

JAVA

GUI v std knihovně

Přehled

- Java GUI
 - Java 1.0 (1996) – AWT
 - použití nativních GUI komponent
 - Java 1.2 (2000) – Swing
 - kompletní GUI v Javě
 - JavaFX (2007)
 - nová technologie
 - běžící nad Java VM
 - ale vlastní jazyk
 - deklarativní
 - zamýšleno jako konkurence pro Flash
 - neujalo se
 - JavaFX 2.0 (2011)
 - zůstalo pouze API (jazyk zahozen)
 - od JDK 7 update 6 součást std JDK (JavaFX 2.2)
 - Java 8 – JavaFX 8
 - Java 11 – JavaFX oddělena z JDK

JAVA

Swing

Swing

- balíky
 - javax.swing....
 - používají se třídy z java.awt...
 - mnoho tříd je potomky tříd z java.awt...
- AWT
 - stále je přítomno
 - kompatibilita,...
 - používá se model událostí
- implementace plně v Javě
 - na všech platformách vypadá a chová se stejně
 - vzhled i chování lze upravovat – přizpůsobovat platformě
- podpora pro 2D grafiku, tisk, drag-and-drop, lokalizace, ...

Hello World

```
import javax.swing.*;

public class HelloWorldSwing {
    private static void createAndShowGUI() {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);
        frame.pack();
        frame.setVisible(true);
    }

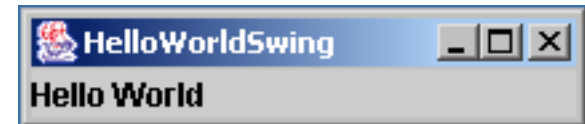
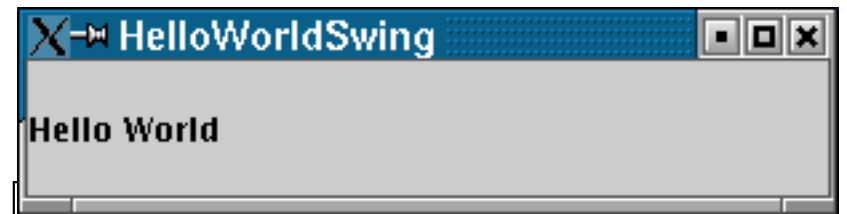
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable()
        {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```



Hello World (2)

```
import javax.swing.*;
```

```
public class HelloWorldSwing {  
    private static void createAndShowGUI() {  
        JFrame.setDefaultLookAndFeelDecorated(true);  
        JFrame frame = new JFrame("HelloWorldSwing");  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        JLabel label = new JLabel("Hello World");  
        frame.getContentPane().add(label);  
        frame.pack();  
        frame.setVisible(true);  
    }  
    public static void main(String[] args) {  
        javax.swing.SwingUtilities.invokeLater(new Runnable() {  
            public void run() {  
                createAndShowGUI();  
            }  
        });  
    }  
}
```



Layout

```
// příklad: cz.cuni.mff.java.gui.ButtonAndLabel  
Container pane = frame.getContentPane();  
pane.setLayout(new GridLayout(0, 1));
```

```
JButton button = new JButton("Click here");  
pane.add(button);
```



```
JLabel label = new JLabel("Hello World");  
pane.add(label);
```

- layout

- určuje velikost a umístění komponent v kontejneru
- určuje změnu velikosti a umístění při změně velikosti kontejneru
- implementuje interface **java.awt.LayoutManager**

Panel a okraje

```
// příklad: cz.cuni.mff.java.gui.ButtonAndLabel2
JPanel panel = new JPanel(new GridLayout(0, 1));
panel.setBorder(BorderFactory.createEmptyBorder(30,
    30, 10, 30));
JButton button = new JButton("Click here");
panel.add(button);
JLabel label = new JLabel("Hello World");
panel.add(label);
...
frame.getContentPane().add(panel);
```

- panel
 - "lightweigth" kontejner
 - kontejnery se vkládají do jiných kontejnerů
- okraj (Border)
 - jak vykreslovat okraje komponent (JComponent)

Look & Feel

```
// příklad: cz.cuni.mff.java.gui.ButtonAndLabel3
String lookAndFeel =
    UIManager.getCrossPlatformLookAndFeelClassName();
UIManager.setLookAndFeel(lookAndFeel);
```

- určuje vzhled a chování GUI
- L&F obsažený v JDK
 - crossplatform (Metal) – na všech platformách stejný vzhled
 - Windows – podobné Windows GUI
 - system
 - na Unixu – metal
 - na Windows – windows
 - Motif
 - GTK+ – od JDK 1.4.2
 - Nimbus – od JDK 6 u10
- lze vytvářet vlastní

Události

Observer
pattern

- GUI se ovládá pomocí událostí (*events*)
 - př. stisknutí tlačítka → událost
- zpracování událostí – *listener*
 - objekt si zaregistruje *listener* → dostává informace o události
- mnoho druhů událostí (a jim odpovídající *listenery*)
 - př. stisk tlačítka, zavření okna, pohyb myši,...

```
public class ButtonAndLabel4 implements ActionListener {  
    ...  
    JButton button = new JButton("Click here");  
    button.addActionListener(this);  
    ...  
    public void actionPerformed(ActionEvent e) {  
        clicks++;  
        label.setText("Hello World: "+clicks);  
    }  
}
```

Události

- jeden *listener* lze zaregistrovat pro více událostí

```
public class TempConvert implements ActionListener {
    ...
    input = new JTextField();
    convertButton = new JButton("Preved");
    convertButton.addActionListener(this);
    input.addActionListener(this);
    ...
    public void actionPerformed(ActionEvent e) {
        int temp = (int)
        ((Double.parseDouble(input.getText()) - 32) * 5 / 9);
        celLabel.setText(temp + " Celsius");
    }
}
```

Události

- implementace listeneru typicky přes anonymní vnitřní třídu nebo lambda výraz

```
button.addActionListener(e ->
    label.setText("Clicked"));
```

Vlákna

- obsluha událostí a vykreslování GUI
 - **jedno** vlákno (event-dispatching thread)
 - zajišťuje postupné obsluhování událostí
 - každá událost se obslouží až po skončení předchozí obsluhy
 - události nepřerušují vykreslování
- `SwingUtilities.invokeLater(Runnable doRun)`
 - statická metoda
 - provede kód v `doRun.run()` pomocí vlákna obsluhujícího GUI
 - počká se, až budou všechny události obslouženy
 - metoda okamžitě skončí
 - nečeká, až se kód provede
 - používá se, pokud program upravuje GUI
- `SwingUtilities.invokeAndWait(Runnable doRun)`
 - jako `invokeLater()`, ale skončí, až se kód provede

Actions

- oddělení komponenty a její funkce
 - pro tlačítka, menu,....
 - stejná akce přiřazená k více komponentám
- **Action**
 - interface
 - lze nastavit
 - zobrazovaný text
 - ikonu
 - popis
 - klávesovou zkratku
 - action listener
 - ...
- **AbstractAction**
 - třída implementující interface **Action**
 - typicky se od ní dědí

Swing

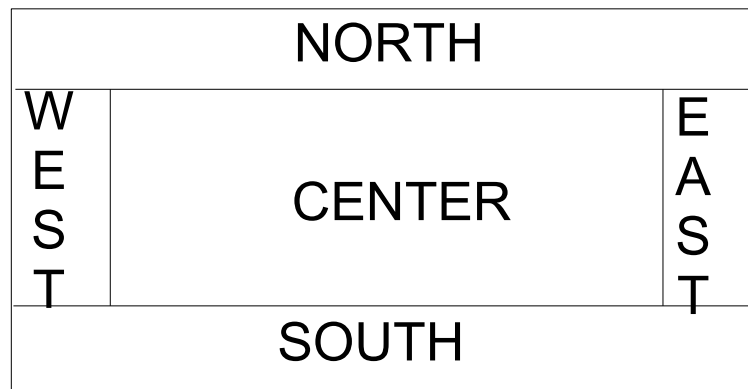
Layouts

Přehled

- vlastnost kontejneru
 - komponenty GUI se umísťují do kontejnerů (frame, dialog, panel,...)
- určuje velikost a umístění komponent v kontejneru
- určuje změnu velikosti a umístění při změně velikosti kontejneru
- implementuje interface **java.awt.LayoutManager**
- `java.awt.Container`
 - `void setLayout(LayoutManager m)`
 - `LayoutManager getLayout()`

BorderLayout

- implicitní layout pro *content pane*
- 5 regionů - north, south, east, west, center

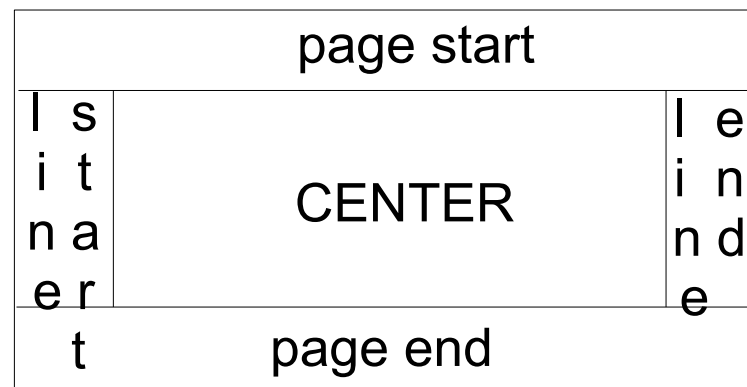


```
JPanel p = new JPanel();  
p.setLayout(new BorderLayout());  
p.add(new Button("Okay"), BorderLayout.SOUTH);
```

```
// následující řádky dělají totéž  
p.add(new Button("Cancel"));  
p.add(new Button("Cancel"), BorderLayout.CENTER);
```

BorderLayout

- relativní určování regionu
 - page start, page end, line start, line end
 - závisí na nastavení **ComponentOrientation**
 - java.awt.Component
 - setComponentOrientation
 - getComponentOrientation
 - java.awt.ComponentOrientation
 - orientace komponent v závislosti na jazyku
 - při ComponentOrientation.LEFT_TO_RIGHT se shoduje s north, south, west, east



BorderLayout

- implicitně – žádné mezery mezi komponenty kontaineru
- konstruktor
 - `BorderLayout(int horizontalGap, int verticalGap)`
- metody
 - `void setVgap(int)`
 - `void setHgap(int)`

FlowLayout

- implicitní layout pro JPanel
- dává komponenty do kontejneru za sebe do řady
- pokud už není místo, začne další řadu

```
contentPane.setLayout(new FlowLayout());
```

```
contentPane.add(new JButton("Button 1"));
```

```
contentPane.add(new JButton("Button 2"));
```

```
contentPane.add(new JButton("Button 3"));
```

```
contentPane.add(new JButton("Long-Named Button 4"));
```

```
contentPane.add(new JButton("5"));
```

FlowLayout

- konstruktory
 - `FlowLayout()`
 - `FlowLayout(int alignment)`
 - `FlowLayout(int alignment, int horizontalGap, int verticalGap)`
 - alignment – kam se budou komponenty zarovnávat
 - `FlowLayout.LEADING` – dopředu
 - `FlowLayout.CENTER` – na střed
 - `FlowLayout.TRAILING` – na konec
 - opět záleží na `ComponentOrientation`
 - Gap – mezery mezi komponentami

GridLayout

- komponenty umísťuje do tabulky
- každá komponenta zabírá celou buňku tabulky
- všetky buňky majú stejnou veľkosť
- špecifikuje sa počet sloupců a řádků
 - GridLayout(int rows, int columns)
 - jeden z rozměrů může být 0
 - oba zároveň ne
 - daný rozměr není špecifikován
 - určí se podle počtu vložených komponent
- pořadí opět podle ComponentOrientation

```
pane.setLayout(new GridLayout(0,2));
```

```
pane.add(new JButton("Button 1"));
```

```
pane.add(new JButton("Button 2"));
```

```
...
```

CardLayout

- umožňuje, aby více komponent (typicky JPanel) sdílelo jedno místo
- v jednu chvíli zobrazena jen jedna komponenta

```
JPanel cards;  
final static String PANEL1 = "Panel1";  
final static String PANEL2 = "Panel2";  
  
JPanel card1 = new JPanel();  
...  
JPanel card2 = new JPanel();  
...  
cards = new JPanel(new CardLayout());  
cards.add(card1, PANEL1);  
cards.add(card2, PANEL2);
```

CardLayout

- přepínání zobrazení

```
CardLayout cl = (CardLayout) (cards.getLayout());  
cl.show(cards, PANEL2);
```

- další metody pro přepínání zobrazení

```
void first(Container)  
void next(Container)  
void previous(Container)  
void last(Container)
```

- **JTabbedPane**

- podobné CardLayout
- není to layout
- je to samostatná komponenty
- sama zobrazuje záložky

GridBagLayout

- nejsložitější a zároveň nejflexibilnější layout
- umísťuje komponenty do tabulky
- jedna komponenta může být přes více řádků nebo sloupců
- sloupce a řádky nemusejí mít stejnou velikost
- umístění komponent se určuje pomocí **GridBagConstraints**

```
JPanel pane = new JPanel(new GridBagLayout());  
GridBagConstraints c = new GridBagConstraints();
```

```
// pro každou komponentu  
//...vytvořit komponentu...  
//...nastavit constraint...  
pane.add(theComponent, c);
```

GridBagConstraints: atributy

- gridx, gridy
 - sloupec a řada levé horní části komponenty
 - sloupec nejvíc vlevo gridx = 0
 - řádek nahoře gridy = 0
 - hodnota GridBagConstraints.RELATIVE (implicitní)
 - komponenta bude umístěna vpravo od předchozí (gridx) nebo pod předchozí (gridy)
 - doporučení – vždy specifikovat hodnoty pro každou komponentu

GridBagConstraints: atributy

- gridwidth, gridheight
 - počet sloupců (gridwidth) a řádků (gridheight), které komponenta zabere
 - implicitní hodnota 1
 - hodnota GridBagConstraints.REMAINDER
 - komponenta bude poslední ve sloupci (gridwidth) nebo řádku (gridheight)
 - hodnota GridBagConstraints.RELATIVE
 - komponenta bude vedle předchozí

GridBagConstraints: atributy

- fill
 - určuje změnu velikosti komponenty, pokud plocha pro zobrazení je větší než komponenta
 - hodnoty (konstanty na GridBagConstraints)
 - NONE (implicitní)
 - velikost komponenty se nemění
 - HORIZONTAL
 - roztáhne komponentu na šířku
 - výšku nemění
 - VERTICAL
 - roztáhne komponentu na výšku
 - šířku nemění
 - BOTH
 - roztáhne komponentu na celou plochu pro zobrazení

GridBagConstraints: atributy

- ipadx, ipady
 - vnitřní doplnění
 - implicitně 0
 - kolik přidat k minimální velikosti komponenty
 - šířka komponenty bude minimálně $2 \cdot \text{ipadx}$
 - doplnění je přidáno na obě strany
 - obdobně výška komponenty bude minimálně $2 \cdot \text{ipady}$
- insets
 - vnější doplnění
 - minimální místo mezi komponentou a hranicí plochy pro zobrazení
 - implicitně žádné
 - hodnota – objekt `java.awt.Insets`
 - konstruktor `Insets(nahore, vlevo, dole, vpravo)`

GridBagConstraints: atributy

- anchor
 - kam umístit komponentu, pokud je menší než velikost plochy pro zobrazení
 - hodnoty – konstanty na GridBagConstraints

```
-----  
| FIRST_LINE_START      PAGE_START      FIRST_LINE_END |  
|  
| LINE_START            CENTER           LINE_END    |  
|  
| LAST_LINE_START      PAGE_END         LAST_LINE_END |  
-----
```

GridBagConstraints: atributy

- weightx, weighty
 - hodnoty mezi 0.0 a 1.0
 - implicitně 0
 - určuje, jak se rozdělí prostor mezi řádky, resp. sloupce
 - pokud všechny $\text{weight}(x|y) = 0$ v řádku resp. sloupci, jsou komponenty soustředěny na střed kontajneru
 - důležité hlavně při změně velikosti kontejneru

GridBagLayout: příklad

- **Button1, Button2, Button3:** weightx = 1.0
- **Button4:** weightx = 1.0, gridwidth = GridBagConstraints.REMAINDER
- **Button5:** gridwidth = GridBagConstraints.REMAINDER
- **Button6:** gridwidth = GridBagConstraints.RELATIVE
- **Button7:** gridwidth = GridBagConstraints.REMAINDER
- **Button8:** gridheight = 2, weighty = 1.0
- **Button9, Button 10:** gridwidth = GridBagConstraints.REMAINDER



GridBagLayout: příklad

Všechna tlačítka: ipadx = 0, fill = GridBagConstraints.HORIZONTAL

Button 1: ipady = 0, weightx = 0.5, weighty = 0.0, gridwidth = 1, anchor = GridBagConstraints.CENTER, insets = new Insets(0,0,0,0), gridx = 0, gridy = 0

Button 2: weightx = 0.5, gridx = 1, gridy = 0

Button 3: weightx = 0.5, gridx = 2, gridy = 0

Button 4: ipady = 40, weightx = 0.0, gridwidth = 3, gridx = 0, gridy = 1

Button 5: ipady = 0, weightx = 0.0, weighty = 1.0, anchor = GridBagConstraints.SOUTH, insets = new Insets(10,0,0,0), gridwidth = 2, gridx = 1, gridy = 2



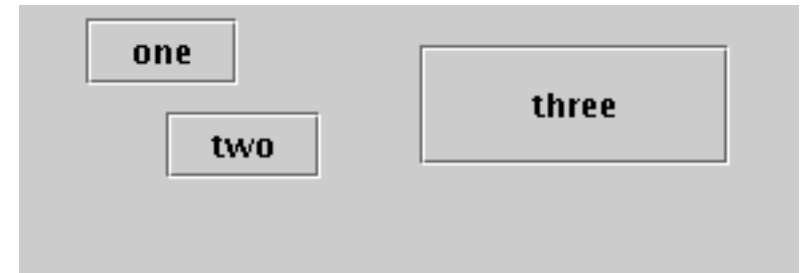
SpringLayout

- přidán od JDK 1.4
- velmi flexibilní
 - dokáže emulovat většinu předchozích layoutů
- nízkoúrovňový
 - určen pro IDE nástroje
 - není určen pro přímé použití
 - nicméně je to možné

Žádný layout

- umístění komponent na absolutní pozice

```
pane.setLayout(null);
JButton b1 = new JButton("one");
JButton b2 = new JButton("two");
JButton b3 = new JButton("three");
pane.add(b1);
pane.add(b2);
pane.add(b3);
Insets insets = pane.getInsets();
Dimension size = b1.getPreferredSize();
b1.setBounds(25 + insets.left, 5 + insets.top,
             size.width, size.height);
size = b2.getPreferredSize();
b2.setBounds(55 + insets.left, 40 + insets.top,
             size.width, size.height);
size = b3.getPreferredSize();
b3.setBounds(150 + insets.left, 15 + insets.top,
             size.width + 50, size.height + 20);
```



Vlastní layout

- implementovat interface **java.awt.LayoutManager**
- metody
 - `void addLayoutComponent (String, Component)`
 - volaná kontejnerem v metodě **add**
 - přidává komponentu do layoutu
 - asociuje komponentu se stringem
 - `void removeLayoutComponent (Component)`
 - volaná kontejnerem v metodách **remove** a **removeAll**
 - `Dimension preferredLayoutSize (Container)`
 - ideální velikost kontejneru
 - `Dimension minimumLayoutSize (Container)`
 - minimální velikost kontejneru
 - `void layoutContainer (Container)`
 - volá se při prvním zobrazení a při každé změně velikosti kontejneru

Swing

Přehled komponent

Label

- třída JLabel
- pro zobrazení
 - krátkého textu
 - obrázku
 - obojího

Tlačítka

- mnoho druhů tlačítek (buttons)
- všechna dědí od **AbstractButton**
 - normální tlačítko (JButton)
 - "klikací" tlačítko
 - toggle button (JToggleButton)
 - přepínací tlačítko (dva stavy)
 - check box (JCheckBox)
 - zaškrťávací tlačítko
 - radio button (JRadioButton)
 - typicky ve skupině, stisknuté jen jedno
- událost – `ActionEvent`
- listener – `ActionListener`

Skupiny tlačítek

- skupina tlačítek – stisknuto jen jedno
 - typicky pro RadioButton
- třída ButtonGroup

```
JRadioButton buttons[] = new JRadioButton [4];
```

```
for (int i=0; i<4; i++) {  
    pane.add(buttons[i] =  
                new JRadioButton("Button "+(i+1)));  
}
```

```
ButtonGroup bg = new ButtonGroup();
```

```
for (int i=0; i<4; i++) {  
    bg.add(buttons[i]);  
}
```


Ikony

- interface **Icon**
 - lze použít na label, tlačítko, menu,...
- třída **ImageIcon**
 - implementuje **Icon**
 - ikona vytvořená z obrázku
 - ze souboru, URL,...
 - jpg, png, gif

```
new JButton("Click", new ImageIcon("ystar.png"));
```

```
new JLabel("Hello", new ImageIcon("gstar.png"),  
SwingConstants.CENTER);
```

Tool tips

- "malá" nápověda
 - "bublina" s textem
 - objeví se při delším podržení myši nad komponentou
- lze nastavit všemu, co dědí od **JComponent**

```
button.setToolTipText("Click here");
```

Textové pole

- třída JTextField
- editace jednoho řádku textu
- po stisknutí klávesy ENTER → `ActionEvent`
- metody
 - `String getText()`
 - vrátí napsaný text
 - `void setText(String text)`
 - nastaví text
- třída JTextArea
 - editace více řádků
 - pro zobrazení posuvníků (scrollbars) nutno vložit do **JScrollPane**
 - `new JScrollPane(new JTextArea())`
 - `JScrollPane` funguje na vše co implementuje `Scrollable`

Combo box

- třída JComboBox
- tlačítko s možnostmi na výběr
 - lze i editovat – `setEditable(boolean b)`
- při změně – `ActionEvent`

```
String[] list = { "aaaa", "bbbb", ... };  
JComboBox cb = new JComboBox(list);  
cb.setEditable(true);
```

List box

- třída JList
- seznam položek
- umožňuje vybírat položky
 - jednu i více naráz (`setSelectionMode(int mode)`)
- metody
 - `int getSelectedIndex()`
 - `Object getSelectedValue()`
- `ListSelectionEvent`
- `ListSelectionListener`

Menu

```
frame.setJMenuBar(createMenu());  
.....  
private static JMenuBar createMenu() {  
    JMenuBar mb = new JMenuBar();  
    JMenu menu = new JMenu("File");  
    JMenuItem item = new JMenuItem("Quit");  
    menu.add(item);  
    mb.add(menu);  
  
    menu = new JMenu("Help");  
    item = new JMenuItem("Content");  
    menu.add(item);  
    menu.add(new JSeparator());  
    .....  
    mb.add(menu);  
  
    return mb;  
}
```

Stromy

- javax.swing.JTree
- zobrazení hierarchických dat
- JTree neobsahuje data přímo
 - pouze data zobrazuje
 - data obsahuje *model (model-view concept)*
- obecně
 - všechny složitější komponenty mají model
 - JTree, JTable, JList, JButton, ...
 - model určuje, jak jsou zobrazovaná data uchovávána a získávána
 - jedna komponenta může mít více modelů
 - př: JList
 - ListModel – spravuje obsah seznamu
 - ListSelectionModel – aktuální výběr v seznamu

JTree: statický obsah

```
DefaultMutableTreeNode top =
    new DefaultMutableTreeNode("Root");
createNodes(top);
tree = new JTree(top);
...
private void createNodes(DefaultMutableTreeNode top) {
    DefaultMutableTreeNode node = null;
    DefaultMutableTreeNode leaf = null;

    node = new DefaultMutableTreeNode("Node1");
    top.add(node);

    leaf = new DefaultMutableTreeNode("Leaf1");
    node.add(leaf);
    leaf = new DefaultMutableTreeNode("Leaf2");
    node.add(leaf);

    node = new DefaultMutableTreeNode("Node2");
    top.add(node);
}
```


JTree: dynamické změny

```
rootNode = new DefaultMutableTreeNode("Root Node");
treeModel = new DefaultTreeModel(rootNode);
treeModel.addTreeModelListener(new MyTreeModelListener());
tree = new JTree(treeModel);
tree.setEditable(true);
tree.getSelectionModel().setSelectionMode
    (TreeSelectionMode.SINGLE_TREE_SELECTION);
...
class MyTreeModelListener implements TreeModelListener {
    public void treeNodesChanged(TreeModelEvent e) {
    }
    public void treeNodesInserted(TreeModelEvent e) {
    }
    public void treeNodesRemoved(TreeModelEvent e) {
    }
    public void treeStructureChanged(TreeModelEvent e) {
    }
}
```

JTree: dynamické změny

```
public DefaultMutableTreeNode addObject(DefaultMutableTreeNode
    parent, Object child, boolean shouldBeVisible) {

    DefaultMutableTreeNode childNode =
        new DefaultMutableTreeNode(child);

    ...
    treeModel.insertNodeInto(childNode, parent,
        parent.getChildCount());

    if (shouldBeVisible) {
        tree.scrollPathToVisible(new TreePath(childNode.getPath()));
    }
    return childNode;
}
```

JTree: vlastní model

- *model-view*
 - Model
 - popisuje data (např. DefaultTreeModel)
 - View
 - Definuje, jak se mají data zobrazit (JTree)
- implicitní model – DefaultTreeModel
- pokud nevyhovuje → vlastní model
 - př. implicitně uzly ve stromu jsou DefaultMutableTreeNode a implementují TreeNode interface
 - vlastní model může přidávat uzly zcela jiného typu
- model musí implementovat TreeModel interface

TreeModel

```
void addTreeModelListener(TreeModelListener l);  
  
Object getChild(Object parent, int index);  
  
int getChildCount(Object parent);  
  
int getIndexOfChild(Object parent, Object child);  
  
Object getRoot();  
  
boolean isLeaf(Object node);  
  
void removeTreeModelListener(TreeModelListener l);  
  
void valueForPathChanged(TreePath path, Object;  
    newValue);
```

Ikony ve stromu

- TreeCellRenderer
 - interface
- setCellRenderer(TreeCellRenderer r)
 - metoda na JTree

```
class MyRenderer extends DefaultTreeCellRenderer {
    public Component
    getTreeCellRendererComponent (JTree
        tree, Object value, boolean sel, boolean expanded,
        boolean leaf, int row, boolean hasFocus) {

        super.getTreeCellRendererComponent(tree, value,
            sel, expanded, leaf, row, hasFocus);
        if (....) {
            setIcon(someIcon);
            setToolTipText("....");
        } else {.....}
        return this;
    }
}
```

Ikony ve stromu

```
ImageIcon leafIcon = createImageIcon("../");

if (leafIcon != null) {
    DefaultTreeCellRenderer renderer =
        new DefaultTreeCellRenderer();

    renderer.setLeafIcon(leafIcon);
    tree.setCellRenderer(renderer);
}
```

JTable

- tabulka
- konstruktory (některé)
 - `JTable(Object[][] rowData, Object[] columnNames)`
 - `JTable(TableModel dm)`

First Name	Last Name	Sport	# of Years	Vegetarian
Kathy	Smith	Snowboarding	5	false
John	Doe	Rowing	3	true
Sue	Black	Knitting	2	false
Jane	White	Speed reading	20	true
Joe	Brown	Pool	10	false

TableModel

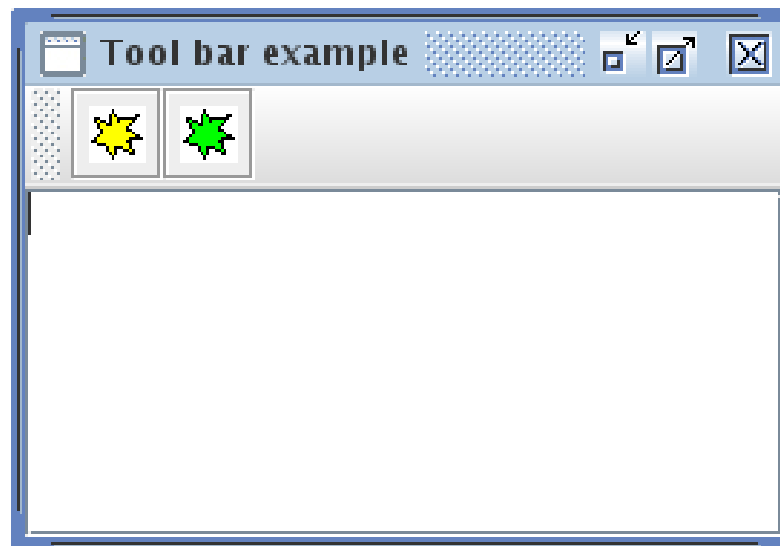
- void addTableModelListener(TableModelListener l)
- Class<?> getColumnClass(int columnIndex)
- int getColumnCount()
- String getColumnName(int columnIndex)
- int getRowCount()
- Object getValueAt(int rowIndex, int columnIndex)
- boolean isCellEditable(int rowIndex, int columnIndex)
- void removeTableModelListener(TableModelListener l)
- void setValueAt(Object aValue, int rowIndex,
int columnIndex)

AbstractTableModel

- předpřipravená implementace modelu
- stačí naimplementovat pouze metody
 - `public int getColumnCount()`
 - `public int getRowCount()`
 - `public Object getValueAt(int row, int col)`

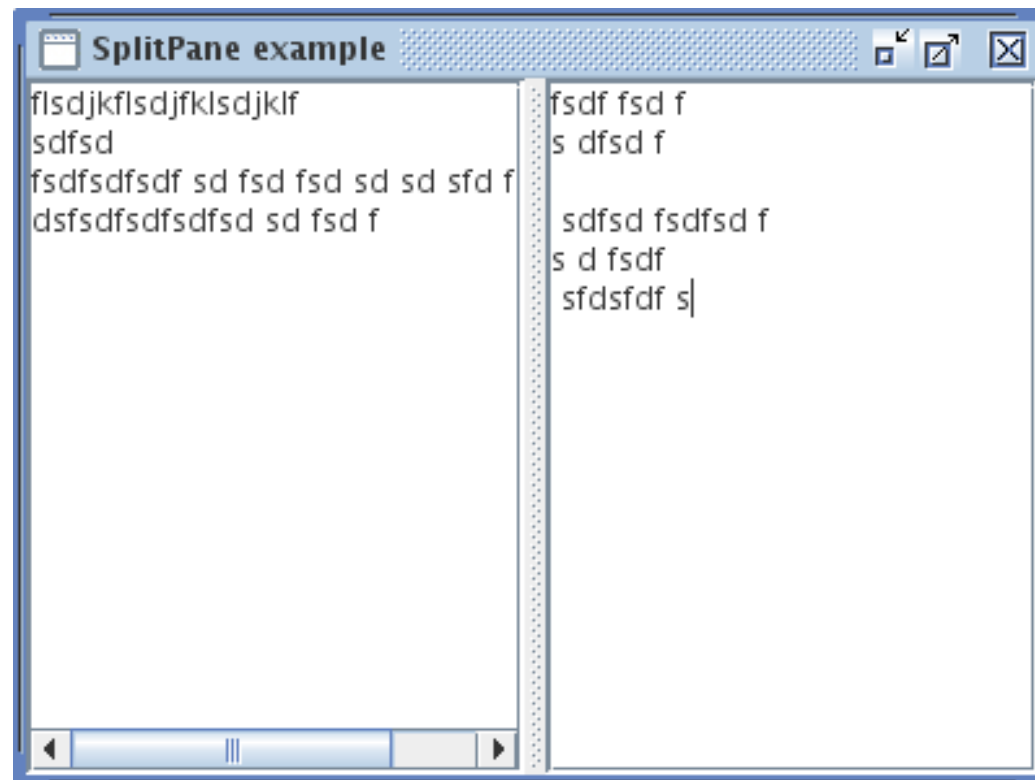
JToolBar

- lišta s tlačítky
- lze přetáhnout na jiné místo
- lze i „vytrhnout“



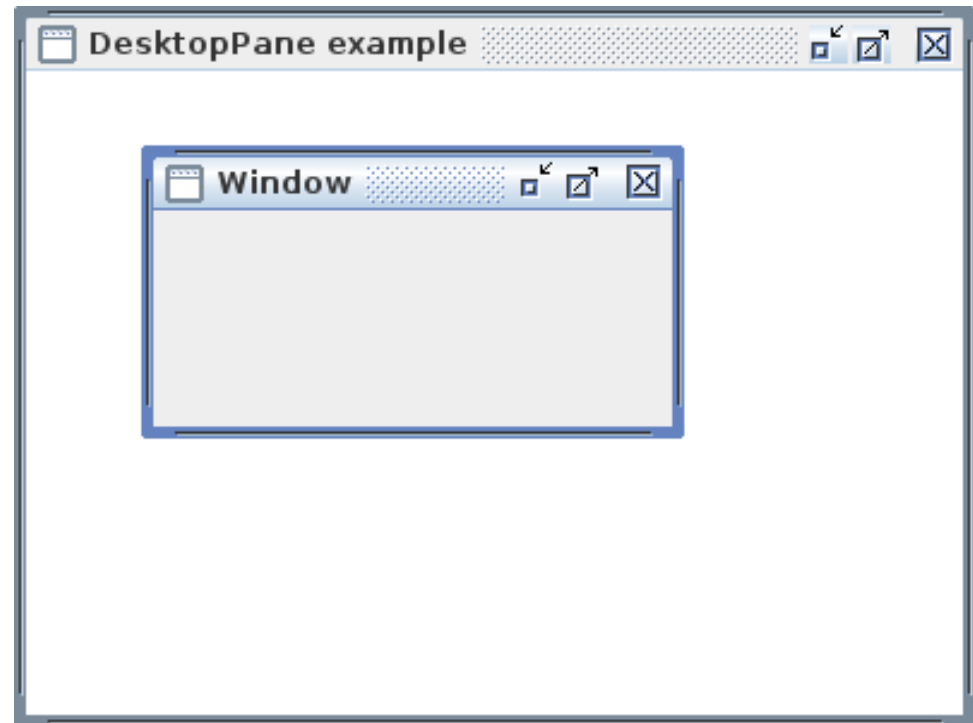
JSplitPane

- zobrazení 2 komponent
 - vedle sebe
 - pod sebe
- oddělovač mezi komponentami lze posouvat



JDesktopPane

- „okna v okně“
- JDesktopPane
 - „plocha“
- JInternalFrame
 - vnitřní okno



Swing

Dialogy

Přehled

- *JDialog*
- *dialog* = okno podobně jako *frame*
- *dialog* je závislý na *frame*
- *dialog* je modální
 - pokud je zobrazen, je zablokovaný vstup do ostatních oken programu
 - lze vytvářet i nemoďální dialogy
- práce s dialogem – stejná jako u *frame*
- *JOptionPane*
 - komponenta usnadňující vytváření standardních dialogů
 - předpřipravené dialogy

JOptionPane



```
//default title and icon  
JOptionPane.showMessageDialog(frame,  
    "Eggs aren't supposed to be green.");
```



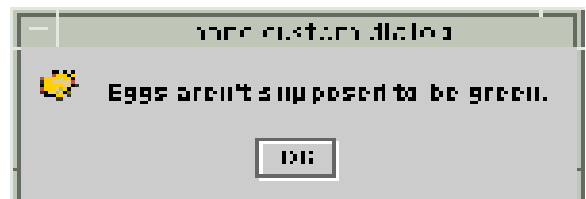
```
//custom title, warning icon  
JOptionPane.showMessageDialog(frame,  
    "Eggs aren't supposed to be green.",  
    "Inane warning",  
    JOptionPane.WARNING_MESSAGE);
```



```
//custom title, error icon  
JOptionPane.showMessageDialog(frame,  
    "Eggs aren't supposed to be green.",  
    "Inane error",  
    JOptionPane.ERROR_MESSAGE);
```



```
//custom title, no icon  
JOptionPane.showMessageDialog(frame,  
    "Eggs aren't supposed to be green.",  
    "A plain message",  
    JOptionPane.PLAIN_MESSAGE);
```



```
//custom title, custom icon  
JOptionPane.showMessageDialog(frame,  
    "Eggs aren't supposed to be green.",  
    "Inane custom dialog",  
    JOptionPane.INFORMATION_MESSAGE,  
    icon);
```

JOptionPane

- předpřipravené dialogy
 - lze je konfigurovat
 - defaultní volba
- statické metody vytvářející dialogy (vždy několik variant jedné metody)
 - showMessageDialog()
 - informační dialog
 - showInputDialog()
 - dialog vyžadující vstup od uživatele
 - vrací String
 - showConfirmDialog()
 - dotazovací dialog (Ano/Ne/Zrušit)
 - vrací int
 - showOptionDialog()
 - Výběr z více možností (Ano-Ne-Možná-Zrušit)

JOptionPane

- lze používat přímo
 - vytvořit JOptionPane objekt
 - několik konstruktorů
 - vložit vytvořený objekt do dialogu

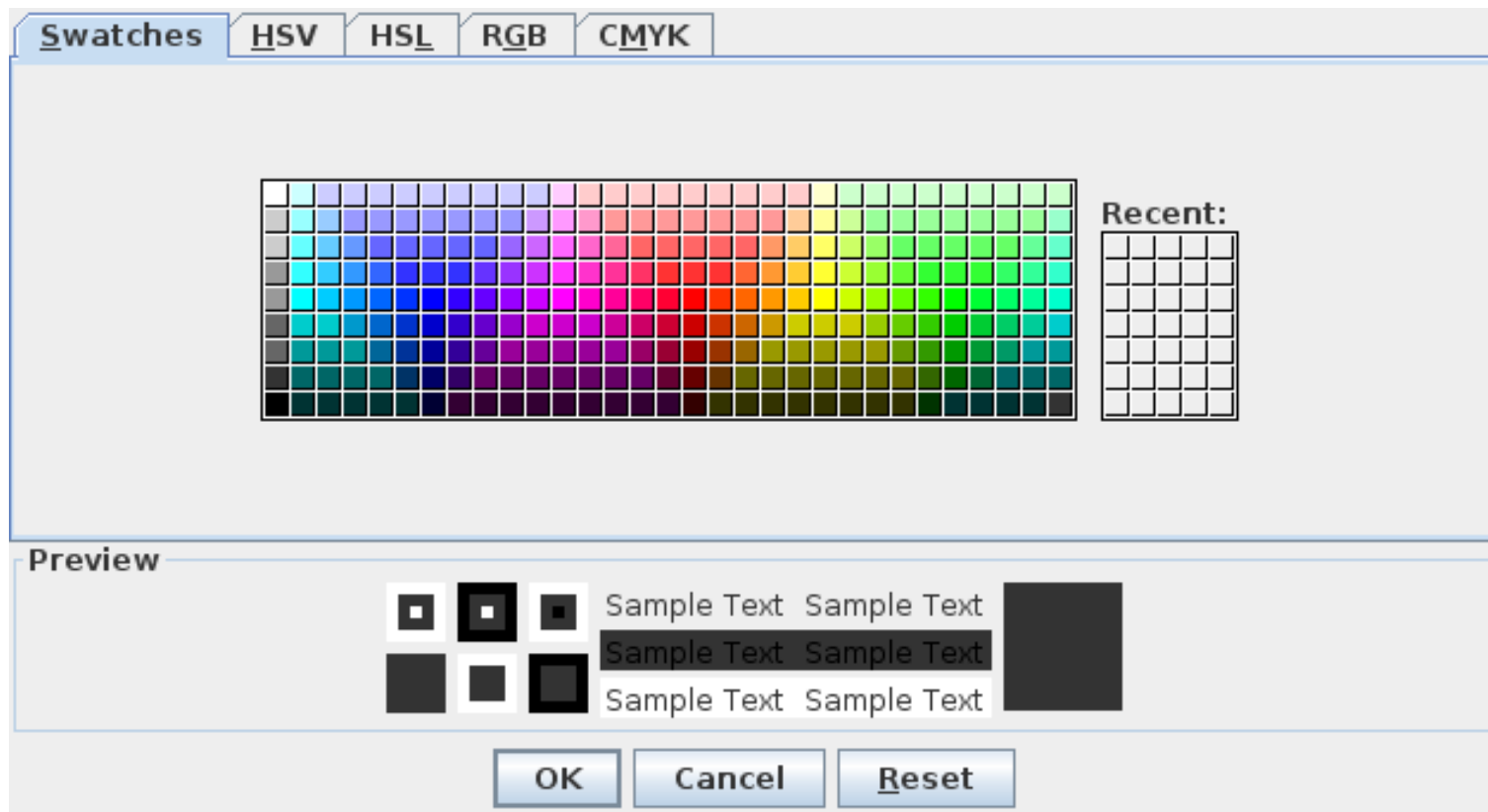
JFileChooser

- standardní dialog pro výběr souborů

```
JFileChooser chooser = new JFileChooser();
chooser.setDialogType(JFileChooser.OPEN_DIALOG)
FileNameExtensionFilter filter =
    new FileNameExtensionFilter(
        "Obrazky", "jpg", "gif");
chooser.setFileFilter(filter);
int returnVal = chooser.showOpenDialog(parent);
if (returnVal == JFileChooser.APPROVE_OPTION) {
    System.out.println("Vybraný soubor: " +
        chooser.getSelectedFile().getName());
}
```

JColorChooser

- výběr barvy
- lze použít
 - jako dialog
 - jako komponenta





Verze prezentace AJ05.cz.2020.01

Tato prezentace podléhá licenci [Creative Commons Uveďte autora-Neužívejte komerčně 4.0 Mezinárodní License](https://creativecommons.org/licenses/by-nc/4.0/).