JAVA

GUI in the std library

Overview

- Java GUI
  - Java 1.0 (1996) – AWT
    - using native GUI components
  - Java 1.2 (2000) – Swing
    - GUI completely in Java
  - JavaFX (2007)
    - new technology
    - running on the Java VM
    - but own language
      - declarative
    - intended as a competitor to Flash
    - failed
  - JavaFX 2.0 (2011)
    - only API (own language abandoned)
  - since JDK 7 update 6 a part of std JDK (JavaFX 2.2)
  - Java 8 – JavaFX 8
  - Java 11 – JavaFX decoupled from JDK

JAVA

Swing

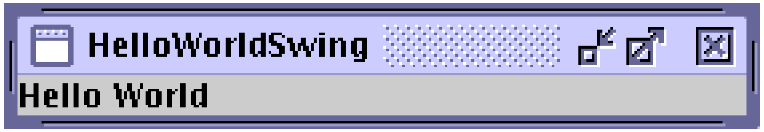
Swing

- packages
  - javax.swing....
  - uses also classes from java.awt...
  - many classes extends classes from java.awt...
- AWT
  - still present
    - compatibility reasons
  - uses the event model
- fully implemented in Java
  - the same look-and-feel on all platforms
    - look-and-feel can be modified – adjusted to a platform
- support for 2D graphics, printing, drag-and-drop, localization, ...

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();
      }
    });
  }
}
```



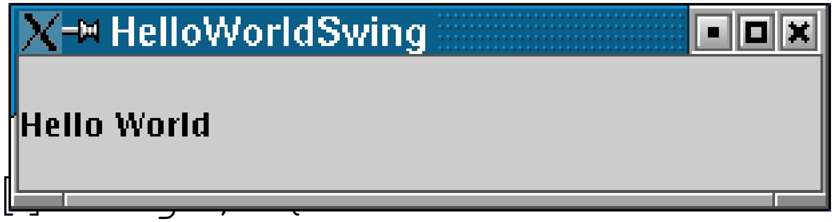
HelloWorldSwing

Hello World

Hello World (2)

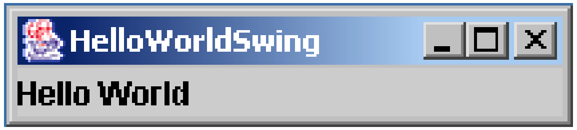
```
import javax.swing.*;

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[
    javax.swing.SwingUtilities.invokeLater(new Runnable()
  {
      public void run() {
        createAndShowGUI();
      }
    });
  }
}
```



HelloWorldSwing

Hello World



HelloWorldSwing

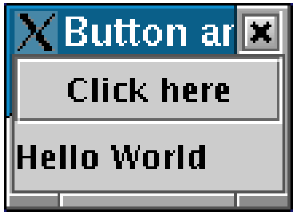
Hello World

Layout

```
// example: 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
  - defines size and placement of components in a container
  - defines changes of size and placement when container size is changed
  - implements the interface **java.awt.LayoutManager**

Panel and borders

```
// example: 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" container
  - container can be inserted to other containers
- border
  - how to paint borders of components (JComponent)

Look & Feel

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

- defines look and behavior of GUI
- L&F included in JDK
  - crossplatform (Metal) – the same GUI on all platforms
  - Windows – similar to the Windows GUI
  - system
    - on Unix – Metal
    - on Windows – Windows
  - Motif
  - GTK+ – since JDK 1.4.2
  - Nimbus – since JDK 6 u10
- own ones can be created

Events

Observer pattern

- GUI is controlled through *events*
  - e.g. click on a button → event
- event processing – *listener*
  - an object registers a *listener* → receives info about events
- many types of events (and of corresponding *listeners*)
  - e.g. button click, window closing, mouse move,...

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

Events

- a single *listener* can be registered for multiple events

```
public class TempConvert implements ActionListener {
  ...
    input = new JTextField();
    convertButton = new JButton("Convert");
    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");
  }
}
```

Events

- listener implementation typically via anonymous inner class or lambda expression

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

Threads

- event processing and GUI painting
  - a **single** thread (event-dipatching thread)
  - ensures subsequent event processing
    - each event is processed after the previous one is finished
    - events do not interrupt painting
- SwingUtilities.invokeLater(Runnable doRun)
  - static method
  - runs code in doRun.run() using the event-processing thread
    - waits until all events are processed
  - the method ends immediately
    - does not wait till the code is run
  - used for GUI modifications
- SwingUtilities.invokeAndWait(Runnable doRun)
  - as invokeLater(), but ends after the code is run

Actions

- separation of a component and its function
  - for buttons, menu,...
  - the same action assigned to several components
- **`Action`**
  - interface
  - can be set
    - displayed text
    - icon
    - description
    - key shortcut
    - action listener
    - ...
- **`AbstractAction`**
  - the class implementing the interface **`Action`**
  - typically this class is extended

Swing

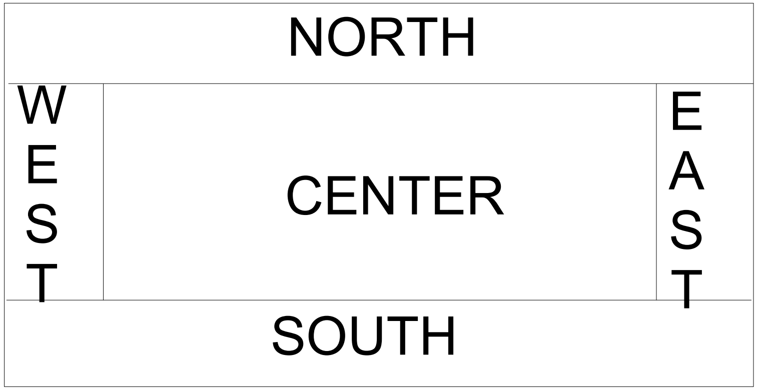
Layouts

Overview

- the container feature
  - components of GUI are placed in a container (frame, dialog, panel,...)
- determines size and placement of components in th container
- determines changes of size and placement when the size of the container is changed
- implements the interface **java.awt.LayoutManager**
- java.awt.Container
  - void setLayout(LayoutManager m)
  - LayoutManager getLayout()

BorderLayout

- default layout for the *content pane*
- 5 regions - north, south, east, west, center

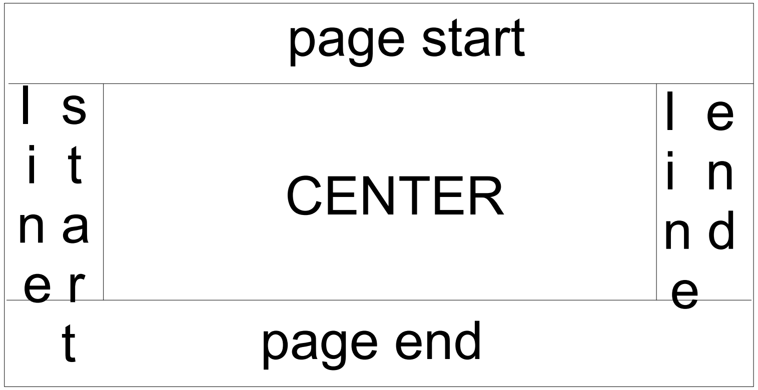


```
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
p.add(new Button("Okay"), BorderLayout.SOUTH);

// following two lines are equivalent
p.add(new Button("Cancel"));
p.add(new Button("Cancel"), BorderLayout.CENTER);
```

BorderLayout

- relative determining the region
  - page start, page end, line start, line end
  - depends on **ComponentOrientation**
    - java.awt.Component
      - setComponentOrientation
      - getComponentOrientation
    - java.awt.ComponentOrientation
      - component orientation related to the used language
  - if ComponentOrientation.LEFT_TO_RIGHT, then it corresponds to north, south, west, east



page start

line start

CENTER

line end

page end

BorderLayout

- default – no gaps between components in the container
- the constructor
  - BorderLayout(int horizontalGap, int verticalGap)
- methods
  - void setVgap(int)
  - void setHgap(int)

FlowLayout

- default layout for JPanel
- arranges components in a directional flow
- if there is no space left in a row, then it starts new row

```
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

- constructors
  - FlowLayout()
  - FlowLayout(int alignment)
  - FlowLayout(int alignment, int horizontalGap, int verticalGap)
    - alignment – alignment of components
      - FlowLayout.LEADING
      - FlowLayout.CENTER
      - FlowLayout.TRAILING
      - depends on the ComponentOrientation
    - Gap – a gap between components

GridLayout

- arranges components in a table
- each component occupies a single cell in the table
- all cells have the same size
- necessary to specify number of columns and rows
  - GridLayout(int rows, int columns)
  - one of the sizes can be 0
    - both cannot
    - the size with 0 is calculated based on the number of inserted components
- ordering of components according to ComponentOrientation

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

pane.add(new JButton("Button 1"));
pane.add(new JButton("Button 2"));
...
```

CardLayout

- allows several components (typically JPanels) occupy the same place
- only one component is visible at a time

```
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

- switching visible components

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

- other methods for switching

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

- **JTabbedPane**
  - similar to CardLayout
  - it is not layout
  - it is a component
  - shows tabs

GridBagLayout

- most complex but most flexible layout
- arranges components in a table
- a single component can occupy several rows and/or columns
- rows and columns can have different sizes

- placing of components determined by **GridBagConstraint**

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

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

GridBagConstraint: atributes

- gridx, gridy
  - column and row of the top left corner of the component
  - the leftmost column gridx = 0
  - the top most row gridy = 0
  - the value GridBagConstraint.RELATIVE (default)
    - the component will be placed on the right side of the previous one (gridx) or below the previous one (gridy)
  - recommendation – always specify particular values for each component

GridBagConstraint: atributes

- gridwidth, gridheight
  - number of columns (gridwidth) and row (gridheight), which the component occupies
  - default value 1
  - hodnota GridBagConstraint.REMAINDER
    - komponenta bude poslední ve sloupci (gridwidth) nebo řádku (gridheight)
  - hodnota GridBagConstraint.RELATIVE
    - komponenta bude vedle předchozí

GridBagConstraint: atributes

- fill
  - defines how to change the component size if the area for the component is bigger than the component
  - values (constants on GridBagConstraint)
    - NONE (default)
      - no changes
    - HORIZONTAL
      - expands the component horizontally
      - no vertical change
    - VERTICAL
      - expands the component vertically
      - no horizontal change
    - BOTH
      - expands the component both horizontally and vertically

GridBagConstraint: atributes

- ipadx, ipady
  - internal padding of the component
  - default 0
  - how much space to add to the minimum size of the component
  - width of the component will be at least 2*ipadx
    - padding will be added to both sides
  - similarly height will be at least 2*ipady
- insets
  - external padding
  - the minimum amount of space between the component and the edges of its display area
  - by default none
  - value – java.awt.Insets
    - the constructor Insets(top, left, bottom, right)

GridBagConstraint: atributes

- anchor
  - where to place the component, when the component is smaller than its display area
  - values – constants on GridBagContraint

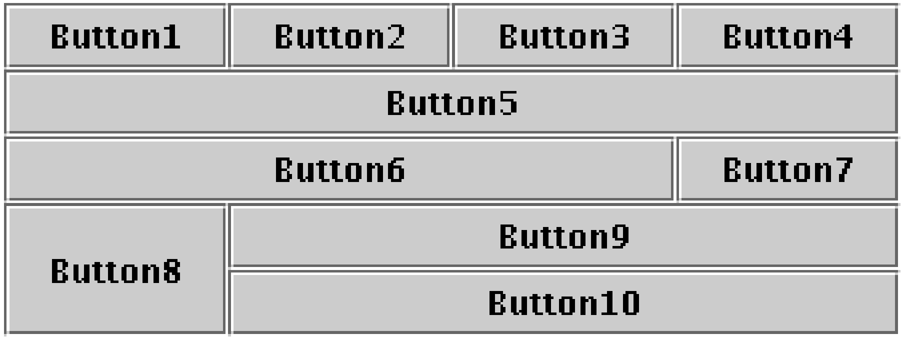
```
 ------------------------------------------------
|FIRST_LINE_START   PAGE_START     FIRST_LINE_END|
|                                                |
|                                                |
|LINE_START           CENTER             LINE_END|
|                                         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

GridBagConstraint: atributes

- weightx, weighty
  - values between 0.0 and 1.0
  - default   0
  - specifies how to distribute extra horizontal/vertical space
  - if all weight(x|y) = 0 in the row resp. column then components are placed in the center of the container
  - important for changes of the container size

GridBagLayout: example

- **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: example

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

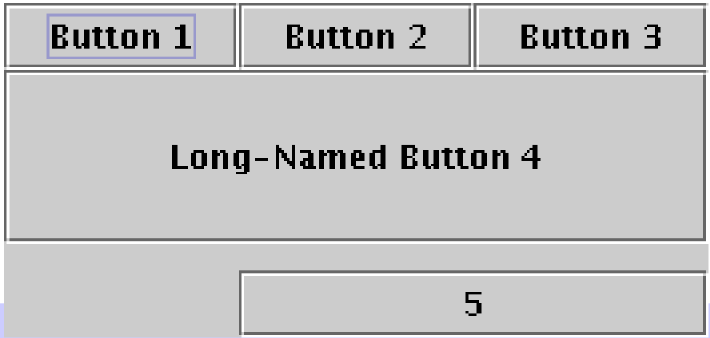
Button 1: ipady = 0, weightx = 0.5, weighty = 0.0, gridwidth = 1, anchor = GridBagConstraints.CENTER, insets = new Insets(0,0,0,0), gridx = 0, gridy = 0

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



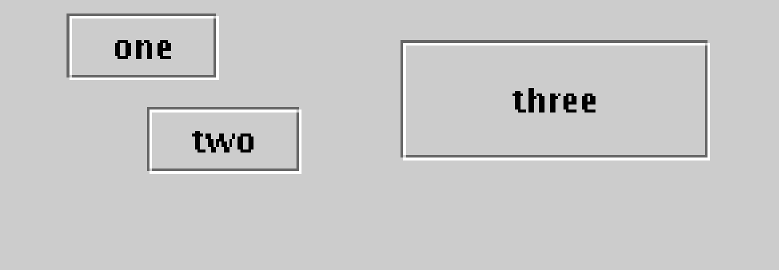
SpringLayout

- since JDK 1.4
- very flexibile
  - can emulate most of the previous layout
- low-level
  - intended for IDEs
  - not intended for direct usage
    - but it is possible

no layout

- placement of components to fixed positions

```
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);
```



Own layout

- implementing the interface **java.awt.LayoutManager**
- methods
  - `void addLayoutComponent(String, Component)`
    - called by the container in the method **add**
    - adds components to the layout
    - associates the component with a string
  - `void removeLayoutComponent(Component)`
    - called by the container in the methods **remove** a **removeAll**
  - `Dimension preferredLayoutSize(Container)`
    - an ideal size of the container
  - `Dimension minimumLayoutSize(Container)`
    - a minimal size of the container
  - `void layoutContainer(Container)`
    - called when firstly shown and after each change of the size of the container

Swing

Component overview

Label

- class JLabel
- for displaying
  - short text
  - image
  - both

Buttons

- many kinds of buttons
- all of them extends **AbstractButton**
  - regular button (JButton)
    - "click" button
  - toggle button (JToggleButton)
    - two-state button (on/off)
  - check box (JCheckBox)
    - selected / deselected box
  - radio button (JRadioButton)
    - typically only one button in a group can be selected
- event – ActionEvent
- listener – ActionListener

Groups of buttons

- a group of buttons – selected can be only one button
  - typically for radio buttons
- the ButtonGroup class

```
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]);
}
```

Icons

- the interface **Icon**
  - can be used with labels, buttons, menus,...
- the class **ImageIcon**
  - implements **Icon**
  - an icon created from an image
    - loaded from file, URL,...
  - jpg, png, gif

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

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

Tool tips

- "small" help
  - a "bubble" with a text
  - displays when the cursor lingers over the component
- can be set to components, which extends **JComponent**

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

Text fileds

- the class JTextField
- an editable single line of text
- after the ENTER key is pressed → ActionEvent
- methods
  - String getText()
    - returns the contained text
  - void setText(String text)
    - sets the text

- the class JTextArea
  - a multi-line editable area
  - have to be inserted to the **JScrollPane** in order to show scrollbars
    - new JScrollPane(new JTextArea)
    - JScrollPane works with anything that implements Scrollable

Combo box

- the class JComboBox
- a button with selection of choices
  - can be edited – setEditable(boolean b)
- generates the ActionEvent when changed

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

List box

- the class JList
- a list of items
- items can be selected
  - a single one or several of them
    - setSelectionMode(int mode)
- methods
  - 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;
  }
```

Trees

- javax.swing.JTree
- displaying hierarchical data
- JTree does not hold data directly
  - only displays data
  - data are hold by a *model (model-view concept)*
- in general
  - all more complex components have a model
    - JTree, JTable, JList, JButton, ...
  - the model determines how the data are stored and retrieved
  - a single component can have multiple models
    - e.g. JList
      - ListModel – holds a content of the list
      - ListSelectionModel – manages current selection

JTree: static content

```
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: dynamic changes

```
rootNode = new DefaultMutableTreeNode("Root Node");
treeModel = new DefaultTreeModel(rootNode);
treeModel.addTreeModelListener(new MyTreeModelListener());
tree = new JTree(treeModel);
tree.setEditable(true);
tree.getSelectionModel().setSelectionMode
        (TreeSelectionModel.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: dynamic changes

```
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: own model

- *model-view*
  - Model
    - describes data (e.g. DefaultTreeModel)
  - View
    - defines how to display data (JTree)
- default model – DefaultTreeModel
- if not suitable → own model
  - e.g., by default, nodes in the tree are DefaultMutableTreeNode and implements the TreeNode interface
    - own model can have nodes of a completely different type
- the model must implement 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);
```

Icons in JTree

- TreeCellRenderer
  - interface
- setCellRenderer(TreeCellRenderer r)
  - method of 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;
```

Icons in JTree

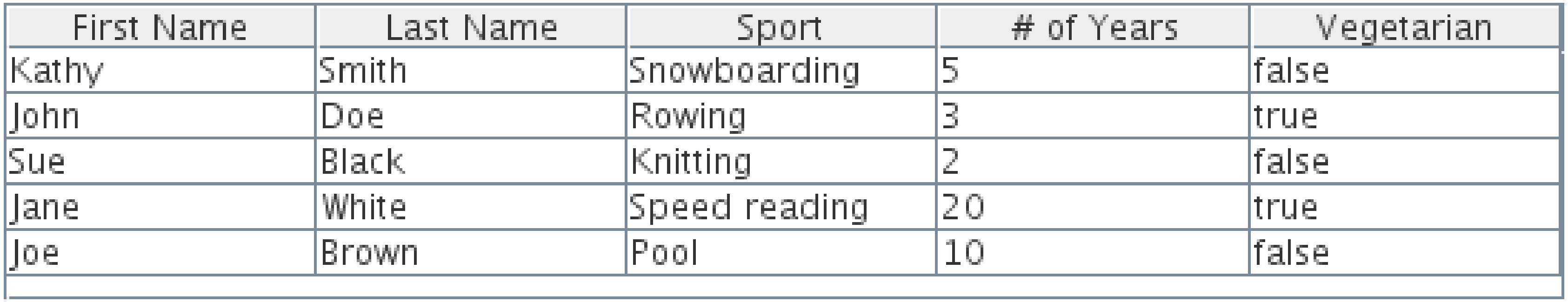
```
ImageIcon leafIcon = createImageIcon("..");

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

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

JTable

- table
- constructors (some of them)
  - 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

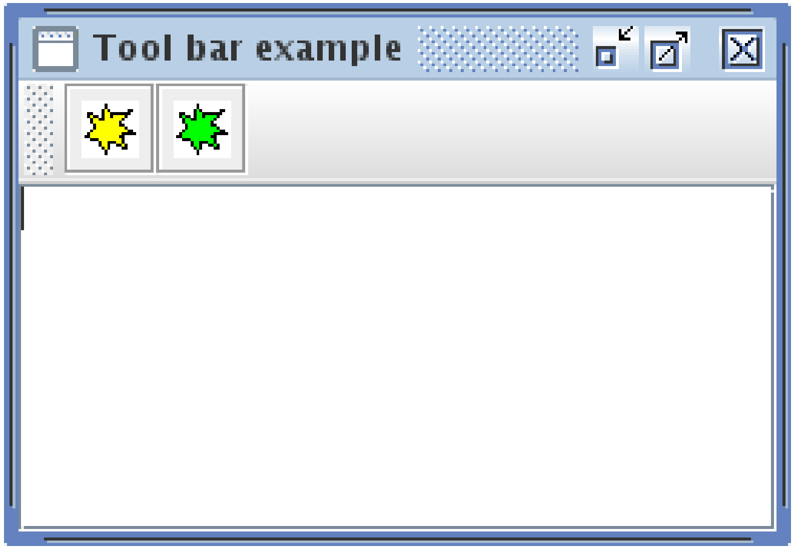
- void addTableModelListener(TableModelListener l)
- Class<?> getColumnClass(int columnIndex)
- int getColumnCount()
- String getColumnName(int columnIndex)
- int getRowCount()
- Object getValueAt(int rowIndex, int columnIndex)
- boolean isCellEditable(int rowIndex, int columnIndex)
- void removeTableModelListener(TableModelListener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

AbstractTableModel

- prepared implementation of a model
- only the following methods have to be implemented
  - public int getColumnCount()
  - public int getRowCount()
  - public Object getValueAt(int row, int col)

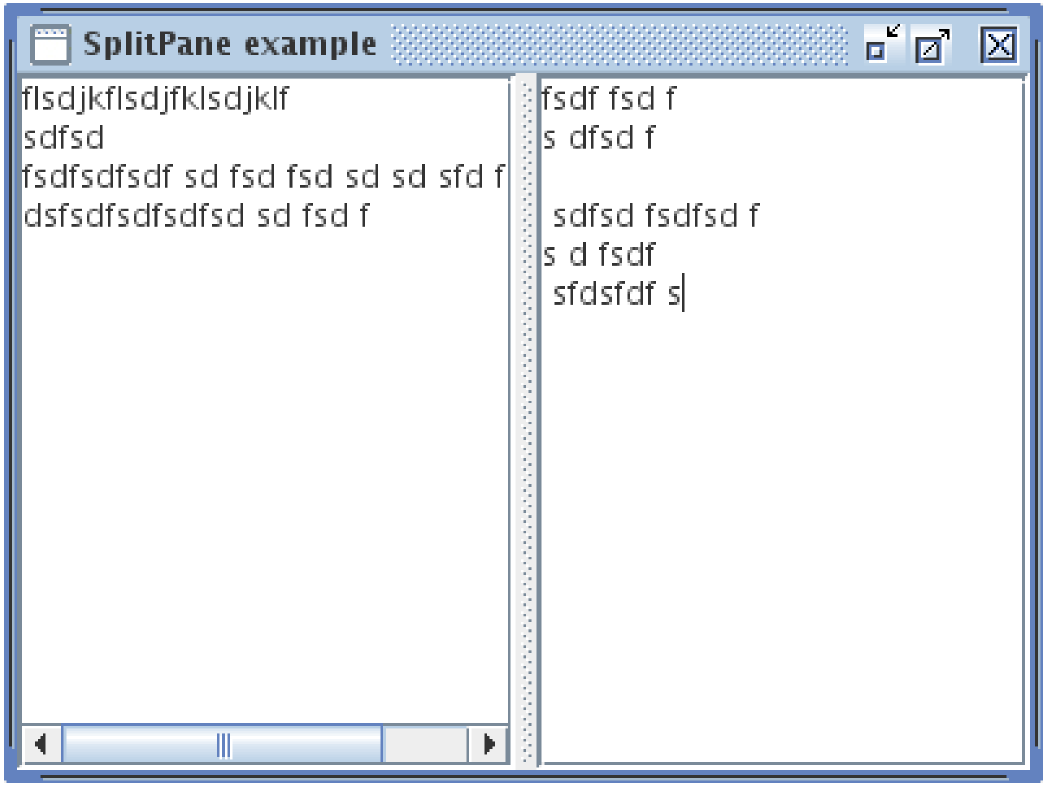
JToolBar

- a bar with buttons
- can be dragged to other place
- can be drag out



JSplitPane

- displays 2 components
  - horizontally
  - vertically
- the separator between components can be moved



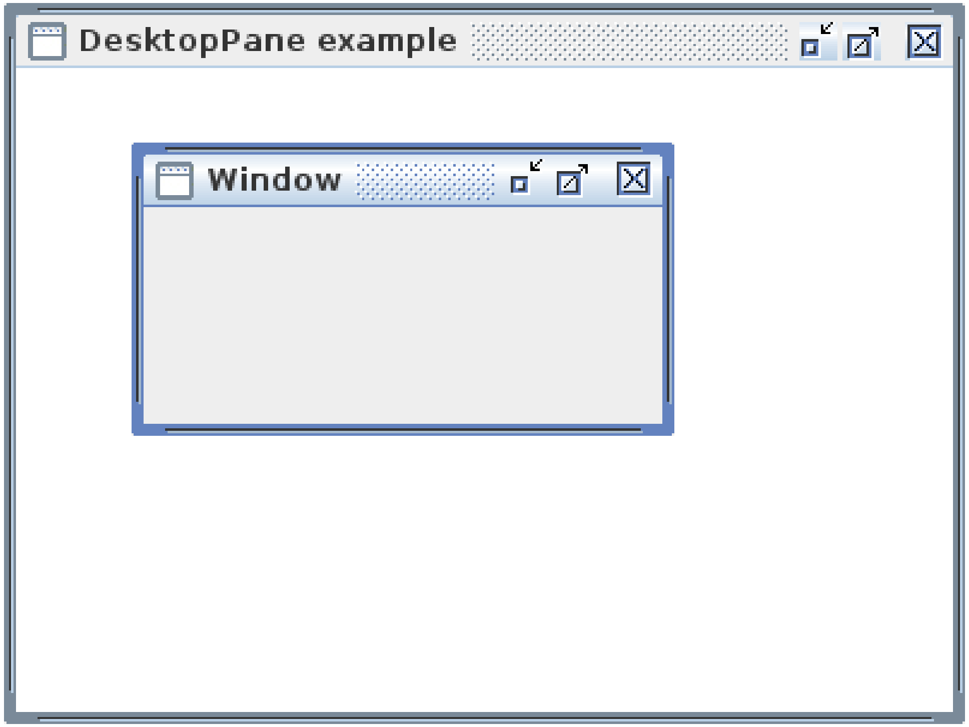
flsdjkflsdjfklsdjklf
sdfsd
fsdfsdfsdf sd fsd fsd sd sd sfd f
dsfsdfsdfsdfsd sd fsd f

fsdf fsd f
s dfsd f

 sdfsd fsdfsd f
s d fsdf
 sfdsfdf s

JDesktopPane

- „a window in a window“
- JDesktopPane
  - desktop
- JInternalFrame
  - inner window



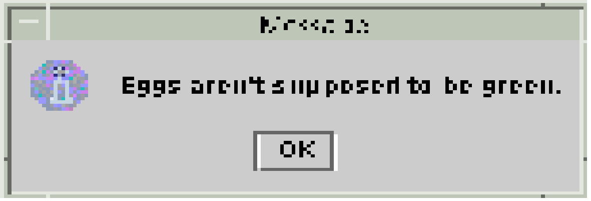
Swing

Dialogs

Overview

- *JDialog*
- *a dialog* = a window similar to the *frame*
- *dialogs* depend on a *frame*
- a dialog is modal
  - if it is displayed, input to other windows of an application is blocked
  - non-modal dialogs can be created also
- managing the dialog – almost the same as for frame
- JOptionPane
  - a component simplifying creation of standard dialogs
  - predefined dialogs

JOptionPane

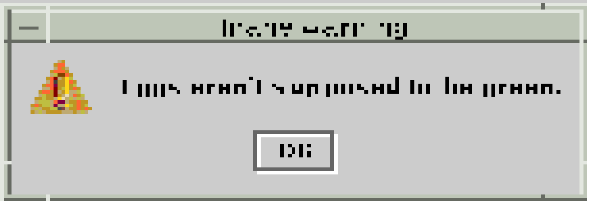


Message

Eggs aren't supposed to be green.

OK

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

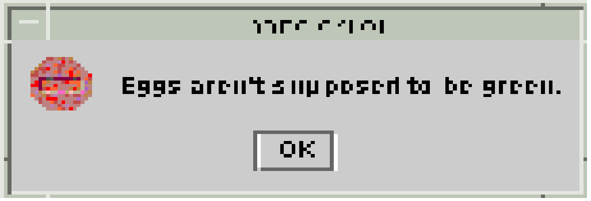


Inane warning

Eggs aren't supposed to be green.

OK

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

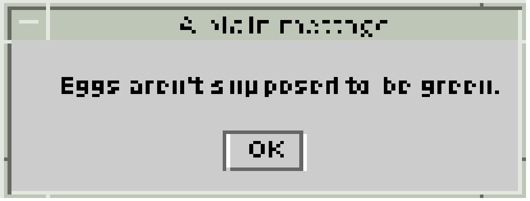


Inane error

Eggs aren't supposed to be green.

OK

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

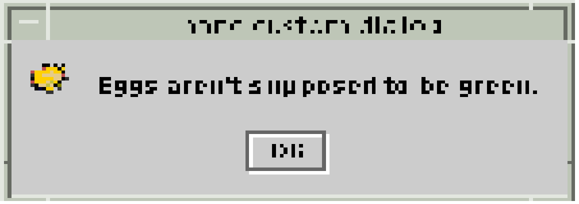


A plain message

Eggs aren't supposed to be green.

OK

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



Inane custom dialog

Eggs aren't supposed to be green.

OK

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

JOptionPane

- predefined dialogs
  - but can be configured
- a set of static methods creating dialogs (always several variants of the single method)
  - showMessageDialog()
    - a dialog with message
  - showInputDialog()
    - a dialog with an input line
    - returns String
  - showConfirmDialog()
    - a dialog with a question (Yes/No/Cancel)
    - returns int
  - showOptionDialog()
    - selection of several choices (Yes-No-Maybe-Cancel)

JOptionPane

- can be also used directly
  - by creating an instance of JOptionPane
    - several constructors
  - the created object can inserted to a dialog

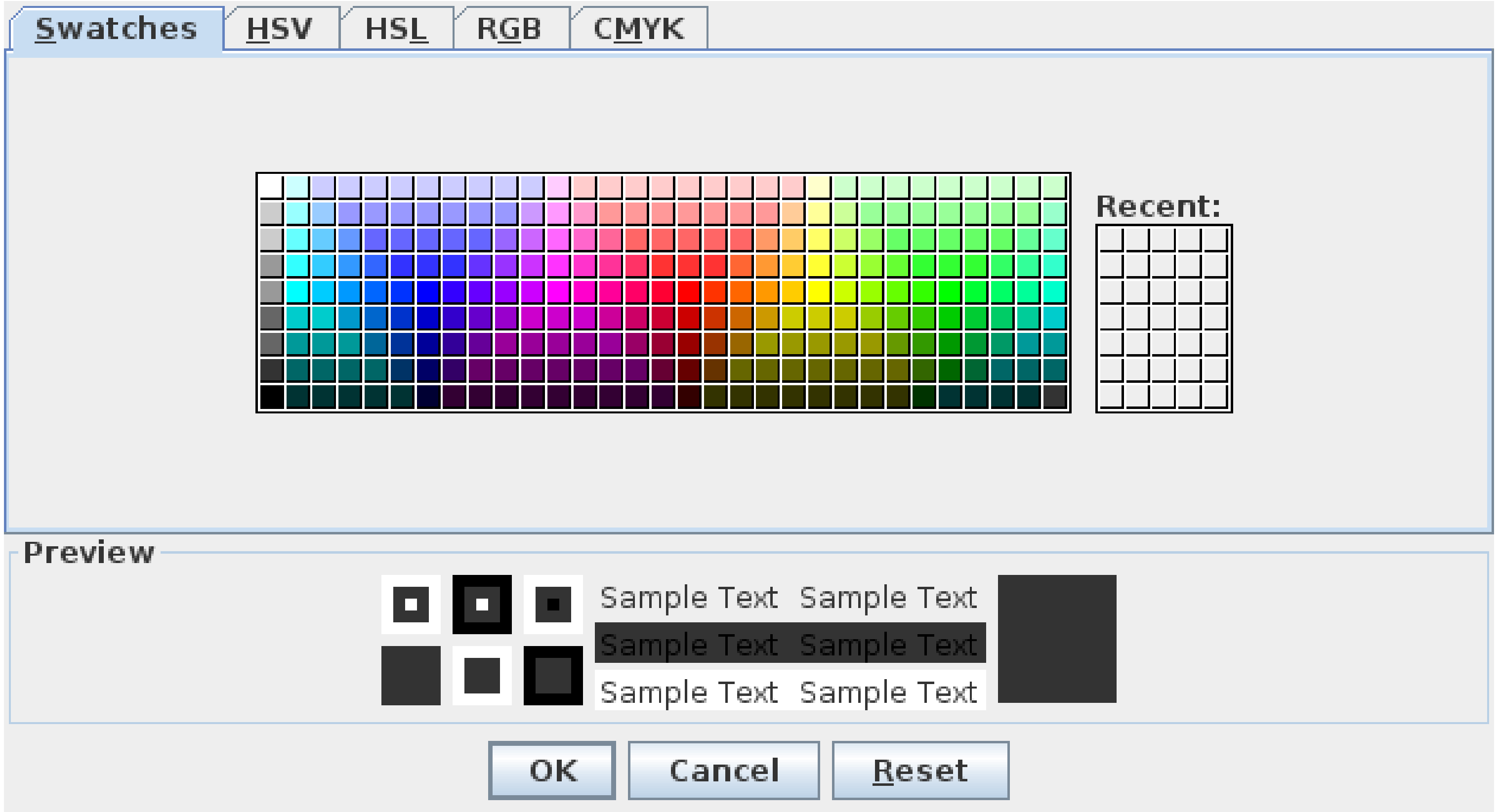
JFileChooser

- a standard dialog for file selection

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

JColorChooser

- choosing colors
- can be used
  - as a dialog
  - as a component



Slides version AJ05.en.2020.01