



Programare orientată pe obiecte

1. Interfețe utilizator grafice (GUIs)

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

1



GUI

- O interfață utilizator grafică - Graphical User Interface (GUI – se pronunță "gu-ii") prezintă un mecanism prietenos pentru interacțiunea utilizatorului cu un program
 - GUI dă programului un aspect ("look") și un mod în care este "simțit" ("feel") caracteristic
 - Permite utilizatorilor să se simtă mai familiarizați cu programul chiar înainte de a-l fi utilizat
 - Reduce timpul de învățare a modului de utilizare

04.05.2006

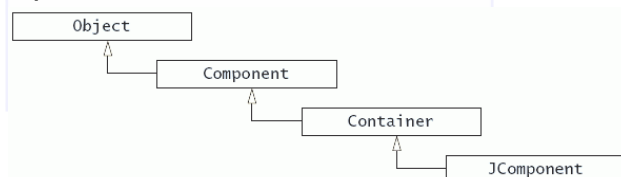
OOP10 - M. Joldoș - T.U. Cluj

2



Containere și componente (din nou)

- Clasa **Component** declară atributele și comportamentele comune tuturor subclaselor sale
 - Metode importante: `paint()`, `repaint()`
- Clasa **Container** gestionează o colecție de componente înrudite
 - În aplicații care folosesc **JFrame** și în applet-uri atașăm componente panoului de conținut (content pane) – care este un container
 - Metode importante: `add()`, `setLayout()`



04.05.2006

3



Clasa Container

- Orice clasă care descinde în clasa **Container** este considerată o clasă container
 - Clasa **Container** se află în pachetul `java.awt`, nu în biblioteca Swing
- Oricărui obiect care aparține unei clase derivate din clasa **Container** (sau din descendenții săi) i se pot adăuga componente
- Clasele **JFrame** și **JPanel** sunt descendente din clasa **Container**
 - De aceea ele și orice alți descendenți ai lor pot servi pe post de container

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

4



Clasa JComponent

- Declară atributele și comportamentele comune ale subclaselor sale, cum sunt:
 - *un look-and-feel* care se poate conecta (*pluggable*) – folosit pentru a adapta aspectul componentelor
 - taste de acces rapid pentru accesul direct la componentele GUI folosind tastatura
 - capabilități comune pentru tratarea evenimentelor pentru cazurile în care mai multe componente GUI inițiază aceleași acțiuni într-un program
 - scurte descrieri ale scopului unei componente GUI (numite *tool tips* – *sugestii referitoare la unealtă*)

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

5



Clasa JComponent

- Oferă suport pentru localizarea interfeței utilizator – adaptarea interfeței pentru a afișa textele în limbi diferite și pentru a folosi convențiile culturale locale.
- Orice descendent al clasei **JComponent** se numește *clasă componentă*
 - Oricare obiect **JComponent** sau *component* poate fi adăugat la orice obiect de clasă container
 - Deoarece este derivată din clasa **Container**, o **JComponent** poate fi adăugată și la alt(ă) **JComponent**

04.05.2006

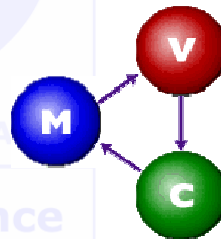
OOP10 - M. Joldoș - T.U. Cluj

6



Swing și arhitectura MVC (Model-Vizualizare-Controller)

- Arhitectura Swing își are rădăcinile în arhitectura *model-view-controller* (*MVC*) care a fost introdus inițial în limbajul SmallTalk.
- Arhitectura MVC cere ca o aplicație vizuală să fie divizată în trei părți separate:
 - Un *model* care reprezintă intern datele aplicației
 - O *vizualizare* (*view*) – reprezentarea vizuală a datelor respective.
 - Un *controlor* (*controller*) care preia intrarea de la utilizator și o transpune în schimbări în model.



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

7



Modelul

- Majoritatea programelor trebuie să facă ceva util, nu să fie "o altă față frumoasă"
 - dar există câteva excepții
 - au existat programe utile cu mult înainte apariției GUI
- **Modelul** este partea care face treaba – adică *modelează* problema care în curs de soluționare prin program
- Modelul ar trebui să fie independent atât de Controlor cât și de Vizualizare
 - *dar poate să le furnizeze amândurora servicii (metode)*
- Independența furnizează flexibilitate și robustețe

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

8



Controlorul

- Controlorul decide ce urmează să facă modelul
- Adesea, utilizatorul are controlul prin intermediul unei GUI
 - în acest caz, GUI și Controlorul sunt adesea același
- Controlorul și Modelul pot fi separate aproape întotdeauna (ce trebuie făcut în raport cu în ce fel trebuie făcut)
- Proiectul Controlorului depinde de model
- **Model nu ar trebui să depindă de Controlor**

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

9



Vizualizarea

- Tipic, utilizatorul trebuie să poată vedea, sau *vizualiza*, ce face programul
- Vizualizarea arată ce face Modelul
 - Vizualizarea este un observator *pasiv*, ea nu ar trebui să afecteze modelul
- Modelul trebuie să fie independent de vizualizare (dar îi poate furniza metode de acces)
- Vizualizarea *nu* trebuie să afișeze ce *crede* Controlorul că se întâmplă

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

10



Combinarea Controlorului și a Vizualizării

- Uneori Controlorul și Vizualizarea sunt combinate, mai ales în programe de mici dimensiuni
- Combinarea Controlorului și a Vizualizării este potrivită dacă cele două sunt foarte interdependente
- Modelul trebuie să rămână independent
- **NU amestecați niciodată codul din Model cu codul GUI!**

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

11



Separarea preocupărilor

- Ca întotdeauna, dorim independența codului
- Modelul nu trebuie contaminat cu cod din control sau din vizualizare
- Vizualizarea trebuie să reprezinte Modelul așa cum este în realitate, nu vreo stare pe care și-o amintește
- Controlorul trebuie să *converseze* cu Modelul și Vizualizarea, nu să le *manipuleze*
 - Controlorul poate seta variabile pe care Modelul și Vizualizarea le pot citi

04.05.2006

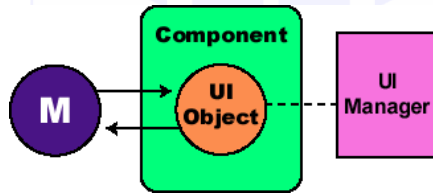
OOP10 - M. Joldoș - T.U. Cluj

12



Arhitectura cu model separabil a Swing

- În Swing părțile de *vizualizare* și *controlor* ale unei componente au necesitat o cuplare strânsă
 - Aceste două entități au fost cuprinse într-un singur obiect UI (user-interface)
 - Acest nou design quasi-MVC este numit uneori *arhitectură cu model separabil*.



04.05.2006

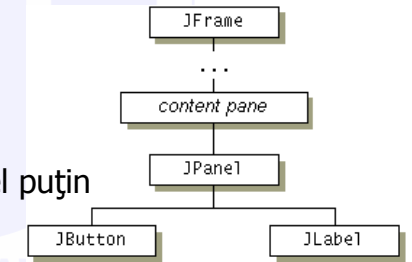
OOP10 - M. Joldoș - T.U. Cluj

13



Ierarhii de conținere

- Containere de nivel înalt
 - Containere intermediare
 - Componente atomice
- Containere de nivel înalt:
 - La rădăcina fiecărei ierarhii de conținere
 - Toate programele Swing au cel puțin unul
 - Panouri de conținut
 - Tipuri de conținere de nivel înalt
 - Cadre (frames)
 - Dialoguri
 - Applet-uri



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

14



Dialoguri

- Mai limitate decât cadrele
- Modalitate
 - Dialogurile modale opresc temporar execuția programului – utilizatorul nu poate continua până când nu s-a închis dialogul
- Tipuri de dialoguri
 - JOptionPane
 - ProgressMonitor
 - JColorChooser
 - JDialog



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

15



Afișarea dialogurilor

- JOptionPane.showXYZDialog(...)
 - Dialoguri de opțiuni și de mesaje
 -  `JOptionPane.showMessageDialog(frame, "Error!", "An error message", JOptionPane.ERROR_MESSAGE);`
 -  `JOptionPane.showOptionDialog(frame, "Save?", "A save dialog", JOptionPane.YES_NO_CANCEL_OPTION);`
 - Intrare, confirmare
- Individualizare
 - showOptionDialog – destul de individualizabil (customizable)
 - JDialog - total individualizabil

04.05.2006

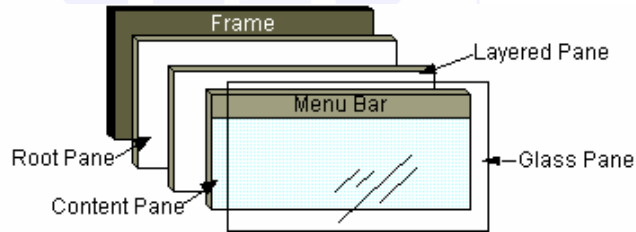
OOP10 - M. Joldoș - T.U. Cluj

16



Containere intermediare – panouri (panels / 'panes')

- Panouri rădăcină (root panes)
 - panou de conținut (content pane)
 - panouri stratificate (layered panes)
 - panouri de sticlă (glass panes)



04.05.2006

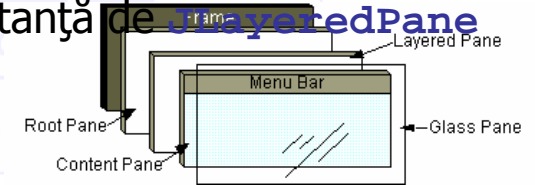
OOP10 - M. Joldoș - T.U. Cluj

17



Panou rădăcină

- Atașat 'invizibil' la containerul de nivel înalt
- Creat de Swing la realizarea cadrului
- Gestionează totul între containerul de nivel înalt și componente
- Plasează bara de meniu și panoul de conținut într-o instanță de `JLayeredPane`



04.05.2006

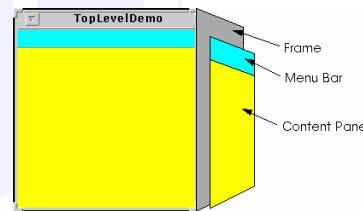
OOP10 - M. Joldoș - T.U. Cluj

18

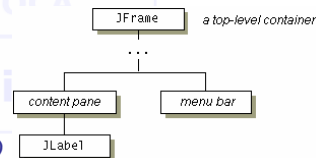


Panouri de conținut

- Folosesc de obicei un `JPanel`
- Conține totul cu excepția barei de meniu pentru majoritatea aplicațiilor Swing
- Poate fi creat explicit sau implicit
 - cod simplificat



```
//Create a panel and add components to it.
JPanel contentPane = new JPanel();
contentPane.add(someComponent);
contentPane.add(anotherComponent);
//Make it the content pane.
contentPane.setOpaque(true);
topLevelContainer.setContentPane(contentPane);
;
```



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

19



Panouri de sticlă

- Nestructurate în componente
 - interceptarea evenimentelor
 - desenare (painting)
- Folosite rar
- Fie sunt create explicit fie se folosește versiunea rădăcină



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

20



Obiecte dintr-un GUI tipic

- Aproape fiecare GUI construit folosind clasele container din Swing vor fi compuse din până la trei feluri de obiecte:
 1. *Containerul* însuși, probabil un obiect panou (*panel*) sau de tip fereastră (*window-like*)
 2. *Componentele* adăugate containerului, cum sunt etichetele (label), butoanele și panourile
 3. Un gestionar de aranjare (*layout manager*) pentru a poziționa componentele în interiorul containerului

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

21



Sugestie: programați aspectul (look) și acțiunile GUI separat

- Sarcina proiectării unui GUI poate fi împărțită în două sub-sarcini principale:
 1. Proiectarea și programarea aspectului GUI pe ecran
 2. Proiectarea și programarea acțiunilor de efectuat ca răspuns la acțiunile utilizatorului
- În particular, este util să implementăm metoda `actionPerformed()` ca *talon* (nu face nimic) până când GUI arată așa cum trebuie

```
public void actionPerformed(ActionEvent e){}
```
- Acest mod este în miezul tehnicii folosite de șablonul *Model-View-Controller*

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

22



Folosirea moștenirii pentru a individualiza (customize) cadrele

- Folosim moștenirea pentru cadrele (frames) complicate pentru a face programele mai ușor de înțeles
- Proiectăm o subclasă a lui `JFrame`
- Stocăm componentele sub forma câmpurilor instanță
- Inițializăm componentele în constructorul subclasei proiectate
- Dacă partea de cod pentru inițializare devine complexă, atunci adăugăm câteva metode ajutătoare

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

23



Gestiunea aranjării

- Până acum am folosit un control limitat asupra aranjării (layout) componentelor
 - Când am folosit un panou, acesta a aranjat componentele de la stânga la dreapta
- Componentele din interfața utilizator sunt aranjate prin plasarea lor în containere
- Fiecare container are un *gestionar de aranjare (layout manager)* care dirijează aranjarea componentelor sale
- Câteva gestionare de aranjare utile:
 - border layout
 - flow layout
 - grid layout
 - box layout

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

24



Gestiunea aranjării

- Implicit, **JPanel** amplasează componentele de la stânga la dreapta și începe un rând nou atunci când este necesar
- Aranjarea panourilor (panel) este efectuată de gestionarul de aranjări **FlowLayout**
- Se pot seta alte gestionare de aranjare
`panel.setLayout(new BorderLayout());`

04.05.2006

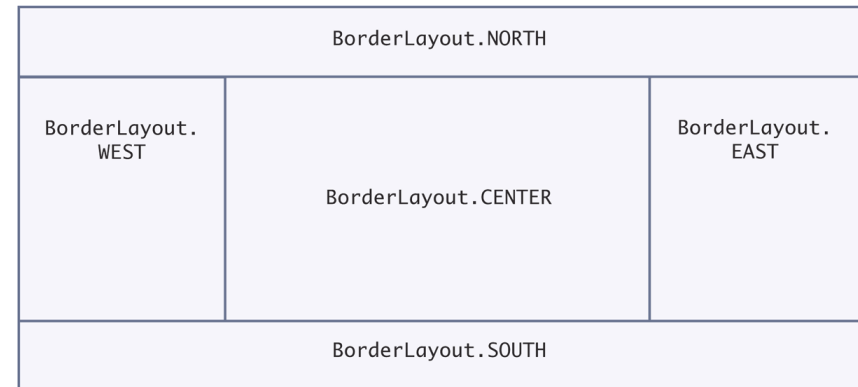
OOP10 - M. Joldoș - T.U. Cluj

25



Border Layout

- Aranjarea după margini (border layout) grupează în cinci zone: centru, nord, vest, sud și est
 - Componentele se extind ca să umple spațiul în această aranjare



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

26



Border Layout

- Este gestionarul de aranjare implicit pentru cadre – "frame" (tehnic, pentru panoul de conținut al cadrului)
- La adăugarea unei componente se specifică poziția astfel:
`panel.add(component, BorderLayout.NORTH);`
- Extinde fiecare componentă pentru a umple toată zona alocată
- Dacă nu doriți aceasta, atunci puneți fiecare componentă într-un panou

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

27



Gestionarul de aranjare FlowLayout

- Gestionarul de aranjare **FlowLayout** aranjează componentele în ordine de la stânga la dreapta și de sus în jos în container
- Constructori:
`public FlowLayout();`
`public FlowLayout(int align);`
`public FlowLayout(int align, int horizontalGap, int verticalGap);`
- Alinierea poate fi **LEFT**, **RIGHT**, sau **CENTER**
- Este implicit pentru **JPanel**

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

28



Gestionarul de aranjare GridLayout

- Aranjează componentele într-o grilă cu număr fix de rânduri și coloane
- Redimensionează fiecare componentă astfel încât ele să aibă toate aceeași mărime
- Extinde fiecare componentă pentru a umple toată zona alocată lui
- Adăugarea de componente, rând cu rând, de la stânga la dreapta:

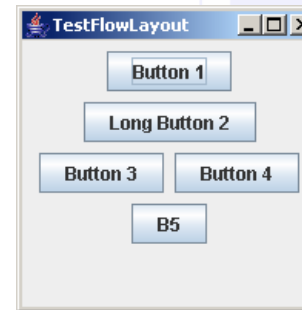
```

JPanel numberPanel = new JPanel();
numberPanel.setLayout(new GridLayout(4, 3));
numberPanel.add(button1);
numberPanel.add(button2);
numberPanel.add(button3);
numberPanel.add(button4);

```

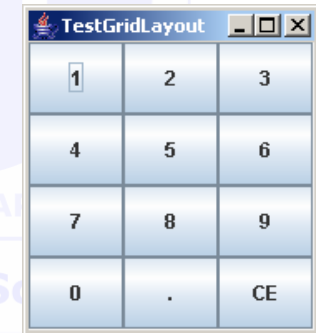


Exemple: FlowLayout și GridLayout



BlueJ: TestFlowLayout

BlueJ: TestGridLayout



Gestionarul de aranjare GridBagLayout

- Aranjare tabelară a componentelor
 - Coloanele pot avea mărimi diferite
 - Componentele se pot întinde pe mai multe coloane
- Destul de complicat de folosit
- Din fericire se pot crea aranjamente care să arate acceptabil prin imbricarea panourilor
 - Dăm fiecărui panou un gestionar de aranjare corespunzător
 - Panourile nu au margini vizibile
 - Folosim câte panouri sunt necesare pentru a organiza componentele



Gestionarul de aranjare BorderLayout

- Gestionarul de aranjare **BoxLayout** aranjează componentele dintr-un container într-un singur rând sau o singură coloană.
- Spațierea și alinierea pe fiecare rând sau coloană poate fi controlată individual
- Containerele care folosesc **BoxLayout** pot fi imbricate unul în altul pentru a produce aranjamente complexe
- Constructor:


```
public BorderLayout(Container c, int direction);
```
- **direction** poate fi **X_AXIS** sau **Y_AXIS**
- Se pot folosi zone rigide (rigid areas) și zone "lipicioase" (glue regions) pentru a spația componentele într-un **BoxLayout**



Exemplu: Crearea unui BorderLayout

```

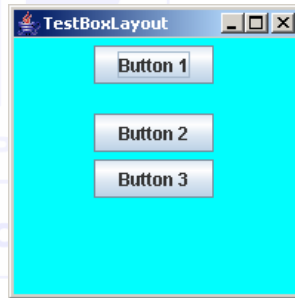
JFrame jf = new JFrame("TextBoxLayout");
jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jf.setSize(new Dimension( 200, 200));
jf.setLocation(300, 300);
// Create a new panel
JPanel p = new JPanel();
// Set the layout manager
p.setLayout(new BorderLayout(p, BorderLayout.Y_AXIS));
// Add buttons
// leave some vertical space before button
p.add( Box.createRigidArea(new Dimension(0,5)) );
addAButton( "Button 1", p );
// vertical space between buttons
p.add( Box.createRigidArea(new Dimension(0,20)) );
addAButton( "Button 2", p );
p.add( Box.createRigidArea(new Dimension(0,5)) );
addAButton( "Button 3", p );
p.setBackground(Color.cyan);
// Add the new panel to the existing container
jf.add( p );
jf.setVisible(true);

```

```

private static void addAButton(
    String text, Container container) {
    JButton button = new
        JButton(text);
    button.setAlignmentX(
        Component.CENTER_ALIGNMENT);
    container.add(button);
}

```



04.05.2006

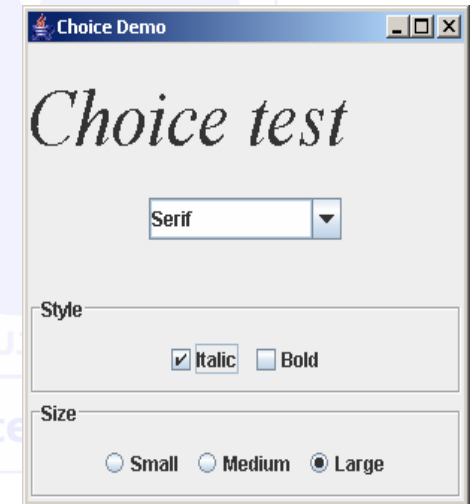
OOP10 - M. Joldoş - T.U. Cluj

33



Controale pentru alegeri

- Butoane radio
- Cutiute de marcare
- Cutii Combo



04.05.2006

OOP10 - M. Joldoş - T.U. Cluj

34



Butoane radio

- Pentru seturi de mici dimensiuni de variante mutual exclusive folosim butoane radio sau o cutie combo
- Într-un set de butoane radio, doar unul poate fi selectat la un moment dat
- Dacă este selectat un alt buton, cel selectat anterior este automat de-selectat (turned off)

04.05.2006

OOP10 - M. Joldoş - T.U. Cluj

35



Butoane radio

- Gruparea butoanelor nu pune butoanele apropiate unul de altul pe container
- Trebuie să le aranjăm noi pe ecran
- `isSelected()`: se apelează pentru a afla dacă un anumit buton este curent selectat sau nu


```
if(largeButton.isSelected()) size = LARGE_SIZE;
```
- Apelăm `setSelected(true)` pe un buton radio din grup înainte de a face vizibil cadrul care conține butoanele

04.05.2006

OOP10 - M. Joldoş - T.U. Cluj

36



Margini

- Punem o margine în jurul panoului pentru a grupa vizual conținutul său
- **EtchedBorder**: efect tridimensional de gravare
- Se poate adăuga margine la oricare componentă, dar cel mai adesea se face pentru panouri:

```
Jpanel panel = new JPanel ();
panel.setBorder(new EtchedBorder ());
```

- **TitledBorder**: o margine cu titlu:

```
Panel.setBorder(new TitledBorder(new EtchedBorder(),
"Size"));
```



Cutii de marcare (bifare)

- Au două stări: marcat (checked) și nemarcat
- Pentru o alegere din doua variante posibilă folosim o cutie de marcare (checkbox)
- Folosim un grup de cutii de marcare atunci când o alegere nu exclude o alta
- Exemplu: "bold" și "italic" la alegerea stilului unui font
- Le construim:

```
JCheckBox italicCheckBox = new JCheckBox("Italic");
```

- Nu le amplasăm în grup de butoane



Cutii Combo

- Pentru un număr mare de opțiuni, folosind o cutie combo (combo box)
 - Folosește mai puțin spațiu decât butoanele radio
- "Combo": combinație de listă cu câmp text
 - Câmpul text afișează numele selecției curente



- Dacă cutia combo este editabilă, atunci utilizatorul poate să-și tasteze propria selecție
 - Folosim metoda `setEditable()`



Cutii Combo

- Textele alegerilor le adăugăm folosind metoda `addItem()`:

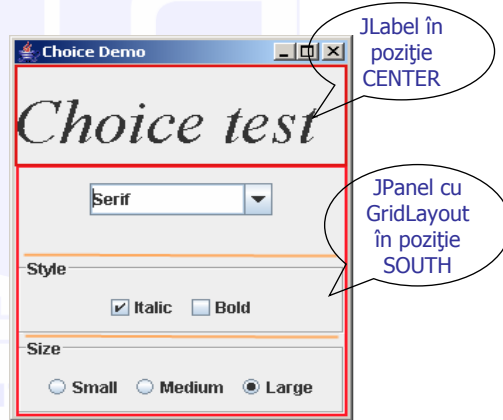
```
JComboBox facenameCombo = new JComboBox();
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
. . .
```
- Obținem alegerea utilizatorului cu `getSelectedItem()` (tipul returnat de aceasta este `Object`)

```
String selectedString =
(String) facenameCombo.getSelectedItem();
```
- Selectăm un element cu `setSelectedItem()`



Butoane radio, cutii de marcare și cutii combo împreună

- Ori de câte ori utilizatorul selectează un element, ele generează un `ActionEvent`
- Exemplu: `ChoiceFrame`
 - Toate componentele notifică același obiect ascultător
 - La clic pe oricare componentă, interogăm fiecare componentă pentru a-i afla conținutul curent
 - Apoi redesenăm textul din exemplu folosind noul font



BlueJ: ChoiceFrameViewer



Gestiunea aranjării

- Pasul 1: Facem o schiță a modului de aranjare dorit
- Pasul 2: Determinăm grupări de componente adiacente cu același mod de aranjare (layout)
- Pasul 3: Identificăm modul de aranjare pentru fiecare grup
- Pasul 4: Grupăm împreună grupurile
- Pasul 5: Scriem codul pentru generarea aranjamentului

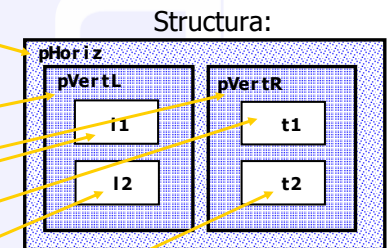


Combinarea gestionarilor de aranjare

- Câteodată e util să creăm mai multe containere unul în altul, fiecare cu propriul gestionar de aranjare
- Spre exemplu, panoul de nivelul cel mai înalt ar putea folosi o aranjare de tipul cutie orizontală, iar în el ar putea fi două sau mai multe panouri cu aranjarea tip cutie verticală
- Rezultatul este controlul complet al spațierii pe *ambele* dimensiuni

```
// Creeaza un nou panou de nivel sus
JPanel pHoriz = new JPanel();
pHoriz.setLayout(new BorderLayout(pHoriz,
BoxLayout.X_AXIS));
add( pHoriz );
// Creeaza doua panouri subordonate
JPanel pVertL = new JPanel();
JPanel pVertR = new JPanel();
pVertL.setLayout(new BorderLayout(pVertL,
BoxLayout.Y_AXIS));
pVertR.setLayout(new BorderLayout(pVertR,
BoxLayout.Y_AXIS));
// Adauga la to pHoriz cu spatiu oozizontal
// intre panouri
pHoriz.add( pVertL );
pHoriz.add( Box.createRigidArea(new
Dimension(20,0) ) );
pHoriz.add( pVertR );
// Creeaza cimpul grade Celsius
l1 = new JLabel("deg C:", JLabel.RIGHT);
pVertL.add( l1 );
t1 = new JTextField("0.0",15);
t1.addActionListener( cHnd );
pVertR.add( t1 );
// Creeaza cimpul grade Fahrenheit
l2 = new JLabel("deg F:", JLabel.RIGHT);
pVertL.add( l2 );
t2 = new JTextField("32.0",15);
t2.addActionListener( fHnd );
pVertR.add( t2 );
```

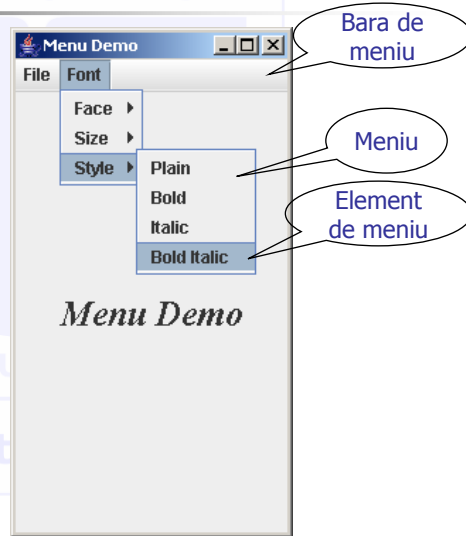
Exemple: Containere și aranjări imbricate





Meniuri

- Cadrul (frame) conține o bară de meniu
- Bara de meniu conține meniuri
- Meniul conține submeniuri și elemente de meniu
 - Meniuri Pull-Down



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

45



Elemente (items) de meniu

- Adăugăm elemente la meniu și la submeniuri cu metoda `add()`:


```
JMenuItem fileExitItem = new JMenuItem("Exit");
fileMenu.add(fileExitItem);
```
- Un element de meniu nu mai are alte submeniuri
- Elementele de meniu generează evenimente acțiune
- Adăugăm câte un ascultător fiecărui element de meniu:


```
fileExitItem.addActionListener(listener);
```
- Adăugăm ascultători de acțiuni doar elementelor de meniu nu și meniurilor și barelor de meniu

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

46



Un exemplu cu meniuri

- Construiește un mic meniu tipic
 - Interceptează evenimentele acțiune de la elementele de meniu
 - Pentru a păstra lizibilitatea programului, folosim o metodă separată pentru fiecare meniu sau fiecare set de meniuri înrudite
 - `createFaceItem()`: creează un element de meniu pentru schimbarea feței fontului (font face)
 - `createSizeItem()`
 - `createStyleItem()`
- BlueJ MenuFrameViewer

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

47



Zone de text

- Folosim `JTextArea` pentru a prezenta mai multe linii de text
- Putem preciza numărul de rânduri și coloane:


```
final int ROWS = 10;
final int COLUMNS = 30;
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
```
- Numărul de caractere pe linie pentru un obiect `JTextField` sau `JTextArea` este numărul de spații *em*
- Un spațiu *em* este spațiul necesar cuprinderii unei litere majuscule **M** (cea mai lată din alfabet)
 - O linie pentru 20 **M** va fi aproape întotdeauna capabilă să conțină mai mult de 20 caractere

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

48



Zone de text

- `setText()`: pentru a seta textul unui câmp sau unei zone de text
- `append()`: pentru a adăuga text la sfârșitul unei zone de text
- Folosim caractere `newline` pentru a separa liniile:

```
textArea.append(account.getBalance() + "\n");
```

- Dacă o folosim doar pentru afișare:

```
textArea.setEditable(false);
// program can call setText and append to change it
```

- Ca să adăugăm bare de defilare (scroll bars) la o zonă text:

```
JTextArea textArea = new JTextArea(ROWS, COLUMNS);
JScrollPane scrollPane = new JScrollPane(textArea);
```

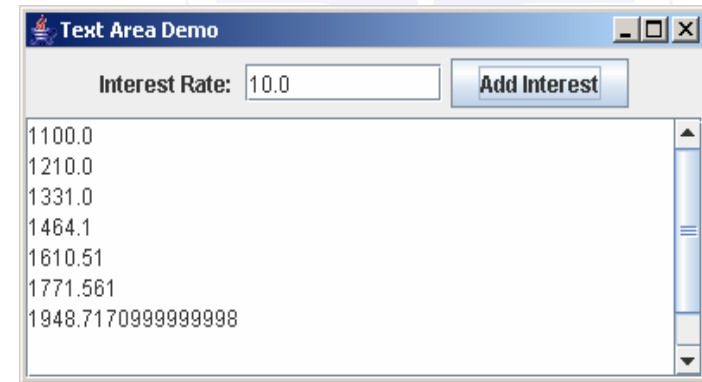
04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

49



Zone de text



BlueJ TextAreaViewer

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

50



Explorarea documentației Swing

- Pentru efecte mai sofisticate, explorăm documentația Swing
- Documentația este vastă, dar nu trebuie să ne descurajăm
- Exemplu care urmează ne arată cum să exploatăm documentația

04.05.2006

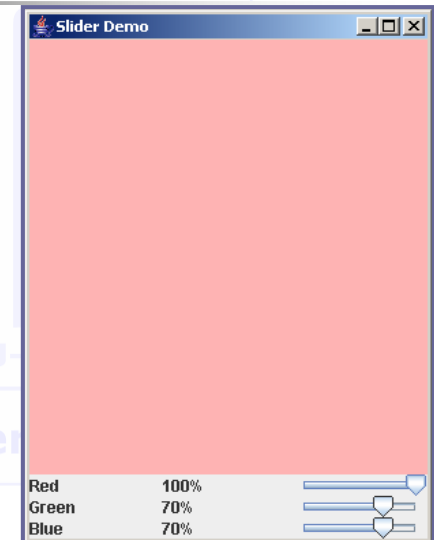
OOP10 - M. Joldoș - T.U. Cluj

51



Exemple: Un amestecător de culori

- Ar trebui să fie distractiv de amestecat propriile culori folosind un slider (glisant) pentru alegerea valorilor de roșu, verde și albastru
- Există peste 50 metode în clasa `JSlider` și peste 250 metode moștenite
- Unele descrieri arată de speriat



04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

52



Cum construiesc un JSlider?

- Căutăm în documentația API Java versiunea 5.0
- Există șase constructori pentru clasa `JSlider`
- Studiem unul sau doi
- Alegem un punct de echilibru între ceva banal și ceva bizar
- Prea limitat: `public JSlider()`
 - Creează un slider orizontal cu gama de la 0..100 și valoarea inițială 50
- Bizar: `public JSlider(BoundedRangeModel brm)`
 - Creează un slider orizontal folosind `BoundedRangeModel` specificat
- Folositor pentru noi:


```
public JSlider(int min, int max, int value)
```

 - Creează un slider orizontal folosind `min`, `max` și `value` (valoarea inițială) precizate.

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

53



Cum pot fi notificat când utilizatorul deplasează cursorul unui JSlider?

- Nu există metodă `addActionListener()`
- Dar este o metodă


```
public void addChangeListener(ChangeListener l)
```
- Clic pe legătura `ChangeListener` pentru a afla mai multe
- Are o singură metodă: `void stateChanged(ChangeEvent e)`
- În aparență, metoda este apelată ori de câte ori utilizatorul mișcă cursorul slider-ului
- Ce este un eveniment `ChangeEvent`?
 - Moștenește metoda `getSource()` din superclasa `EventObject`
 - `getSource()`: ne spune care componentă a generat acest eveniment

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

54



Cum pot fi notificat când utilizatorul deplasează cursorul unui JSlider?

- Acum știm cum să facem:
 - Adăugăm un ascultător pentru evenimentul schimbare (change event) la fiecare slider
 - La modificarea poziției cursorului este apelată metoda, `stateChanged()`
 - Aflăm noua valoare a slider-ului
 - Re-calculăm valoarea culorii
 - Redesenăm panoul cu culoarea
- Avem nevoie de valoarea curentă a slider-ului
- Ne uităm la toate metodele care încep cu `get`; găsim:

```
public int getValue()
```

întoarce valoarea sliderului

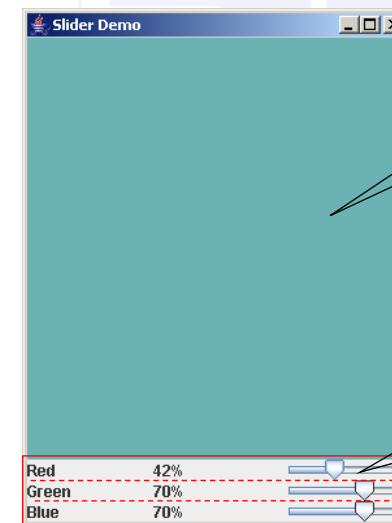
04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

55



Componentele sliderFrame



JPanel in poziție CENTER

JPanel cu GridLayout in poziție SOUTH

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

56



Icoane

- **JLabels**, **JButtons**, și **JMenuItems** pot avea reprezentări iconice (icoane)
 - O *icoană* nu este decât o mică imagine (de obicei)
 - Nu se cere să fie mică
- O icoana este un obiect de clasă **ImageIcon**
 - Se bazează pe un fișier imagine digitală cum sunt **.gif**, **.jpg**, sau **.tiff**
- Etichetele (**JLabel**), butoanele (**JButton**) și elementele de meniu (**JMenuItem**) pot afișa un sir, o icoană, amândouă sau nimic

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

57



Icoane

- Clasa **ImageIcon** se folosește pentru a converti un fișier cu imagine la o icoană Swing

```
ImageIcon dukeIcon = new
ImageIcon("duke_waving.gif");
```

 - Fișierul care conține imaginea trebuie să se afle în același director ca și clasa în care apare acest fragment de cod, sau trebuie dată calea completă sau relativă la el
 - Remarcați că numele de fișier este dat sub forma unui șir de caractere
- Atașarea unei icoane la o etichetă se face cu metoda **setIcon** astfel:

```
JLabel dukeLabel = new JLabel("Mood check");
dukeLabel.setIcon(dukeIcon);
```

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

58



Icoane

- Altfel, icoana poate fi dată ca argument constructorului lui **JLabel**:

```
JLabel dukeLabel = new JLabel(dukeIcon);
```
- Textul poate fi adăugat etichetei folosind metoda **setText**:

```
dukeLabel.setText("Mood check");
```
- Icoanele și textul pot fi adăugate la **JButton** și **JMenuItem** la fel ca pentru **JLabel**

```
JButton happyButton = new JButton("Happy");
ImageIcon happyIcon = new
ImageIcon("smiley.gif");
happyButton.setIcon(happyIcon);
```

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

59



Icoane

- Butoanele sau elementele de meniu se pot crea numai cu icoană dând obiectul de tip **ImageIcon** ca argument constructorului lui **JButton** sau **JMenuItem**

```
ImageIcon happyIcon = new ImageIcon("smiley.gif");
JButton smileButton = new JButton(happyIcon);
JMenuItem happyChoice = new JMenuItem(happyIcon);
```

 - Butoanele sau elementele de meniu create fără text trebuie să folosească metoda **setActionCommand()** pentru a seta explicit comanda acțiunii deoarece nu avem șir de caractere

04.05.2006

OOP10 - M. Joldoș - T.U. Cluj

60



Rezumat

- GUI
- Containere și componente
- MVC și Swing
- Gestiunea aspectului (aranjării) management
- Butoane radio
- Cutii de marcare (bifare)
- Cutii combo
- Gruparea butoanelor
- Meniuri
- Zone de text
- Explorarea documentației: folosirea lui JSlider
- Icoane – setarea icoanelor și a textului