Aplicatii JAVA

5

JAVA
Interfata grafica AWT (partea II)

Adrian Runceanu
www.runceanu.ro/adrian

Curs 5

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

5. Interfata grafica AWT

1. Privire de ansamblu asupra interfetei grafice
2. Componente AWT
3. Gestionari de pozitionare
4. Gruparea componentelor (Clasa Panel)

(prezentate in cursul 4)

1. Tratarea evenimentelor
2. Folosirea ferestrelor in AWT

5. Tratarea evenimentelor

Un *eveniment* este produs de o actiune a utilizatorului asupra unei componente grafice si reprezinta *mecanismul prin care utilizatorul comunica efectiv cu programul*.

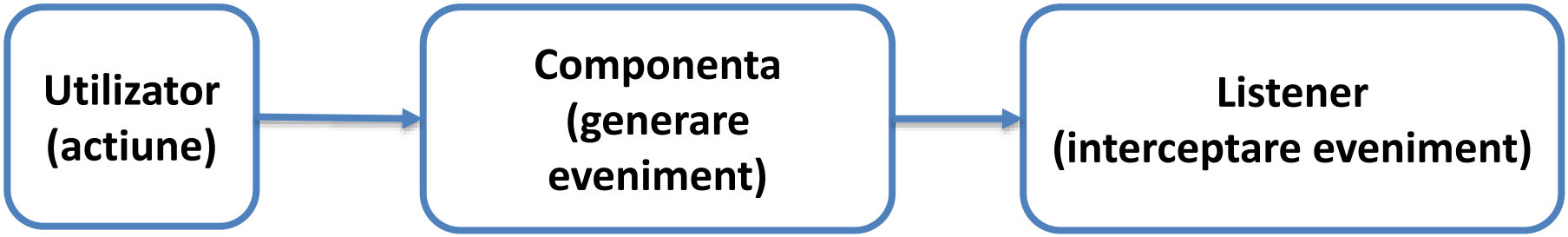
Exemple de evenimente sunt:

- apasarea unui buton
- modificarea textului într-un control de editare
- închiderea, redimensionarea unei ferestre, etc.

Componentele care genereaza anumite evenimente se mai numesc si surse de evenimente.

5. Tratarea evenimentelor

- Interceptarea evenimentelor generate de componentele unui program se realizeaza prin intermediul unor clase de tip **listener** (ascultator, consumator de evenimente).
- In Java, orice obiect poate "consuma" evenimentele generate de o anumita componenta grafica.



5. Tratarea evenimentelor

Asadar, pentru *a scrie cod care sa se execute în momentul în care utilizatorul interactioneaza cu o componenta grafica* trebuie sa facem urmatoarele 2 lucruri:

1. *sa scriem o clasa de tip* listener *care sa "asculte" evenimentele produse de acea componenta* si *în cadrul acestei clase sa implementam metode specifice pentru tratarea lor*
2. sa comunicam componentei sursa ca respectiva clasa îi "asculta" evenimentele pe care le genereaza, cu alte cuvinte sa înregistram acea clasa drept "consumator" al evenimentelor produse de componenta respectiva.

5. Tratarea evenimentelor

- *Evenimentele sunt, ca orice altceva în Java, obiecte*.
- *Clasele care descriu aceste obiecte se împart în mai multe tipuri în functie de componenta care le genereaza, mai precis în functie de actiunea utilizatorului asupra acesteia.*

- Pentru fiecare tip de eveniment exista o clasa care instantiaza obiecte de acel tip (exemple):
  - evenimentul generat de actionarea unui buton este implementat prin clasa ActionEvent
  - evenimentul generat de modificarea unui text prin clasa TextEvent, etc.
- Toate aceste clase au ca superclasa comuna clasa **AWTEvent**.

5. Tratarea evenimentelor

- O clasa consumatoare de evenimente (listener) poate fi orice clasa care specifica în declaratia sa ca doreste sa asculte evenimente de un anumit tip.

- *Acest lucru se realizeaza prin implementarea unei interfete specifice fiecarui tip de eveniment.*

5. Tratarea evenimentelor

- Astfel, pentru ascultarea evenimentelor de tip ActionEvent clasa respectiva trebuie sa implementeze interfata ActionListener, pentru TextEvent interfata care trebuie implementata este TextListener, etc.

- Toate aceste interfete au suprainterfata comuna EventListener.
- Exemple:

```
class AscultaButoane implements ActionListener

class AscultaTexte implements TextListener
```

5. Tratarea evenimentelor

- Intrucât o clasa poate implementa oricâte interfete ea va putea sa asculte evenimente de mai multe tipuri:

```
class Ascultator implements ActionListener, TextListener
```

- Vom vedea în continuare metodele fiecarei interfete pentru a sti ce trebuie sa implementeze o clasa consumatoare de evenimente.

5. Tratarea evenimentelor

Exemplu de tratare a evenimentelor

- Inainte de a detalia aspectele prezentate mai sus, sa consideram un exemplu de tratare a evenimentelor.

- Vom crea o fereastra care sa contina doua butoane cu numele "OK", repectiv "Cancel".

- La apasarea fiecarui buton vom scrie pe bara titlu a ferestrei mesajul " Ati apasat butonul ...".

5. Tratarea evenimentelor

Exemplu: Ascultarea evenimentelor de la doua butoane

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame {
        public Fereastra(String titlu) {
                super(titlu);
        }
        public void initializare() {
        setLayout(new FlowLayout()); // se stabileste gestionarul
        setSize(200, 100);   // se dimensioneaza fereastra
```

5. Tratarea evenimentelor

```
        Button b1 = new Button("OK");
        add(b1);        // se adauga primul buton
        Button b2 = new Button("Cancel");
        add(b2);        // se adauga al doilea buton
        Ascultator listener = new Ascultator(this);
        b1.addActionListener(listener);
        b2.addActionListener(listener);
// ambele butoane sunt ascultate de obiectul "listener" instanta a clasei Ascultator, definita ulterior
        }
}
```

5. Tratarea evenimentelor

```
class Ascultator implements ActionListener {
private Fereastra f;
public Ascultator (Fereastra f) {
        this.f = f;
}
        // metoda interfetei ActionListener
        public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        // numele comenzii este numele butonului apasat
        System.out.println(e.toString());
        if (command.equals("OK")) f.setTitle("Ati apasat OK");
        else
           if (command.equals("Cancel")) f.setTitle("Ati apasat Cancel");
        }
}
```

5. Tratarea evenimentelor

```
public class TestEvent { // fereastra principala

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

5. Tratarea evenimentelor

- Nu este obligatoriu sa definim clase speciale pentru ascultarea evenimentelor.

- In exemplul de mai sus am definit o clasa speciala "Ascultator" pentru a intercepta evenimentele produse de cele doua butoane si din acest motiv a trebuit sa trimitem ca parametru acestei clase instanta la fereastra noastra.

5. Tratarea evenimentelor

Mai corect ar fi fost sa folosim chiar clasa "Fereastra" pentru a-si asculta evenimentele produse de componentele sale:

```
class Fereastra extends Frame implements ActionListener{
        public Fereastra(String titlu) {
                super(titlu);
        }
        public void initializare() {
                . . .
                b1.addActionListener(this);
                b2.addActionListener(this);
//ambele butoane sunt ascultate chiar din clasa Fereastra
//deci ascultatorul este instanta curenta: this
}
```

5. Tratarea evenimentelor

```
public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
// numele comenzii este numele butonului apasat
        System.out.println(e.toString());
        if (command.equals("OK")) this.setTitle("Ati apasat OK");
        else
           if (command.equals("Cancel")) this.setTitle("Ati apasat
Cancel");
        }
}
. . .
```

Asadar, *orice clasa poate asculta evenimente de orice tip cu conditia sa implementeze interfetele specifice acelor evenimente*.

5. Tratarea evenimentelor

Tipuri de evenimente si componentele care le genereaza.

- In tabelul de mai jos sunt prezentate în stânga tipurile de evenimente si interfetele iar în dreapta lista componentelor ce pot genera evenimente de acel tip precum si o scurta explicatie despre motivul care le provoaca:

5. Tratarea evenimentelor

Eveniment/Interfata	Componente care genereaza acest eveniment
ActionEvent ActionListener	Button, List, TextField, MenuItem, CheckboxMenuItem, Menu, PopupMenu Actiuni asupra unui control
AdjustmentEvent AdjustmentListener	Scrollbar si orice clasa care implementeaza interfata Adjustable Modificarea unei valori variind între doua limite
ComponentEvent ComponentListener	Component si subclasele sale Redimensionari, deplasari, ascunderi ale componentelor
ContainerEvent ContainerListener	Container si subclasele sale Adaugarea, stergerea componentelor la un container
FocusEvent FocusListener	Component si subclasele sale Preluarea, pierderea focusului de catre o componenta
KeyEvent KeyListener	Component si subclasele sale Apasarea, eliberarii unei taste când focusul este pe o anumita componenta.
MouseEvent MouseListener	Component si subclasele sale Click, apasare, eliberare a mouse-ului pe o componenta, intrarea, iesirea mouse-ului pe/de pe suprafata unei componente
MouseEvent MouseMotionListener	Component si subclasele sale Miscarea sau "târârea" (drag) mouse-ului pe suprafata unei componente
WindowEvent WindowListener	Window si subclasele sale Dialog, FileDialog, Frame Inchiderea, maximizarea, minimizarea, redimensionarea unei ferestre
ItemEvent ItemListener	Checkbox, CheckboxMenuItem, Choice, List si orice clasa care implementeaza interfata ItemSelectable
	Selectia, deselecttia unui articol dintr-o lista sau meniu.
TextEvent TextListener	Orice clasa derivata din TextComponent cum ar fi : TextArea, TextField Modificarea textului dintr-o componenta de editare a textului

5. Tratarea evenimentelor

Evenimente suportate de o componenta

Urmatorul tabel prezinta o clasificare a evenimentelor în functie de componentele care le suporta:

Componenta	Evenimente suportate de componenta
Adjustable	AdjustmentEvent
Applet	ContainerEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Button	ActionEvent, FocusEvent, KeyEvent, MouseEvent, ComponentEvent
Canvas	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
ScrollPane	ContainerEvent, 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

5. Tratarea evenimentelor

Metodele interfetelor de tip "listener"

Orice clasa care trateaza evenimente trebuie sa implementeze obligatoriu metodele interfetelor corespunzatoare evenimentelor pe care le trateaza. Tabelul prezinta, pentru fiecare interfata, metodele puse la dispozitie si care trebuie implementate de clasa ascultator.

Interfata	Metodele interfetei
ActionListener	actionPerformed(ActionEvent)
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)
ComponentListener	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)
ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
WindowListener	windowOpened(WindowEvent) windowClosing(WindowEvent) windowClosed(WindowEvent) windowActivated(WindowEvent) windowDeactivated(WindowEvent) windowIconified(WindowEvent) windowDeiconified(WindowEvent)
ItemListener	itemStateChanged(ItemEvent)
TextListener	textValueChanged(TextEvent)

5. Tratarea evenimentelor

Folosirea adaptorilor si a claselor interne în tratarea evenimentelor

- Am vazut ca o clasa care trateaza evenimente de un anumit tip trebuie sa implementeze interfata corespunzatoare acelui tip.
- Aceasta înseamna ca *trebuie sa implementeze obligatoriu toate metodele definite de acea interfata, chiar daca nu specifica nici un cod pentru unele dintre ele*.
- Sunt însa situatii când acest lucru este suparator, mai ales atunci când nu ne intereseaza decât o singura metoda a interfetei.

5. Tratarea evenimentelor

Un exemplu sugestiv este urmatorul: *o fereastra care nu are specificat cod pentru tratarea evenimentelor sale nu poate fi închisa cu butonul standard marcat cu 'x' din coltul dreapta sus si nici cu combinatia de taste Alt+F4*.
Pentru a realiza acest lucru trebuie interceptat evenimentul de închidere a ferestrei în metoda windowClosing si apelata apoi metoda dispose de închidere a ferestrei, eventual urmata de iesirea din program, în cazul când este vorba de fereastra principala a aplicatiei.
Aceasta înseamna ca trebuie sa implementam interfata WindowListener care are nu mai putin de sapte metode.

5. Tratarea evenimentelor

```
// Crearea unei ferestre cu ascultarea evenimentelor sale
// folosind implementarea directa a interfetei WindowListener

import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame implements
WindowListener {
        public Fereastra(String titlu) {
                super(titlu);
                this.addWindowListener(this);
        }
```

5. Tratarea evenimentelor

```
// metodele interfetei WindowListener
      public void windowOpened(WindowEvent e) {}
      public void windowClosing(WindowEvent e) {
            dispose();              // inchidere fereastra
            System.exit(0);         // terminare program
      }
      public void windowClosed(WindowEvent e) {}
      public void windowIconified(WindowEvent e) {}
      public void windowDeiconified(WindowEvent e) {}
      public void windowActivated(WindowEvent e) {}
      public void windowDeactivated(WindowEvent e) {}
      }

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

5. Tratarea evenimentelor

- Observati ca trebuie sa implementam toate metodele interfetei, chiar daca nu scriem nici un cod pentru ele.
- Singura metoda care ne intereseaza este windowClosing în care specificam ce trebuie facut atunci când utilizatorul doreste sa închida fereastra.
- Pentru a evita scrierea inutila a acestor metode exista o serie de clase care implementeaza interfetele de tip "listener" fara a specifica nici un cod pentru metodele lor.
- Aceste clase se numesc *adaptori*.

5. Tratarea evenimentelor

Folosirea adaptorilor

- *Un adaptor este o clasa abstracta care implementeaza o interfata de tip "listener".*

- Scopul unei astfel de clase este ca la crearea unui "ascultator" de evenimente, în loc sa implementa o anumita interfata si implicit toate metodele sale, *sa extindem adaptorul corespunzator interfetei respective (daca are!) si sa supradefinim doar metodele care ne intereseaza (cele în care vrem sa scriem o anumita secventa de cod).*

5. Tratarea evenimentelor

Adaptorul interfetei WindowListener este WindowAdapter iar folosirea acestuia este data în exemplul de mai jos:

// Crearea unei ferestre cu ascultarea evenimentelor sale folosind extinderea clasei WindowAdapter

```
import java.awt.*;
import java.awt.event.*;
class Fereastra extends Frame {
        public Fereastra(String titlu) {
                super(titlu);
                this.addWindowListener(new Ascultator());
        }
}
class Ascultator extends WindowAdapter {
// supradefinim metodele care ne intereseaza
        public void windowClosing(WindowEvent e) {
                System.exit(0);
        }
}
```

5. Tratarea evenimentelor

Avantajul clar al acestei modalitati de tratare a evenimentelor este reducerea codului programului, acesta devenind mult mai usor lizibil.

5. Tratarea evenimentelor

Insa exista si doua dezavantaje majore:

1. Dupa cum ati observat, fata de exemplul anterior clasa "Fereastra" nu poate extinde WindowAdapter deoarece ea extinde deja clasa Frame si din acest motiv am construi o noua clasa numita "Ascultator".

Vom vedea însa ca acest dezavantaj poate fi eliminat prin folosirea unei clase interne.

2. Un alt dezavantaj este ca *orice greseala de sintaxa în declararea unei metode a interfetei nu va produce o eroare de compilare dar nici nu va supradefini metoda interfetei* ci, pur si simplu, va crea o metoda a clasei respective.

5. Tratarea evenimentelor

```
class Ascultator extends WindowAdapter {
// in loc de windowClosing scriem WindowClosing
// nu supradefinim vreo metoda a clasei WindowAdapter
// nu da nici o eroare
// nu face nimic !
public void WindowClosing(WindowEvent e) {
                    System.exit(0);
        }
}
```

5. Tratarea evenimentelor

In tabelul de mai jos sunt dati toti adaptorii interfetelor de tip "listener" - se observa ca o interfata XXXListener are un adaptor de tipul XXXAdapter. Interfetele care nu au un adaptor sunt cele care definesc o singura metoda si prin urmare crearea unei clase adaptor nu îsi are rostul.

Interfata "listener"	Adaptor
ActionListener	nu are
AdjustmentListener	nu are
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
ItemListener	nu are
KeyListener	KeyAdapter
MouseListener	Mouse
MouseMotionListener	MouseMotionAdapter
TextListener	nu are
WindowListener	WindowAdapter

5. Tratarea evenimentelor

Folosirea claselor interne (anonime)

- Stim ca o *clasa interna este o clasa declarata în cadrul altei clase* iar *clasele anonime sunt acele clase interne folosite doar pentru instantierea unui singur obiect de acel tip*.
- Un exemplu tipic de folosire a lor este instantierea adaptorilor direct în corpul unei clase care contine componente ale caror evenimente trebuie interceptate.

5. Tratarea evenimentelor

Clasa "Fereastra" din exemplul anterior poate fi scrisa astfel:

```
class Fereastra extends Frame {
        public Fereastra(String titlu) {
                super(titlu);
        this.addWindowListener (new WindowAdapter() {
                //corpul clasei anonime
                public void windowClosing(WindowEvent e) {
                        System.exit(0);         }
                });
        }
}
```

Se observa cum codul programului a fost redus substantial prin folosirea unui adaptor si a unei clase anonime.

5. Interfata grafica AWT

1. Privire de ansamblu asupra interfetei grafice
2. Componente AWT
3. Gestionari de pozitionare
4. Gruparea componentelor (Clasa Panel)
5. Tratarea evenimentelor
6. Folosirea ferestrelor in AWT

6. Folosirea ferestrelor in AWT

Dupa cum am vazut suprafetele de afisare ale componentelor grafice (containerele) sunt extensii ale clasei **Container**.

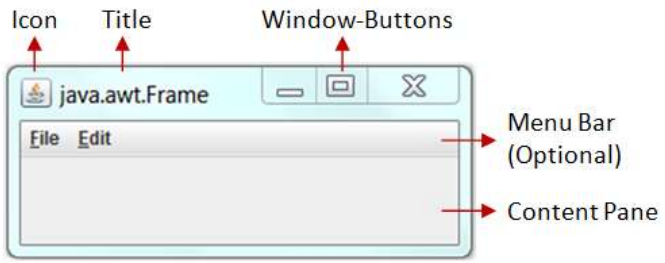
- O categorie aparte a acestor containere o reprezinta ferestrele.
- Spre deosebire de un applet care își poate plasa componentele direct pe suprafata de afisare a browser-ului în care ruleaza, o aplicatie independenta are nevoie de propriile ferestre pe care sa faca afisarea componentelor sale grafice.
- Pentru dezvoltarea aplicatiilor care folosesc grafica se vor folosi clasele Window si subclasele sale directe Frame si Dialog.

6. Folosirea ferestrelor in AWT

1. Clasa Window

Clasa Window este rar utilizata în mod direct.

- Ea permite crearea unor ferestre top-level care nu au chenar si nici bara de meniuri.
- Pentru a crea ferestre mai complexe se utilizeaza clasele Frame si Dialog.



(*) http://stackoverflow.com/questions/8976874/show-two-dialogs-on-top-of-each-other-using-java-swing

(*) http://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html

6. Folosirea ferestrelor in AWT

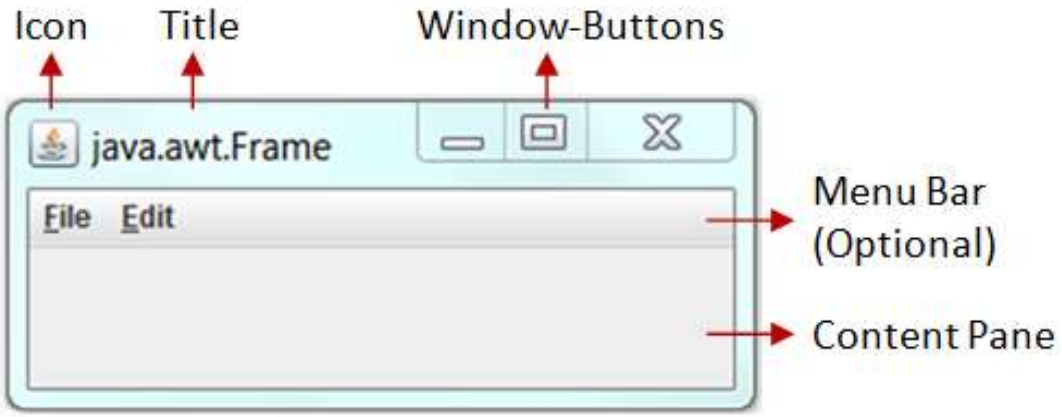
Metodele mai importante ale clasei Window (mostenite de toate subclasele sale) sunt date în tabelul de mai jos:

`void dispose()`	Distruge (închide) fereastra si si elibereaza toate resursele acesteia
`Component getFocusOwner()`	Returneaza componenta ferestrei care are focus-ul daca si numai daca fereastra este activa
`Window getOwnedWindows()`	Returneaza un vector cu toate ferestrele subclase ale ferestrei respective
`Window getOwner()`	Returneaza parintele (superclasa) ferestrei
`void hide()`	Face fereastra invizibila fara a o distruge însa. Pentru a redeveni vizibila apelati metoda show
`boolean isShowing()`	Testeaza daca fereastra este vizibila sau nu
`void pack()`	Redimensioneaza automat fereastra la o suprafata optima care sa cuprinda toate componentele sale. Trebuie apelata, în general, dupa adaugarea tuturor componentelor pe suprafata ferestrei.
`void show()`	Face vizibila o fereastra creata. Implicit o fereastra nou creata nu este vizibila.
`void toBack()` `void toFront()`	Trimite fereastra în spatele celorlalte ferestre deschise Aduce fereastra în fata celorlalte ferestre deschise

6. Folosirea ferestrelor in AWT

2. Clasa Frame

- Este subclasa directa a clasei Window si este folosita pentru crearea de ferestre independente si functionale, eventual continând bare de meniuri.
- *Orice aplicatie grafica independenta trebuie sa aiba cel putin o fereastra, numita si fereastra principala, care va fi afisata la pornirea programului.*



6. Folosirea ferestrelor in AWT

Constructorii clasei Frame sunt:

1. Frame () - Construieste o fereastra, fara titlu, initial invizibila.

2. Frame(String title) - Construieste o fereastra, cu titlul specificat, initial invizibila.

6. Folosirea ferestrelor in AWT

Asadar, o fereastra nou creata este invizibila.
Pentru a fi facuta vizibila se va apela metoda show definita în superclasa Window.
Exemplu:

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

6. Folosirea ferestrelor in AWT

Crearea ferestrelor prin instantierea obiectelor de tip Frame este mai putin uzuala.
De obicei, ferestrele unui program vor fi definite în clase separate care extind clasa Frame, ca în exemplul de mai jos:

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

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

6. Folosirea ferestrelor in AWT

Gestionarul de pozitionare implicit al clasei Window este BorderLayout.

- Din acest motiv, în momentul în care fereastra este creata dar nici o componenta grafica nu este pusa pe suprafata ei, suprafata de afisare a ferestrei va fi nula.

- Acelasi efect îl vom obtine daca o redimenionam si apelam apoi metoda pack care determina dimeniunea suprafetei de afisare în functie de componentele grafice afisate pe ea.
- Se observa de asemenea ca butonul de închidere a ferestrei nu este functional.
- Interceptarea evenimentelor se face prin implementarea interfetei WindowListener si prin adaugarea în lista ascultatorilor ferestrei (uzual) chiar a obiectului care implementeaza fereastra sau prin folosirea unor adaptori si clase anonime.

6. Folosirea ferestrelor in AWT

Metodele mai folosite ale clasei Frame sunt:

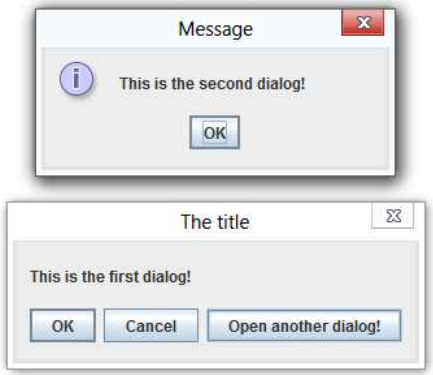
`static Framest getFrames()`	Metoda statica ce returneaza lista tuturor ferestrelor deschise ale unei aplicatii
`Image getIconImage()` `void setIconImage(Image img)`	Afla/seteaza imaginea(iconita) care sa fie afisata atunci când fereastra este minimizata
`MenuBar getMenuBar()` `void setMenuBar(MenuBar mb)`	Afla/seteaza bara de meniuri a ferestrei
`int getState()` `void setState(int s)`	Returneaza/seteaza starea ferestrei. O fereastra se poat gasi în doua stari, descrise de constantele: Frame.ICONIFIED (daca este minimizata) Frame.NORMAL (daca nu este minimizata).
`String getTitle()` `void setTitle()`	Afla/seteaza titlul ferestrei
`boolean isResizable()` `void setResizable(boolean r)`	Determina/stabileste daca fereastra poate fi redimenionata de utilizator.

6. Folosirea ferestrelor in AWT

3. Clasa Dialog

Toate interfetele grafice ofera un tip special de *ferestre destinate preluarii datelor de la utilizator*.

- Acestea se numesc *ferestre de dialog sau casete de dialog* si sunt implementate prin intermediul clasei Dialog, subclasa directa a clasei Window.
- Diferenta majora între ferestrele de dialog si ferestrele normale (obiecte de tip Frame) consta în faptul ca o fereastra de dialog este dependenta de o alta fereastra (normala sau tot fereastra dialog), numita si fereastra parinte.



6. Folosirea ferestrelor in AWT

- Cu alte cuvinte, *ferestrele de dialog nu au o existenta de sine statatoare.*

- Când fereastra parinte este distrusa sunt distruse si ferestrele sale de dialog, când este minimizata ferestrele sale de dialog sunt facute invizibile iar când este maximizata acestea sunt aduse la starea în care se gaseau în momentul minimizarii ferestrei parinte.

6. Folosirea ferestrelor in AWT

Ferestrele de dialog pot fi de doua tipuri:

1. **modale**: care blocheaza accesul la fereastra parinte în momentul deschiderii lor - de exemplu, *ferestre de introducere a unor date*, de *alegere a unui fisier în vederea deschideriii*, de *selectare a unei optiuni*, *mesaje de avertizare*, etc;

2. **nemodale**: care nu blocheaza fluxul de intrare catre fereastra parinte - de exemplu, *ferestrele de cautare a unui cuvânt într-un fisier*.

6. Folosirea ferestrelor in AWT

Implicit o fereastra de dialog este nemodala si invizibila.

Constructorii clasei Dialog sunt:

Dialog(Frame parinte)

Dialog(Frame parinte, String titlu)

Dialog(Frame parinte, String titlu, boolean modala)

Dialog(Frame parinte, boolean modala)

Dialog(Dialog parinte)

Dialog(Dialog parinte, String titlu)

Dialog(Dialog parinte, String titlu, boolean modala)

Unde:

- "parinte" reprezina o instanta ferestrei parinte,
- "titlu" reprezinta titlul ferestrei
- iar prin argumentul "modala" specificam daca fereastra de dialog creata va fi modala (true) sau nemodala (false - valoarea implicita).

6. Folosirea ferestrelor in AWT

Pe lânga metodele mostenite de la superclasa Window clasa Dialog mai contine metodele:

`boolean isModal()`	Determina daca fereastra de dialog este modala sau nu.
`void setModal(boolean modala)`	Specifica tipul ferestrei de dialog: modala (true) sau nemodala (false)

- Crearea unei ferestre de dialog este relativ simpla si se realizeaza prin crearea unei clase care sa extinda clasa Dialog.

- Mai complicat este însa modul în care se implementeaza *comunicarea între fereastra de dialog si fereastra parinte*, pentru ca aceasta din urma sa poata folosi datele introduse (sau optiunea specificata) în caseta de dialog.

6. Folosirea ferestrelor in AWT

Exista doua abordari generale:

1. *obiectul care reprezinta dialogul poate sa capteze evenimentele de la componentele de pe suprafata sa* si sa seteze valorile unor variabile ale ferestrei parinte în momentul în care dialogul este încheiat

2. obiectul care creeaza dialogul (fereastra parinte) sa se înregistreze ca ascultator al evenimentelor de la butoanele care determina încheierea dialogului, iar fereastra de dialog sa ofere metode publice prin care datele introduse sa fie preluate din exterior.

6. Folosirea ferestrelor in AWT

Exemplu: *sa se creeze o fereastra de dialog modala pentru introducerea unui sir de caractere*.

- *Fereastra principala a aplicatiei va fi parintele casetei de dialog, va primi sirul de caractere introdus si îsi va modifica titlul ca fiind sirul primit.*
- Deschiderea ferestrei de dialog se va face la apasarea unui buton al ferestrei principale numit "Schimba titlul".
- Dialogul va mai avea doua butoane OK si Cancel pentru terminarea sa cu confirmare, respectiv renuntare.

6. Folosirea ferestrelor in AWT

Cele doua ferestre vor arata astfel:

6. Folosirea ferestrelor in AWT

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

// Fereastra principala a aplicatiei
class FerPrinc extends Frame implements ActionListener{
        public FerPrinc(String titlu) {
                super(titlu);
        this.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                        System.exit(0);
                }
        });
}
```

6. Folosirea ferestrelor in AWT


```
public void initializare() {
        setLayout(new FlowLayout());
        setSize(300, 100);
        Button b = new Button("Schimba titlul");
        add(b);
        b.addActionListener(this);
}
//metoda interfetei ActionListener
public void actionPerformed(ActionEvent e) {
        FerDialog d = new FerDialog(this, "Titlu", true);
        // se verifica inchiderea ferestrei modale de dialog
        if (d.raspuns == null) return;
        setTitle(d.raspuns);
        }
}
```

6. Folosirea ferestrelor in AWT

```
// fereastra de dialog
class FerDialog extends Dialog implements ActionListener {
        public String raspuns = null;
        private TextField text;
        public FerDialog(Frame parinte, String titlu, boolean
modala) {
                super(parinte, titlu, modala);
         this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                raspuns = null;
                dispose();
            }
         });
```

6. Folosirea ferestrelor in AWT

```
        setLayout(new FlowLayout());
        Button ok, cancel;
        ok = new Button("OK");
        cancel = new Button("Cancel");
        text = new TextField("", 50);
        add(cancel);add(text);add(ok);pack();
        ok.addActionListener(this);
        cancel.addActionListener(this);
        show();
}
```

6. Folosirea ferestrelor in AWT

```
// metoda interfetei ActionListener
public void actionPerformed(ActionEvent e) {
// se verifica ce buton a fost apasat
        String buton = e.getActionCommand();
        if (buton.equals("OK")) raspuns = text.getText();
        else
                if (buton.equals("Cancel")) raspuns = null;
        dispose();
        }
}
```

6. Folosirea ferestrelor in AWT

```
// clasa principala
public class TestDialog {
        public static void main(String args[]) {
                FerPrinc f = new FerPrinc("Fereastra
principala");
                f.initializare();
                f.show();
        }
}
```

Referinte

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

Întrebări?