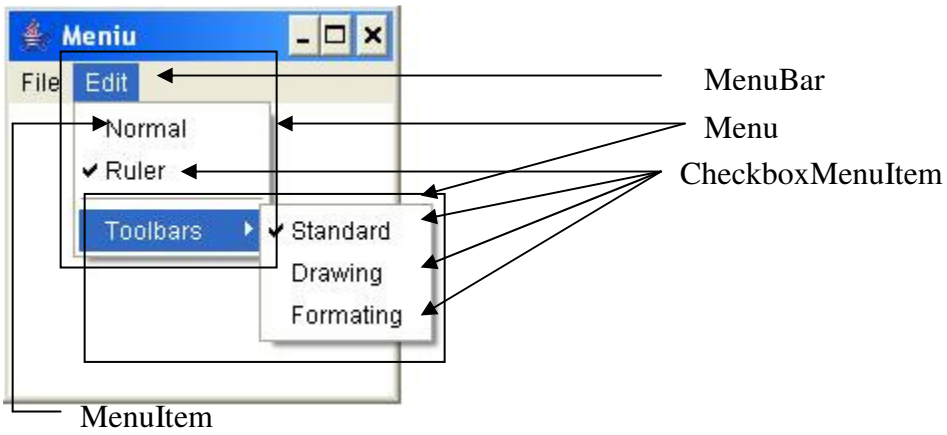
Program **awt02d.java**

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

public class awt02d {
  public static void main ( String args []) {
    // Crearea unui obiect de tip Frame
    Frame f = new Frame ("Fereastra");
```

```
    // Precizarea modului de pozitionare a componentelor
    f.setLayout (new FlowLayout());
    // Crearea butoanelor
    Button b1 = new Button("Primul");
    Button b2 = new Button("Al doilea");
    Button b3 = new Button("Al treilea");
    // Adaugarea butoanelor create la forma
    f.add(b1);
    f.add(b2);
    f.add(b3);
    // crearea obiectului de tratare a evenimentului
    f.addWindowListener(new WindowAdapter (){
      public void windowClosing(WindowEvent e) {
        // Terminare program
        System.exit(0);
      }
    });
    // dimensionarea formei
    f.pack ();
    // Afisarea fereastrei
    f.setVisible(true);
  }
}
```

La o aplicație se poate defini cel mult un meniu fix (meniu bară, meniu orizontal), care se afişează în partea superioară a ferestrei (interfeței cu utilizatorul) şi care este o **instanță a clasei MenuBar**. Opțiunile dintr-un meniu sunt instanțe ale clasei MenuItem sau CheckboxMenuItem şi pot avea un nou meniu asociat (meniu derulant) - gestionat cu clasa Menu. In afara acestui meniu fix, permanent, se pot defini şi meniuri de context (meniuri popup), care se activează cu un right-click de mouse. Pentru alegerea rapidă a unei opțiuni dintr-un meniu se poate folosi o combinație de taste. Aceste comenzi rapide se pot gestiona cu ajutorul clasei MenuShortcut. In fig.2 este prezentată ierarhia de clase care permite gestiunea acestor componente.



- Rădăcina ierarhiei este **clasa abstractă MenuComponent**.
  Dintre metodele acestei clase amintim:
- `public Font` **`getFont`**`()` – dă fontul componentei de meniu;
- `public void` **`setFont`**`(Font f)` – setează fontul care va fi folosit pentru componenta de meniu;

Meniul bară (care apare în partea superioară a ferestrei) se gestionează cu **clasa MenuBar** şi conține unul sau mai multe meniuri (instanțe ale clasei Menu). Constructorul clasei este:

- `public` **`MenuBar`**`() –` creaza un meniu bară vid

Pentru ataşarea unui meniu bară la o fereastră (la un Frame) se foloseşte metoda:

- `public void` **`setMenuBar`**`(MenuBar m)`

Dintre metodele proprii ale clasei MenuBar amintim:

- `public Menu` **`add`**`(Menu m) –` adauga la meniul bară curent un nou meniu şi întoarce o referinta la acesta;
- `public void` **`remove`**`(int index) –` elimina din meniul bară meniul cu indicele precizat;
- `public void` **`remove`**`(MenuComponent m) –` elimina componenta de meniu m;
- `public int` **`getMenuCount`**`() –` întoarce numarul de meniuri din meniul bară;
- `public Enumeration` **`shortcuts`**`() –` întoarce o enumerare a comenzilor rapide existente în meniul bară;
- `public MenuItem` **`getShortcutMenuItem`**`(MenuShortcut sc) –` întoarce opţiunea de meniu corespunzator comenzii rapide sc;
- `public void` **`deleteShortcut`**`(MenuShortcut s) –` elimina comanda rapida sc.

Un meniu este alcatuit din una sau mai multe opţiuni, care sunt instante ale **clasei MenuItem**. Constructorii clasei sunt:

- `public` **`MenuItem`**`() –` creaza o opţiune din meniu fara eticheta şi fara o comanda rapida asociată;
- `public` **`MenuItem`**`(String label) –` creaza o opţiune din meniu cu eticheta dată ca parametru şi fara o comanda rapida asociată;
- `public` **`MenuItem`**`(String label, MenuShortcut sc) –` creaza o opţiune din meniu cu eticheta dată ca parametru şi cu comanda rapida asociată sc;

Dintre metode clasei MenuItem amintim:

- `public String` **`getLabel`**`() –` întoarce eticheta opţiunii;
- `public void` **`setLabel`**`(String label) –` setează eticheta opţiunii;
- `public boolean` **`isEnabled`**`() –` întoarce *true* daca opţiunea este activă şi false în caz contrar;
- `public void` **`setEnabled`**`(boolean b) –` setează daca opţiunea este activă sau nu. La crearea opţiunii, ea este implicit activă;
- `public MenuShortcut` **`getShortcut`**`() –` întoarce comanda rapida asociata opţiunii;
- `public void` **`setShortcut`**`(MenuShortcut sc) –` setează comanda rapida asociata opţiunii;
- `public void` **`deleteShortcut`**`() –` elimina comanda rapida asociata opţiunii;
- `public String` **`getActionCommand`**`() –` întoarce numele evenimentului de actiune generat de acest articol de meniu;
- `public void` **`setActionCommand`**`(String command) –` modifica numele evenimentului de actiune generat de catre acest articol de meniu (implicit numele evenimentuluide actiune este identic cu eticheta articolului);

Clasa CheckboxMenuItem permite gestiunea unui control checkbox ce va fi inclus ca opţiune într-un meniu. Prin selectarea unei astfel de opţiuni se schimbă starea opţiunii din on în off sau din off în on. Constructorii acestei clase sunt:

- **`CheckboxMenuItem`**`() –` creaza o opţiune checkbox din meniu fara eticheta;
- **`CheckboxMenuItem`**`(String label) –` creaza o opţiune checkbox din meniu cu eticheta dată;
- **`CheckboxMenuItem`**`(String label, boolean state) –` creaza o opţiune checkbox din meniu cu eticheta şi starea dată.

Fiind subclasă a clasei MenuItem, moşteneşte metodele acestei clase. Dintre metodele proprii amintim următoarele două, care se referă la starea opțiunii:

- `public void` **`setState`**`(boolean b);`
- `public boolean` **`getState`**`().`

Clasa Menu gestionează *meniuri derulante,* adică meniuri se afişează pe ecran sub forma unei liste verticale de opțiuni. Aceste opțiuni sunt instante ale uneia din clasele *MenuItem*, *CheckboxMenuItem* sau *Menu*. Această posibilitate de definire permite crearea de meniuri în cascada. Intre opțiunile unui meniu pot fi incluşi *separatori* (afişate ca linii orizontale). Optional, meniurile pot fi *tractabile* (engleza: *tear-off menu*). Astfel de meniuri pot fi trase cu mouse-ul şi deplasate în alta parte a ecranului. Daca platforma pe care ruleaza programul nu ofera posibilitatea tractarii, aceasta optiune este ignorata.

Constructorii clasei sunt:

- `public` **`Menu`**`()` – construieste un meniu fara eticheta;
- `public` **`Menu`**`(String label)` – construieste un meniu cu eticheta dată;
- `public` **`Menu`**`(String label, boolean tearOff)` – construieste un meniu cu eticheta dată, iar al doilea parametru indica daca opțiunea este sau nu tractabilă (poate fi trasă cu mouse-ul şi deplasate în alta parte);

Dintre metodele clasei amintim:

- `public int` **`getItemCount`**`()` – întoarce numarul de opțiuni;
- `public boolean` **`isTearOff`**`()` – întoarce *true* sau false după cum meniul este tractabil sau nu;
- `public MenuItem` **`getItem`**`(int index)` – întoarce opțiunea de pe pozitia *index*;
- `public MenuItem` **`add`**`(MenuItem mi)` – adauga la meniu opțiuneam*i*;
- `public void` **`add`**`(String label)` – adauga la meniu o opțiune cu eticheta *label*;
- `public void` **`insert`**`(MenuItem m, int index)` – insereaza opțiunea *i* pe pozitia *index*;
- `public void` **`insert`**`(String label, int index)` – insereaza pe pozitia *index* o opțiune cu eticheta *label*;
- `public void` **`addSeparator`**`()` – se adauga un separator;
- `public void` **`insertSeparator`**`(int index)` – se insereaza un separator pe pozitia *index*;
- `public void` **`remove`**`(int index)` – se elimina opțiunea de pe pozitia *index*;
- `public void` **`remove`**`(MenuItem m)` – se elimina opțiunea m;
- `public void` **`removeAll`**`()` – se elimina din meniu toate opțiunile.

In programul următor se dau instrucțiunile care generează meniul ce apare în figura de mai sus.

Programul **`m01.java`**

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

public class m01 {
  public static void main ( String args []) {
    Frame f = new Frame ("Meniu");
    MenuBar m = new MenuBar ();

    m.add(new Menu ("File"));

    Menu toolbar = new Menu ("Toolbars");
    toolbar.add(new CheckboxMenuItem ("Standard", true));
    toolbar.add(new CheckboxMenuItem ("Drawing"));
```

```
    toolbar.add(new CheckboxMenuItem ("Formating"));

    Menu edit = new Menu("Edit");

    edit.add(new MenuItem ("Normal"));
    edit.add(new CheckboxMenuItem ("Ruler", true));
    edit.addSeparator();
    edit.add(toolbar );

    m.add(edit);

    f.setMenuBar(m);
    f.setSize(200, 200) ;
    f.setVisible(true);
  }
}
```

Meniurile de context (meniurile verticale) se gestionează cu **clasa PopupMenu**, subclasă a clasei Menu. Ele se afişează la apariţia unui eveniument (de obicei la apăsarea butonului drept de mouse). Constructorii clasei sunt:

- **`PopupMenu`**`()` - crează un meniu cu nume vid;
- **`PopupMenu`**`(String label)` - crează un meniu cu numle dat.

Dintre metodele proprii amintim:

- **`show`**`(Component origin, int x, int y)` - afişează meniul la poziţia (x,y) relativă cu componenta dată ca prim argument (colţul din stânga sus a componentei are coordonatele (0,0)).

La un moment dat pot să fie definite mai multe meniuri de context. Pentru ca să fie afişat un anumit meniu (cu metoda show de mai sus) de către o componentă, acesta trebuie să adăugat la componentă cu metoda add şi eventual eliminat cu metoda remove (o componentă poate avea cel mult un meniu de context ataşat la un anumit moment).

Relativ la meniuri apar următoarele evenimente:

- **`ActionEvent`** - la alegerea unei opţiuni ce este instanţă MenuItem;
- **`ItemEvent`** - la alegerea unei opţiuni ce este instanţă CheckboxMenuItem. Evenimentul ItemEvent are câmpurile SELECTED şi DESELECTED prin care se precizează că opţiunea a fost selectată sau deselectată.