AWT II

Componente

Clasa Frame

Frame-ul este un fel de *Window* care permite adăugarea unui *MenuBar*, a titlului ferestrei, și altor caracteristici ale unei ferestre (cum ar fi redimensionarea, maximizarea, minimizarea, meniu de fereastră etc).

Constantele unui Frame

Clasa conține o serie de constante pentru a specifica tipul cursorului. În acest scop se poate folosi metoda *Component.setCursor()* pentru a schimba cursorul mouse-ului atunci când acesta se află deasupra *frame*-ului. Mai jos este o listă cu aceste constante:

```
public final static int DEFAULT_CURSOR
public final static int CROSSHAIR_CURSOR
public final static int TEXT_CURSOR
public final static int WAIT_CURSOR
public final static int SW_RESIZE_CURSOR
public final static int SE_RESIZE_CURSOR
public final static int NW_RESIZE_CURSOR
public final static int NE_RESIZE_CURSOR
public final static int N_RESIZE_CURSOR
public final static int S_RESIZE_CURSOR
public final static int W_RESIZE_CURSOR
public final static int E_RESIZE_CURSOR
public final static int HAND_CURSOR
public final static int MOVE_CURSOR
```

Constructorii clasei Frame

```
public Frame ()
```

Constructorul implicit creează o fereastră ascunsă cu numele “Untitled” sau un *String* gol. Ca și în cazul clasei *Window*, managerul de aspect implicit, al *Frame*-ului comportă cursorul `BorderLayout.DEFAULT_CURSOR`. Deoarece *Frame* este ascunsă la creare, trebuie apelată metoda *show()* sau *setVisible(true)* pentru a permite vizualizarea acesteia.

```
public Frame (String title)
```

Această versiune de constructor este identică primei, dar permite setarea titlului ferestrei prin specificarea parametrului de tip *String*.

Metodele clasei Frame

```
public String getTitle ()
public void setTitle (String title)
```

Metoda `getTitle()` returnează titlul *Frame*-ului curent, iar `setTitle()` modifică titlul Frame-ului curent.

```
public Image getIconImage ()
public void setIconImage (Image image)
```

Metoda `getIconImage()` returnează imaginea folosită ca iconiță a ferestrei. Inițial, aceasta este *null*, dar poate fi setată ulterior. Pe unele platforme, acest concept nu este implementat.

A doua metodă permite modificarea acestei iconițe.

```
public MenuBar getMenuBar ()
public synchronized void setMenuBar (MenuBar bar)
```

Prima metodă returnează bara de meniu a *Frame*-ului curent. A doua metodă modifică bara de meniu cu parametrul `bar`. Se poate ca o aplicație să aibă mai multe bare de meniu, aceasta depinzând de context.

```
public synchronized void remove (MenuComponent component)
```

Această metodă șterge componenta specificată ca parametru, din cadrul Frame-ului, în cazul în care `component` este bara de meniuri a Frame-ului. Aceasta este echivalent cu apelul metodei `setMenuBar` cu parametru *null*.

```
public synchronized void dispose ()
```

Metoda `dispose()` eliberează resursele folosite de *Frame*. Dacă vre-una din clasele *Dialog* sau *Window* sunt asociate acestui *Frame,* atunci și resursele acestora sunt de asemenea eliberate. Se obișnuiește ascunderea ferestrei înainte de eliberarea resurselor acesteia, pentru ca utilizatorii sa nu realizeze acest proces.

```
public boolean isResizable ()
public void setResizable (boolean resizable)
```

Prima metodă returnează faptul că *Frame*-ul curent este sau nu redimensionabil. A doua metodă modifică starea *Frame*-ului. O valoare *true* a parametrului înseamnă că starea ferestrei este redimensionabilă în timp ce *false* semnifică faptul că utilizatorul nu poate redimensiona fereastra.

```
public void setCursor (int cursorType)
public int getCursorType ()
```

Metoda `setCursor` permite specificarea cursorul Frame-ului curent.

Variabila `cursorType` trebuie să fie una din constantele de mai sus. Dacă se specifică altă valoare în locul acestor constante, va fi aruncată excepția *IllegalArgumentException*.

A doua metodă permite preluarea cursorului actual.

```
public synchronized void addNotify ()
```

Această metodă creează un *peer* pentru Frame-ul curent. Fiecare componentă de interfață AWT are propriul ei *peer*. Un *peer* este implementarea acelei componente în mediul nativ. De exemplu, componenta *Choice* corespunde unor obiecte ce permit utilizatorului să selecteze un obiect dintr-o listă.

Metoda `addNotify` creează *peer* pentru *Frame* automat, la apelul metodei *show()*. Se poate suprascrie această metodă însă trebuie să apelăm mai întâi *super.addNotify()*.

Evenimentele Frame-ului

Evenimentele care pot fi generate de un *Frame*, sunt cele generate de *Component*, deoarece *Frame* moștenește această clasă. Pe lângă acestea, *Frame* poate genera evenimente de tip *Window*. Acestea sunt: WINDOW_DESTROY, WINDOW_EXPOSE, WINDOW_ICONIFY, WINDOW_DEICONIFY, și WINDOW_MOVED.

Un eveniment des întâlnit, WINDOW_DESTROY, este generat atunci când utilizatorul încearcă să închidă *Frame*-ul, selectând *Quit*, sau *Exit* din meniu. Implicit acest eveniment nu face nimic. Trebuie să oferiți un *handler,* adică o metodă care să trateze acest eveniment.

```
enableEvents (AWTEvent.WINDOW_EVENT_MASK);
```

Metoda garantează primirea tuturor evenimentelor de fereastră, chiar dacă nu sunt ascultători abonați. Pentru a închide o fereastră se poate folosi metoda *processWindowEvent* și în cazul Frame-ului:

```
protected void processWindowEvent(WindowEvent e)
{
	if (e.getID() == WindowEvent.WINDOW_CLOSING)
	{
	//Notificăm si pe altii că se inchide
		if (windowListener != null)
		windowListener.windowClosing(e);
		System.exit(0);
	} else
	{
		super.processEvent(e);
	}
}
```

În cazul în care uităm să apelăm metoda `enableEvents`,se poate ca metoda `processWindowEvent` să nu fie apelată niciodată, astfel că închiderea ferestrei nu va avea loc.

Cum construim din Window o noua componentă

Am discutat despre clasa Window și clasa Frame, iar acum putem să vedem cum sunt folosite în exemplul de mai jos. Construim o serie de butoane pop-up. De asemenea este folosit *Toolkit*-ul unui *Frame* pentru a încărca imagini într-o aplicație. Butoanele apar atunci când utilizatorul apasă butonul dreapta al mouse-ului, asemănător exemplului din cursul precedent.

```
import java.awt.*;
import java.awt.image.ImageObserver;
public class PopupButtonFrame extends Frame implements ImageObserver
{
    Image im;
    Window w = new PopupWindow (this);
    PopupButtonFrame ()
    {
        super ("PopupButton Example");
        im = getToolkit().getImage ("1.jpg");

        MediaTracker mt = new MediaTracker (this);
        mt.addImage (im, 0);

        Dimension d = new Dimension(400,400);
        this.resize(d);
        this.show();
        try {
         mt.waitForAll();
        }
        catch (Exception e) {e.printStackTrace(); }
    }
    public static void main (String args[])
    {
        Frame f = new PopupButtonFrame ();
    }
    public void paint (Graphics g)
    {
        if (im != null)
        g.drawImage (im, 20, 20, this);
    }
    public boolean mouseDown (Event e, int x, int y)
    {
        if (e.modifiers == Event.META_MASK)
        {
            w.setLocation(getLocation().x+x, getLocation().y+y);
            w.setVisible(true);
            return true;
        }
        return false;
    }
```

```
}
class PopupWindow extends Window
{
    PopupWindow (Frame f)
    {
        super (f);
        Panel p = new Panel ();
        p.add (new Button ("About"));
        p.add (new Button ("Save"));
        p.add (new Button ("Quit"));
        add ("North", p);
        setBackground (Color.gray);
        pack();
    }
    public boolean action (Event e, Object o)
    {
        if ("About".equals (o))
         System.out.println ("About");
        else if ("Save".equals (o))
         System.out.println ("Save Me");
        else if ("Quit".equals (o))
         System.exit (0);
        hide();
        return true;
    }
}
```

Noutatea față de programul din cursul anterior, este că, încărcăm o imagine în cadrul Frame-ului, folosind un obiect de tip `Image` ce este desenat pe metoda de `paint()`.

Dialoguri

Clasa *Dialog* oferă implementarea unei ferestre speciale, care este în mod normal, folosită la mesaje pop-up pentru ca utilizatorul să poată introduce date. Ea trebuie asociată cu *Frame*, și ori de câte ori ceva se întâmplă Frame-ului, același eveniment va avea loc și pe *Dialog*. De exemplu, dacă părintele *Frame* are iconiță, *Dialog*-ul dispare până când iconița este ștearsă din *Frame*.

Dialog-ul poate fi modal sau amodal. Un *Dialog* amodal este acela în care utilizatorul poate interacționa cu *Frame*-ul și cu *Dialogul* în același timp. Un *Dialog* modal este acela care blochează orice interacțiune cu alte componente ale aplicației, inclusiv *Frame*-ul asociat cu acesta.

În figura de mai jos este exemplificat un astfel de dialog.

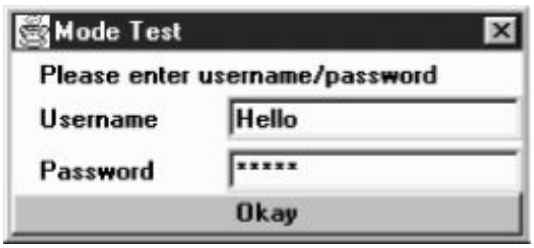

Figura 11.1 Exemplu de Dialog

Constructor și metode

```
public Dialog (Frame parent)
public Dialog (Frame parent, boolean modal)
public Dialog (Frame parent, String title)
public Dialog (Frame parent, String title, boolean modal)
```

Primul constructor creează o instanță de *Dialog*, fără titlu cu părintele dat de `parent`. Acest dialog nu este *modal* și este redimensionabil.

Al doilea constructor specifică părintele prin parametrul `parent`. Se poate specifica și faptul că este sau nu *modal*. Inițial, dialogul este redimensionabil.

Al treilea constructor permite specificarea titlului dialogului. Dialogul nu este *modal*.

Al patrulea constructor este cel mai complex și permite specificarea atât a titlului cât și a faptului că este sau nu *modal*.

```
public String getTitle ()
public void setTitle (String title)
```

Cele două metode permit aflarea titlului dialogului curent, respectiv modificarea titlului.

```
public boolean isResizable ()
public void setResizable (boolean resizable)
```

Aceste două metode permit aflarea faptului că dialogul curent este redimensionabil sau nu, respectiv setarea acestui fapt.

```
public boolean isModal ()
public void setModal (boolean mode)
```

Aceste două metode permit interogarea/setarea faptului că *Dialog*-ul curent este modal sau nu. Data viitoare când dialogul este afișat folosind metoda *show(),* dacă este setat *modal*, doar el va fi activ, în caz contrar și alte componente sunt accesibile.

Evenimentele unui Dialog sunt aceleași ca și cele ale unui *Frame*.

```
import java.awt.*;

interface DialogHandler
{
    void dialogCreator (Object o);
}

class modeTest extends Dialog
{
    TextField user;
    TextField pass;
    modeTest (DialogHandler parent)
    {
        //crearea unui dialog cu parinte, parent
        //nume Mode Test, modal
        super ((Frame)parent, "Mode Test", true);
        //adaugarea unei etichete
        add("North", new Label ("Please enter username/password"));
        //adaugarea unui panel cu doua etichete
        Panel left = new Panel ();
        left.setLayout (new BorderLayout ());
        left.add ("North", new Label ("Username"));
        left.add ("South", new Label ("Password"));
        //panelul este adugat la stanga
        add ("West", left);
        //alt panel cu doua campuri pentru
        //introducerea unui text
        Panel right = new Panel ();
        right.setLayout (new BorderLayout ());
        user = new TextField (15);
        pass = new TextField (15);
        //parola este ascunsa utilizatorului
        pass.setEchoCharacter ('*');
        right.add ("North", user);
        right.add ("South", pass);
        //panelul este adaugat la stanga
        add ("East", right);
        add ("South", new Button ("Okay"));
        resize (250, 125);
    }
    public boolean handleEvent (Event e)
    {
        if (e.id == Event.WINDOW_DESTROY)
        {
            dispose();
            return true;
        }
        else if ((e.target instanceof Button) &&
```

```
        (e.id == Event.ACTION_EVENT))
        {
            //cand are loc un eveniment este transmis
            //parintelui pentru afisarea datelor citite
            ((DialogHandler)getParent ()).dialogCreator(e.arg);
        }
        return super.handleEvent (e);
     }
}
public class modeFrame extends Frame implements DialogHandler
{
    modeTest d;
    modeFrame (String s)
    {
        super (s);
        resize (100, 100);
        d = new modeTest (this);
        d.show ();
    }
    public static void main (String []args)
    {
        Frame f = new modeFrame ("Frame");
    }
    public boolean handleEvent (Event e)
    {
        if (e.id == Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            System.exit (0);
        }
        return super.handleEvent (e);
    }
    public void dialogCreator(Object o)
    {
        // crearea unui dialog
        // in care sunt adaugate ca etichete
        // valorile text ale membrilor lui modeTest
        d.dispose();
        add ("North", new Label (d.user.getText()));
        add ("South", new Label (d.pass.getText()));
         show ();
    }
}
```

Explicațiile despre modul cum funcționează acest program sunt date ca și comentariu. După cum se poate observa, tratarea evenimentelor și unele modele conceptuale sunt conform modelului Java 1.0, așa că acest exemplu este mai mult informativ, nu trebuie utilizat la implementare.

Mai jos este un exemplu, de asemenea cu explicații, în care tratarea evenimentelor și modul de lucru este mai aproape de cel versiunilor actuale de JDK.

```
import java.awt.*;
//se renunta la interfata DialogHandler
class modeTest11 extends Dialog
{
    TextField user;
    TextField pass;
    //Crearea GUI al aplicatiei
    modeTest11 (Frame parent)
    {
        super (parent, "Mode Test", true);
        add ("North", new Label ("Please enter username/password"));
        //adaugarea panel-elor in mod asemanator
        Panel left = new Panel ();
        left.setLayout (new BorderLayout ());
        left.add ("North", new Label ("Username"));
        left.add ("South", new Label ("Password"));
        add ("West", left);
        Panel right = new Panel ();
        right.setLayout (new BorderLayout ());
        user = new TextField (15);
        pass = new TextField (15);
        pass.setEchoCharacter ('*');
        right.add ("North", user);
        right.add ("South", pass);
        add ("East", right);
        add ("South", new Button ("Okay"));
        resize (250, 125);
    }
    //suprascrierea functiei de tratare a
    //evenimentelor de fereastra
    public boolean handleEvent (Event e)
    {
    if (e.id == Event.WINDOW_DESTROY) {
    dispose();
    return true;
    } else if ((e.target instanceof Button) &&
    (e.id == Event.ACTION_EVENT)) {
    hide();
    }

    return super.handleEvent (e);
    }
}
//fereastra ce afiseaza datele
public class modeFrame11 extends Frame
```

```
{
    modeFrame11 (String s)
    {
        super (s);
        resize (100, 100);
    }
    public static void main (String []args)
    {
        Frame f = new modeFrame11 ("Frame");
        modeTest11 d;
        //dialogul este afisat primul
        //el este "pornit" de frame
        d = new modeTest11 (f);
        //f.show ();
        //ce se intampla daca as decomenta instructiunea
        //de mai sus. De ce nu am acces la frame?
        //Pentru ca dialogul este modal.Daca ar fi amodal?
        d.show ();
        d.dispose();
        //dupa ce dialogul este inlaturat
        //este afisat frame-ul
        f.add ("North", new Label (d.user.getText()));
        f.add ("South", new Label (d.pass.getText()));
        f.show ();
    }
    //aici sunt tratate evenimentele frame-ului
    public boolean handleEvent (Event e)
    {
        if (e.id == Event.WINDOW_DESTROY)
        {
            hide();
            dispose();
            System.exit (0);
        }
        return super.handleEvent (e);
    }
}
```

Clasa FileDialog

Aceasta este o subclasă a *Dialog*-ului ce permite selectarea fișierelor sau salvarea acestora. *FileDialog* este întotdeauna un *Dialog modal*, cu alte cuvinte aplicația apelantă va fi blocată până la selectarea unui fișier sau până la închiderea dialogului de fișier.

Metodele clasei FileDialog

Un *FileDialog* are două moduri: unul pentru încărcarea fișierelor și unul pentru salvarea acestora. Următoarele variabile descrise oferă posibilitatea specificării acestui mod de lucru. Diferența vizibilă este că pe ecran va apare *Load* sau *Save*.

```
public final static int LOAD
public final static int SAVE
```

În continuare vom descrie constructorii acestei clase

```
public FileDialog (Frame parent)
public FileDialog (Frame parent, String title)
public FileDialog (Frame parent, String title, int mode)
```

Primul constructor creează un dialog pentru fișiere ce are specificat *Frame*-ul părinte ca parametru. Titlul este inițial gol.

Al doilea constructor permite specificarea și a titlului inițial.

Al treilea constructor permite specificarea, pe lângă parametrii descriși, și a modului dialogului: *Save* sau *Load*.

```
public String getDirectory ()
public void setDirectory (String directory)
```

Aceste două metode permit preluarea sau setarea a directorului curent în care se va face căutarea fișierului. Metoda `setDirectory` trebuie apelată înainte de afișarea dialogului.

```
public String getFile ()
public void setFile (String file)
```

Unele dintre cele mai importante metode, acestea permit preluarea numelui fișierului selectat (în cazul metodei `getFile`). Metoda `setFile` permite setarea numelui fișierului selectat implicit la afișarea dialogului.

```
public FilenameFilter getFilenameFilter ()
public void setFilenameFilter (FilenameFilter filter)
```

Metoda `getFilenameFilter` permite preluarea filtrului de nume al fișierului. Clasa `FilenameFilter` face parte din pachetul *java.io* și permite limitarea căutării unui fișier după diverse filtre. De exemplu, dacă filtrăm după extensie, putem avea un filtru compus din *.jpg*, *.gif*, sau *.mpeg.*

A doua metodă permite setarea acestui filtru pentru dialogul ce va fi afișat.

```
public int getMode ()
public void setMode (int mode)
```

Aceste două metode permit preluarea / setarea modului de lucru: *Save* sau *Load*.

Mai jos avem un exemplu pentru folosirea acestor tipuri de dialog, foarte folositoare în cazul aplicațiilor ce lucrează cu fișiere.

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
public class FileDialogTest extends Frame implements ActionListener
{
    //zona de editarea textului
    TextArea myTextArea;
    Label myLabel;
    //butonul pentru deshiderea dialocului Load
    Button loadButton;
    //butonul pentru deshiderea dialocului Save
    Button saveButton;
    //se construieste fereastra de editare text
    FileDialogTest ()
    {
        super ("File Dialog Tester");
        Panel p = new Panel ();
        p.add (loadButton = new Button ("Load"));
        p.add (saveButton = new Button ("Save"));
        //butoanele sunt abonate la evenimentul
        //de apasare
        loadButton.addActionListener(this);
        saveButton.addActionListener(this);
        add ("North", myLabel = new Label ());
        add ("South", p);
        add ("Center", myTextArea = new TextArea (10, 40));
        pack();
    }
    public static void main (String args[])
    {
        //instantierea ferestrei principale
        FileDialogTest f = new FileDialogTest();
        //si afisarea acesteia
        f.show();
    }
    public boolean handleEvent (Event e)
    {
        if (e.id == Event.WINDOW_DESTROY)
        {
            hide();
            dispose ();
            System.exit(0);
            return true; // never gets here
        }
```

```
    return super.handleEvent (e);
}
//cand apare un eveniment
public void actionPerformed(ActionEvent e)
{
    //daca sursa evenimentului este de un buton
    if (e.getSource() instanceof Button)
    {
        int state;
        String msg;
        //daca este cel de Load
        if (e.getSource() == loadButton)
        {
            state = FileDialog.LOAD;
            msg = "Load File";
        }
        else //sau este cel de Save
        {// if (e.target == saveButton)
            state = FileDialog.SAVE;
            msg = "Save File";
        }
        //Se deschide un dialog de fisier
        //cu parintele this
        FileDialog file = new FileDialog (this, msg, state);
        file.setFile ("*.java"); // filtrul este extensia java
        file.show(); // afisarea dialogului si asteptare
        String curentFile;
        byte[] data = null;
        //daca s-a selectat un fisier se va prelua numele lui
        if ((curentFile = file.getFile()) != null)
        {
            //se compune calea completa a fisierului
            String filename = file.getDirectory() + curentFile;
            setCursor (Frame.WAIT_CURSOR);
            //daca dialogul a fost de Load
            if (state == FileDialog.LOAD)
            {
                try
                {
                    //se incearca deschiderea fisierului
                    //si citirea datelor din el
                    data =ReadFile(filename);
                }
                catch (FileNotFoundException exc)
                {
                    System.out.println("File Not Found: " + filename);
                }
                catch (IOException exc)
```

```
                {
                    System.out.println("IOException: " + filename);
                }
                myLabel.setText ("Load: " + filename);

            }
            else//trebuie sa salvam...
            {
                //Stergem".*.*" daca apare acest text
                //inseamna ca fisierul nu exista
                 if (filename.indexOf (".*.*") != -1) {
                filename = filename.substring (0, filename.length()-4);
                }
                try
                {
                    //se incearca scrierea in fisier
                    data = WriteInFile(filename);
                }
                catch (IOException exc)
                {
                    System.out.println("IOException: " + filename);
                }
                myLabel.setText ("Save: " + filename);
                //numele fisierului este afisat la sfarsit
            }
            //daca s-a scris sau s-a citit corect
            if (data!=null)
                //putem plasa acea informatie in casuta de text
                myTextArea.setText (new String (data, 0));
            //revenim la cursorul initial
            setCursor (Frame.DEFAULT_CURSOR);
        }
    }
}
//metoda citeste din fisierul dat ca parametru
//si returneaza ceea ce a reusit sa citeasca
public byte[] ReadFile(String filename) throws IOException
{
    File f = new File (filename);
    byte[] data;
    FileInputStream fin = new FileInputStream (f);
    int filesize = (int)f.length();
    data = new byte[filesize];
    fin.read (data, 0, filesize);
    return data;
}
//metoda scrie din fisierul dat ca parametru
//si returneaza ceea ce a reusit sa scrie
```

```
    public  byte[] WriteInFile(String filename) throws IOException
    {
        byte[] data;
        File f = new File (filename);
        FileOutputStream fon = new FileOutputStream (f);
        String text = myTextArea.getText();
        int textsize = text.length();
        data = new byte[textsize];
        //se scrie in data ce se afla in myTextArea
        text.getBytes (0, textsize, data, 0);
        fon.write (data);
        fon.close ();
        return data;
    }
}
```

Aplicația permite introducerea într-o fereastră ca cea de mai jos a unui text, care poate fi "încărcat" dintr-un fișier sau poate salva textul editat într-un fișier. Explicațiile despre cum funcționează acest program se găsesc sub forma unor comentarii inserate în dreptul instrucțiunilor.

Layout

Fiecare *Container* are un manager de layout (adică al aspectului componentelor). Acesta este responsabil pentru poziționarea componentelor în cadrul recipientului, indiferent de platformă sau mărimea ecranului. Practic aceste clase elimină necesitatea de a calcula poziția în care plasăm o componentă. Chiar și pentru o componentă simplă cum ar fi un buton, calcularea poziției și a dimensiunilor relative la fereastra curentă, se poate întinde pe zeci poate sute de linii de cod. În continuare vom descrie câteva managere de aspect care ușurează poziționarea acestor componente pe ecran.

Interfața LayoutManager

Această interfață definește responsabilitățile unei clase ce va afișa componente din cadrul unui *Container*. Sarcina acestei interfețe este să determine poziția și mărimea fiecărei componente din cadrul *Container*-ului. Nu se apela direct metoda unui *LayoutManager*, deoarece apelurile acestor clase sunt ascunse de programatori. Metodele acestei interfețe se pot suprascrie atunci când implementăm un nou manager de aspect, diferit de cele oferite de Java.

Metodele interfeței LayoutManager

```
public abstract void addLayoutComponent (String name, Component component)
```

Această metodă este apelată când programul atribuie un nume componentei atunci când o adaugă Layout-ului (de exemplu când apelează *add*(*String*,*Component*) sau *add*(*Component,Object*)).

```
public abstract void removeLayoutComponent (Component component)
```

Metoda va șterge componenta `component` din memorie. Aceasta este apelată de obicei la eliberarea *Container*-ului în care se află componenta.

```
public abstract Dimension preferredLayoutSize (Container parent)
```

Metoda determină mărimea preferată a componentelor din cadrul *Container*-ului. Metoda va returna un obiect *Dimension* ce conține lățimea și lungimea necesară afișării componentelor.

```
public abstract Dimension minimumLayoutSize (Container parent)
```

Metoda `minimumLayoutSize` este apelată pentru a determina mărimea minimă a componentelor din cadrul *Container*.

```
public abstract void layoutContainer (Container parent)
```

Această metodă ajută la poziționarea tuturor componentelor ale `parent`.

Interfața LayoutManager2

Odată cu introducerea Java 1.1, au apărut modificări interfeței mai sus menționată.

Noua interfață rezolvă o problemă care apare atunci când lucrăm cu *GridBagLayout.* Deși metoda *addLayoutComponent(String, Component)*, funcționează bine pentru *BorderLayout* și *CardLayout,* nu poate fi folosită pentru *GridBagLayout*. Poziția unei componente într-un *GridBagLayout* este controlată de un număr de constrângeri, care sunt încapsulate în obiectul de tip *GridBagConstraints*. Pentru a asocia aceste constrângeri, trebuie apelată metoda *setConstraints()*.

Interfața *LayoutManager2* definește o versiune a metodei *addLayoutComponent()* ce poate fi folosită de toți managerii de aspect, indiferent de constrângerile particulare. Metoda are un parametru de tip *Object* care poate reprezenta orice tip de informație, cum ar fi tipul de constrângeri de mai sus.

Metodele interfeței LayoutManager2

```
public abstract void addLayoutComponent(Component comp, Object constraints)
```

Metoda este apelată când programul asignează `constraints` componentei `comp` atunci când o adaugă layout-ului. Este treaba layout-ului să decidă ce va face cu `constraints`.

```
public abstract Dimension maximumLayoutSize(Container target)
```

Această metodă returnează mărimea maximă a `target` în cadrul acestui manager de layout.

Anterior, doar mărimea minimă sau cea preferată se putea obține.

```
public abstract float getLayoutAlignmentX(Container target)
public abstract float getLayoutAlignmentY(Container target)
```

Metodele returnează aliniamentul `target` de-a lungul axei *X* respectiv *Y*. Valoarea returnată este între 0.0 și 1. Valorile apropiate de 0 înseamnă o poziționare mai aproape de stânga, dacă vorbim de prima metodă, în timp ce valorile mai aproape de 1 semnifică o poziționare mai la dreapta. Dacă ne referim la a doua metodă, o valoare aproape de 0 înseamnă o poziționare mai sus, în timp ce o valoare mai aproape de jos.

```
public abstract void invalidateLayout(Container target)
```

Metoda spune managerului de aspect că orice informație de layout, care are ca țintă `target` este invalidată.

FlowLayout

Acesta este managerul de Layout implicit pentru un *Panel*. Un *FlowLayout* adaugă componente container-ului curent, pe rânduri, de la stânga la dreapta. Dacă nu se mai găsesc componente în rând, ca începe un nou rând. Atunci când *Container*-ul este redimensionat, componentele sunt repoziționate pe același principiu.

```
public final static int LEFT
public final static int CENTER
public final static int RIGHT
```

Acestea sunt constantele acestei clase, constante ce returnează aliniamentul.

Metodele FlowLayout

```
public FlowLayout ()
public FlowLayout (int alignment)
public FlowLayout (int alignment, int hgap, int vgap)
```

Primul constructor creează un layout de acest tip, folosind setări implicite, aliniament centrat cu o pauză orizontală și verticală între componente de cinci pixeli. De obicei acest constructor este apelat în cadrul metodei *setLayout()*:*setLayout (new FlowLayout()).*

Al doilea constructor permite setarea aliniamentului pentru componentele adăugate. Astfel în figura de mai jos avem exemple pentru cele trei tipuri de aliniamente.

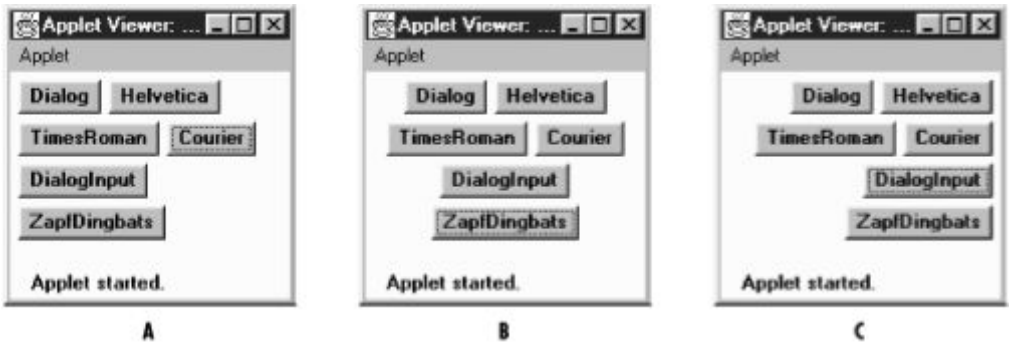

Figura 11.2 FlowLayout (A) stânga B) centrat C) dreapta)

Al treilea constructor, permite setarea spațiului dintre componente, pe orizontală și pe verticală. În figura de mai jos avem o reprezentarea a acestui tip de aliniament.

Figura 11.3 Aliniament cu *hgap* de 0 si *vgap* de 20

```
public int getAlignment ()
public void setAlignment (int alignment)
```

Cele două metode permit returnarea, respectiv setarea aliniamentului layout-ului curent. Aliniamentul poate fi una din cele trei constante mai sus menționate.

```
public int getHgap ()
public void setHgap (int hgap)
public int getVgap ()
public void setVgap (int hgap)
```

Aceste metode permit preluarea respectiv setarea spațiului vertical și orizontal dintre componente. După apelul metodei `setVgap` sau `setHgap` trebuie apelată metoda de *validate()* a Container-ului. Începând cu versiunea 1.2, Java admite alte două constante pentru orientare și anume TRAILING și LEADING.

În exemplul de mai jos se poate înțelege cum se efectuează afișarea componentelor pe rânduri. Pentru o mai bună înțelegere a acestor aliniamente recomand redimensionarea ferestrei.

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

public class FlowLayoutDemo extends Frame implements ActionListener
{
    //panel-ul unde vor fi plasate butoanele
    final Panel compsToExperiment = new Panel();
    //cele doua butoane de jos
    //care vor decide aliniamentul celor de sus
    Button RtoLbutton;
    Button LtoRbutton;
    //manager-ul de layout principal
    FlowLayout experimentLayout = new FlowLayout();
    final String RtoL = "Right to left";
    final String LtoR = "Left to right";

    public boolean handleEvent(Event e)
    {
    //daca evenimentul este cel de inchidere a ferestrei
        if (e.id == Event.WINDOW_DESTROY)
        {
            dispose();
            System.exit(1);
        }
        return super.handleEvent (e);
    }

    public void actionPerformed(ActionEvent e)
    {
            String command = e.getActionCommand();
            //Check the selection
            if (command.equals("Left to right")) {
                //setarea orientarii stanga -> dreapta
                compsToExperiment.setComponentOrientation(
                        ComponentOrientation.LEFT_TO_RIGHT);
            } else {
                //setarea orientarii stanga <- dreapta
                compsToExperiment.setComponentOrientation(
                        ComponentOrientation.RIGHT_TO_LEFT);
            }
            //update al layout-ului panelului
            compsToExperiment.validate();
            compsToExperiment.repaint();

    }
```

```
//constructorul Frame-ului
public FlowLayoutDemo(String name)
{
    super(name);
    addComponentsToFrame();
}
public void addComponentsToFrame()
{
    //panelul primeste ca layout experimentLayout
    compsToExperiment.setLayout(experimentLayout);
    experimentLayout.setAlignment(FlowLayout.TRAILING);
    //in acest panel vor sta butoanele din
    //partea de jos a ferestrei
    Panel controls = new Panel();
    controls.setLayout(new FlowLayout());

    LtoRbutton = new Button(LtoR);
    LtoRbutton.setActionCommand(LtoR);
    RtoLbutton = new Button(RtoL);
    RtoLbutton.setActionCommand(RtoL);
    //se aboneaza la ascultarea actiunilor
    LtoRbutton.addActionListener(this);
    RtoLbutton.addActionListener(this);
    //adaug butoane panelului compsToExperiment
    compsToExperiment.add(new Button("Button 1"));
    compsToExperiment.add(new Button("Button 2"));
    compsToExperiment.add(new Button("Button 3"));
    compsToExperiment.add(new Button("Long-Named Button 4"));
    compsToExperiment.add(new Button("5"));
    //initial se seteaza orientarea stanga -> dreapta
    compsToExperiment.setComponentOrientation(
            ComponentOrientation.LEFT_TO_RIGHT);

    controls.add(LtoRbutton);
    controls.add(RtoLbutton);
    //se adauga panelul compsToExperiment in centrul frame-ului
    this.add(compsToExperiment, BorderLayout.CENTER);
    //se adauga panelul controls in partea de jos a frame-ului
    this.add(controls, BorderLayout.SOUTH);

}
private static void createAndShowGUI() {
    //se instantiaza si ruleaza fereastra
    FlowLayoutDemo frame = new FlowLayoutDemo("FlowLayoutDemo");
    frame.pack();
    frame.setVisible(true);
}
```

```
    public static void main(String[] args) {
        createAndShowGUI();
    }

}
```

BorderLayout

Acesta este managerul implicit pentru un *Window*. Oferă un mod ușor de a poziționa componentele conform unor limite ale ferestrei, de unde vine și numele.

Următorul apel: *setLayout(newBorderLayout()* modifică managerul de aspect astfel încât layout-ul să fie unul de tip *BorderLayout*.

Constantele BorderLayout

```
public static final String CENTER
public static final String EAST
public static final String NORTH
public static final String SOUTH
public static final String WEST
```

Aceste constante vor indica poziția cardinală a locului unde va fi plasat pe Container, noua componentă.

Metodele BorderLayout

```
public BorderLayout ()
public BorderLayout (int hgap, int vgap)
```

Primul constructor creează un *BorderLayout* folosind o setare de delimitatori de zero pixeli pe verticală și zero pixeli pe orizontală. Dimensiunea delimitatorului înseamnă spațiul adiacent dintre componente.

Al doilea constructor permite crearea unui *BorderLayout* cu delimitatori a căror dimensiune este specificată de parametrii lui. Se poate ca acești parametrii să fie negativi, în cazul acesta componentele suprapunându-se unele peste altele.

```
public int getHgap ()
public void setHgap (int hgap)
public int getVgap ()
public void setVgap (int hgap)
```

Aceste metode permit preluarea/schimbarea dimensiunii delimitatorului pe orizontală respectiv pe verticală.

În general metodele acestui tip de *Layout* sunt aceleași ca și în cazul *FlowLayout*, diferă doar comportamentul aspectului și anume faptul a acest *Layout* poziționează pe regiuni cardinale, componentele pe *Container-ul* pe care îl tratează.

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

public class BorderLayoutDemo extends Frame
{
    public static boolean RIGHT_TO_LEFT = false;
    public BorderLayoutDemo()
    {
        super("BorderLayout Demo");
        //container-ul unde vom adauga toate
        //butoanele
        Panel allcomp = new Panel();
        //neaparat sa aiba un layout de acest tip
        allcomp.setLayout(new BorderLayout());
        //adaugam butoanele panelului
        addComponentsTo(allcomp);
        //panelul este adaugat frame-ului curent
        add(allcomp,BorderLayout.CENTER);
        //pentru evenimentul de inchidere al frame-ului
        addWindowListener(
                new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {
                        System.exit(0);
                    }
                }
        );
    }

    public static void addComponentsTo(Container pane) {

        if (!(pane.getLayout() instanceof BorderLayout)) {
            pane.add(new Label("Container doesn't use BorderLayout!"));
            return;
        }
        //orientarea este stanga -> dreapta
        if (RIGHT_TO_LEFT) {
            pane.setComponentOrientation(
                    java.awt.ComponentOrientation.RIGHT_TO_LEFT);
        }
        //se adauga cele 5 butoane
        Button button = new Button("Button 1 (PAGE_START)");
        pane.add(button, BorderLayout.PAGE_START);
        //componenta din centru este de obicei cea mai mare
```

```
        button = new Button("Button 2 (CENTER)");
        button.setPreferredSize(new Dimension(200, 100));
        pane.add(button, BorderLayout.CENTER);

        button = new Button("Button 3 (LINE_START)");
        pane.add(button, BorderLayout.LINE_START);

        button = new Button("Long-Named Button 4 (PAGE_END)");
        pane.add(button, BorderLayout.PAGE_END);

        button = new Button("5 (LINE_END)");
        pane.add(button, BorderLayout.LINE_END);

    }

    private static void createAndShowGUI() {

      //instantiez si afisez fereastra
        BorderLayoutDemo frame = new BorderLayoutDemo();
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args)
    {
        createAndShowGUI();
    }
}
```

Mai există o serie de alte clase Layout: GridLayout ,CardLayout ce pot fi studiate mai departe. Aceste *Layout*-uri pot fi utilizate în combinație, în funcție de context. Iată un mic exemplu de cum putem combina mai multe aspecte:

```
import java.awt.*;
public class ManyLayouts extends java.applet.Applet
{
    public void init()
    {
        Panel s = new Panel();
        Panel e = new Panel();
        setLayout (new BorderLayout ());
        add ("North", new Label ("Enter text", Label.CENTER));
        add ("Center", new TextArea ());
        e.setLayout (new GridLayout (0,1));
        e.add (new Button ("Reformat"));
        e.add (new Button ("Spell Check"));
        e.add (new Button ("Options"));
        add ("East", e);
```

```
        s.setLayout (new FlowLayout ());
        s.add (new Button ("Save"));
        s.add (new Button ("Cancel"));
        s.add (new Button ("Help"));
        add ("South", s);
    }
}
```

Componente tip lista

Choice

Vom discuta în continuare despre trei componente AWT: *Choice*, *List* și *CheckBox*. Toate cele trei clase implementează interfața *ItemSelectable*. *Choice* și *List* sunt similare, ambele oferă posibilitatea alegerii unui obiect dintr-o listă. Tipul de listă este însă diferit, Choice fiind o list de tip pull-down iar *List* fiind una pe care poate fi efectuată acțiunea de *scroll*.

Metodele Choice

```
public Choice ()
```

Există un singur constructor pentru această clasă. Componenta, la creare, este inițial goală. La adăugarea unui item, folosind *addItem()* sau *add()*, atunci acesta poate fi vizualizat și selectat.

```
public int getItemCount ()
public int countItems ()
```

Aceste metode returnează obiectele selectabile din componenta *Choice*.

```
public String getItem (int index)
```

Această metodă returnează textul unui item de la poziția dată de `index`. Dacă `index` este mai mic ca zero sau mai mare ca numărul de elemente selectabile atunci va fi aruncată excepția `ArrayIndexOutOfBoundsException`.

```
public synchronized void add (String item)
public synchronized void addItem (String item)
```

Aceste metode permit adăugarea unui nou element în lista *Choice*-ului. Dacă parametrul este un null, atunci va fi aruncată excepția *NullPointerException*.

```
public synchronized void insert (String item, int index)
```

Metoda permite adăugarea unui nou element la poziția indicată de `index`. Dacă `index` este ilegal, atunci va fi aruncată excepția `IllegalArgumentException`..

```
public synchronized void remove (String item)
public synchronized void remove (int position)
public synchronized void removeAll ()
```

Metodele permit ștergerea unui element sau a tuturora din lista de elemente disponibile. Prima metodă permite selectarea elementului de șters, după numele dat ca parametru.

A doua metodă permite ștergerea de la o anumită poziție, atâta timp cât este una validă.

Ultima metodă va șterge toate elementele din lista *Choice*.

```
public String getSelectedItem ()
```

Metoda `getSelectedItem()` returnează elementul selectat, sub forma unui *String*. Dacă *Choice* este goală funcția va returna *null*.

```
public Object[] getSelectedObjects ()
```

Metoda permite aflarea elementului selectat sub forma unui șir de *Object*. Metoda este impusă de interfața *ItemSelectable*, și este folositoare cu adevărat în cazul clasei *List*.

```
public int getSelectedIndex ()
```

Metoda returnează poziția din listă a elementului selectat. Primul element se află pe poziția 0.

```
public synchronized void select (int position)
```

Această metodă forțează ca item-ul de pe poziția `position` să fie cel selectat.

```
public void select (String string)
```

Același efect este obținut doar că elementul este specificat prin numele său.

Evenimentele Choice

Evenimentul cel mai des întâlnit este cel de selectarea a unui element. Aceasta poate fi captat printr-un *ItemListener*. De asemenea se pot folosi evenimente de mouse sau tastatură.

```
public boolean keyDown (Event e, int key)
public boolean keyUp (Event e, int key)
```

Metodele sunt apelate atunci când utilizatorul apasă o tastă și focusul este pe *Choice*. Prin obiectul *e* de tip *Event* se poate interoga date specifice despre eveniment, iar variabila *key* poate furniza date despre tasta apăsată.

```
public void addItemListener(ItemListener listener)
public void removeItemListener(ItemListener listener)
```

Poate cele mai importante metode, permit inregistrarea/deînregistrarea unui obiect de tip Choice la/de la ascultarea evenimentelor de modificare a acestui obiect.

```
protected void processEvent(AWTEvent e)
```

Metoda este apelată atunci când un eveniment de tip `AWTEvent` are loc. Suprascrierea acestei este echivalentă cu suprascrierea *handleEvent()* din vechea versiune Java.

În exemplul de mai jos se poate observa adăugarea unor elemente în lista Choice, precum și tratarea evenimentelor de selecție ce apar.

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
class MyChoice extends Choice
{
    MyChoice ()
    {
        super ();
        //trebuie sa
        //activam ascultarea evenimentelor
        enableEvents (AWTEvent.ITEM_EVENT_MASK);
    }
    //suprascriem processItemEvent
    protected void processItemEvent(ItemEvent e)
    {
        //atunci cand are loc un eveniment
        ItemSelectable ie = e.getItemSelectable();
        System.out.println ("Item Selected: " + ie.getSelectedObjects()[0]);
        super.processItemEvent (e);
    }
}
public class ChoiceDemo extends Applet implements ItemListener
{
    Choice c;
    public void init ()
    {
        String []fonts;
        //preluam fonturile existente pe sistem
        //pentru a fi incarcare in Choice
        fonts = Toolkit.getDefaultToolkit().getFontList();
        c = new MyChoice();
        for (int i = 0; i < fonts.length; i++)
        {
            //metoda de adaugare de elemente
            //in ComboList
```

```
            c.add (fonts[i]);
        }
        //componenta Choice este adaugata Applet-ului
        add (c);
        //ascultam un eveniment de selectie
        c.addItemListener (this);
    }
    //ce se intampla cand selectam un element
    public void itemStateChanged(ItemEvent e)
    {
        ItemSelectable ie = e.getItemSelectable();
        System.out.println ("State Change: " + ie.getSelectedObjects()[0]);
    }
}
```

Liste

Clasa *List* oferă un mod de a afișa o secvență de elemente, din care utilizatorul poate selecta unul sau mai multe. În mod normal, un *scrollbar* este asociat acestei componente.

Metode

```
public List ()
public List (int rows)
public List (int rows, boolean multipleSelections)
```

Primul constructor creează o listă goală cu patru linii vizibile. Utilizatorul poate selecta în acest caz, un singur element.

Al doilea constructor permite specificarea numărului de linii vizibile, prin parametrul `rows`.

Al treilea constructor permite de asemenea specificarea numărului de linii, și totodată, specificarea faptului că utilizatorul poate selecta mai multe elemente sau nu.

```
public int getItemCount ()
public int countItems ()
```

Cele două metode returnează numărul de elemente din listă. A doua metodă este numele metodei din versiunea veche.

```
public String getItem (int index)
```

Metoda returnează reprezentarea String a *item*-ului de la poziția *index*.

```
public String[] getItems ()
```

Metoda returnează un șir ce conține toate elementele din listă.

Nu contează dacă elementul este sau nu selectat.

```
public synchronized void add (String item)
public synchronized void addItem (String item)
public synchronized void add (String item, int index)
public synchronized void addItem (String item, int index)
```

Primele două metode permit adăugarea unui item la sfârșitul listei. Ultimele două elemente permit adăugarea unui element pe o poziție specificată de `index`. Dacă indexul este mai mic ca zero sau mai mare ca numărul de elemente de listă, elementul este adăugat la sfârșit.

```
public synchronized void removeAll ()
public synchronized void clear ()
```

Metodele permit ștergerea tuturor elementelor dintr-o listă.

```
public synchronized void remove (String item)
public synchronized void remove (int position)
public synchronized void delItem (int position)
```

Prima metodă permite ștergerea unui element după numele său, iar celelalte două permit ștergerea unui element după poziția lui în listă.

```
public synchronized int getSelectedIndex ()
```

O altă metodă des folosită, `getSelectedIndex()` permite aflarea poziției în listă a elementului selectat. Dacă mai multe elemente sunt selectate, metoda returnează valoarea -1.

```
public synchronized int[] getSelectedIndexes ()
```

Metoda returnează un sir de întregi ce reprezintă indecșii elementelor selectate.

```
public synchronized String getSelectedItem ()
public synchronized String[] getSelectedItems ()
```

Aceste metode sunt corespondentele metodelor de mai sus, returnând numele elementelor selectate.

```
public synchronized Object[] getSelectedObjects ()
```

Metoda returnează elementele selectate sub forma unui șir de obiecte, conform interfeței *ItemSelectable*. Dacă nici un item nu este selectat, șirul este gol.

```
public synchronized void select (int index)
public synchronized void deselect (int index)
```

Metodele permit selectarea/deselectarea unui element de la poziția `index`.

Dacă lista este setată pe selecție unică, la apelul metodei `select()`, elementul anterior selectat este automat deselectat. Dacă lista este de tipul multiselecție, atunci metoda `select()` nu are acest efect. Metoda `deselect()` deselectează elementul de la poziția `index`.

```
public boolean isIndexSelected (int index)
public boolean isSelected (int index)
```

Metodele permit interogarea unui element cu privire la faptul că este sau nu selectat.

```
public boolean isMultipleMode ()
public boolean allowsMultipleSelections ()
```

Metodele returnează starea curentă a listei, daca este de tipul multiselecție sau nu.

```
public void setMultipleMode (boolean value)
public void setMultipleSelections (boolean value)
```

Metodele permit schimbarea stării curente a listei: dacă valoarea parametrului este *true* atunci lista va trece în mod multiselecție, dacă valoarea parametrului este *false*, atunci lista va fi cu selecție unică.

```
public int getRows ()
```

Metoda returnează numărul de rânduri care a fost transmis la instanțierea listei. Nu returnează numărul de rânduri vizibile.

```
public Dimension getPreferredSize (int rows)
public Dimension preferredSize (int rows)
```

Metodele returnează dimensiunea preferată, pentru a ști înălțimea rândurilor.

```
public Dimension getPreferredSize ()
public Dimension preferredSize ()
```

Metodele returnează dimensiunea preferată a listei. Pentru a calcula această dimensiune, se vor folosi numărul de rânduri specificate la instanțierea listei.

Evenimentele listei

Se poate înregistra un *ItemListener* cu metoda *addItemListener()* sau un *ActionListener* cu metoda *addActionListener()* pentru a asculta evenimentele ce apar pe această componentă. Există totuși o altă metodă numită *action()* care înregistrează evenimente apărute pe o listă:

```
public boolean action (Event e, Object o)
```

Metoda este apelată atunci când utilizatorul efectuează un dublu-click pe un item din listă, iar *o* reprezintă acel obiect. Dacă lista este de tip multiselecție, prinderea acestui eveniment poate duce la confuzii din cauză că nu este clar dacă utilizatorul a vrut să selecteze elementul sau selecția se referă la toate elementele alese. Următorul exemplu prezintă un mod de a rezolva această situație, prin implicarea unui alt element și anume un buton.

```
import java.awt.*;
import java.applet.*;
public class ListDemo extends Applet
{
    List l;
    public void init ()
    {
        String fonts[];
        //se preiau fonturile din sistem
        fonts = Toolkit.getDefaultToolkit().getFontList();
        //se creaza o lista cu multiselectie
        l = new List(4, true);
        for (int i = 0; i < fonts.length; i++)
        {
            //se adauga elementele in lista
            l.add(fonts[i]);
        }
        setLayout (new BorderLayout (10, 10));
        //se adauga componentele pe fereastra
        add ("North", new Label ("Pick Font Set"));
        add ("Center", l);
        add ("South", new Button ("Submit"));
        resize (preferredSize());
        validate();
    }
    public boolean action (Event e, Object o)
    {
        // la apasarea butonlui
        if (e.target instanceof Button)
        {
            //se preiau elementele selectate
            String chosen[] = l.getSelectedItems();
            for (int i=0;i<chosen.length;i++)
                //pentru a fi afisate
                System.out.println (chosen[i]);
        }
        return false;
    }
}

public void addItemListener(ItemListener listener)
public void removeItemListener(ItemListener listener)
```

Modelul nou de tratare a evenimentelor implică folosirea unor ascultători, iar pentru aceasta avem metodele de abonare/dezabonare la evenimentele de tip *item*.

```
public void addActionListener(ActionListener listener)
public void removeActionListener(ActionListener listener)
```

Aceste metode se ocupă cu înregistrarea/dezabonarea unei liste la evenimente de acțiune.

În general modelul este cel întâlnit la componente, deci regulile se aplică asemenea.

Pentru a exemplifica utilizarea acestor evenimente, precum și folosirea unei liste cu selecție unică, avem exemplul de mai jos:

```
public class ListDemo1 extends Applet implements ItemListener {
  /*Membrii clasei*/
  private LayoutManager Layout;
  private List Selector;
  private Font SansSerif;
  //constructorul initializeaza forma
  public ListDemo1 () {
    /*intr-un sir tinem culorile*/
    String [] ColorList;
    int i;
    /* instantierea membrilor */
    ColorList = new String [9];
    SansSerif = new Font ("SansSerif", Font.BOLD, 14);
    Layout = new FlowLayout ();
    Selector = new List (6);
    //sunt initializate valorile din sir
    ColorList [0] = "Red";
    ColorList [1] = "Magenta";
    ColorList [2] = "Blue";
    ColorList [3] = "Cyan";
    ColorList [4] = "Green";
    ColorList [5] = "Yellow";
    ColorList [6] = "White";
    ColorList [7] = "Gray";
    ColorList [8] = "Black";
    for (i = 0; i < ColorList.length; ++i) {
      Selector.add (ColorList [i]);
    }
    Selector.setBackground (Color.LIGHT_GRAY);
    Selector.setForeground (Color.red);
    Selector.setFont (SansSerif);
    // se adauga  pe forma, lista
    setLayout (Layout);
    add (Selector);
    //ascultam la evenimentele de pe lista
    Selector.addItemListener (this);
    //setari initiale
    Selector.select (2);
    setBackground (Color.blue);
  }
  public void itemStateChanged(ItemEvent e)
  {
      //daca s-a selectat un element
    int Selection;
```

```
    //se preia indexul acestuia
    Selection = Selector.getSelectedIndex();
    //si in functie de elementul selectat
    //se modifica fundalul formei
    if (Selection == 0) {
      setBackground (Color.red);
    } else if (Selection == 1) {
      setBackground (Color.magenta);
    } else if (Selection == 2) {
      setBackground (Color.blue);
    } else if (Selection == 3) {
      setBackground (Color.cyan);
    } else if (Selection == 4) {
      setBackground (Color.green);
    } else if (Selection == 5) {
      setBackground (Color.yellow);
    } else if (Selection == 6) {
      setBackground (Color.white);
    } else if (Selection == 7) {
      setBackground (Color.gray);
    } else if (Selection == 8) {
      setBackground (Color.black);
    }
  }
}
```

Meniuri

Mai sus, am menționat că un *Frame* poate avea un meniu. Pentru a asigura un meniu aplicației, acesta va sta într-un Frame. Implementarea unui meniu în cadrul unui *Frame*, implică o îmbinare a mai multor elemente și anume: *MenuBar*, *Menu*, *MenuItem*.

Pentru a afișa un meniu, acesta va fi pus într-un *MenuBar*, care la rândul lui va fi adăugat unui Frame. Un meniu poate conține un *MenuItem*, dar poate conține și alte meniuri care formează submeniurile sale. Componentele MenuItem sunt materialele construite ca și panel-ele care formează cortinele. Cortina este *Menu*. Meniurile sunt atașate de un MenuBar. Această bară de meniuri va sta de obicei în partea de sus a Frame-ului ca în figura de mai jos:

Figura 11.4 Componentele unui meniu

În figura de mai jos avem toate aceste componente plasate in ierarhia AWT.

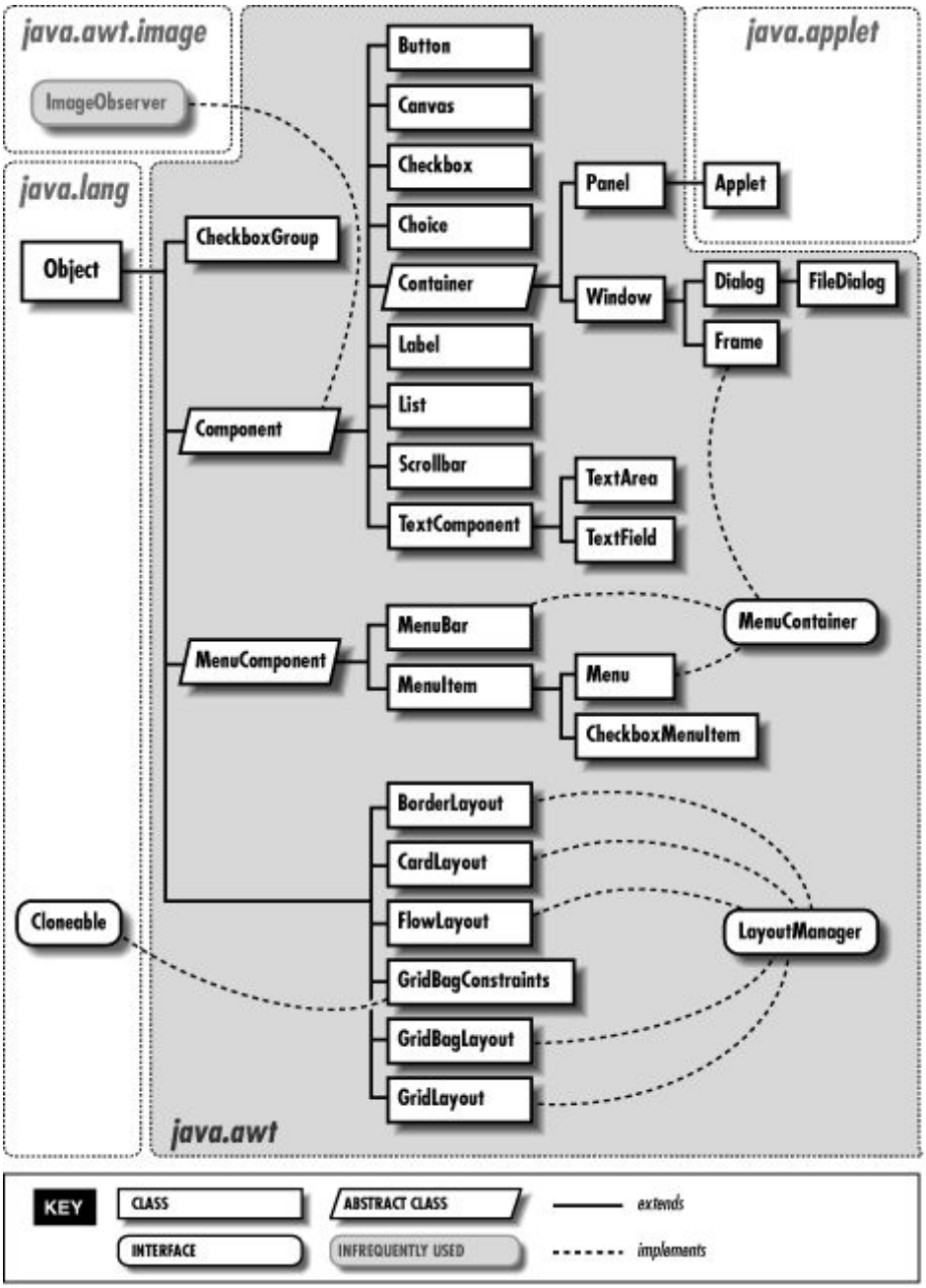

Figura 11.5 Ierarhia componentelor de meniu

MenuComponent

Aceasta este o clasă abstractă ce reprezintă părintele obiectelor legate de meniuri. Nu se va crea niciodată o instanță a acestei clase, ci se poate eventual moșteni. Utilitatea acesteia, este aceea de a conține o serie de metode valabile pentru celelalte componente de care vom vorbi.

```
public Font getFont ()
public void setFont (Font f)
```

Cele două metode permit preluarea/setarea fontului asociat unei componente de meniu. Atunci când se creează o componentă de meniu, fontul inițial ala acesteia este *null*.

```
public String getName ()
public void setName (String name)
```

Metodele permit preluarea/setarea numelui componentei de meniu. Fiecare obiect de subclasă a *MenuComponent* va avea un nume la instanțiere. A doua metodă permite schimbarea acestui nume.

```
public MenuComponentPeer getPeer ()
```

Această metode permite preluarea peer-ului asociat componentei de meniu.

```
public boolean postEvent (Event e)
```

Metoda este folosită pentru a transmite un eveniment *e*, componentei *MenuComponent*. Evenimentul este transmis *Frame*-ului, la obiectul din vârful ierarhiei ce îl conține pe *MenuComponent*. Dacă suprascriem această metodă suntem obligați să transmitem mai departe celorlalte componente, prin apelul return `super.postEvent (e);`.

```
protected void processEvent(AWTEvent e)
```

Această metodă primește toate evenimentele *AWTEvents* ce au ca țintă o subclasă a *MenuComponent*. Mai apoi, acestea pot fi transmise folosind metoda *processEvent()*.

MenuContainer

Aceasta este o interfață implementată de patru tipuri de Container-e și anume *Frame*, *Meniu* și *MenuBar* și *Container*.

Metodele

```
public abstract Font getFont ()
```

Metoda returnează fontul aferent componentei de meniu. Clasa *MenuItem* implementează această metodă, astfel că toate subclasele o moștenesc.

```
public abstract boolean postEvent (Event e)
```

Metoda va transmite evenimentul obiectului care implementează această metodă.

```
public abstract void remove (MenuComponent component)
```

Metoda permite ștergerea componentei specificată parametru din obiectul clasei ce implementează această metodă.

MenuShortcut

Aceasta este o clasă ce reprezintă o scurtătură de taste pentru un *MenuItem*. Atunci când au loc aceste evenimente, un eveniment de acțiune este generat astfel încât el declanșează acțiunea unei componente de meniu.

Metode

```
public MenuShortcut (int key)
```

Constructorul creează un obiect *MenuShortcut* cu cheia `key`. Această cheie reprezintă valoarea întreagă returnată de evenimentul KEY_PRESS.

```
public MenuShortcut(int key, boolean useShiftModifier)
```

Acest constructor permite, în plus, specificarea faptului că este folosită o combinație de taste gen *Shift* sau nu.

```
public int getKey ()
```

Metode returnează codul cheii care a declanșat *MenuShortcut* actual.

```
public boolean usesShiftModifier()
```

Această metodă specifică faptul că obiectul *MenuShortcut* impune ca tasta Shift să fie apăsată sau nu.

MenuItem

În principiu acesta este elementul ce se găsește într-un meniu.

Metodele

```
public MenuItem ()
public MenuItem (String label)
public MenuItem (String label, MenuShortcut shortcut)
```

Prima metodă creează un item de meniu cu o etichetă goală și nici o scurtătură de taste.

Al doilea constructor creează un meniu ce permite specificarea unei etichete, în timp ce al treilea mai adaugă și o scurtătură de taste.

```
public String getLabel ()
public void setLabel (String label)
```

Aceste două metode permit preluarea/setarea etichetei asociate cu item-ul curent.

```
public MenuShortcut getMenuShortcut ()
public void setShortcut (MenuShortcut shortcut)
```

Aceste metode permit preluarea/setarea scurtăturile de taste.

```
public void deleteMenuShortcut ()
```

Metoda permite ștergerea unor scurtături de taste asociate cu item-ul curent.

```
public boolean isEnabled ()
public synchronized void setEnabled(boolean b)
```

Metoda `isEnabled` verifică dacă item-ul de meniu este activat. Un item activat poate fi selectat de utilizator, iar unul dezactivat apare scris cu gri.

Metoda `setEnabled` activează/dezactivează item-ul de meniu, conform parametrului.

```
public synchronized void enable ()
public synchronized void disable ()
```

Metoda permite activarea/dezactivarea unui item de meniu.

```
public String getActionCommand()
public void setActionCommand(String command)
```

Metoda `getActionCommand()` permite preluarea comenzii asociate cu item-ul de meniu actual. Comanda, în mod automat, este eticheta item-ului. Aceasta se poate modifica folosind metoda `setActionCommand` ce poate specifica alt *String* ce va trece drept noua comandă.

```
public void addActionListener(ItemListener listener)
public void removeActionListener(ItemListener listener)
```

Aceste metode permit abonarea/dezabonarea de la evenimente de tip *action*. În general sunt valabile evenimentele asociate componentelor AWT.

Menu

Acestea sunt obiectele care conțin elementele de meniu descrise mai sus.

Metodele

```
public Menu ()
public Menu (String label)
public Menu (String label, boolean tearOff)
```

Primul constructor permite instanțierea unui meniu care nu are etichetă și care nu poate fi oprit.

Al doilea constructor permite instanțierea unui meniu cu etichetă, care nu poate fi oprit.

Al treilea constructor creează un meniu cu eticheta `label` dar ce poate fi oprit sau nu. Aceasta se va seta folosind al treilea parametru.

```
public int getItemCount()
public int countItems ()
```

Aceste metode permit aflarea numărului de elemente dintr-un meniu.

```
public MenuItem getItem (int index)
```

Metoda returnează elementul de la poziția dată de `index`. Dacă `index` este invalid, atunci metoda aruncă excepția `ArrayIndexOutOfBoundsException`.

```
public synchronized MenuItem add (MenuItem item)
```

Această metodă permite adăugarea unui obiect `item` în meniu. Eticheta asociată cu `item` este afișată în meniu. Dacă `item` se află deja într-un alt meniu atunci este șters de acolo.

```
public void add (String label)
```

Metoda permite crearea unui item de meniu cu eticheta `label` și adăugarea acestuia la meniu.

```
public synchronized void insert(MenuItem item, int index)
public synchronized void insert(String label, int index)
```

Metodele permit adăugarea unui item la indexul specificat. Dacă indexul nu este valid atunci va fi aruncată excepția *IllegalArgumentException*.

```
public void addSeparator ()
public void insertSeparator(int index)
```

Meniul nu va genera un eveniment atunci când este selectat, evenimentul fiind generat de selectarea unui *MenuItem*.

MenuBar

Aceasta este componenta ce va fi adăugată *Frame*-ului.

Metodele

```
public MenuBar()
```

Constructorul permite crearea unui *ManuBar* gol. Pentru a adăuga meniuri acestuia se va folosi metoda *add()*.

```
public int getMenuCount ()
public int countMenus ()
```

Metodele returnează numărul de meniuri aflate în cadrul barei de meniuri.

```
public Menu getMenu (int index)
```

Metoda returnează meniul de la poziția `index`. Dacă `index` nu este valid, metoda aruncă excepția *ArrayIndexOutOfBoundsException*.

```
public synchronized Menu add (Menu m)
```

Metoda permite adăugarea unui meniu `m` pe bara de meniuri. Eticheta care a fost folosită pentru crearea lui `m` va fi afișată pe bara de meniuri. Dacă `m` se găsește pe o bară de meniuri deja, el va fi șters.

```
public synchronized void remove (int index)
public synchronized void remove (MenuComponent component)
```

Metoda `remove` șterge componenta de la poziția `index`, dacă este vorba de prima metodă. În cazul celei de-a doua metodă se șterge meniul `component` din bara de meniuri.

```
public MenuItem getShortcutMenuItem (MenuShortcut shortcut)
```

Metoda returnează MenuItem asociat cu scurtătura de taste `shortcut`. Dacă acest item de meniu nu există, se returnează *null*.

```
public synchronized Enumeration shortcuts()
```

Pentru a afla scurtăturile de taste din obiectele asociate barei de meniuri actuale, avem metoda *shortcuts()*.

```
public Menu getHelpMenu ()
public synchronized void setHelpMenu (Menu m)
```

Metoda returnează meniul ce a fost desemnat ca meniu de ajutor prin intermediul celei de-a doua metode. Metoda `setHelpMenu` setează meniul de ajutor, și anume îl va deplasa maxim la stânga pe `m`.

În exemplul de mai jos sunt cuprinse toate elementele prezentate mai sus.

```
import java.awt.*;
import java.awt.event.*;
//clasa proprie de item de meniu
class MyMenuItem extends MenuItem
{
    //in care adaugam un ascultator
    //obiectului al
    //care va fi unul de tip MenuDemo
    public MyMenuItem (String s, ActionListener al)
    {
        super (s);
        addActionListener (al);
    }
}
public class MenuDemo extends Frame implements ActionListener, ItemListener
{
    public MenuDemo ()
    {
        super ("MenuTest");
        MenuItem mi;
        //se creeaza meniu file
        Menu file = new Menu ("File", true);
        //caruia is se adauga un meniu Open
        file.add (new MyMenuItem ("Open", this));
        //si un meniu Close (care este dezactivat)
        mi = file.add (new MyMenuItem ("Close", this));
        mi.setEnabled (false);
        //cream un submeniu Extras cu trei meniuri
        Menu extras = new Menu ("Extras", false);
        //unul de tip checkbox
        CheckboxMenuItem ck = new CheckboxMenuItem ("What");
        ck.addItemListener(this);
        mi = extras.add (ck);
        //iar celelalte normale ( A1 si A2)
        mi = extras.add (new MyMenuItem ("Second submenu (A1)", this));
```

```
        mi.setActionCommand ("A1");
        mi = extras.add (new MyMenuItem ("Third submenu (A2)", this));
        mi.setActionCommand ("A2");
        file.add (extras);
        //se adauga o linie de separatie
        file.addSeparator();
        //si ultimul meniu Quit
        file.add (new MyMenuItem ("Quit", this));
        //dupa meniul file, barei de meniuri
        //i se adauga meniul help
        Menu help = new Menu("Help");
        //cu o optiune, About
        help.add (new MyMenuItem ("About", this));
        MenuBar mb = new MenuBar();
        //adugam barei de meniuri cele doua meniuri
        mb.add (file);
        mb.add (help);
        mb.setHelpMenu (help);
        //si bara o atasam frame-ului
        setMenuBar (mb);
        setSize (200, 200);
        enableEvents (AWTEvent.WINDOW_EVENT_MASK);
    }
    //atunci cand apasam unul din meniuri
    public void actionPerformed(ActionEvent e)
    {
        if (e.getActionCommand().equals("Quit"))
        {
            System.exit(0);
        }
        //afisam selectia efectuata
        System.out.println ("User selected " + e.getActionCommand());

    }
     public void itemStateChanged(ItemEvent e)
     {
         //daca meniul este de tip selectabil CheckBoxmenu
         if (e.getSource() instanceof ItemSelectable)
        {
            ItemSelectable is = (ItemSelectable)e.getSource();
            System.out.println ("The value is: " +
(is.getSelectedObjects()!=null));
        }
     }
    protected void processWindowEvent(WindowEvent e)
    {
        if (e.getID() == WindowEvent.WINDOW_CLOSING)
        {
```

```
        //notificam ascultatorii de eveniment
        super.processWindowEvent(e);
        //si se inchide procesul
        System.exit(0);
        }
        else
        {
            super.processWindowEvent(e);
        }
    }
    public static void main (String []args)
    {
        MenuDemo f = new MenuDemo();
        f.setVisible(true);
    }
}
```