Programare orientată pe obiecte

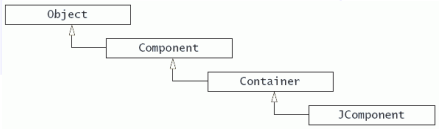
1. Interfeţe utilizator grafice (GUIs)

GUI

- O interfaţă utilizator grafică - Graphical User Interface (GUI – se pronunţă "gu-ii") prezintă un mecanism prietenos pentru interacţiunea utilizatorului cu un program
  - GUI dă programului un aspect ("look") şi un mod în care este "simţit" ("feel") caracteristic
  - Permite utilizatorilor să se simtă mai familiarizaţi cu programul chiar înainte de a-l fi utilizat
  - Reduce timpul de învăţare a modului de utilizare

Containere şi componente (din nou)

- Clasa **`Component`** declară atributele şi comportamentele comune tuturor subclaselor sale
  - Metode importante: **`paint(), repaint()`**
- Clasa **`Container`** gestionează o colecţie de componente înrudite
  - în aplicaţii care folosesc **`JFrame`** şi în applet-uri ataşăm componente panoului de conţinut (content pane) – care este un container
  - Metode importante: **`add(), setLayout()`**



Clasa `Container`

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

Clasa `JComponent`

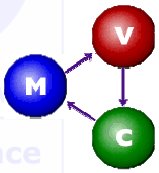
- Declară atributele şi comportamentele comune ale subclaselor sale, cum sunt:
  - *un look-and-feel* care se poate conecta (*pluggable*) – folosit pentru a adapta aspectul componentelor
  - taste de acces rapid pentru accesul direct la componentele GUI folosind tastatura
  - capabilităţi comune pentru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

Clasa `JComponent`

- Oferă suport pentru localizarea interfeţei utilizator – adaptarea interfeţei pentru a afişa textele în limbi diferite şi pentru a folosi convenţiile culturale locale.
- Orice descendent al clasei **`JComponent`** se numeşte *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 şi la alt(ă) **`JComponent`**

Swing şi arhitectura MVC (Model-Vizualizare-Controlor)

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



Modelul

- Majoritatea programelor trebuie să facă ceva util, nu să fie "o altă faţă frumoasă"
  - dar există câteva excepţii
  - au existat programe utile cu mult înaintea apariţiei GUI
- Modelul este partea care face treaba – adică *modelează* problema care în curs de soluţionare prin program
- Modelul ar trebui să fie independent atât de Controlor cât şi de Vizualizare
  - *dar poate să le furnizeze amândurora servicii (metode)*
- Independenţa furnizează flexibilitate şi robusteţe

Controlorul

- Controlorul decide ce urmează să facă modelul
- Adesea, utilizatorul are controlul prin intermediul unei GUI
  - în acest caz, GUI şi Controlorul sunt adesea acelaşi
- Controlorul şi 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

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 afişeze ce *crede* Controlorul că se întâmplă

Combinarea Controlorului şi a Vizualizării

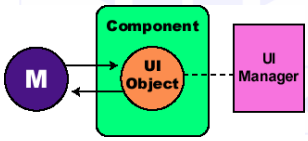
- Uneori Controlorul şi Vizualizarea sunt combinate, mai ales în programe de mici dimensiuni
- Combinarea Controlorului şi a Vizualizării este potrivită dacă cele două sunt foarte interdependente
- Modelul trebuie să rămână independent
- *NU amestecaţi niciodată* codul din Model cu codul GUI!

Separarea preocupărilor

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

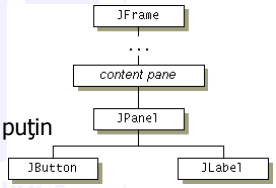
Arhitectura cu model separabil a Swing

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



Ierarhii de conţinere

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



Dialoguri

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

Afişarea dialogurilor

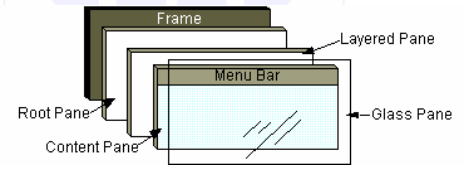
- JOptionPane.showXYZDialog(...)
  - Dialoguri de opţiuni şi de mesaje

  - `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

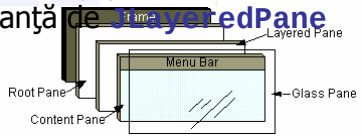
Containere intermediare – panouri (panels / 'panes')

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



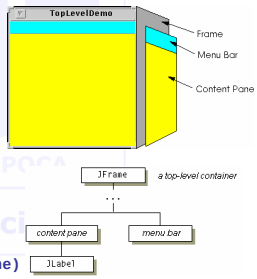
Panou rădăcină

- Ataşat 'invizibil' la containerul de nivel înalt
- Creat de Swing la realizarea cadrului
- Gestionează totul între containerul de nivel înalt şi componente
- Plasează bara de meniu şi panoul de conţinut într-o instanţă de `JLayeredPane`



Panouri de conţinut

- Folosesc de obicei un **`JPanel`**
- Conţine totul cu excepţia barei de meniu pentru majoritatea aplicaţiilor 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(anotherComponet);
//Make it the content pane.
contentPane.setOpaque(true);
topLevelContainer.setContentPane(contentPane)
;
```


Panouri d sticlă

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

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* însuşi, probabil un obiect panou (*panel*) sau de tip fereastră (*window-like*)
  2. *Componentele* adăugate containerului, cum sunt etichetele (label), butoanele şi panourile
  3. Un gestionar de aranjare (*layout manager*) pentru a poziţiona componentele în interiorul containerului

Sugestie: programaţi aspectul (look) şi acţiunile GUI separat

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

```
public void actionPerformed(ActionEvent e){}
```

- Acest mod este în miezul tehnicii folosite de şablonul *Model-View-Controller*

Folosirea moştenirii pentru a individualiza (customize) cadrele

- Folosim moştenirea pentru cadrele (frames) complicate pentru a face programele mai uşor de înţeles
- Proiectăm o subclasă a lui **`JFrame`**
- Stocăm componentele sub forma câmpurilor instanţă
- Iniţializăm componentele în constructorul subclasei proiectate
- Dacă partea de cod pentru iniţializare devine complexă, atunci adăugăm câteva metode ajutătoare

Gestiunea aranjării

- Până acum am folosit un control limitat asupra aranjării (layout) componentelor
  - Când am folosit un panou, acesta a aranjat componentele de da stânga la dreapta
- Componentele din interfaţa 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

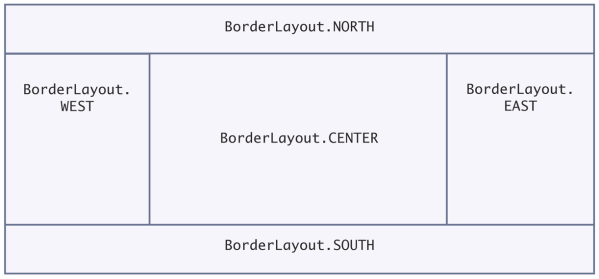
Gestiunea aranjării

- Implicit, **`JPanel`** amplasează componentele de la stânga la dreapta şi 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());
```


Border Layout

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



Border Layout

- Este gestionarul de aranjare implicit pentru cadre – "frame" (tehnic, pentru panoul de conţinut al cadrului)
- La adăugarea unei componente se specifică poziţia astfel:

```
panel.add(component, BorderLayout.NORTH);
```

- Extinde fiecare componentă pentru a umple toată zona alocată
- Dacă nu doriţi aceasta, atunci puneţi fiecare componentă într-un panou

Gestionarul de aranjare `FlowLayout`

- Gestionarul de aranjare **`FlowLayout`** aranjează componentele în ordine de la stânga la dreapta şi de sus în jos în container
- Constructori:

```
public FlowLayout();
public FlowLayout(int align);
public FlowLayout(int align, int horizontalGap,
                  int verticalGap);
```

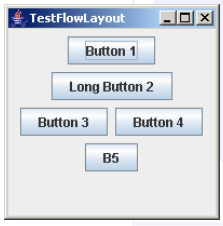
- Alinierea poate fi **`LEFT`**, **`RIGHT`**, sau **`CENTER`**
- Este implicit pentru **`JPanel`**

Gestionarul de aranjare `GridLayout`

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

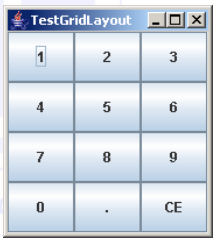
```
JPanel numberPanel = new JPanel();
numberPanel.setLayout(new GridLayout(4, 3));
numberPanel.add(button1);
numberPanel.add(button2);
numberPanel.add(button3);
numberPanel.add(button4);
. . .
```


Exemple: `FlowLayout` şi `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 `BoxLayout`

- Gestionarul de aranjare `BoxLayout` aranjează componentele dintr-un container într-un singur rând sau o singură coloană.
- Spaţierea şi 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 BoxLayout(Container c, int direction);
```

- `direction` poate fi `X_AXIS` sau `Y_AXIS`
- Se pot folosi zone rigide (rigid areas) şi zone "lipicioase" (glue regions) pentru a spaţia componentele într-un `BoxLayout`

Exemplu: Crearea unui `BoxLayout`

```
JFrame jf = new JFrame("TestBoxLayout");
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 BoxLayout(p, BoxLayout.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);
}
```

Controale pentru alegeri

- Butoane radio
- Cutiuţe de marcare
- Cutii Combo

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)

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 conţine butoanele

Margini

- Punem o margine în jurul panoului pentru a grupa vizual conţinutul 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)

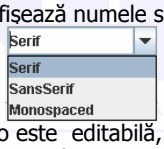
- Au două stări: marcat (checked) şi nemarcat
- Pentru o alegere din doua variante posibilă folosim o cutie de marcare (checkbox)
- Folosim un grup de cutii de marcare atunci când o alegere nu exclude o alta
- Exemplu: "bold" şi "italic" la alegerea stilului unui font
- Le construim:

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

- Nu le amplasăm în grup de butoane

Cutii Combo

- Pentru un număr mare de opţiuni, folosind o cutie combo (combo box)
  - Foloseşte mai puţin spaţiu decât butoanele radio
- "Combo": combinaţie de listă cu câmp text
  - Câmpul text afişează numele selecţiei curente



- Dacă cutia combo este editabilă, atunci utilizatorul poate să-şi tasteze propria selecţie
  - Folosim metoda **`setEditable()`**

Cutii Combo

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

```
JComboBox facenameCombo = new JComboBox();
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
. . .
```

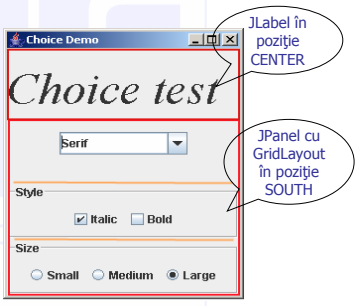
- Obţinem alegerea utilizatorului cu **`getSelectedItem()`** (tipul returnat de aceasta este **`Object`**)

```
String selectedString =
   (String) facenameCombo.getSelectedItem();
```

- Selectăm un element cu **`setSelectedItem()`**

Butoane radio, cutii de marcare şi cutii combo împreună

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



BlueJ: ChoiceFrameViewer

Gestiunea aranjării

- Pasul 1: Facem o schiţă a modului de aranjare dorit
- Pasul 2: Determinăm grupări de componente adiacente cu acelaşi 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

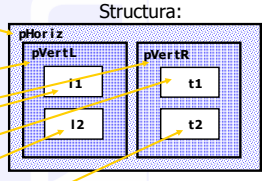
Combinarea gestionarilor de aranjare

- 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 spaţierii pe *ambele* dimensiuni

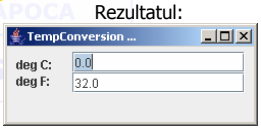
Exemple: Containere şi aranjări imbricate

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

Structura:

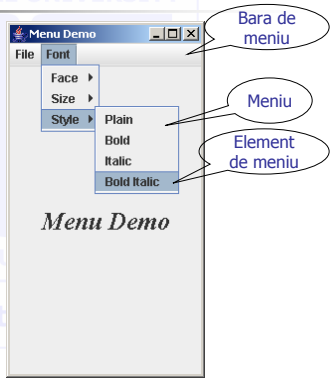


Rezultatul:



Meniuri

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



Elemente (items) de meniu

- Adăugăm elemente la meniu şi 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 acţiune
- Adăugăm câte un ascultător fiecărui element de meniu:

```
fileExitItem.addActionListener(listener);
```

- Adăugăm ascultători de acţiuni doar elementelor de meniu nu şi meniurilor şi barelor de meniu

Un exemplu cu meniuri

- Construieşte un mic meniu tipic
- Interceptează evenimentele acţiune 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 feţei fontului (font face)
  - **`createSizeItem()`**
  - **`createStyleItem()`**

BlueJ MenuFrameViewer

Zone de text

- Folosim **`JTextArea`** pentru a prezenta mai multe linii de text
- Putem preciza numărul de rânduri şi 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 spaţii *em*
- Un spaţiu *em* este spaţiul necesar cuprinderii unei litere majuscule **`M`** (cea mai lată din alfabet)
  - O linie pentru 20 **`M`** va fi aproape întotdeauna capabilă să conţină mai mult de 20 caractere

Zone de text

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

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

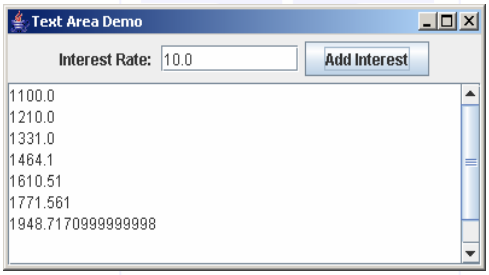
- Dacă o folosim doar pentru afişare:

```
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);
```


Zone de text



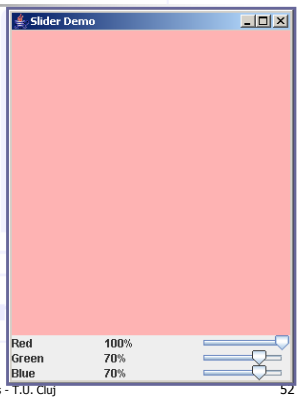
BlueJ TextAreaViewer

Explorarea documentaţiei Swing

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

Exemple: Un amestecător de culori

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



Cum construiesc un `JSlider`?

- Căutăm în documentaţia API Java versiunea 5.0
- Există şase constructori pentru clasa `JSlider`
- Studiem unul sau doi
- Alegem un punct de echilibru între ceva banal şi ceva bizar
- Prea limitat: `public JSlider()`
  - Creează un slider orizontal cu gama de la 0..100 şi valoarea iniţială 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 şi value (valoarea iniţială) precizate.

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 aparenţă, metoda este apelată ori de câte ori utilizatorul mişcă cursorul slider-ului
- Ce este un eveniment `ChangeEvent`?
  - Moşteneşte metoda `getSource()` din superclasa `EventObject`
  - `getSource()`: ne spune care componentă a generat acest eveniment

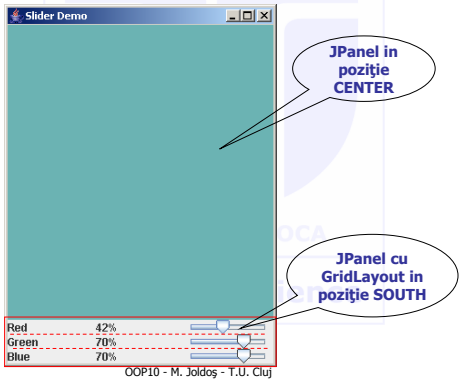
Cum pot fi notificat când utilizatorul deplasează cursorul unui `JSlider`?

- Acum ştim cum să facem:
  - Adăugăm un ascultător pentru evenimentul schimbare (change event) la fiecare slider
  - La modificarea poziţiei 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

Componentele `SliderFrame`



Icoane

- `JLabels`, `JButtons`, şi `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 fişier imagine digitală cum sunt `.gif`, `.jpg`, sau `.tiff`
- Etichetele (`JLabel`), butoanele (`JButton`) şi elementele de meniu (`JMenuItem`) pot afişa un sir, o icoană, amândouă sau nimic

Icoane

- Clasa `ImageIcon` se foloseşte pentru a converti un fişier cu imagine la o icoană Swing

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

  - Fişierul care conţine imaginea trebuie să se afle în acelaşi director ca şi clasa în care apare acest fragment de cod, sau trebuie dată calea completă sau relativă la el
  - Remarcaţi că numele de fişier este dat sub forma unui şir de caractere
- Ataşarea unei icoane la o etichetă se face cu metoda `setIcon` astfel:

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


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 şi textul pot fi adăugate la `JButton` şi `JMenuItem` la fel ca pentru `JLabel`

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


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 acţiunii deoarece nu avem şir de caractere

Rezumat

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