

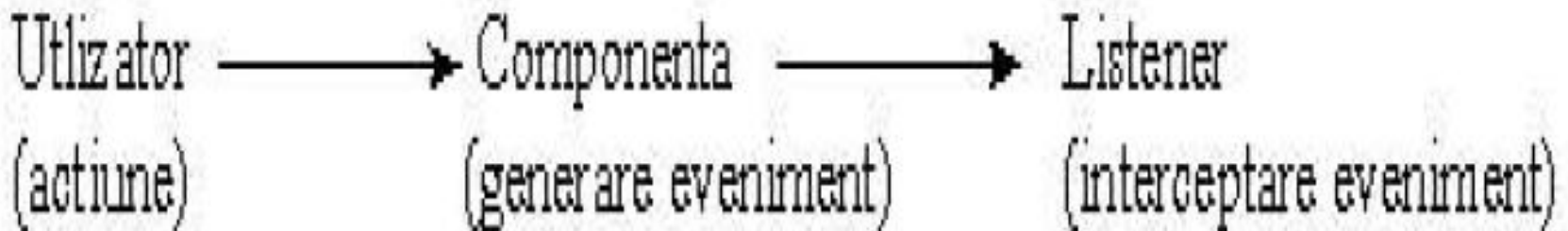
Interfața grafică cu utilizatorul - AWT

Programare Orientată pe Obiecte



Tratarea evenimentelor

- Eveniment: apăsarea unui buton, modificarea textului, închiderea unei ferestre, etc.
- Sursă: componentă care generează un eveniment.
- Interceptarea evenimentelor generate de componentele unui program se realizează prin intermediul unor clase de tip listener
- Evenimentele - AWTEvent:
 ActionEvent, TextEvent.



Ascultătorii - EventListener:

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

Ascultătorii - EventListener:

ActionListener, TextListener.

```
class AscultaButoane implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        // Metoda interfetei ActionListener
        ...
    }
}

class AscultaTexte implements TextListener {
    public void textValueChanged(TextEvent e) {
        // Metoda interfetei TextListener
        ...
    }
}

class Ascultator implements ActionListener, TextListener {
    public void actionPerformed(ActionEvent e) {
        ...
    }
    public void textValueChanged(TextEvent e) {
        ...
    }
}
```

Ascultătorii (2):

- Inregistrarea/eliminarea unei clase în lista ascultătorilor unei componente:
 - addTipEvenimentListener,
 - removeTipEvenimentListener.
- Tratarea evenimentelor se desfășoară astfel:
- Componentele generează evenimente când ceva "interesant" se întâmplă;
- Sursele evenimentelor permit oricărei clase să "asculte" evenimentele sale prin metode de tip addTIPListener;
- O clasă care ascultă evenimente trebuie să implementeze interfețe specifice fiecărui tip de eveniment – acestea descriu metode ce vor fi apelate automat la apariția evenimentelor.

Exemplu: Ascultarea evenimentelor a două butoane

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

class Fereastră extends Frame {
    public Fereastră(String titlu) {
        super(titlu);
        setLayout(new FlowLayout());
        setSize(200, 100);
        Button b1 = new Button("OK");
        Button b2 = new Button("Cancel");
        add(b1);
        add(b2);

        Ascultator listener = new Ascultator(this);
        b1.addActionListener(listener);
        b2.addActionListener(listener);
        // Ambele butoane sunt ascultate de obiectul listener
        // instanta a clasei Ascultator, definita mai jos
    }
}

class Ascultator implements ActionListener {
    private Fereastră f;
    public Ascultator(Fereastră f) {
        this.f = f;
    }

    // Metoda interfeței ActionListener
    public void actionPerformed(ActionEvent e) {
        f.setTitle("Ați apasat " + e.getActionCommand());
    }
}

public class TestEvent1 {
    public static void main(String args[]) {
        Fereastră f = new Fereastră("Test Event");
        f.show();
    }
}
```

Tratarea evenimentelor în fereastră

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

class Fereastră extends Frame implements ActionListener {
    Button ok = new Button("OK");
    Button exit = new Button("Exit");
    int n=0;

    public Fereastră(String titlu) {
        super(titlu);
        setLayout(new FlowLayout());
        setSize(200, 100);
        add(ok);
        add(exit);

        ok.addActionListener(this);
        exit.addActionListener(this);
        // Ambele butoane sunt ascultate în clasa Fereastră
        // deci ascultătorul este instanța curentă: this
    }

    // Metoda interfeței ActionListener
    public void actionPerformed(ActionEvent e) {

        if (e.getSource() == exit)
            System.exit(0); // Terminăm aplicația

        if (e.getSource() == ok) {
            n++;
            this.setTitle("Ați apasat OK de " + n + " ori");
        }
    }
}

public class TestEvent2 {
    public static void main(String args[]) {
        Fereastră f = new Fereastră("Test Event");
        f.show();
    }
}
```

Tipuri de evenimente

Evenimente de nivel jos - apăsare de tastă, mișcarea mouse-ului, etc.

ComponentEvent	Ascundere, deplasare, redimensionare, afișare
ContainerEvent	Adăugare pe container, eliminare
FocusEvent	Obținere, pierdere focus
KeyEvent	Apăsare, eliberare taste, tastare
MouseEvent	Operațiuni cu mouse-ul: click, drag, etc.
WindowEvent	Operațiuni asupra ferestrelor: minimizare, maximizare, etc.

Evenimente semantice

- interacțiunea cu o componentă GUI:
apăsarea unui buton, selectarea unui articol
dintr-o listă, etc.

ActionEvent	Acțiunare
AdjustmentEvent	Ajustarea unei valori
ItemEvent	Schimbarea stării
TextEvent	Schimbarea textului

Componentele AWT și tipurile de evenimente generate

Component	ComponentListener FocusListener KeyListener MouseListener MouseMotionListener
Container	ContainerListener
Window	WindowListener
Button List MenuItem TextField	ActionListener
Choice Checkbox List CheckboxMenuItem	ItemListener
Scrollbar	AdjustmentListener
TextField TextArea	TextListener

Interfețe ascultător

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

Evenimente de la surse multiple

- metoda getSource returnează obiectul responsabil cu generarea evenimentului.

```
public void actionPerformed(ActionEvent e) {
    Object sursa = e.getSource();
    if (sursa instanceof Button) {
        // A fost apasat un buton
        Button btn = (Button) sursa;
        if (btn == ok) { // A fost apasat butonul 'ok'
        }
        ...
    }
    if (sursa instanceof TextField) {
        // S-a apasat Enter dupa editarea textului
        TextField tf = (TextField) sursa;
        if (tf == nume) { // A fost editata componenta 'nume'
        }
        ...
    }
}
```

- ActionEvent conține metoda getActionCommand care, implicit, returnează eticheta butonului care a fost apăsat.

Adaptori și a clase anonime

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

class Fereastră extends Frame implements WindowListener {
    public Fereastră(String titlu) {
        super(titlu);
        this.addWindowListener(this);
    }

    // Metodele interfetei WindowListener
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {
        // Terminare program
        System.exit(0);
    }
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
}

public class TestWindowListener {
    public static void main(String args[]) {
        Fereastră f = new Fereastră("Test WindowListener");
        f.show();
    }
}
```

Folosirea unui adaptor

```
this.addWindowListener(new Ascultator());
```

```
...
```

```
class Ascultator extends WindowAdapter {  
    // Supradefinim metodele care ne intereseaza  
    public void windowClosing(WindowEvent e) {  
        System.exit(0);  
    }  
}
```

- TipXListener - TipXAdapter
- Folosirea unei clase anonime:

```
this.addWindowListener(new WindowAdapter() {  
    public void windowClosing(WindowEvent e) {  
        // Terminam aplicatia  
        System.exit(0);  
    }  
});
```

Structura generală a unei ferestre

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

class Fereastră extends Frame implements ActionListener {

    // Constructorul
    public Fereastră(String titlu) {
        super(titlu);

        // Tratăm evenimentul de închidere a ferestrei
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose(); // închidem fereastră
                // sau terminăm aplicația
                System.exit(0);
            }
        });

        // Eventual, schimbăm gestionarul de poziționare
        setLayout(new FlowLayout());

        // Adăugăm componentele pe suprafața ferestrei
        Button exit = new Button("Exit");
        add(exit);

        // Facem înregistrarea claselor listener
        exit.addActionListener(this);

        // Stabilim dimensiunile
        pack(); // implicit

        //sau explicit
        // setSize(200, 200);
    }

    // Implementăm metodele interfețelor de tip listener
    public void actionPerformed(ActionEvent e) {
```

Structura generală a unei ferestre

```
        System.exit(0);
    }
}

public class TestFrame {
    public static void main(String args[]) {
        // Cream fereastra
        Fereastră f = new Fereastră("0 fereastră");

        // O facem vizibila
        f.show();
    }
}
```

Metode

- getFrames
- setIconImage
- setMenuBar
- setTitle
- setResizable

Tratarea evenimentelor - Exemplu

1. Program pentru afisarea unui buton cu inscriptia "Click Me" si afisarea unei casete de dialog cu titlul "Event Fired" la fiecare clic pe buton (cu mouse). Afisarea casetei de dialog se face astfel:
`JOptionPane.showMessageDialog(new JFrame(), "", "Event Fired!", JOptionPane.PLAIN_MESSAGE);`

Se vor examina pe rand urmatoarele variante de definire a clasei ascultator la evenimente generate de buton:

- a) Cu trei clase separate: clasa ascultator, clasa derivata din "JFrame" care contine si un buton, clasa cu "main" (care afiseaza fereastra).
- b) Cu doua clase: clasa ascultator si clasa derivata din "JFrame" si care contine metoda "main".
- c) Cu o singura clasa: clasa ascultator cu nume inclusa in clasa ce contine metoda "main"
- d) Cu o singura clasa : clasa ascultator anonima, inclusa intr-un bloc (metoda "addActionListener") din clasa ce contine metoda "main".
- e) Cu doua clase: O subclasa a clasei "JButton" care contine si metoda "actionPerformed" si clasa care contine metoda "main".
- f) Cu doua clase : clasa ascultator inclusa intr-o subclasa a clasei "JFrame" (separata de clasa ce contine metoda "main")
- g) O singura clasa care extinde pe "JFrame" si implementeaza "ActionListener" (clasa este si generator si ascultator la evenimente).

a. Cu trei clase separate: clasa ascultator, clasa derivata din "JFrame" care contine si un buton, clasa cu "main" (care afiseaza fereastra).

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class ascultator implements ActionListener{
    public void actionPerformed(ActionEvent e){
        JOptionPane.showMessageDialog(new JFrame(),"Event
        Fired !",JOptionPane.PLAIN_MESSAGE);
    }
}
class jframe extends JFrame{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(new ascultator());
        setSize(250,250);
        setVisible(true);
    }
}
class test{
    public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

b. Cu doua clase: clasa ascultator si clasa derivata din "JFrame" si care contine metoda "main".

```
class ascultator implements ActionListener{
    public void actionPerformed(ActionEvent e){
        JOptionPane.showMessageDialog(new
            JFrame(), "", "Event Fired!",
            JOptionPane.PLAIN_MESSAGE);
    }
}
class jframe extends JFrame{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(new ascultator());
        setSize(250,250);
        setVisible(true);
    }
    public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

c. Cu o singura clasa: clasa ascultator cu nume inclusa in clasa ce contine metoda "main"

```
class JFrame extends JFrame{
    public JFrame(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(new ascultator());
        setSize(250,250);
        setVisible(true);
    }
    class ascultator implements ActionListener{
        public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog(new
            JFrame(),"","Event Fired !",JOptionPane.PLAIN_MESSAGE);
        }
    }
    public static void main(String arg[]){
        JFrame jf=new JFrame();
    }
}
```

d. Cu o singura clasa : clasa ascultator anonima, inclusa intr-un bloc (metoda "addActionListener") din clasa ce contine metoda "main".

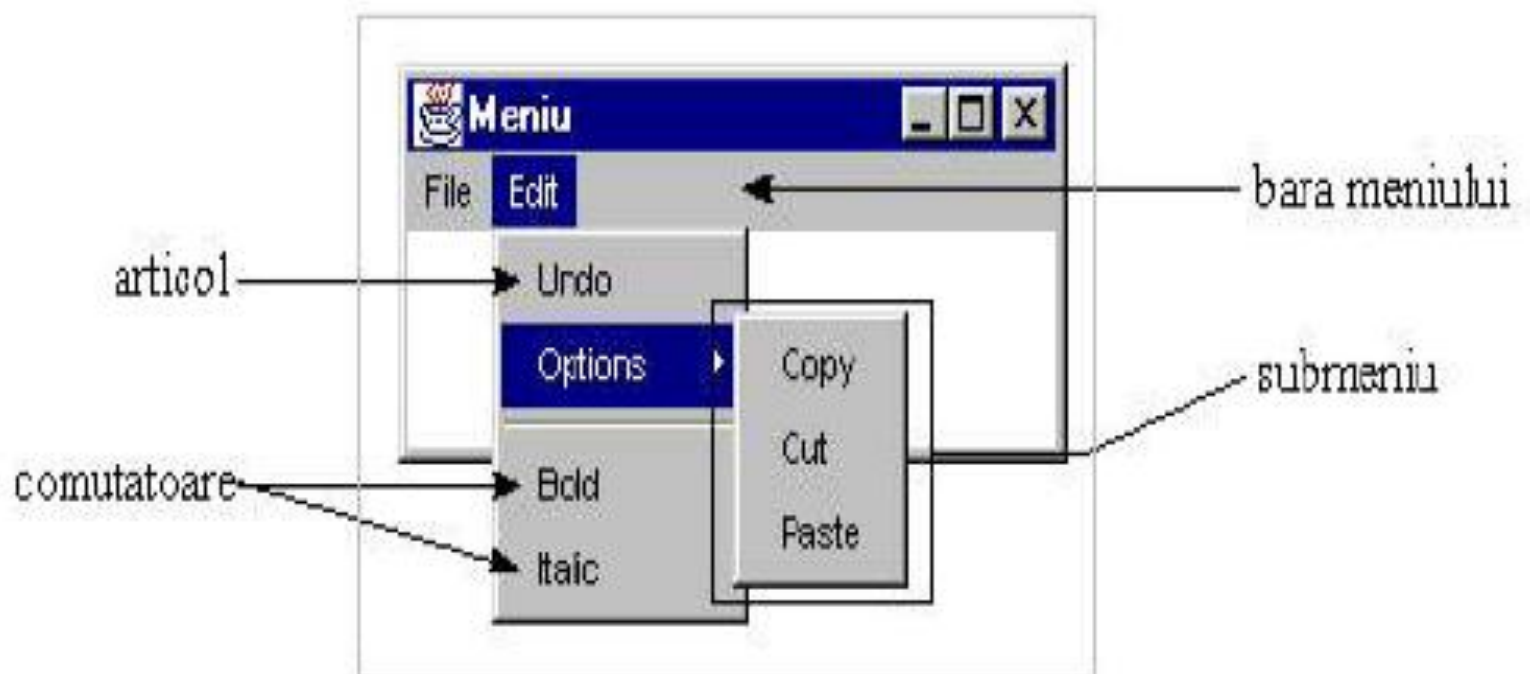
```
class jframe extends JFrame{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                JOptionPane.showMessageDialog(new
                    JFrame(),"","Event Fired!",
                    JOptionPane.PLAIN_MESSAGE);
            }
        });
        setSize(250,250);
        setVisible(true);
    }
    public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

g. O singura clasa care extinde pe "JFrame" si implementeaza "ActionListener" (clasa este si generator si ascultator la evenimente).

```
class jframe extends JFrame implements ActionListener{
    public jframe(){
        JButton jb=new JButton("Click me!");
        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(jb);
        jb.addActionListener(this);
        setSize(250,250);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e){
        JOptionPane.showMessageDialog(new
            JFrame(),"","Event Fired !",
            JOptionPane.PLAIN_MESSAGE);
    }
    public static void main(String arg[]){
        JFrame jf=new jframe();
    }
}
```

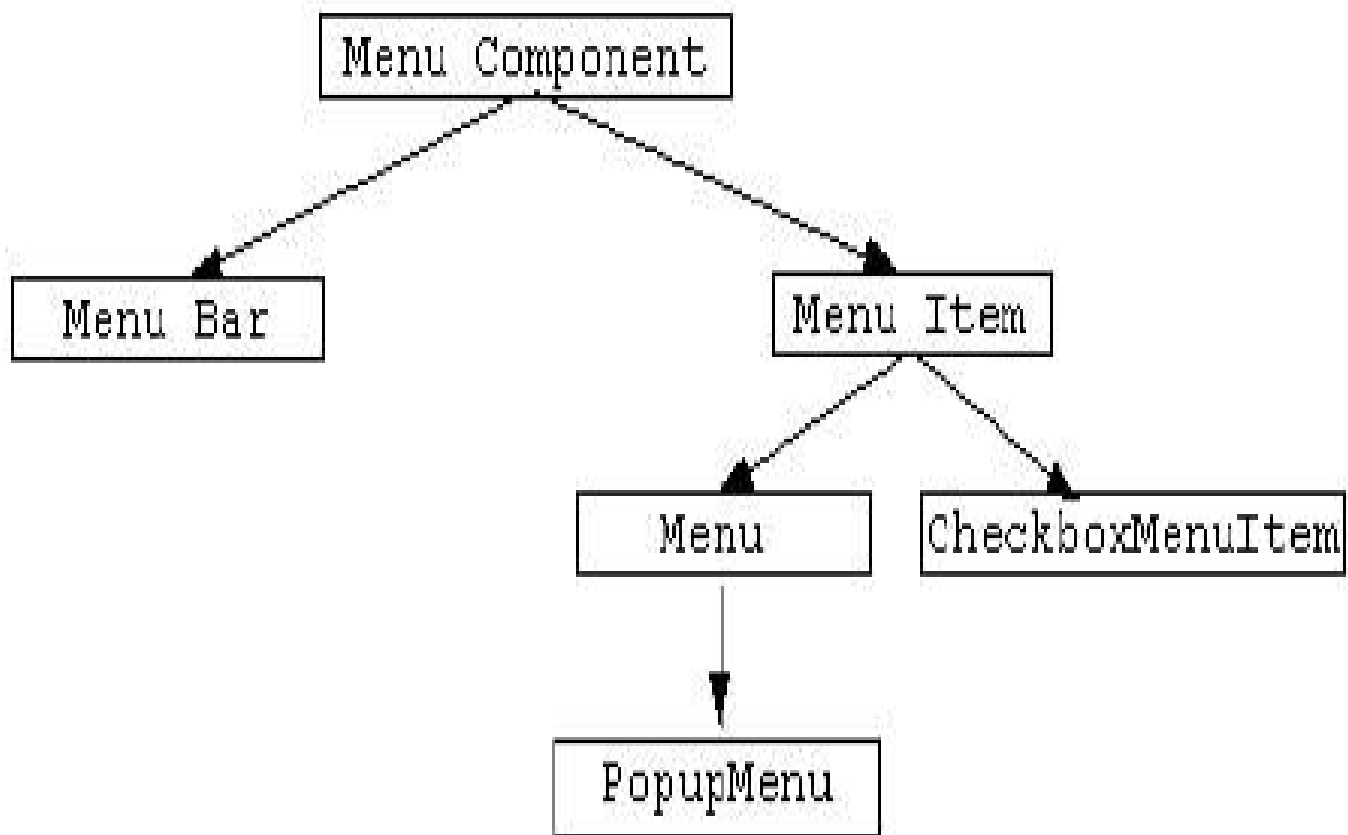
Folosirea meniurilor

- Componentele unui meniu sunt derivate din MenuComponent.
- Meniuri fixe (vizibile permanent)
- Meniuri de context (popup)



addMenuBar

Ierarhia claselor ce descriu meniuri



Exemplu

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

public class TestMenu {
    public static void main(String args[]) {
        Frame f = new Frame("Test Menu");

        MenuBar mb = new MenuBar();

        Menu fisier = new Menu("File");
        fisier.add(new MenuItem("Open"));
        fisier.add(new MenuItem("Close"));
        fisier.addSeparator();
        fisier.add(new MenuItem("Exit"));

        Menu optiuni = new Menu("Options");
        optiuni.add(new MenuItem("Copy"));
        optiuni.add(new MenuItem("Cut"));
        optiuni.add(new MenuItem("Paste"));

        Menu editare = new Menu("Edit");
        editare.add(new MenuItem("Undo"));
        editare.add(optiuni);

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

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

        f.setMenuBar(mb);
        f.setSize(200, 100);
        f.show();
    }
}
```

Tratarea evenimentelor generate de meniuri

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

public class TestMenuEvent extends Frame
    implements ActionListener, ItemListener {

    public TestMenuEvent(String titlu) {
        super(titlu);

        MenuBar mb = new MenuBar();
        Menu test = new Menu("Test");
        CheckboxMenuItem check = new CheckboxMenuItem("Check me")
            ;
        test.add(check);
        test.addSeparator();
        test.add(new MenuItem("Exit"));

        mb.add(test);
        setMenuBar(mb);

        Button btnExit = new Button("Exit");
        add(btnExit, BorderLayout.SOUTH);
        setSize(300, 200);
        show();
    }
}
```

Tratarea evenimentelor generate de meniuri

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

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

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

public static void main(String args[]) {
    TestMenuEvent f = new TestMenuEvent("Tratare evenimente
        meniuri");
    f.show();
}
}
```

Meniuri de context (popup)

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

...

```
popup.show(Component origine, int x, int y)  
fereastră.add(popup1);
```

...

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

isPopupTrigger() :

- dacă acest eveniment de mouse este evenimentul care poate afișa un meniu popup
- trebuie testat în ambele metode mousePressed și mouseReleased pentru că depinde de platformă

Folosirea unui meniu de context (popup)

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

class Fereastră extends Frame implements ActionListener{
    // Definem meniul popup al ferestrei
    private PopupMenu popup;
    // Pozitia meniului va fi relativa la fereastră
    private Component origin;
    public Fereastră(String titlu) {
        super(titlu);
        origin = this;

        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        this.addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                if (e.isPopupTrigger())
                    popup.show(origin, e.getX(), e.getY());
            }
            public void mouseReleased(MouseEvent e) {
                if (e.isPopupTrigger())
                    popup.show(origin, e.getX(), e.getY());
            }
        });
        setSize(300, 300);
    }
}
```

Folosirea unui meniu de context (popup) (2)

```
// Cream meniul popup
popup = new PopupMenu("Options");
popup.add(new MenuItem("New"));
popup.add(new MenuItem("Edit"));
popup.addSeparator();
popup.add(new MenuItem("Exit"));
add(popup); //atasam meniul popup ferestrei
popup.addActionListener(this);
}

public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.equals("Exit"))
        System.exit(0);
}

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

Acceleratori



Clasa MenuShortcut

Ctrl + Tasta sau Ctrl + Shift + Tasta

```
// Ctrl+O
```

```
new MenuItem("Open",new MenuShortcut(  
    KeyEvent.VK_O));
```

```
// Ctrl+P
```

```
new MenuItem("Print",new MenuShortcut('p'));
```

```
// Ctrl+Shift+P
```

```
new MenuItem("Preview",new MenuShortcut('p'),  
    true);
```