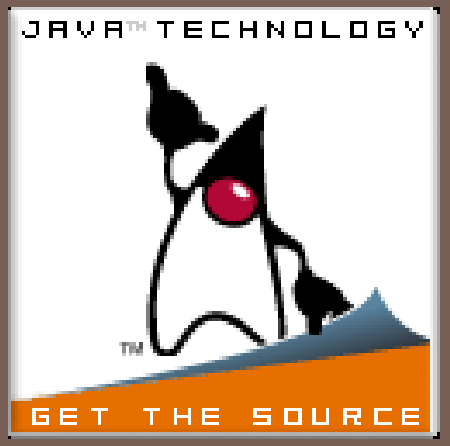
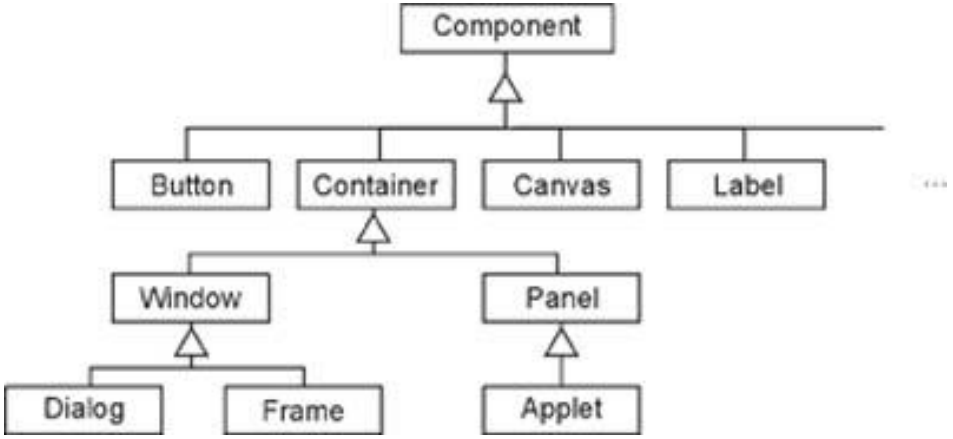


TVORBA GRAFICKÉHO POUŽÍVATEĽSKÉHO ROZHRANIA

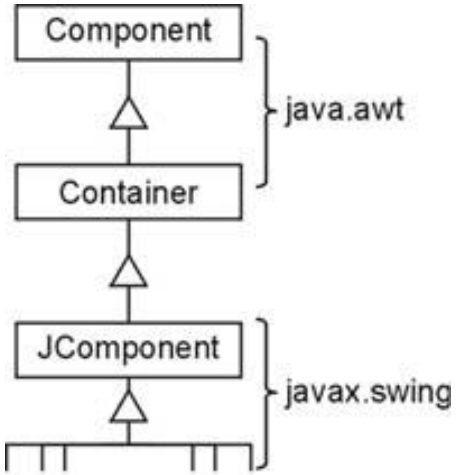
Používateľské rozhranie

- Od verzie 1.0 poskytuje platforma Java prostredníctvom **Abstract Windowing Toolkit (AWT)** nástroje pre tvorbu GUI aplikácií
- AWT mal byť nástroj pre vytváranie aplikácií, ktoré budú mať rovnaký vzhľad na všetkých platformách – preto definuje len spoločnú podmnožinu ovládacích prvkov
- Ovládacie prvky jednotlivých platforiem zobrazovalo príslušné okenné prostredie, AWT len poskytovalo prístup k týmto prvkom
- Táto snaha neuspela, aplikácie mali chudobné rozhranie



Používateľské rozhranie *pokr.*

- Od verzie Java 1.2 je k dispozícii nové prostredie pre vývoj GUI aplikácií – **Swing**
- Toto prostredie zaviedlo odľahčené komponenty, *ktoré sa vykresľujú prostredníctvom JVM*
- Umožňujú zabezpečiť **plnú kontrolu** nad zobrazovaním grafických komponentov, aplikácie majú teda jednotný vzhľad na ľubovoľnej platforme
- Swing komponenty sú postavené na AWT, preto mnoho rysov je rovnakých



Používateľské rozhranie *pokr.*

Swing obsahuje:

- Rozhranie pre 2D grafiku (Java 2D API)
- Množinu GUI komponentov
- Možnosť jednoduchej *zmeny vzhľadu aplikácie* (Pluggable Look and Feel)
- Technológie pre *prenos údajov* (kopírovanie, vkladanie,...)
- Technológie pre jednoduchú tvorbu aplikácií *s podporou národných zvyklostí*
- Technológie pre neobmedzený počet operácií *vracajúcich späť vykonané zmeny (undo a redo)*
- Možnosť vytvárania aplikácií pre hendikepovaných používateľov

MVC architektúra

Swing API architecture follows loosely based MVC architecture in the following manner.

- **Model** reprezentuje dáta komponentu
- **View** reprezentuje vizuálnu reprezentáciu dát komponentu
- **Controller** spracováva a odpovedá na udalostí, napr. vstupné dáta od používateľa v pohľade (view) a môže vyvolať zmeny v dátach komponentu
- Swing-ove komponenty majú Model ako samostatný prvok a View a Controller sú zahrnuté v prvkoch užívateľského rozhrania.
- Takto Swing získava „pluggable look-and-feel" architektúru.

MVC architektúra pokr.

V Swing-u každý komponent má svoj model, dokonca aj najjednoduchšia prvky akými sú tlačidlá.

Existujú 2 druhy modelov v Swing-u:

- **Stavové** – obsluhujú stav komponentu – napr. či komponent bol vybraný/stlačený
- **Dátové** – spracovávajú dáta, s ktorými komponent pracuje

*Trieda **JComponent***

Základným prvkom, definujúcim správanie komponentov Swing je trieda **JComponent**

java.lang.Object
↳ java.awt.Component
↳ java.awt.Container
↳ **javax.swing.JComponent**

Trieda ***JComponent***

Všetci potomkovia *JComponent* dedia vlastnosti:

- Plávajúcu **nápovedu** (tool tips)
- **Vykresľovanie a okraj** (border)
- **Meniteľný vzhľad** (pluggable look and feel)
- Používateľsky **nastaviteľné vlastnosti** (putClientProperty, getClientProperty)
- Podpora pre **správu rozloženia** (setMinimumSize, getMinimumSize,....)
- **Podpora pre hendikepovaných používateľov**
- Podpora pre funkciu **ťahaj a pusti** (drag and drop)
- **Dvojité vyrovnávacie pamäte** pre prekresľovanie prvkov (double buffering)
- **Viazanie činností na klávesy** (Key bindings, napr. medzera pre stlačenie default tlačidla,...)

Trieda JComponent *pokr.*

Nastavovanie vzhľadu komponentu:

- void **setBorder(Border), Border getBorder()** – nastavenie okrajov komponentu
- void **setForeground(Color), Color getForeground()** – farba popredia komponentu (zvyčajne farba, ktorou sa píše)
- void **setBackground(Color) , Color getBackground()** - farba pozadia komponentu (ak je nepriehľadný)
- void **setOpaque(boolean), boolean isOpaque()** - nepriehľadný komponent vyplní pozadie farbou pozadia
- void **setFont(Font), Font getFont()** – font komponentu
- void **setCursor(Cursor), Cursor getCursor()** – kurzor komponentu, napr. aPanel.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

Trieda JComponent *pokr.*

Stav komponentu:

- void **setComponentPopupMenu(JPopupMenu)** - nastaví vyskakovacie menu pre komponent
- void **setToolTipText(String)** – nastaví text plávajúcej nápovede
- void **setName(String),** String **getName()** – meno komponentu, používa sa pre asociáciu textu pre komponenty, ktoré nezobrazujú text
- boolean **isShowing()** – určuje, či je komponent zobrazený na obrazovke
- void **setEnabled(boolean),** boolean **isEnabled()** – určuje, či je komponent aktívny, t.j. reaguje na udalosti
- void **setVisible(boolean),** boolean **isVisible()** – určuje, či je komponent viditeľný

Trieda JComponent *pokr.*

Vykresľovanie komponentu:

- void **repaint(),** void **repaint(int, int, int, int)** – požiadavka na obnovenie vizuálnej reprezentácie komponentu
- void **repaint(Rectangle)** - požiadavka na obnovenie vizuálnej reprezentácie komponentu v zadanej oblasti
- void **revalidate()** – požiadavka na opätovne rozmiestnenie prvkov po zmene rozmerov prvku
- void **paintComponent(Graphics)** – používateľské vykreslenie komponentu. Táto metóda sa prekrýva, *ak treba vykresliť komponent ináč ako štandardne.*

Trieda JComponent *pokr.*

Obsah komponentu:

- Component **add(Component)**
  Component **add(Component, int)**
  void **add(Component, Object)**
  pridá špecifikovaný komponent do kontajnera, hodnota *int* určuje danú pozíciu v kontajneri, *Object* predstavuje parameter pre príslušného správcu rozloženia
- void **remove(int),** void **remove(Component),** void **removeAll()**
  odstráni prvok z kontajnera, hodnota *int* predstavuje pozíciu
- JRootPane **getRootPane()** – vráti RootPane daného kontajnera
- Container **getTopLevelAncestor()**
  vráti najvrchnejší kontajner, v ktorom sa komponent nachádza (JFrame, JApplet, Jdialog, null)
- Container **getParent()** – vráti rodičovský kontajner

Trieda JComponent *pokr.*

Obsah komponentu:

- int **getComponentCount()** – počet prvkov daného kontajnera

- Component **getComponent(int)**
- Component[] **getComponents()**

  vráti komponent daný indexom, resp. všetky komponenty

- Component **getComponentZOrder(int)**

  Component[] **getComponentZOrder()**

  vráti komponenty v z-poradí, komponent s najnižším indexom sa kreslí posledný (najviac vpredu)

Trieda JComponent *pokr.*

Rozloženie komponentu:

- void **setPreferredSize(Dimension),** void **setMaximumSize(Dimension)**
- void **setMinimumSize(Dimension)** - nastaví rozmery komponentu, je to však iba odporúčanie, správca rozloženia ho môže ignorovať
- Dimension **getPreferredSize(),** Dimension **getMaximumSize(),** Dimension **getMinimumSize()** – vráti rozmery komponentu
- void **setAlignmentX(float),** void **setAlignmentY(float)** – zarovnanie komponentu, hodnota medzi 0 a 1, 0 = vľavo hore; 0.5 = stred; 1 = vpravo dole
- float **getAlignmentX(),** float **getAlignmentY()** – vracia požadované zarovnanie
- void **setLayout(LayoutManager),** LayoutManager **getLayout()** – určuje správcu rozloženia daného komponentu
- void **applyComponentOrientation(ComponentOrientation)** void **setComponentOrientation(ComponentOrientation)** – nastaví orientáciu kontajnera a jeho prvkov (zľava – doprava, alebo naopak)

Trieda JComponent *pokr.*

Pozícia a rozmery komponentu:

- int **getWidth(),**

  int **getHeight()** – aktuálna šírka a výška komponentu
- Dimension **getSize(),**

  Dimension **getSize(Dimension)** – vráti rozmery ako inštanciu Dimension
- int **getX(),** int **getY()** – nastaví počiatočné hodnoty pozície komponentu vzhľadom na rodiča
- Rectangle **getBounds(),**

  Rectangle **getBounds(Rectangle)** - vráti hranice komponentu

  – ľavý horný roh, výšku a šírku
- Point **getLocation(),** Point **getLocation(Point)**

  – vráti pozíciu komponentu (ľavý horný roh)

Trieda JComponent *pokr.*

Pozícia a rozmery komponentu:

- Point **getLocationOnScreen()** – vráti pozíciu vzhľadom na obrazovku
- Insets **getInsets()** – vracia veľkosť okrajom komponentu
- void **setLocation(int, int),** void **setLocation(Point)**
- void **setSize(int, int),** void **setSize(Dimension)**
- void **setBounds(int, int, int, int),** void **setBounds(Rectangle)**

Udalosti

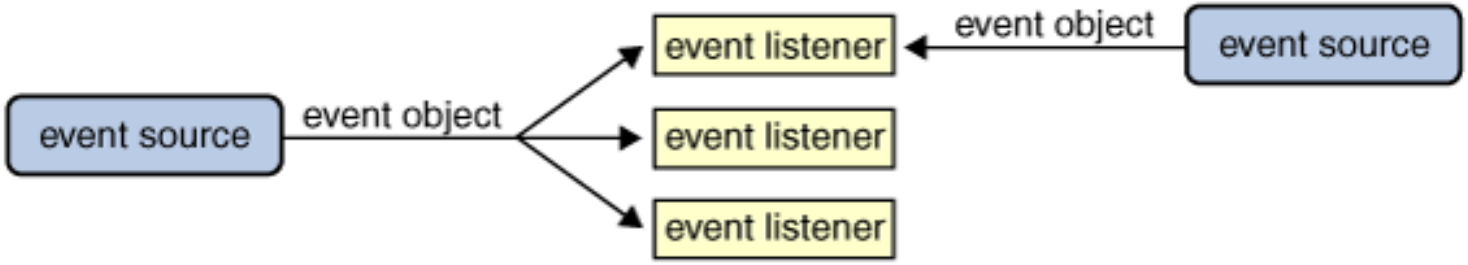
GUI aplikácie **sú riadené udalosťami** (event driven), generovanými používateľom.

Udalosťou je:

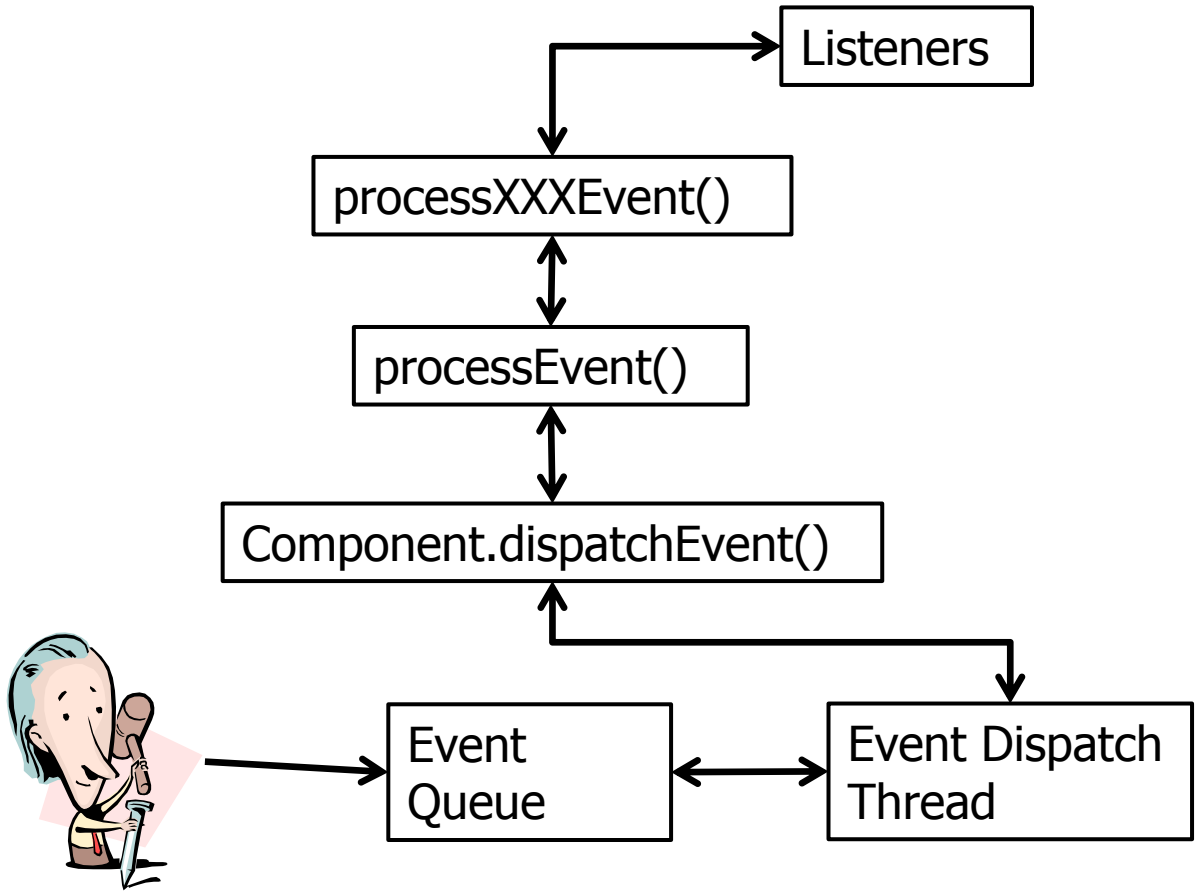
- **Akcia používateľa** (pohyb myšou, stlačenie klávesy,...)
- **Zmena stavu komponentu** (stlačenie tlačidla,...)

Zdrojom udalosti (event source) môže byť ľubovoľný komponent

Informácia o výskyte udalosti je distribuovaná všetkým komponentom, ktoré sa **zaregistrujú** u zdroja udalosti (event listener).



Udalosti *pokr.*



Spracovanie udalosti

Udalosti *pokr.*

Tvorba vlastnej udalosti:

1. **Definovať triedu**, obsahujúcu informáciu o udalosti (potomok java.util.EventObject), **XXXEvent**
2. **Definovať rozhranie XXXListener**, potomok rozhrania java.util.EventListener
3. V triede, ktorá bude generovať danú udalosť definovať metódy:
   a) **addXXXListener()** – registrácia prijímateľa správy o udalosti
   b) **removeXXXListener()** – zrušenie registrácie

Trieda EventObject:

EventObject(Object source) – konštruktor
Object **getSource**() – vráti objekt, ktorý je zdrojom udalosti
String - **toString**() – reťazcová reprezentácia udalosti

Rozhranie EventListener – značkovacie rozhranie, nemá žiadne metódy

Udalosti *pokr.*

Dva druhy udalostí:

1. **Nízkoúrovňové udalosti, generované používateľom**. Pohyb myšou, stláčanie klávesov, tlačidiel myši, odovzdávanie fokusu, ...
2. **Vysokoúrovňové udalosti, zapríčinené zmenou stavu objektu.** Mnohokrát vznikajú ako dôsledok nízkoúrovňových udalostí. Kliknutím na tlačidlo myši nad tlačidlom (butonom) sa odštartuje akcia tlačidla, ktorá môže spôsobiť zmenu údajov v tabuľke, a pod.

Nízkoúrovňové udalosti

Udalosť	Názov	Zdroj
Udalosti od klávesnice	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent) - funkčné klávesy	Potomkovia triedy java.awt.Component
Udalosti od tlačidiel myši	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)	Potomkovia triedy java.awt.Component
Pohyb myši	mouseDragged(MouseEvent) mouseMoved(MouseEvent)	Potomkovia triedy java.awt.Component
Koliesko myši	mouseWheelMoved(MouseEvent)	Potomkovia triedy java.awt.Component

Nízkoúrovňové udalosti

Udalosť	Názov	Zdroj
Fokus	focusGained(FocusEvent) focusLost(FocusEvent)	Potomkovia triedy java.awt.Component
Komponenty	componentAdded(ContainerEvent) componentRomoved(ContainerEvent)	Potomkovia triedy java.awt.Container
Zmena stavu okna	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)	Okná (potomkovia triedy java.awt.Window)

Vysokoúrovňové udalosti

Udalosť	Názov	Zdroj
Vlastnosť objektu	propertyChanged(PropertyChangeEvent)	Triedy Swing
Zmena stavu	stateChanged(ChangeEvent)	Triedy Swing
Aktivácia prvku	actionPerformed(ActionEvent)	Triedy Swing, ktoré predstavujú nejakú činnosť (tlačidlo, menu)

Obsluha udalosti

Pre obsluhu udalostí, generovaných komponentom môžeme použiť **rôzne techniky:**

- **Definovať triedu, implementujúcu príslušné rozhranie**
  1. Ako anonymnú vnútornú triedu
  2. Ako bežnú triedu

- **Použiť adaptér,** implementujúci rozhranie a prekryť v ňom potrebné metódy (*pre udalosti s veľkým množstvom metód*)

- **Metódou komponentu addXXXListener zaregistrovať inštanciu triedy** implementujúcej rozhranie udalosti

Obsluha udalosti *pokr.*

Vytvoríme obsluhu udalosti actionPerformed:

1. Definujeme triedu, ktorá implementuje rozhranie ActionListener:

   public class MyButtonActionListener **implements** ActionListener {…}

2. V tejto triede implementujeme **jedinú metódu rozhrania**

   public void actionPerformed(ActionEvent e) {…}

3. Zaregistrujeme pri komponente, ktorý generuje udalosť záujem o informovanie

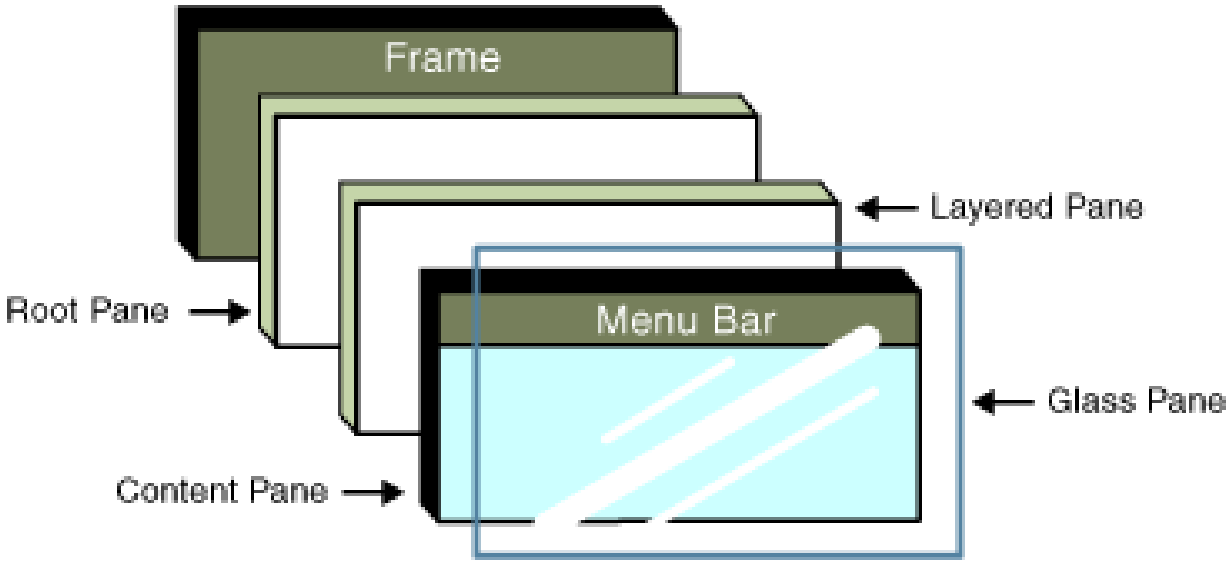
   button.addAdctionListener(**new** MyButtonActionListener());

Hlavné kontajnery

Predstavujú základ GUI aplikácie, využívajú prostriedky hosťujúceho operačného systému pre získanie plochy na zobrazenie svojho obsahu:

- **JFrame** – okno GUI aplikácie
- **JDialog** – modálne okno, určené pre tvorbu dialógov (modálne okno sa neuzavrie, kým neodpoviete na položenú otázku)
- **JApplet** – základ appletov

Úlohy jednotlivých grafických vrstiev

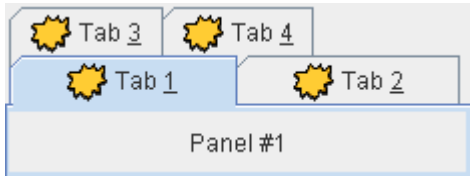


Content pane obsahuje všetky viditeľné GUI komponenty rámca
Root pane spravuje vnútro rámca vrátane content pane, glass pane a ďalši.
Glass pane sa môže použiť na zachytenie udalosti myši alebo na kreslenie ponad grafické prvky aplikácie.
Layered pane - dovoľuje dať komponenty nad alebo za inými komponentmi.

Kontajnery strednej úrovne

Umožňujú organizovať ovládacie prvky GUI aplikácie:

- **JPanel**
- **JScrollPane**
- **JTabbedPane**
- **JSplitPane**
- **JToolBar**

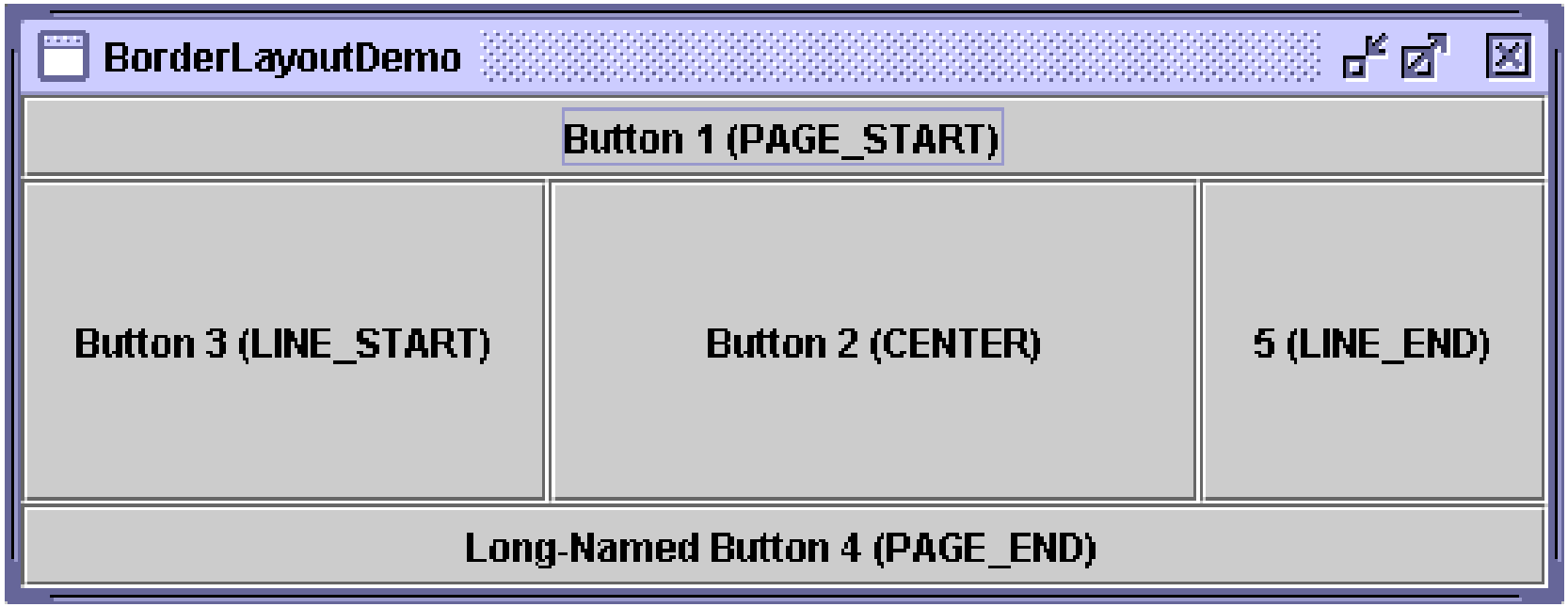


Správcovia rozloženia

- **Určujú spôsob, akým sa rozmiestnia prvky v kontajneri.**
- Cieľom bolo poskytnúť vývojárom nástroj na jednoduché a efektívne rozmiestňovanie prvkov na ploche kontajnera v závislosti na jeho veľkosti
- Je možné vytvoriť vlastného správcu rozloženia
- Existuje množstvo správcov rozložení, od jednoduchých, určených na priame použitie až po komplexné nástroje na vizuálnu tvorbu rozhrania
- Každý kontajner môže mať vlastného správcu rozloženia:

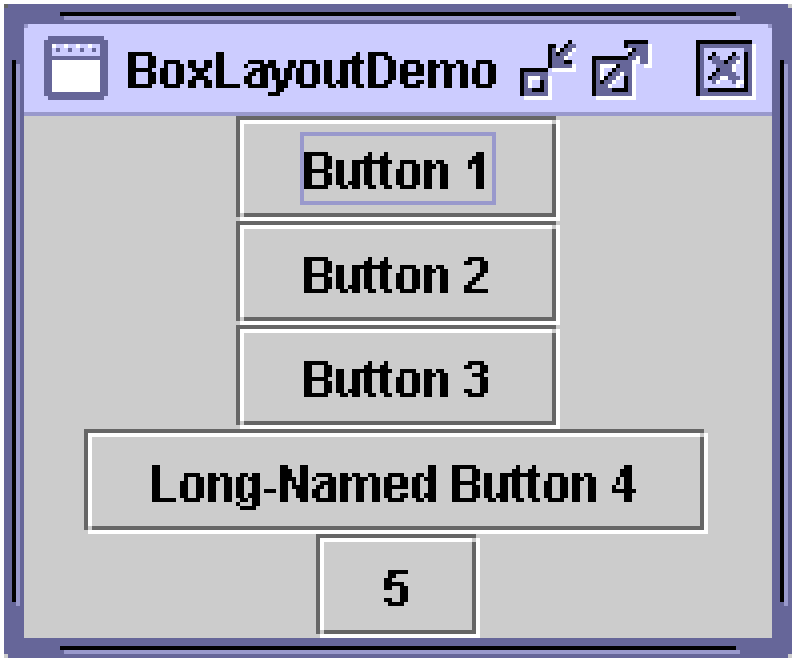
```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());

panel.add(aComponent, BorderLayout.PAGE_START);
```

Správcovia rozloženia *pokr.*



Border Layout – ukladá komponenty do piatich častí plochy

Správcovia rozloženia *pokr.*



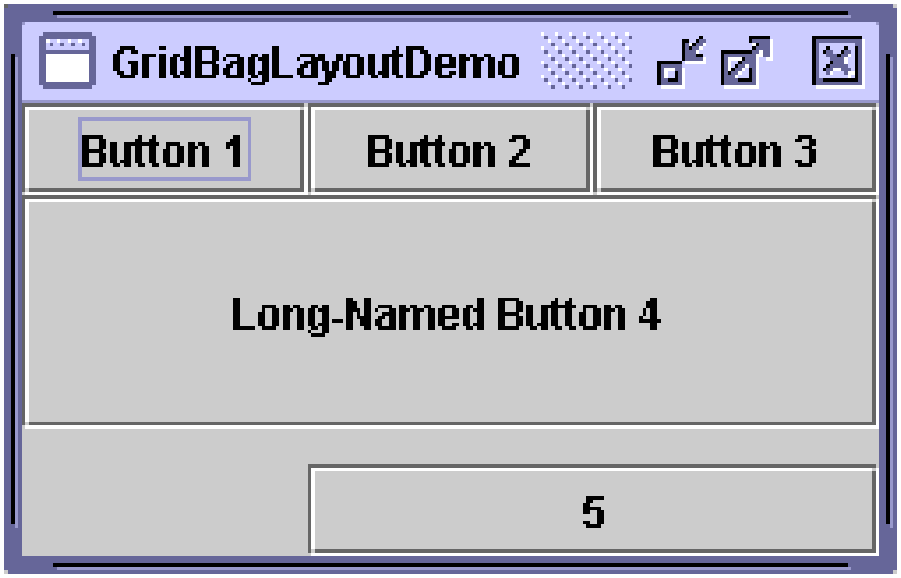
Box Layout – ukladá komponenty do riadku, alebo do stĺpca

Správcovia rozloženia *pokr.*

Flow Layout – ukladá komponenty za sebou v riadku, ak nie je miesto, tak do ďalšieho riadku

Grid Layout – ukladá komponenty do mriežky

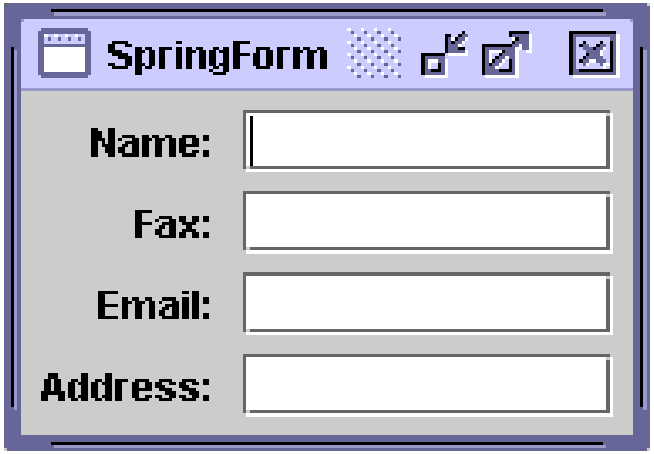
Správcovia rozloženia *pokr.*



GridBagLayout – **najkomplexnejší správca,** určený hlavne pre použitie pomocou nástrojov na vizuálny návrh GUI

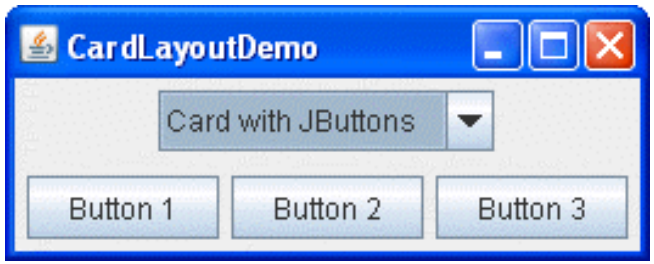
Každý komponent spravovaný pomocou GridBagLayout má priradený objekt typu GridBagConstraints, ktorý špecifikujú kde v mriežke bude objekt umiestnený, jeho min a max rozmery a kde v zobrazovanej ploche sa ukáže.

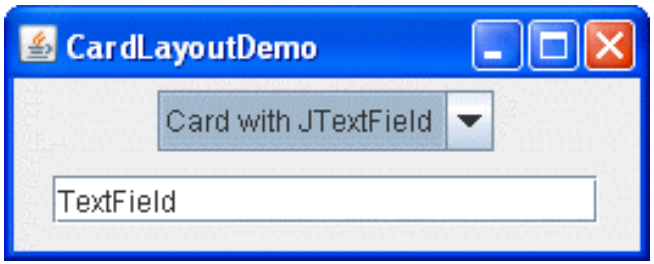
Správcovia rozloženia *pokr.*



Spring Layout – umožňuje definovať relatívnu polohu prvkov, používaný hlavne nástrojmi na vizuálny návrh GUI.

Správcovia rozloženia *pokr.*



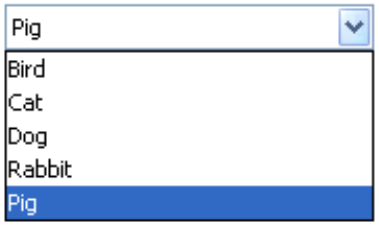


Card Layout – umožňuje „prepínať" medzi viacerými komponentmi, z ktorých práve jeden je viditeľný

Základné ovládacie prvky

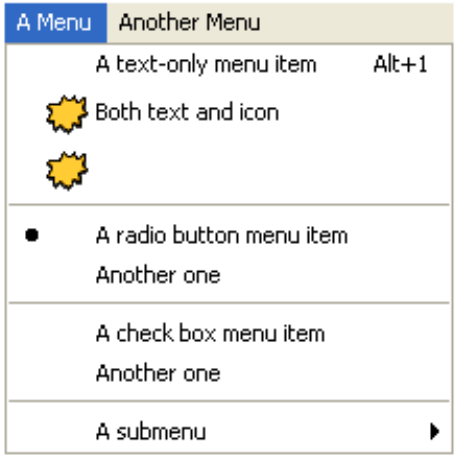
JButton

JCheckBox

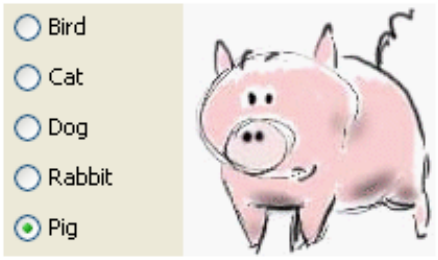


JComboBox

JList



JMenu



JRadioButton

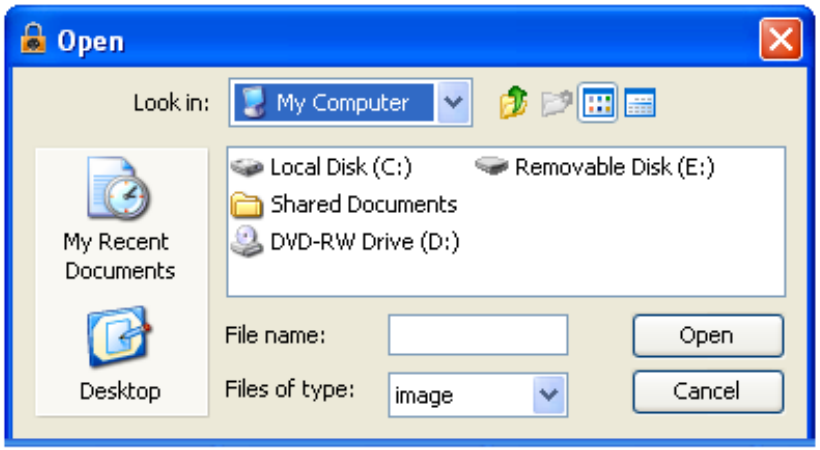
JSlider

JSpinner

JTextField

JPasswordField

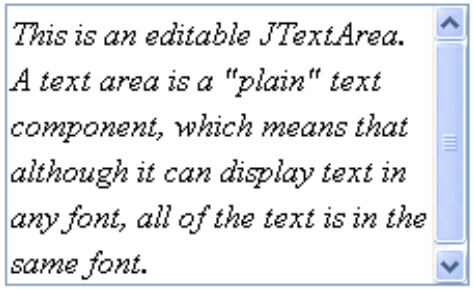
Základné ovládacie prvky *pokr.*



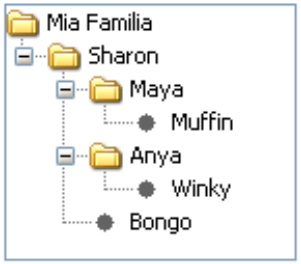
JFileChooser

Host	User	Password	Last Modified
Biocca Games	Freddy	!#asf6Awwzb	Mar 16, 2006
zabble	ichabod	Tazb!34$fZ	Mar 6, 2006
Sun Developer	fraz@hotmail.com	AasW541!fbZ	Feb 22, 2006
Heirloom Seeds	shams@gmail.com	bkz[ADF78!	Jul 29, 2005
Pacific Zoo Shop	seal@hotmail.com	vbAf124%z	Feb 22, 2006

JTable



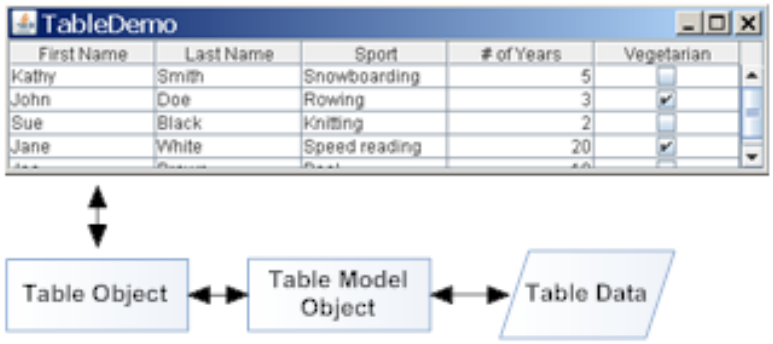
JTextArea



JTree

Tvorba tabuliek

Trieda **JTable** umožňuje zobraziť údaje, organizované vo forme dvojrozmernej tabuľky:



Tvorba tabuliek *pokr.*

Trieda JTable predstavuje **pohľad** na tabuľku, reprezentovanú **modelom.**

Údaje v tabuľke môžu byť zadané

- pri konštrukcii:

  JTable(Object[][] rowData, Object[] columnNames)

  JTable(Vector rowData, Vector columnNames)

- určením dátového modelu definovaného rozhraním TableModel, napr. vytvorením potomka triedy AbstractTableModel:

  JTable(TableModel dm)

Tvorba tabuliek *pokr.*

Trieda **AbstractTableModel** umožňuje implementovať model tabuľky:

- int **findColumn(String columnName)**
- int **getColumnCount()**
- String **getColumnName(int columnIndex)**
- int **getRowCount()**
- Object **getValueAt**(int rowIndex, int columnIndex)
- Class<?> **getColumnClass(int columnIndex)**
- boolean **isCellEditable(int rowIndex, int columnIndex)**
- void **setValueAt(Object aValue, int rowIndex, int columnIndex)**

Tvorba tabuliek *pokr.*

Zmena modelu sa oznamuje metódami:

- void **addTableModelListener**(TableModelListener l)
- void **fireTableCellUpdated**(int row, int column)
- void **fireTableChanged**(TableModelEvent e)
- void **fireTableDataChanged**()
- void f**ireTableRowsDeleted**(int firstRow, int lastRow)
- void **fireTableRowsInserted**(int firstRow, int lastRow)
- void **fireTableRowsUpdated**(int firstRow, int lastRow)
- void **fireTableStructureChange**d()
- <T extends EventListener> T[] **getListeners**(Class<T> listenerType)
- TableModelListener[] **getTableModelListeners**()
- void r**emoveTableModelListener(**TableModelListener l)

Rozhranie TableModelListener definuje metódu:

- void **tableChanged**(TableModelEvent e)

Tvorba tabuliek *pokr.*

Trieda **TableModelEvent** definuje udalosť, vzniknutú v modeli:

- static int ALL_COLUMNS
- static int DELETE
- static int HEADER_ROW
- static int INSERT
- static int UPDATE
- int **getColumn**() - stĺpec, kde udalosť nastala
- int **getFirstRow**() – prvý riadok, kde udalosť nastala
- int **getLastRow**() – posledný riadok, kde udalosť nastala
- int **getType**() – typ udalosti (INSERT, UPDATE, DELETE)

Tvorba tabuliek *pokr.*

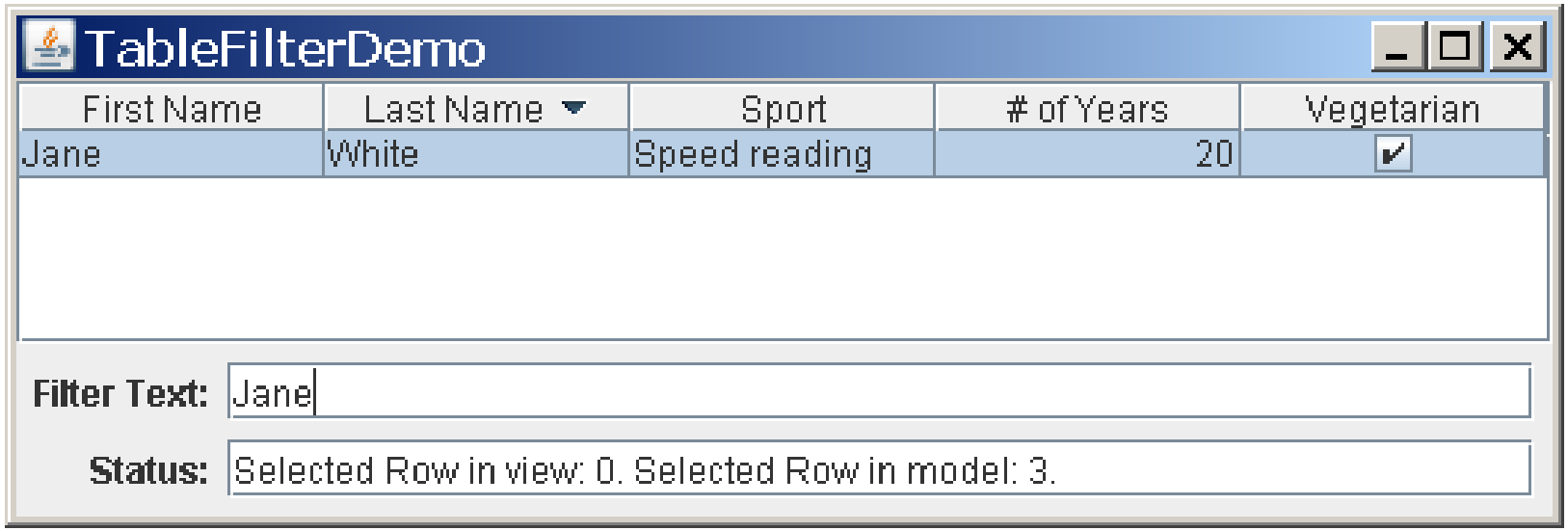
Zobrazovanie buniek prebieha nasledovne:

1. Určí sa, či pre daný stĺpec existuje používateľom definovaná metóda zobrazenia. Ak áno, použije sa.
2. Pomocou metódy **getColumnClass** sa určí typ údajov v stĺpci. Podľa typu sa použije príslušná metóda:
   - Boolean – zaškrtávacie políčko
   - Number – vpravo zarovnaný reťazec
   - Double, Float – text naformátovaný pomocou NumberFormat
   - Date – text naformátovaný pomocou DateFormat
   - ImageIcon, Icon – centrovaný popisok (label)
   - Object – textová reprezentácia

Tvorba tabuliek *pokr.*

Triedenie podľa stĺpcov:

Aktivovaním atribútu **autoCreateRowSorter** sa povolí triedenie stĺpcov podľa prirodzeného usporiadania hodnôt v stĺpci



Tvorba tabuliek *pokr.*

Výber riadkov tabuľky – metóda **JTable.setSelectionMod**e:

- SINGLE_SELECTION – výber iba jedného riadku
- SINGLE_INTERVAL_SELECTION – výber spojitého intervalu riadkov
- MULTIPLE_INTERVAL_SELECTION – výber riadkov nespojitých intervalov

Pre informácie o výbere treba zaregistrovať inštanciu rozhrania ListSelectionListener:

- JTable.getSelectionModel().addListSelectionListener

Tvorba tabuliek *pokr.*

Tlač tabuliek

...

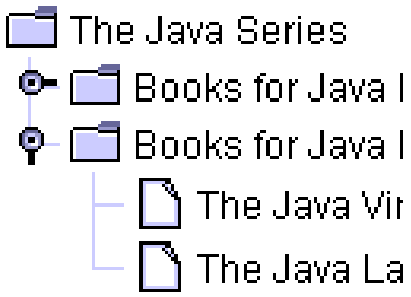
```
table.print();
```

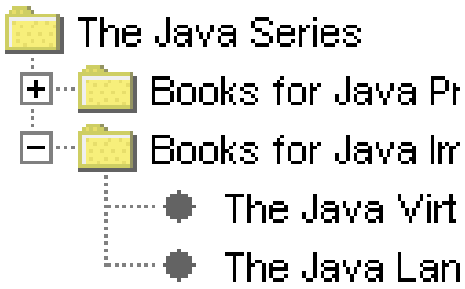
...

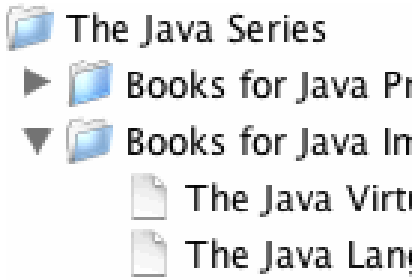
Možnosť pridania záhlavia a ďalších prvkov pre papierový výstup.

Zobrazovanie hierarchických údajov

Trieda **JTre**e umožňuje zobraziť údaje, organizované v stromovej štruktúre:







Trieda JTree

Podobne ako ostatné prvky knižnice Swing pracuje s dátovým modelom, štandardne je to inštancia **triedy DefaultTreeModel**. Jednotlivé uzly sú reprezentované triedou, implementujúcou rozhranie TreeNode:

- Enumeration **children**()
- boolean **getAllowsChildren**()
- TreeNode g**etChildAt**(int childIndex)
- int **getChildCount**()
- int **getIndex**(TreeNode node)
- TreeNode **getParent**()
- boolean **isLeaf**()

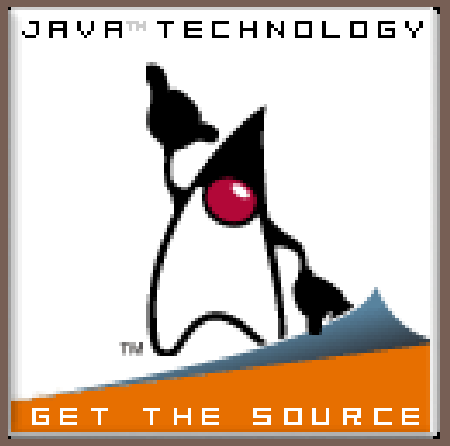
Trieda JTree *pokr.*

Tvorba stromu:

```
public TreeDemo() {
    ...
    DefaultMutableTreeNode top = new DefaultMutableTreeNode("Java");
    createNodes(top);
    tree = new JTree(top);
    ...
...
}
```

Trieda JTree *pokr.*


```
private void createNodes(DefaultMutableTreeNode top) {
    DefaultMutableTreeNode category = null;
    DefaultMutableTreeNode book = null;

    category = new DefaultMutableTreeNode("Books for Java Programmers");
    top.add(category);

    //original Tutorial
    book = new DefaultMutableTreeNode(new BookInfo
        ("The Java Tutorial: A Short Course on the Basics", "tutorial.html"));
     category.add(book);
...
    category = new DefaultMutableTreeNode("Books for Java Implementers");
    top.add(category);
...
}
```



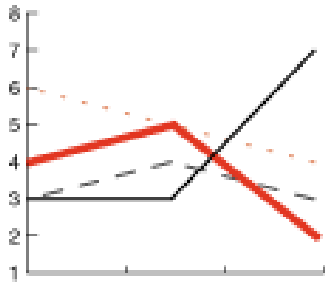
2D GRAFIKA

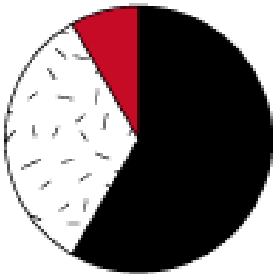
2D grafika

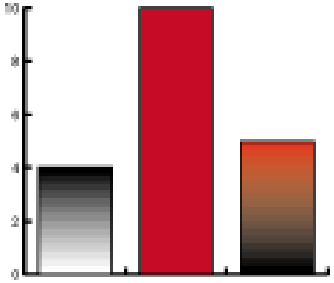
2D Java API poskytuje:

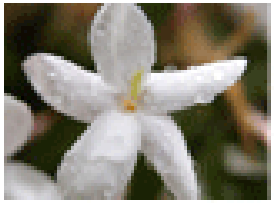
- Jednotný model pre vykresľovanie na displeji a tlačiarni
- Mnoho grafických primitív (krivky, obdĺžniky, elipsy,...)
- Vyplňovanie tvarov farbou, textúrou,...
- Zobrazovanie textov v rôznych fontoch
- Mechanizmus detekcie výberu tvarov, textu a obrázkov
- Mechanizmus riadenia zobrazenia prekrývajúcich sa objektov
- Správu farieb

Príklady použitia 2D grafiky

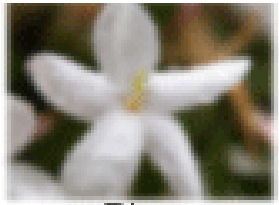




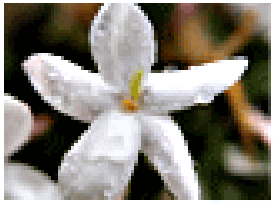




Image



Blur

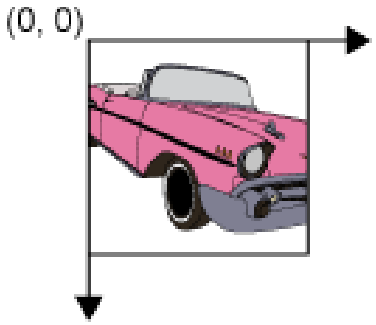


Sharpen

2D grafika *pokr.*

Dve súradnicové sústavy:

- **Používateľská** – nezávislá na zariadení, logický systém súradníc.



- **Súradnicový systém zariadenia** – rozlíšenie závisí na vlastnostiach zariadenia

Vykresľovanie vlastnej grafiky

Trieda JComponent obsahuje metódu:

protected void **paintComponent(Graphics g)**

Jej prekrytím je možné vytvoriť *vlastnú grafickú reprezentáciu komponentu*.

Triedy Graphics a Graphics2D predstavujú **plátno komponentu**, do ktorého sa kreslí.

Možností triedy Graphics2D

Obrysy

Výrez vykresľovanej oblasti

Výplň

Prevod textových reťazcov na glyfy.

Kompozícia

Určenie preferencií v kompromise medzi rýchlosťou a kvalitou vykresľovania

Transformácia atribútov

Grafické primitíva - bod

java.awt.**Point** – predstavuje bod v dvojrozmernom priestore:

- Point() – vytvorí bod so súradnicami (0,0)
- Point(int x, int y) - vytvorí bod so súradnicami (x,y)
- Point(Point p) – vytvorí nový bod s rovnakými súradnicami ako vzor
- boolean **equals**(Object obj) – porovnanie dvoch bodov, vracia *true*, ak sú rovnaké súradnice x aj y oboch bodov
- Point **getLocation**()
- double **getX**()
- double **getY**()
- void **move**(int x, int y) – presunie bod na pozíciu (x,y)
- void **setLocation**(double x, double y) – ako move, hodnoty sú zaokrúhlené a orezané tak, aby sa zmestili do intervalu <Integer.MIN_VALUE, Integer.MAX_VALUE>
- void **setLocation**(int x, int y) – ako move
- void **setLocation**(Point p) – ako move
- String **toString**()
- void **translate**(int dx, int dy) - posunie bod na pozíciu (x + dx,y + dy)

Grafické primitíva - úsečka

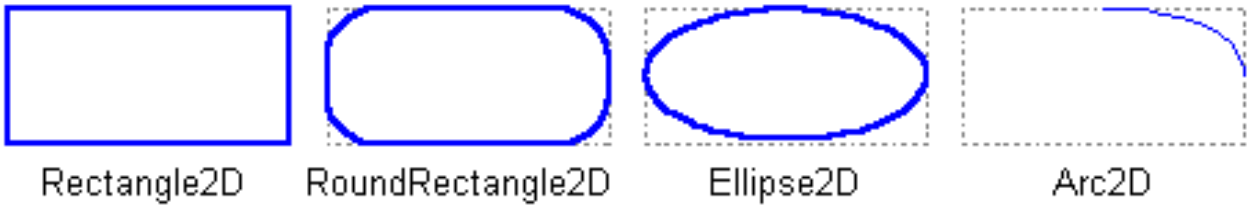
Abstraktná trieda Line2D

boolean **contains**(double x, double y) – určí, či daný bod je bodom úsečky
Rectangle **getBounds**()
abstract Point2D **getP1**()
abstract Point2D **getP2**()
abstract double **getX1**()
abstract double **getX2**()
abstract double **getY1**()
abstract double **getY2**()
boolean **intersects**(double x, double y, double w, double h)
boolean **intersectsLine**(double x1, double y1, double x2, double y2)
double **ptLineDist**(double px, double py)
abstract void **setLine**(double x1, double y1, double x2, double y2)
void **setLine**(Line2D l)
void **setLine**(Point2D p1, Point2D p2)

Nakreslenie:

```
g2.draw(new Line2D.Double(x1, y1, x2, y2));
```

Ďalšie grafické primitíva



- **Štvoruholník**:
  - g2.draw(new **Rectangle2D**.Double(x, y, rectwidth, rectheight));
- **Elipsa**
  - g2.draw(new **Ellipse2D**.Double(x, y, rectwidth, rectheight));
- **Oblúk**
  - g2.draw(new **Arc2D**.Double(x, y, width, height, 90, 135, Arc2D.OPEN));

Ďalšie grafické primitíva

- **Polygón**:

```
// draw GeneralPath (polyline)  - kreslenie lomenej čiary

int x2Points[] = {0, 100, 0, 100};
int y2Points[] = {0, 50, 50, 0};
GeneralPath polyline = new GeneralPath(GeneralPath.WIND_EVEN_ODD,
                                       x2Points.length);
polyline.moveTo (x2Points[0], y2Points[0]);
for (int index = 1; index < x2Points.length; index++) {
        polyline.lineTo(x2Points[index], y2Points[index]);
 };
g2.draw(polyline);
```

Trieda *GeneralPath* umožňuje vykresliť ľubovoľný tvar zadaním rad pozícií pozdĺž hraníc obrazca. Tieto pozície môžu byť spojené úsečkami, kvadratickými krivkami, alebo kubickými (Bézierovymi) krivkami.

Kreslenie obrysov

Pre každý geometrický tvar je možné nakresliť jeho obrys.
Rozhranie Stroke, resp. jeho potomok BasicStroke:

```
final static float dash1[] = {10.0f};
final static BasicStroke dashed = new BasicStroke(1.0f,
                                                  BasicStroke.CAP_BUTT,
                                                  BasicStroke.JOIN_MITER,
                                                  10.0f, dash1, 0.0f);
g2.setStroke(dashed);
g2.draw(new RoundRectangle2D.Double(x, y, rectWidth, rectHeight,10, 10));
```

Kreslenie obrysov *pokr.*

Každý obrys má určený spôsob spájania čiar:

- JOIN_BEVEL 
- JOIN_MITER 
- JOIN_ROUND 

A ukončovania prerušovania čiar:

- CAP_BUTT 
- CAP_ROUND 
- CAP_SQUARE 

Kreslenie výplne

Pre každý geometrický tvar je možné vyplniť jeho plochu:

```
redtowhite = new GradientPaint(0,0,color.RED,100, 0,color.WHITE);
g2.setPaint(redtowhite);
g2.fill (new Ellipse2D.Double(0, 0, 100, 50));
```

Fonty

Java poskytuje päť logických rodín písiem:

- Dialog
- DialogInput
- Monospaced
- Serif
- SansSerif

```
Font font = new Font("Dialog", Font.PLAIN, 12);
```

Metriky fontov


```
// get metrics from the graphics
    FontMetrics metrics = graphics.getFontMetrics(font);

// get the height of a line of text in this font and render context
    int hgt = metrics.getHeight();

// get the advance of my text in this font and render context
   int adv = metrics.stringWidth(text);

// calculate the size of a box to hold the text with some padding.
   Dimension size = new Dimension(adv+2, hgt+2);
```