PROGRAMARE AVANSATĂ PE OBIECTE

Conf.univ.dr. Radu Boriga

Proiectarea interfețelor grafice în Java

- **GUI (Graphical User Interface)** – o modalitate de interacțiune vizuală între utilizator și aplicație, folosind componente grafice specifice (butoane, liste, meniuri etc.)

- Java oferă o infrastructură de clase și pachete destinate realizării interfețelor grafice:
  - AWT (Abstract Windowing Toolkit)
  - JFC (Java Foundation Classes)
  - JavaFx

Java Foundation Classes

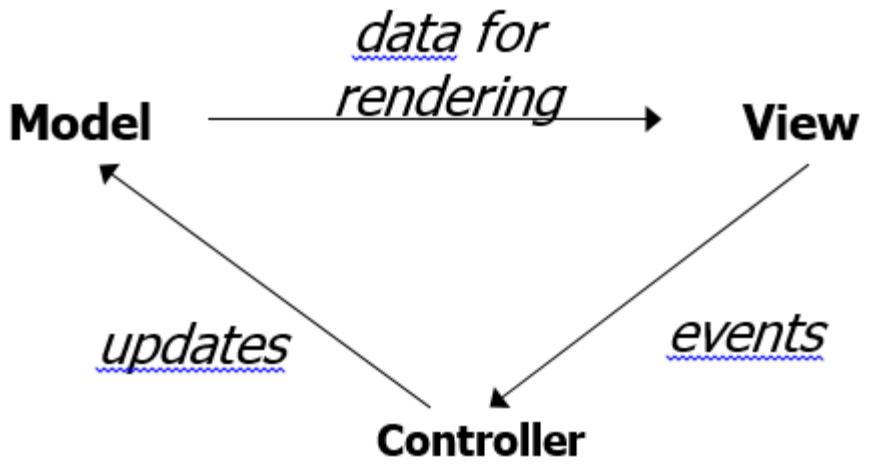
- **JFC** – o infrastructură complexa de pachete cu clase și interfețe dedicate realizării interfețelor grafice
  - Swing
  - Look and feel
  - Accessibility API
  - Java2D API
  - Internalization

Swing

- O arhitectură complexă de clase și interfețe care extinde modelul AWT.
- Conține 18 pachete, cel de bază fiind javax.swing.
- Componentele Swing nu depind de sistemul de operare, fiind implementate direct în Java.
- **Avantaje:**
  - flexibilitate, deoarece desenarea componentelor se efectuează integral din Java
  - paleta de componentele grafice este mult extinsă, iar componentele sunt mai elaborate
- **Dezavantaje**
  - viteza mai redusă comparativ cu AWT, deoarece desenarea componentelor se efectuează integral prin cod Java
  - aspectul componentelor nu se modifica automat odată cu îmbunătățirile aduse de noile sisteme de operare

Swing

- **Modelul** gestionează datele și înștiințează controller-ul în momentul în care datele sunt modificate.
- **View-ul** are rolul de a reprezenta grafic datele din model și de a facilita interacțiunea cu utilizatorul.
- **Controller-ul** definește modul în care interfața reacționează la acțiunile utilizatorului, recepționează mesajele primite de la view după apariția unui anumit eveniment și trimite mesaje modelului pentru a actualiza datele afișate de view.

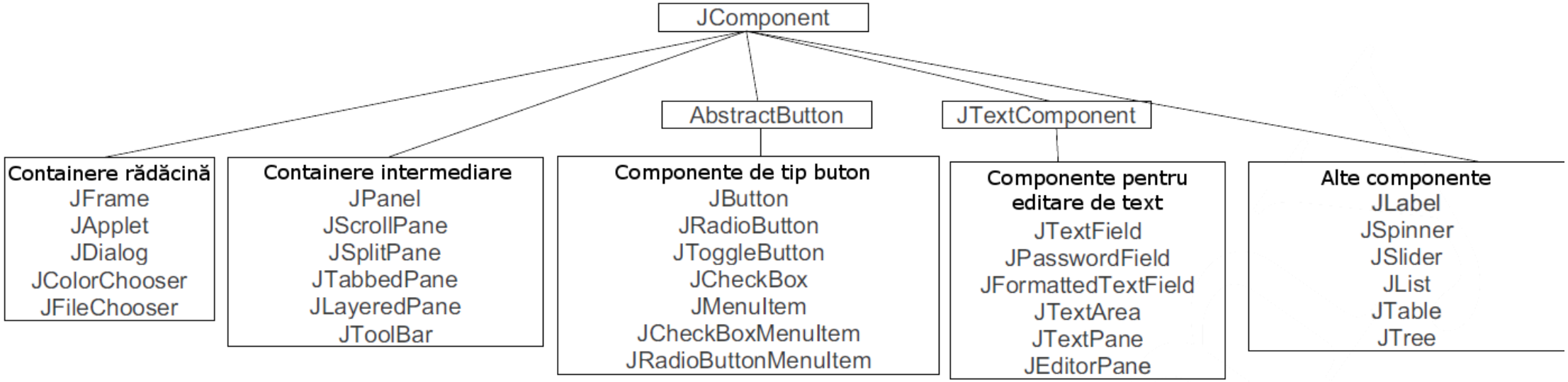


În Swing, View-ul și Controller-ul au fost unificate, obținându-se o arhitectură cu model separabil!

Etapele realizării unei GUI

- Crearea unui container rădăcină (formular principal)
- Adăugarea unor containere intermediare în containerul rădăcină (în afara celui implicit)
- Adăugarea unor componente grafice în containerele intermediare
- Poziționarea/alinierea componentelor în containerele intermediare folosind gestionari de poziționare (layout manager)
- Specificarea acțiunilor care trebuie efectuate în momentul apariției unui anumit eveniment

Componente grafice



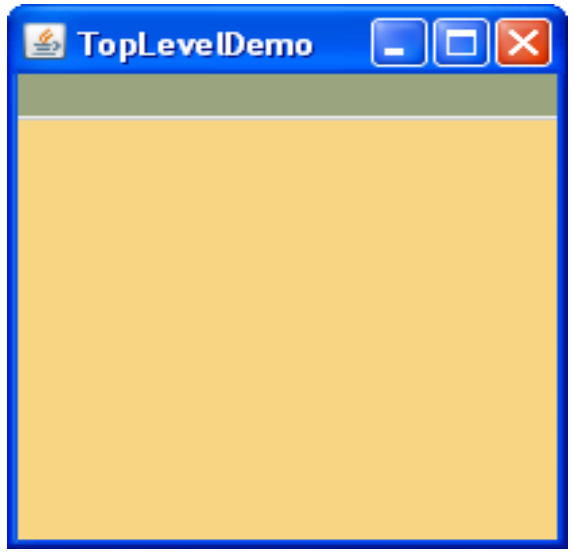
Componente de tip container

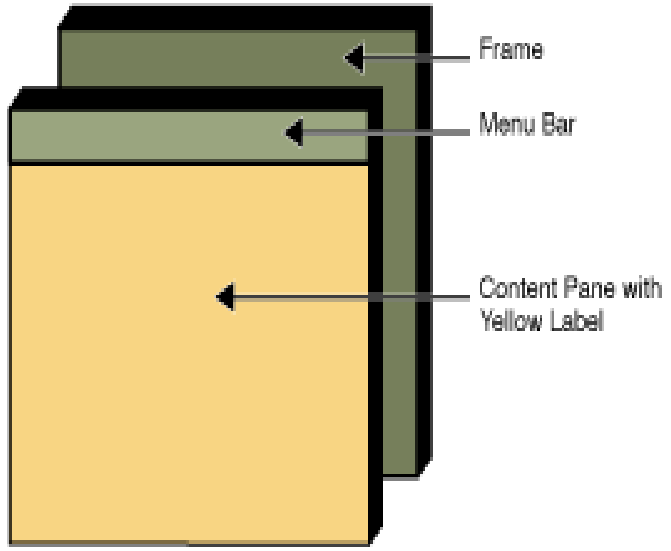
- **Containere rădăcină (root containers)** – sunt singurele care pot fi afișate direct pe ecran. Acestea sunt: **ferestrele** (JFrame), **dialogurile** (JDialog) și **applet-urile** (JApplet).

- **Containere intermediere** – au rol de container pentru alte componente grafice, dar trebuie să fie plasate într-un container rădăcină (JPanel, JScrollPane, JTabbedPane, JSplitPane, JLayeredPane, JDesktopPane, JInternalFrame)

➢ Clasele au metode care permit:
  - adăugarea sau eliminarea unei componente grafice (add/remove)
  - modificarea aspectului containerului (dimensiuni - setSize, font - setFont, culori - setBackground/setForeground)
  - stabilirea modului de aliniere al componentelor grafice incluse (setLayout)

Container-ul rădăcină JFrame

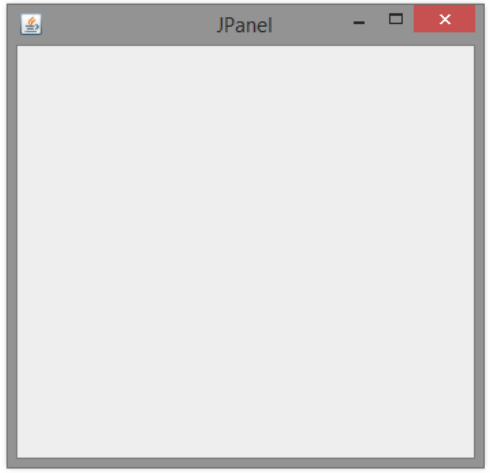




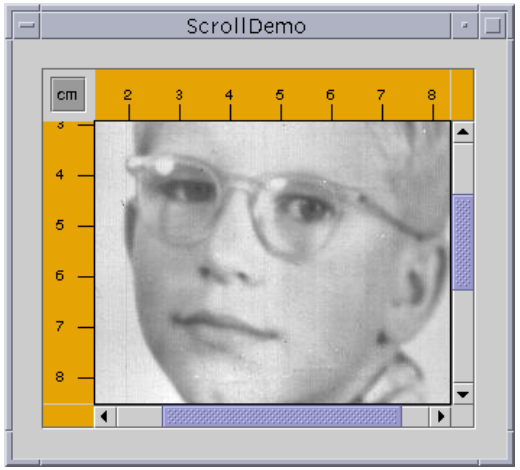
JFrame conține

- icon
- o bară de titlu
- butoane predefinite pentru operațiile de redimensionare sau închidere a ferestrei
- un container intermediar implicit de tip JPanel
- o bară de meniu (opțional)

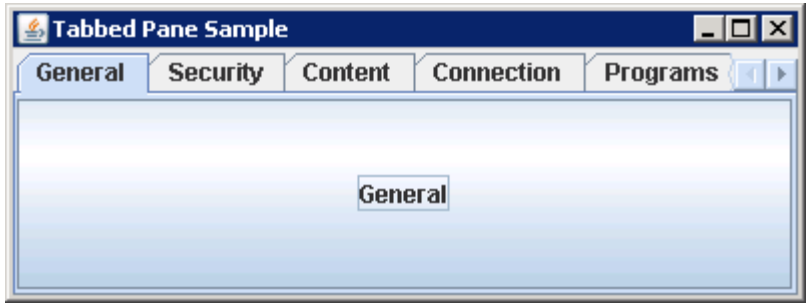
Containere intermediare



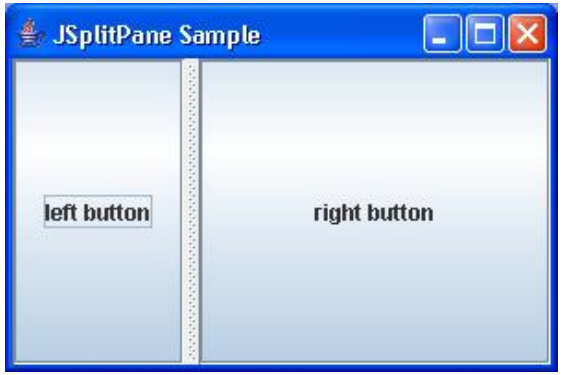
JPanel



JScrollPane

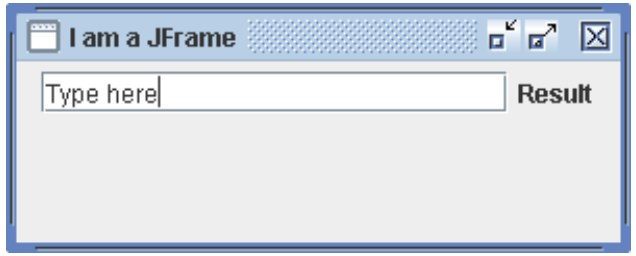


JTabbedPane

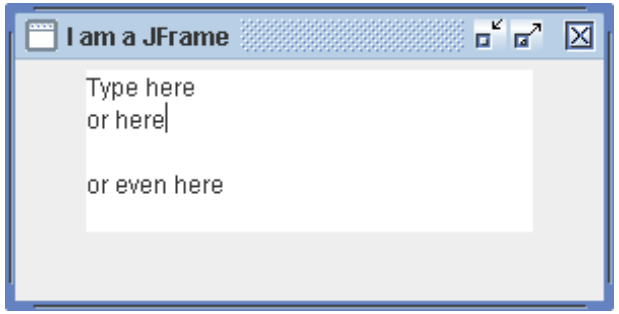


JSplitPane

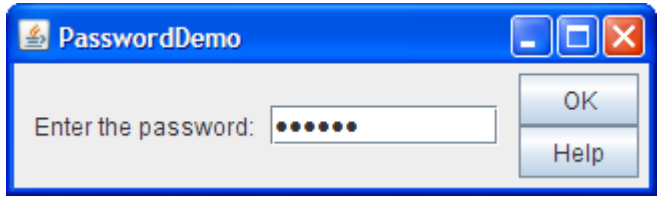
Componente grafice elementare



JTextField



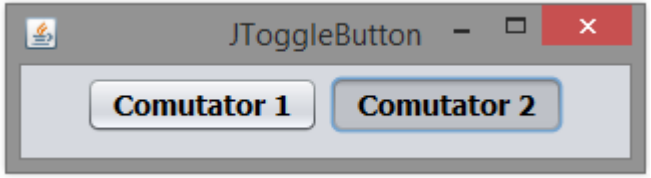
JTextArea



JPasswordField

JLabel

JButton

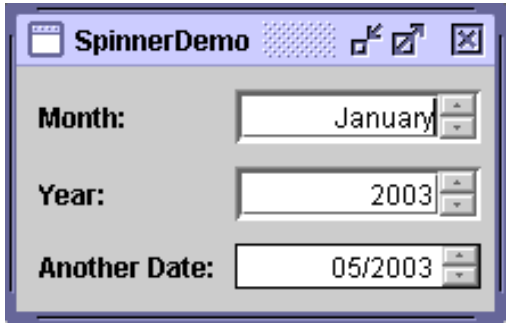


JToggleButton

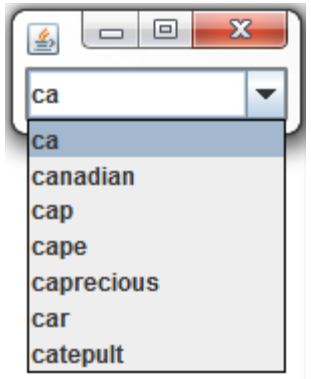
Componente grafice elementare

JRadioButton

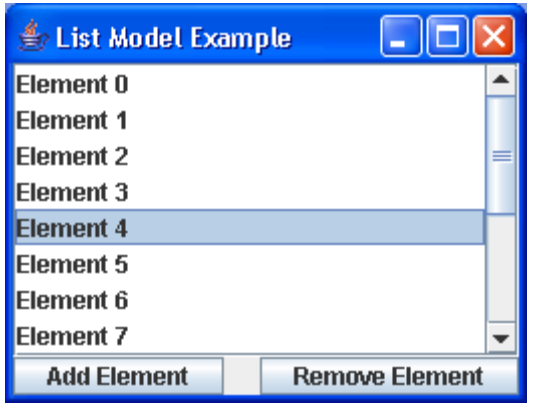
JCheckBox



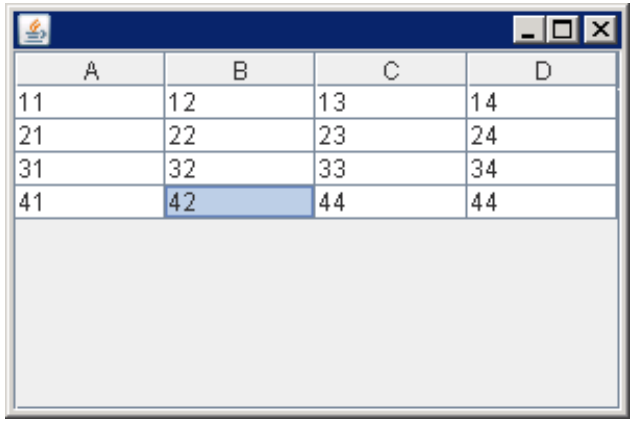
JSpinner



JComboBox



JList



JTable

Gestionari de poziționare (layout manager)

- Rolul unui gestionar de poziționare este acela de a stabili, automat, poziția și dimensiunea fiecărei componente grafice dintr-un container.

Poziționare inițială	Redimensionare
	

Tipuri de gestionari de poziționare

- În Swing sunt definiți mai mulți gestionari de poziționare:
  - FlowLayout
  - BorderLayout
  - GridLayout
  - CardLayout
  - GridBagLayout
  - SpringLayout
  - GroupLayout
- Gestionarul de poziționare al unui container se stabilește folosind metoda

void setLayout(LayoutManager manager)

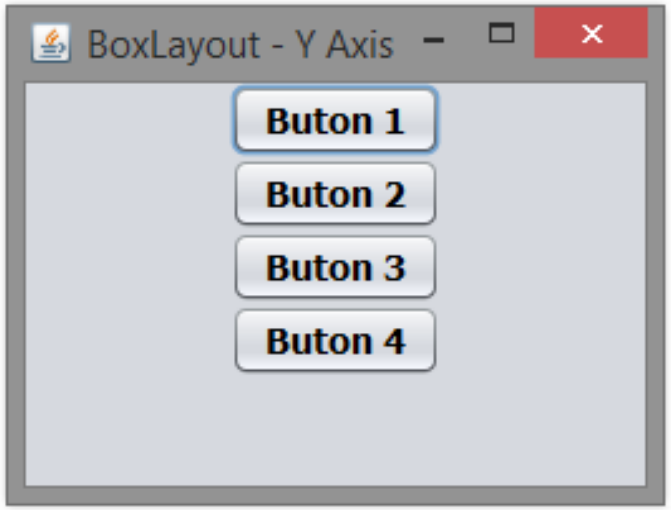
- Implicit, un container de tip JPanel are asociat un container de tip FlowLayout, în timp ce un JFrame are asociat un BorderLayout.

Gestionarul FlowLayout

- Acest gestionar așează componentele pe suprafața de afișare în flux liniar.
- Componentele sunt adăugate una după alta pe linii, în limita spațiului disponibil.
- Adăugarea componentelor se face de la stânga la dreapta pe linie, iar alinierea obiectelor în cadrul unei linii poate fi de trei feluri: la stânga, la dreapta și pe centru.

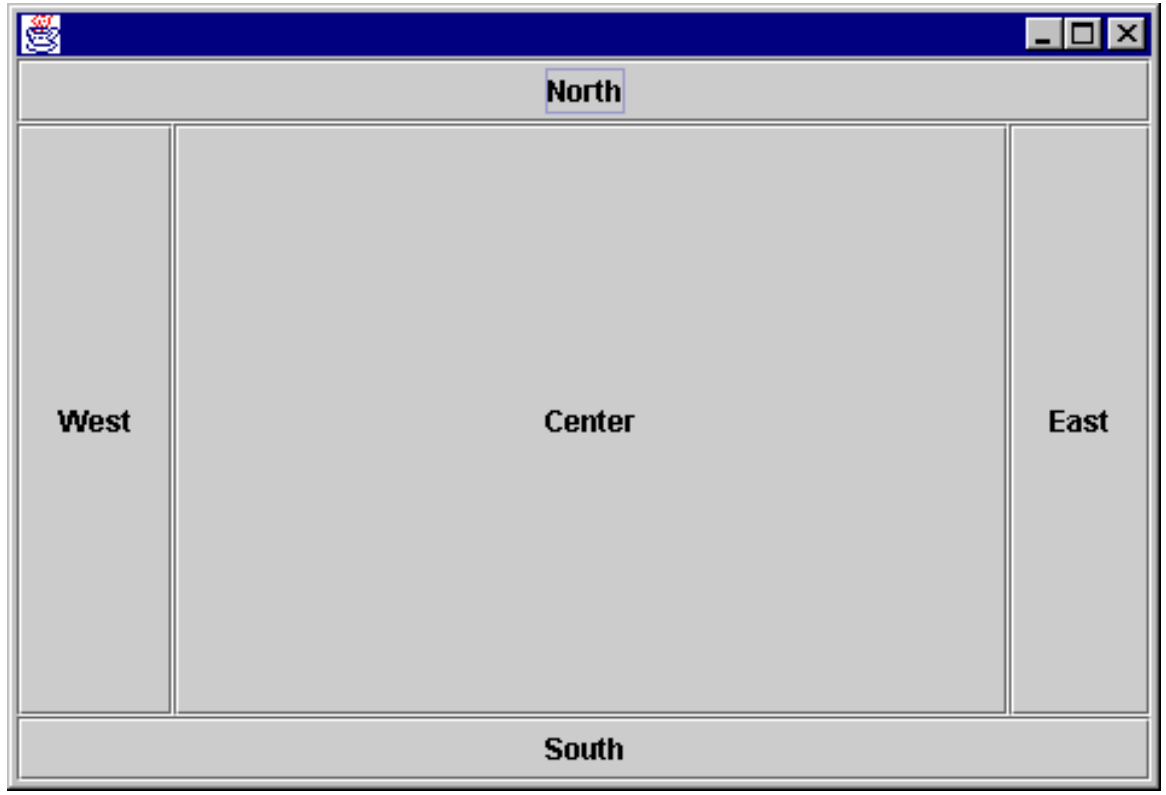
Gestionarul BoxLayout

- Gestionarul **BoxLayout** poziționează componentele în container într-un flux liniar, pe orizontală (X Axis) sau pe verticală (Y Axis).



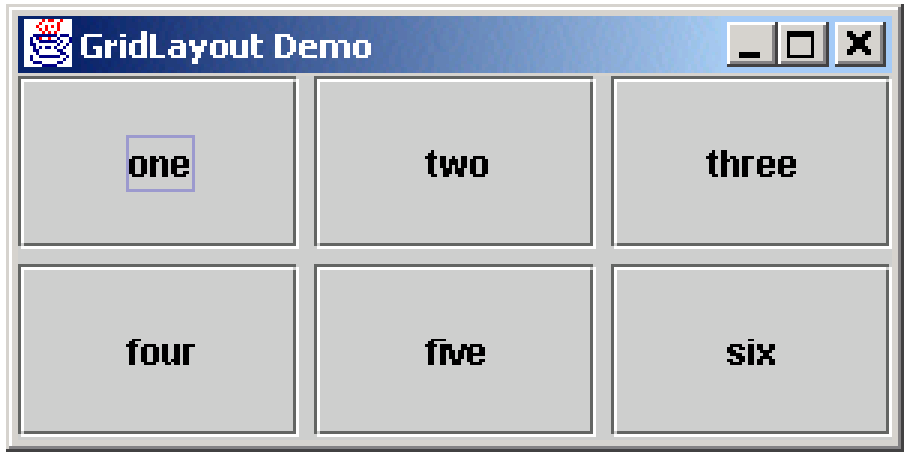
Gestionarul BorderLayout

- Gestionarul **BorderLayout** împarte suprafața de afișare în cinci regiuni, corespunzătoare celor patru puncte cardinale și centrului.



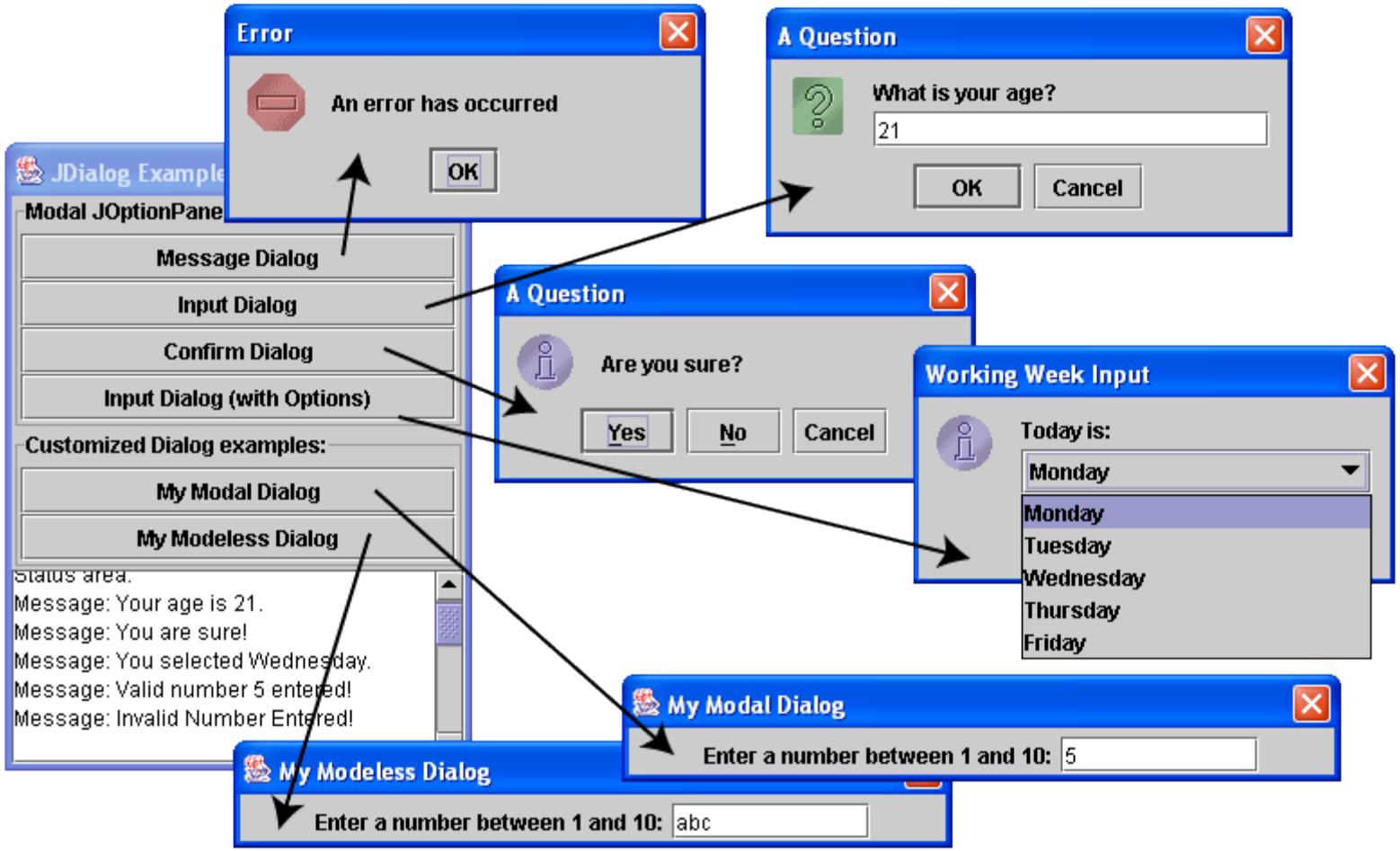
Gestionarul GridLayout

- Gestionarul **GridLayout** organizează containerul sub forma unui tabel cu rânduri și coloane.
- Se specifică numărul de linii și de coloane în constructor.
- Celulele au aceeași dimensiune.



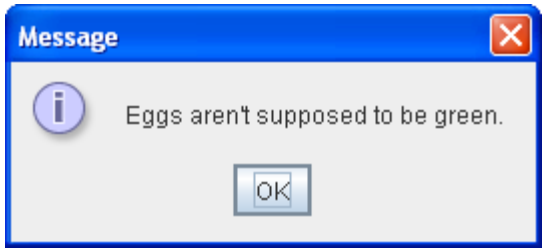
Dialoguri

› Dialogurile sunt containere rădăcină folosite pentru a informa utilizatorul despre anumite evenimente apărute în cadrul unei aplicații sau pentru a cere utilizatorului confirmarea unei anumite acțiuni.



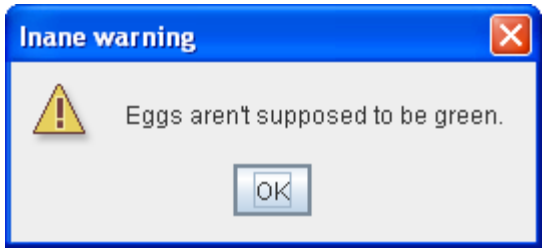
Dialoguri

- Pentru a simplifica procesul de design al unui dialog, în pachetul **javax.swing** a fost introdusă clasa **JOptionPane**.
- Crearea dialogurilor se realizează folosind următoarele metode statice ale clasei **JOptionPane**:
  - **void showMessageDialog(Component parent, Object message)**




```
JOptionPane.showMessageDialog(null,"Eggs
aren't supposed to be green.");
```

  - **void showMessageDialog(Component parent, Object message, String title, int type)**



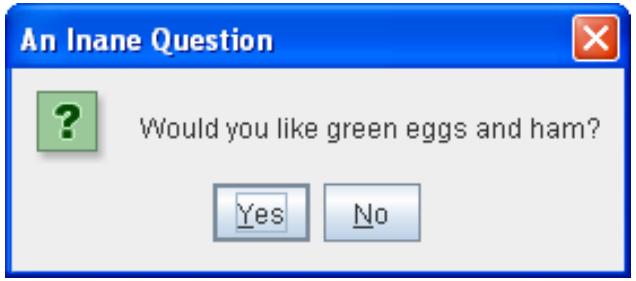

```
JOptionPane.showMessageDialog(null, "Eggs
aren't supposed to be green.", "Inane
warning", JOptionPane.WARNING_MESSAGE);
```

Dialoguri

- **int showConfirmDialog(Component parent, Object message)**


```
int r=JOptionPane.showConfirmDialog(null, "Do you like bacon?");
```

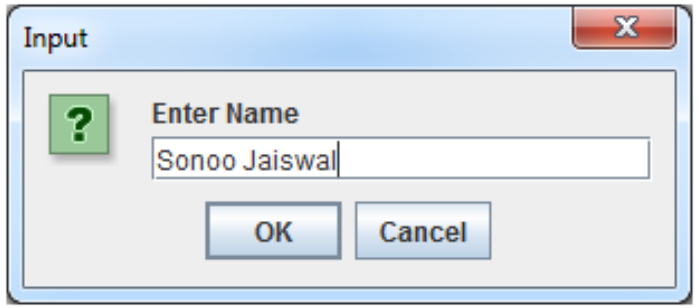
- **static int showConfirmDialog(Component parent, Object message, String title, int type)**




```
int r = JOptionPane.showConfirmDialog(null, "Would you like green
eggs and ham?", "An Inane Question", JOptionPane.YES_NO_OPTION);
```

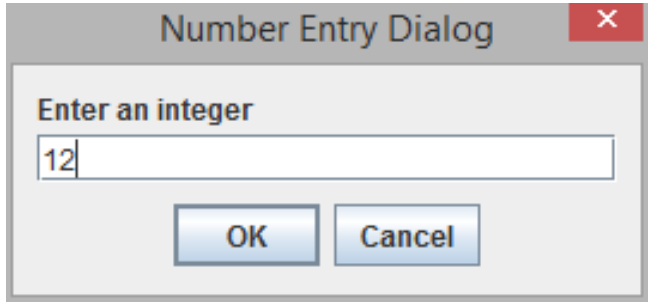
Dialoguri

- **int showInputDialog(Component parent, Object message)**




```
String name = JOptionPane.showInputDialog(null,"Enter Name");
```

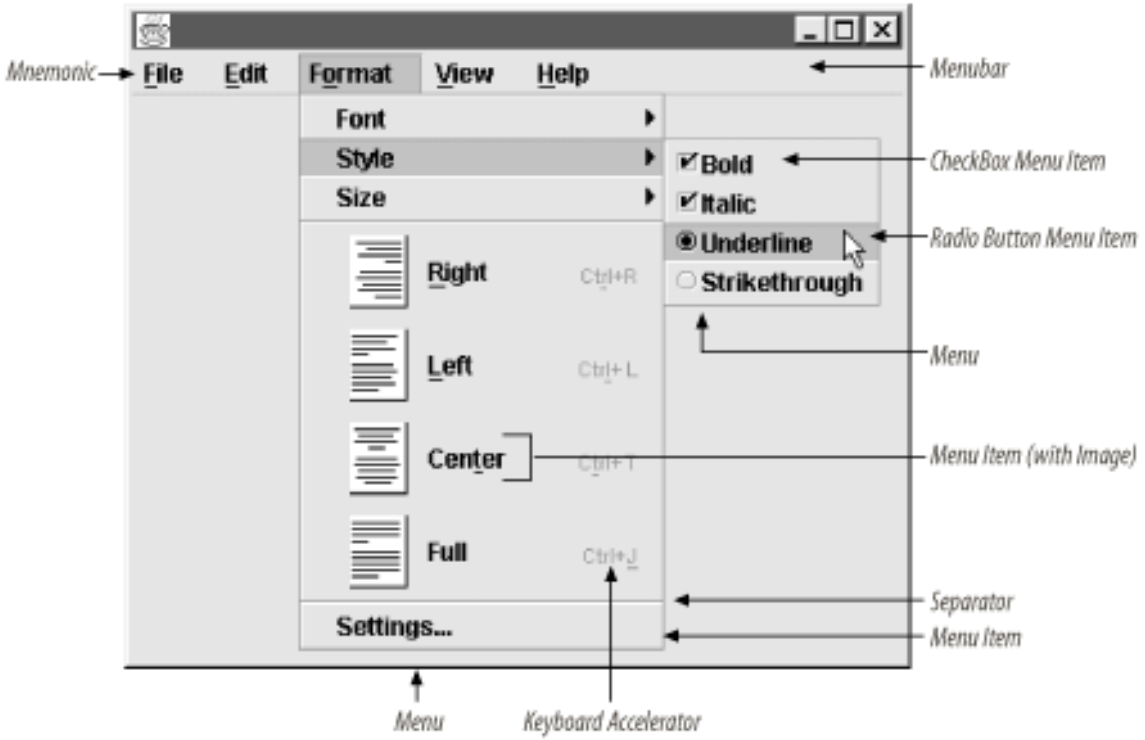
- **static int showInputDialog(Component parent, Object message, String title, int type)**



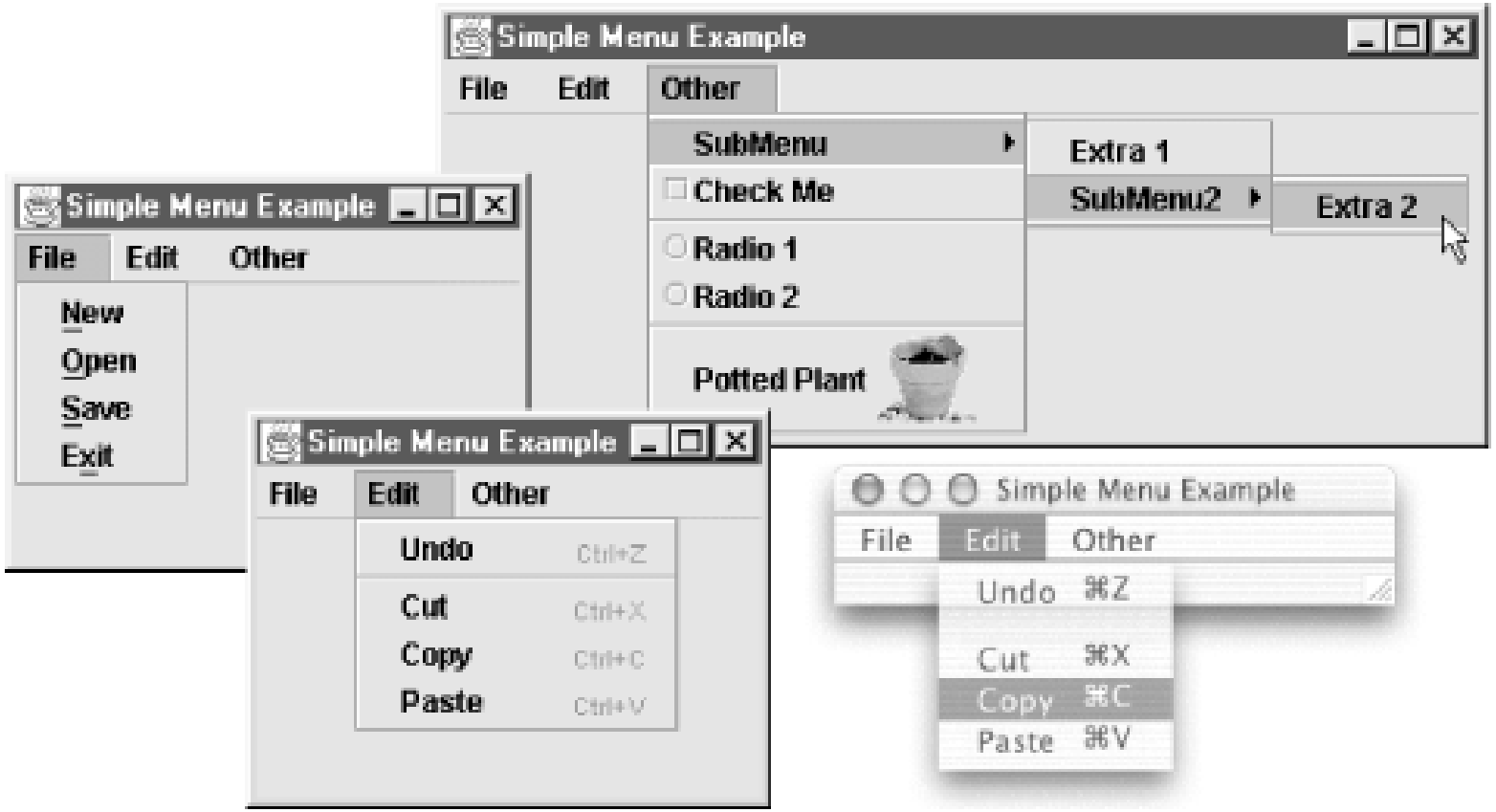

```
String nr = JOptionPane.showInputDialog(null, "Enter an integer",
"Number Entry Dialog", JOptionPane.PLAIN_MESSAGE);
```

Meniuri

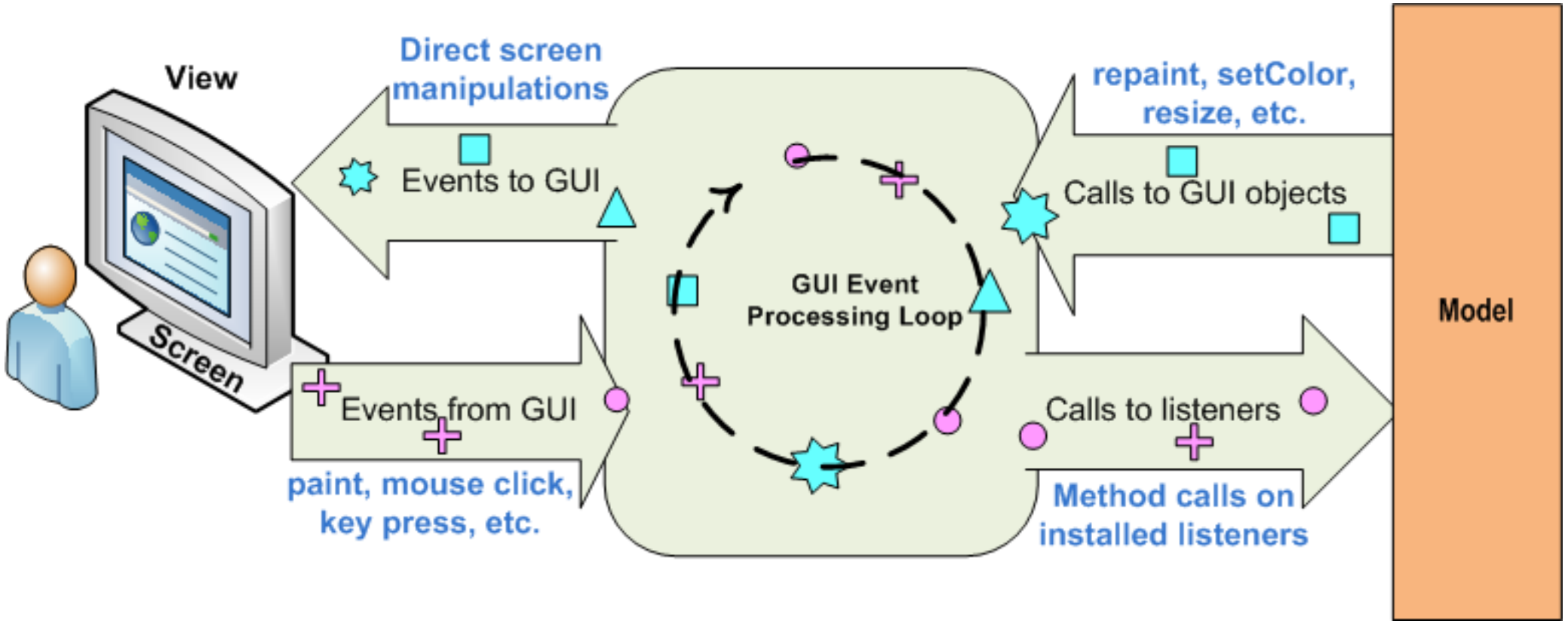
- **Elementele unui meniu orizontal:**
  - `JMenuBar`
  - `JMenu`
  - `JMenuItem`
  - `JSeparator`
  - `JCheckBoxMenuItem / JRadioButtonMenuItem`



Meniuri


Simple Menu Example
File
Edit
Other
SubMenu
Check Me
Radio 1
Radio 2
Potted Plant
Extra 1
SubMenu2
Extra 2
Simple Menu Example
File
Edit
Other
New
Open
Save
Exit
Simple Menu Example
File
Edit
Other
Undo Ctrl+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Simple Menu Example
File
Edit
Other
Undo ⌘Z
Cut ⌘X
Copy ⌘C
Paste ⌘V

Tratarea evenimentelor



Tratarea evenimentelor

- În momentul în care utilizatorul interacționează cu o componentă grafică se vor genera evenimente, adică obiecte **TipEvent** (ActionEvent, ItemEvent, KeyEvent, MouseEvent, WindowEvent, TextEvent).

- **Exemple:** apăsarea unui buton, modificarea textului dintr-un editor, închiderea sau redimensionarea unei ferestre etc.

- Interceptarea și tratarea evenimentelor se realizează prin intermediul unor obiecte **TipListener**, care implementează interfețe specifice evenimentelor generate (ActionListener, KeyListener, MouseListener etc.).

Tratarea evenimentelor

- Obiectele **TipListener** sunt create de către programator prin două modalități:

1. **Implementarea interfeței dedicate**

- Fiecare interfață definește una sau mai multe metode care vor fi apelate automat la apariția unui eveniment:

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

class AscultaTexte implements TextListener {
public void textValueChanged(TextEvent e) {...}}
```

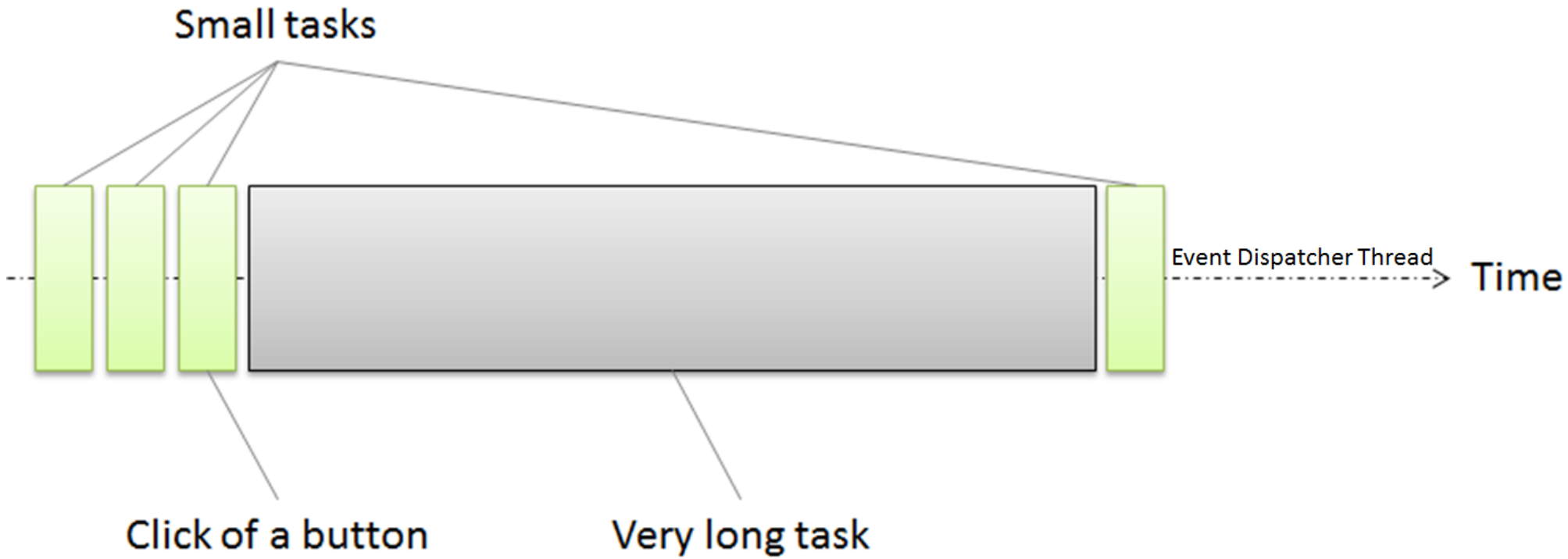
Tratarea evenimentelor

2. Extinderea unor clase adaptor au numele de forma **EvenimentAdapter**

Interfaţa	Adaptor
ActionListener	nu are
AdjustemnrListener	nu are
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
ItemListener	nu are
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
TextListener	nu are
WindowListener	WindowAdapter

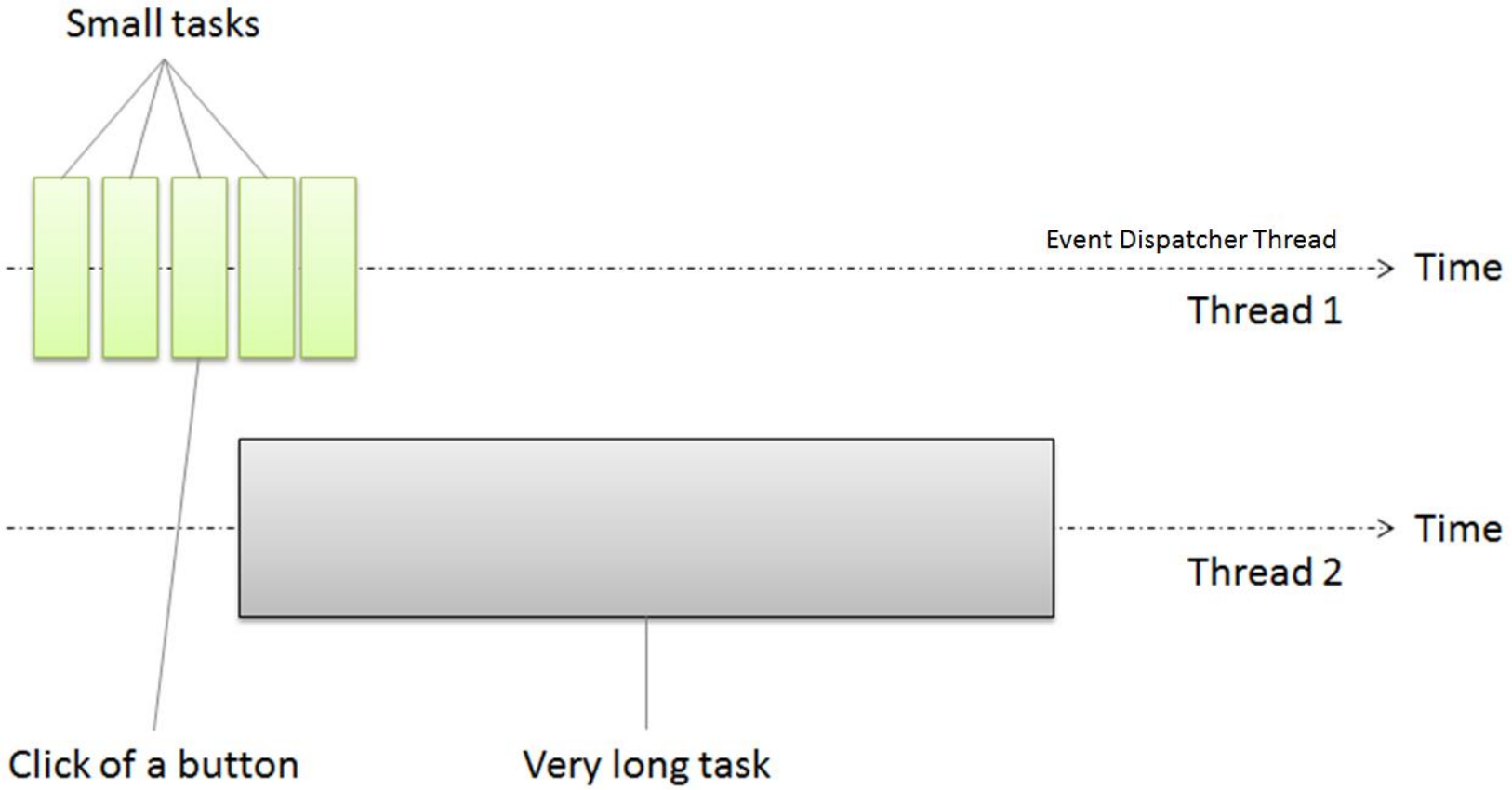
SwingWorker

- Dacă într-o interfață grafică sunt efectuate prelucrări de durată în cadrul firului asociat (**Event Dispatcher Thread - EDT**), atunci interfața grafică se va bloca ("va îngheța").



SwingWorker

- Pentru a evita acest fenomen, trebuie să implementăm prelucrarea consumatoare de timp într-un fir de executare separat, astfel încât să nu blocăm firul interfeței grafice (EDT).

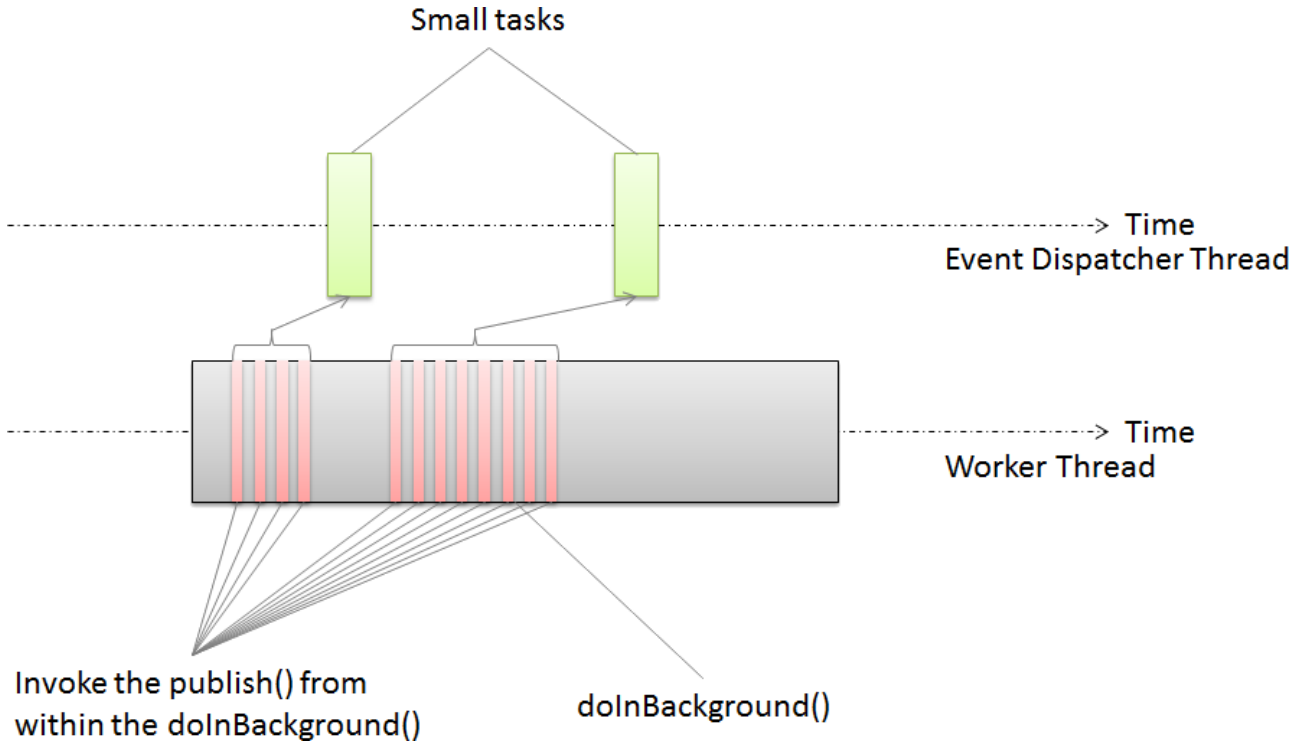


SwingWorker

› În Java 6 a fost introdusă clasa abstractă generică

SwingWorker<TD_1, TD_2>

› Clasa conține metoda **`TD_1 doInBackground()`**, în cadrul căreia se va implementa prelucrarea consumatoare de timp, pe un fir separat.



SwingWorker

- Metoda **`doInBackground()`** va fi planificată pentru executare pe un fir worker, separat de EDT, în momentul apelării metodei **`void execute()`** din EDT.
- Executarea metodei **`doInBackground()`** poate fi întreruptă folosind metoda **`boolean cancel(boolean b)`**.
- După terminarea executării metodei **`doInBackground()`**, în mod natural sau nu, firul interfeței grafice EDT va apela metoda **`void done()`**.
- Valorea returnată de metoda **`doInBackground()`** poate fi accesată folosind metoda **`TD_1 get()`**.
- Pentru a testa dacă metoda **`doInBackground()`** a fost întreruptă sau nu, se poate utiliza metoda **`boolean isCancelled()`**.
- Pentru a testa dacă metoda **`doInBackground()`** s-a terminat se poate utiliza metoda **`boolean isDone()`**.

SwingWorker

- Dacă în timpul executării metodei **`doInBackground()`** dorim ca o valoare intermediară să fie afișată în GUI (în cadrul EDT!), atunci o vom "publica", folosind metoda **`void publish(TD_2 valoare)`**.
- Prelucrarea valorilor intermediare publicate se realizează în cadrul metodei **`void process(List<TD_2> valori)`**.
- Metodele **`publish()`** și **`process()`** lucrează într-un mod asincron, deci metoda **`process()`** poate fi apelată de EDT după mai multe apeluri ale metodei **`publish()`**!!!
- Un obiect **SwingWorker** poate fi rulat o singură dată, respectiv se va crea unul nou de fiecare dată când este nevoie!