Components

Ji Zhenyan & Åke Malmberg

How to Make Dialogs

- Several classes support *dialogs*. To create simple, standard dialogs, use the JOptionPane class. The ProgressMonitor class can put up a dialog that shows the progress of an operation. Two other classes, JColorChooser and JFileChooser, also supply standard dialogs. To bring up a print dialog, use either the getPrintJob method defined in the Toolkit class or using the Java 2 platform, the Printing API. To create custom dialogs, use the JDialog class directly.
- Every dialog is dependent on a frame. When that frame is destroyed, so are its dependent dialogs. When the frame is iconified, its dependent dialogs disappear from the screen. When the frame is deiconified, its dependent dialogs return to the screen.

- The dialogs that JOptionPane provides are modal. When a modal dialog is visible, it blocks user input to all other windows in the program.
- To create a non-modal dialog, you must use the JDialog class directly.
- Even when you use JOptionPane to implement a dialog, you're still using a JDialog behind the scenes. The reason is that JOptionPane is simply a container that can automatically create a JDialog and add itself to the JDialog's content pane.
- **DialogDemo.java**

Panels

- The JPanel class provides general-purpose containers for lightweight components.
- By default, panels are opaque. You can make a panel transparent by invoking setOpaque(false).
- A transparent panel draws no background, so that any components underneath show through.
- uses panels in several ways:
  - serves as a content pane for the application's frame.
  - are used to both contain components and coordinate communication between components.
  - are used to group components and restrict their size.

- a panel's layout manager is an instance of FlowLayout
- add components to a panel, use the add method.
- When the layout manager is FlowLayout, BoxLayout, GridLayout, or GridBagLayout, use the one-argument add method, like this:

```
aFlowPanel.add(aComponent);
aFlowPanel.add(anotherComponent);
```

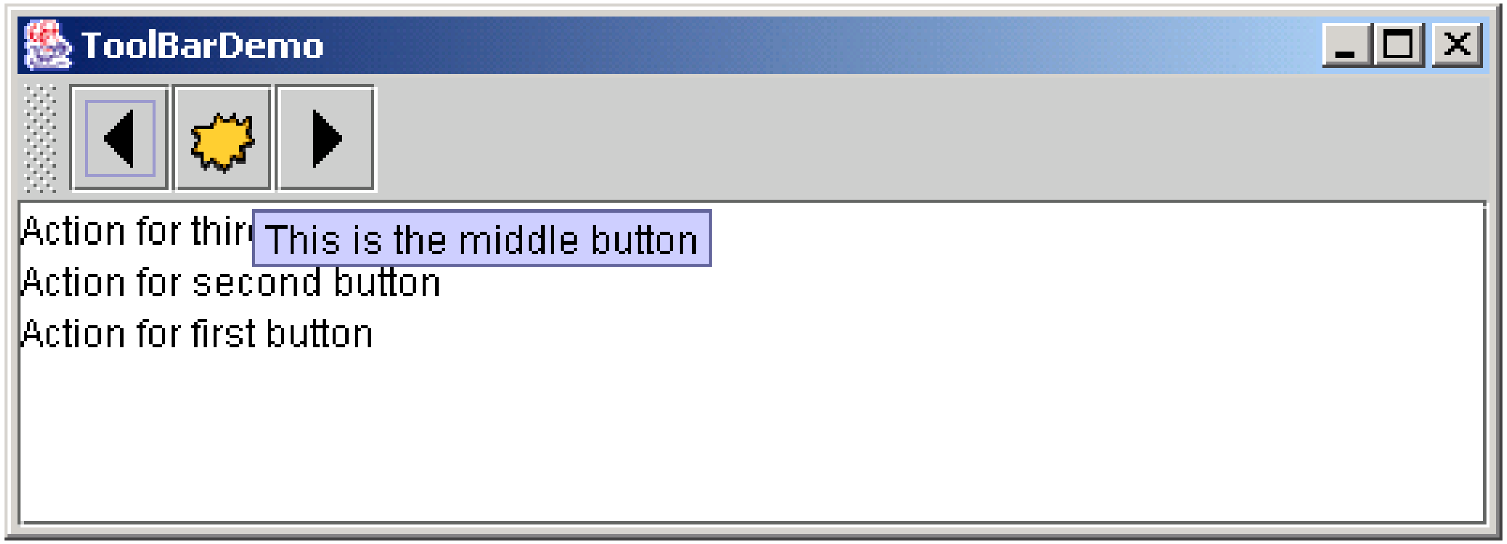
- When the layout manager is BorderLayout, provide a second argument specifying the added component's position within the panel. For example:

```
aBorderPanel.add(aComponent, BorderLayout.CENTER);
aBorderPanel.add(anotherComponent, BorderLayout.SOUTH);
```

Tool Bars

- A JToolBar is a container that groups several components -- usually buttons with icons -- into a row or column.
- tool bars provide easy access to functionality that is also in menus.
- By default, the user can drag the tool bar to a different edge of its container or out into a window of its own.
- For the drag-out behavior to work correctly, the tool bar must be in a container that uses BorderLayout. The component that the tool bar affects is generally in the center of the container. The tool bar must be the only other component in the container; it must not be in the center.
- If any buttons in your tool bar duplicate functionality of other components, such as menu items, then you should probably create and add the tool-bar buttons as described in How to Use Actions.

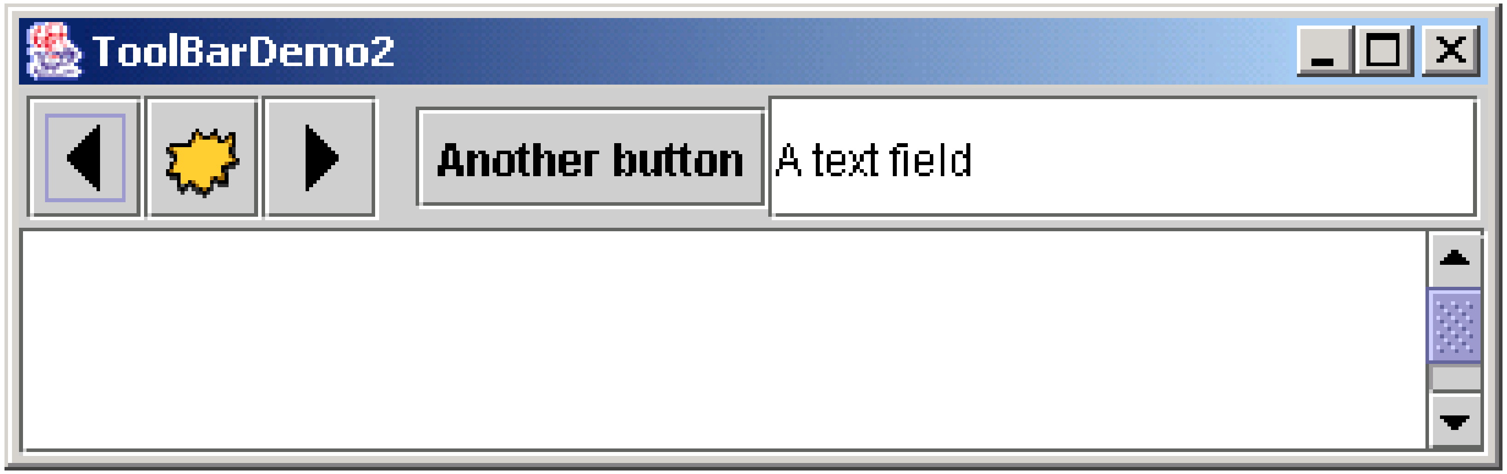
ToolBarDemo.java



Action for thir
This is the middle button
Action for second button
Action for first button

- Using the setFloatable(false) to make a tool bar immovable.
- Adding a separator to a tool bar.
- Adding a non-button component to a tool bar.
- You can easily make the components in a tool bar be aligned along their tops or bottoms, instead of centered, by invoking the setAlignmentY method. For example, to align the tops of all the components in a tool bar, invoke setAlignmentY(TOP_ALIGNMENT) on each component. Similarly, you can use the setAlignmentX method to specify the alignment of components when the tool bar is vertical. This flexibility of layout is possible because tool bars use BoxLayout to position their components.

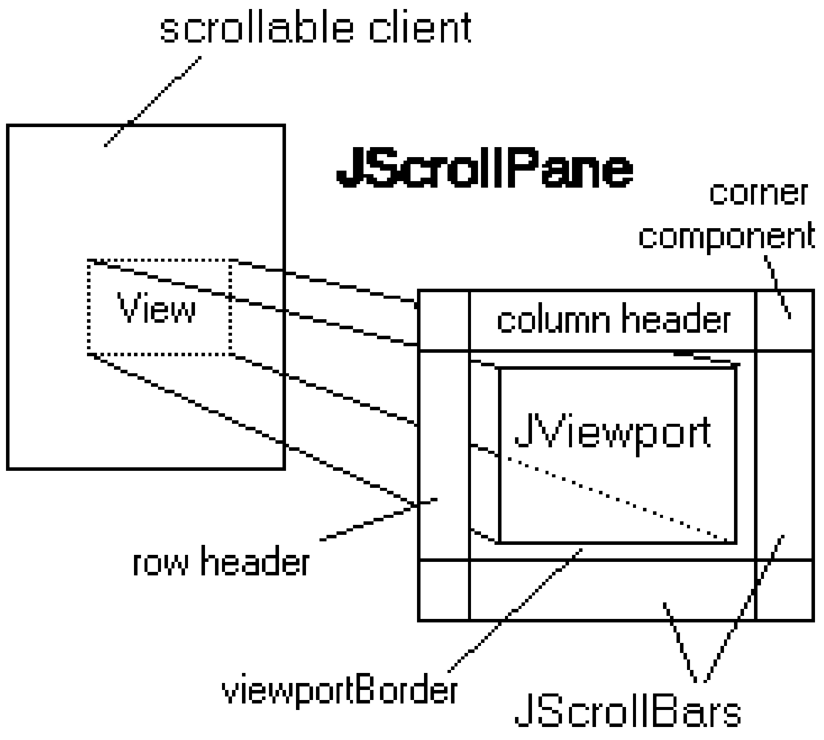
ToolbarDemo2.java



Scroll Panes

- A JScrollPane provides a scrollable view of a component.
- The scroll pane handles everything else: creating the scroll bars when necessary, redrawing the client when the user moves the scroll knobs, and so on.
- **contentPane.setPreferredSize(new Dimension(400, 100));**
- **JScrollPane**(Component view, int vsbPolicy, int hsbPolicy)
- **JScrollPane**(int vsbPolicy, int hsbPolicy)
- void setHorizontalScrollBarPolicy(int)
  void setVerticalScrollBarPolicy(int)

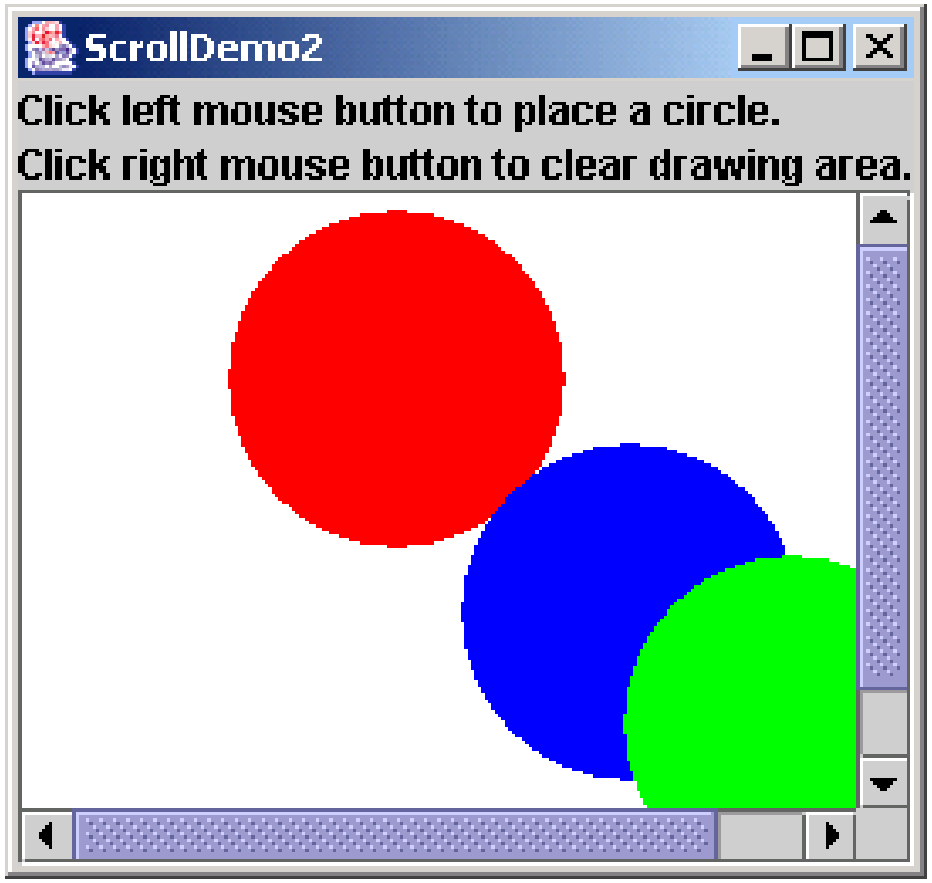
Scroll Panes



- **Policy**
- VERTICAL_SCROLLBAR_AS_NEEDED HORIZONTAL_SCROLLBAR_AS_NEEDED
- The default.
- VERTICAL_SCROLLBAR_ALWAYS HORIZONTAL_SCROLLBAR_ALWAYS
- VERTICAL_SCROLLBAR_NEVER HORIZONTAL_SCROLLBAR_NEVER

- **MouseInputAdapter**
- The adapter which receives mouse events and mouse motion events. The methods in this class are empty; this class is provided as a convenience for easily creating listeners by extending this class and overriding only the methods of interest.

ScrollDemo2.java



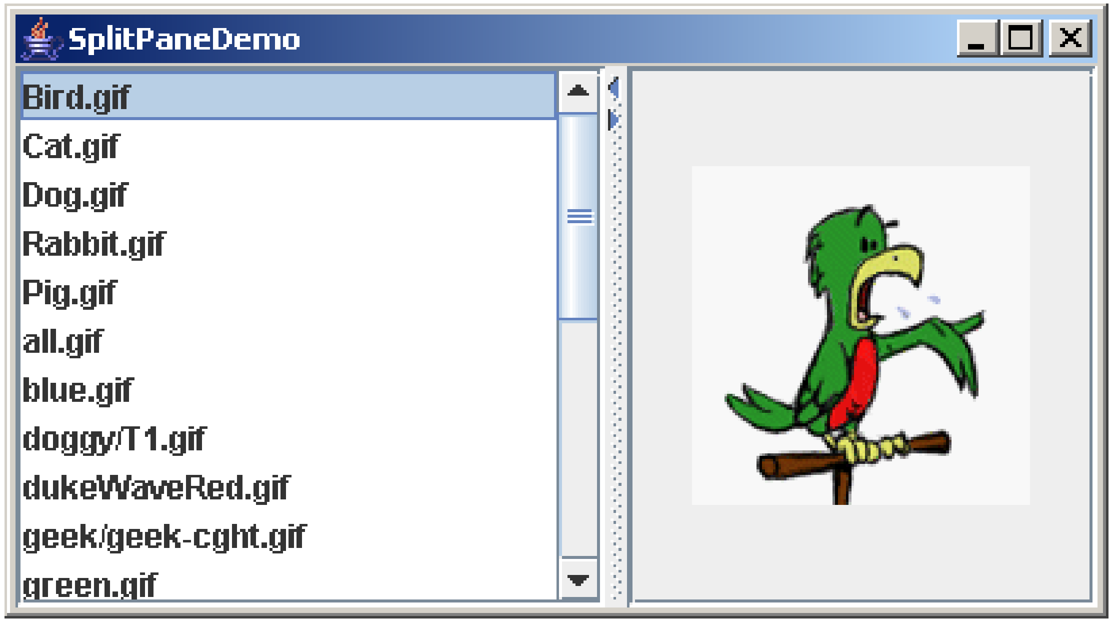
Click left mouse button to place a circle.
Click right mouse button to clear drawing area.

Split Panes

- A JSplitPane displays two components, either side by side or one on top of the other.
- can divide screen space among three or more components by putting split panes inside of split panes
- By dragging the divider that appears between the components, the user can specify how much of the split pane's total area goes to each component.

Split Panes

SplitPaneDemo.java



Resource bundles

- Resource bundles contain locale-specific objects.
- This allows you to write programs that can:
  - be easily localized, or translated, into different languages
  - handle multiple locales at once
  - be easily modified later to support even more locales

Delimiter

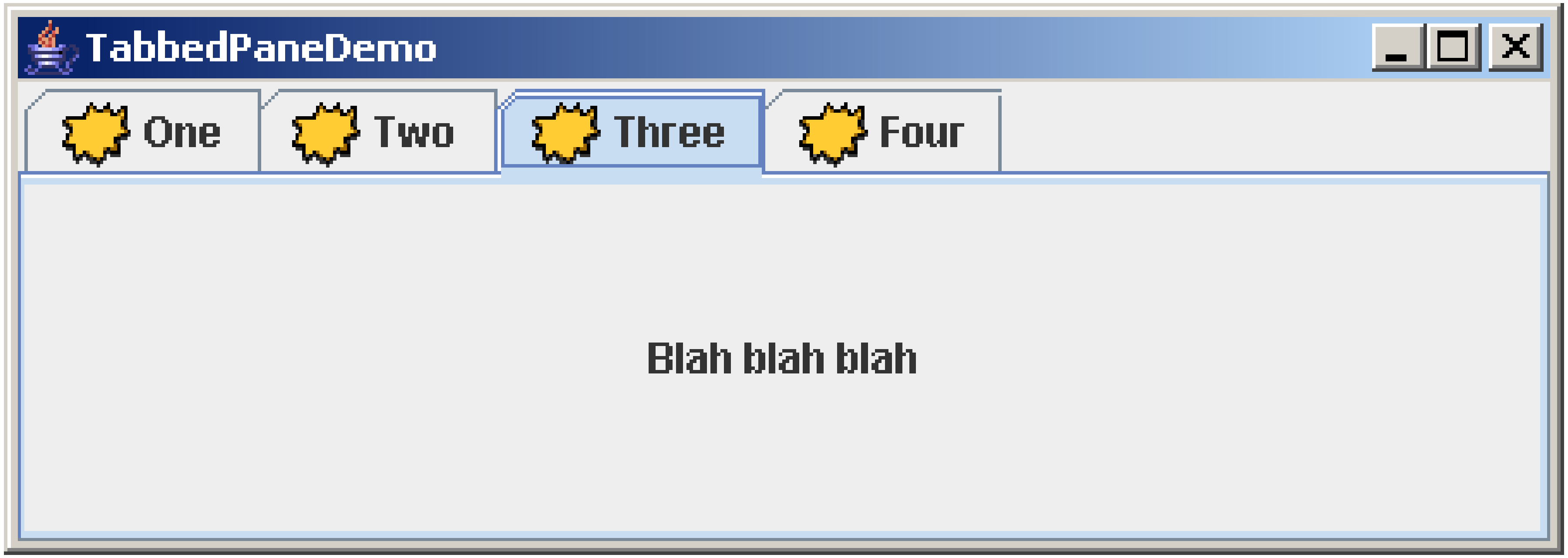
- If there are different resources for different countries, you can make specializations: for example, "MyResources_de_CH" contains objects for the German language (de) in Switzerland (CH).

I18NSample.java

Tabbed Panes

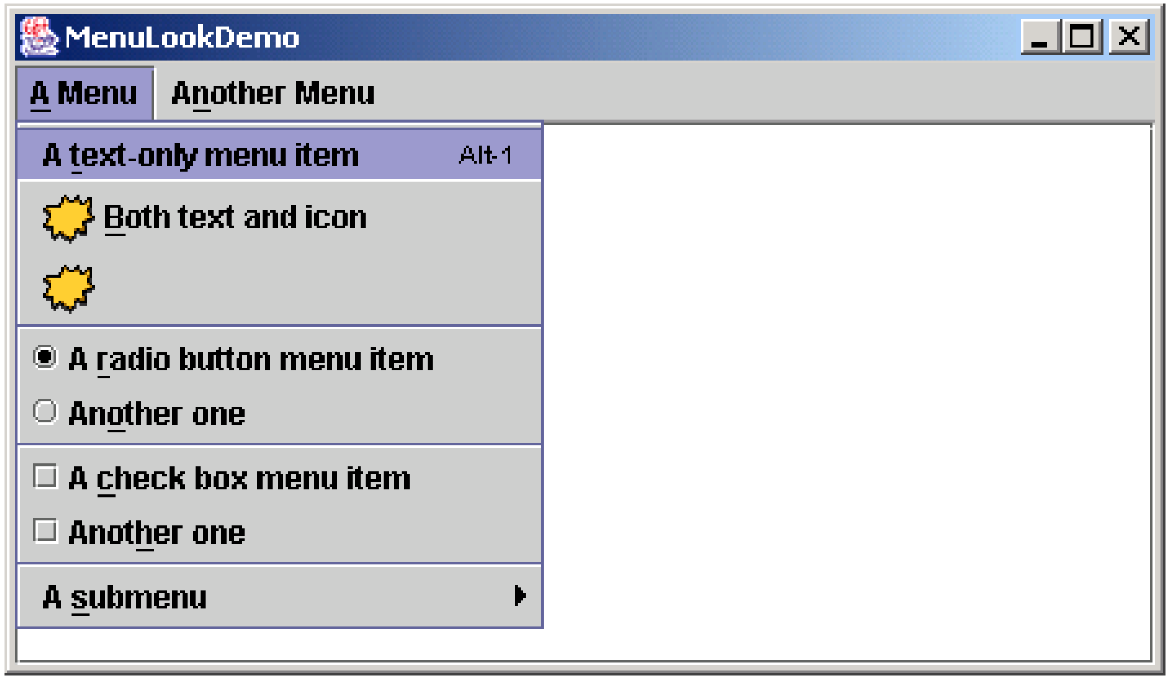
- With the JTabbedPane class, you can have several components (usually panels) share the same space. (CardLayout)
- The user chooses which component to view by selecting the tab corresponding to the desired component.

TabbedPaneDemo.java



Blah blah blah

MenuLookDemo.java

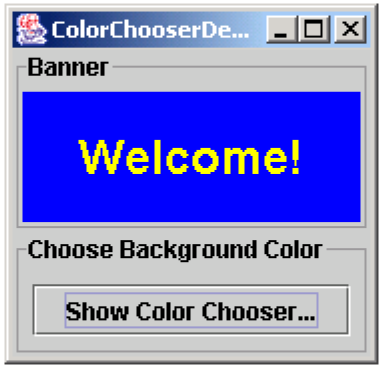


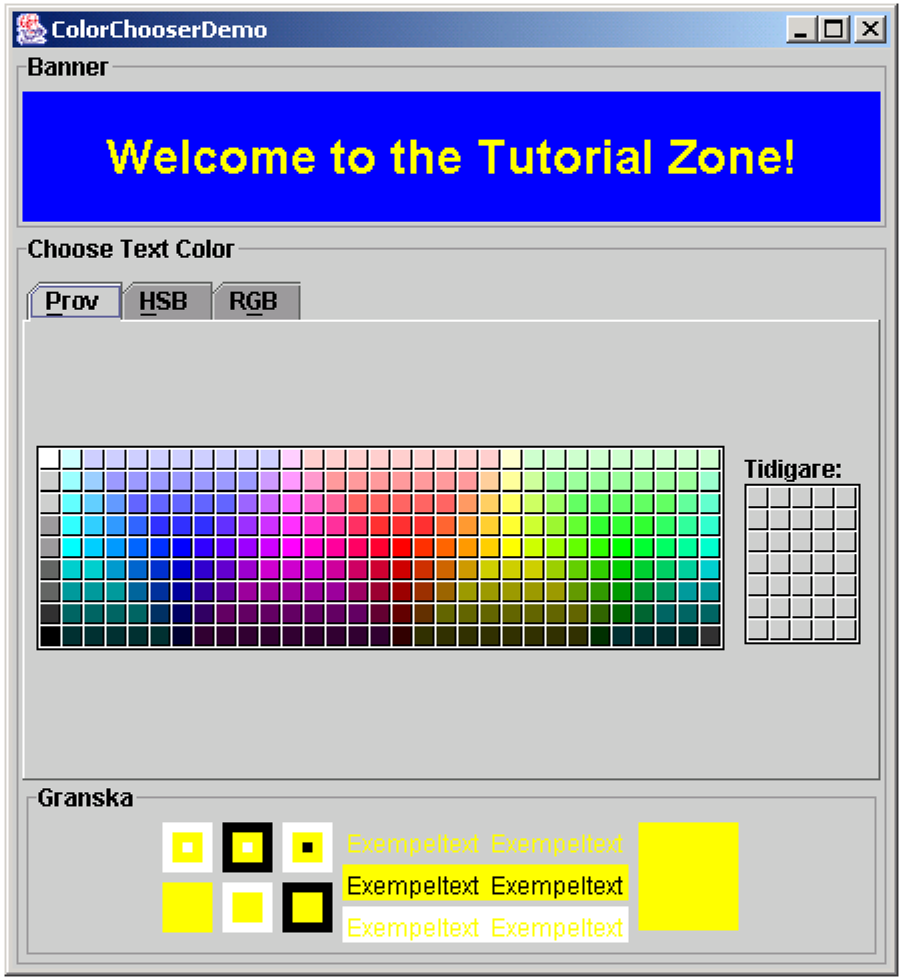
Color Choosers

- the JColorChooser class to provide users with a palette of colors to choose from.
- A color chooser is a component that you can place anywhere within your program's GUI.
- A color chooser uses an instance of ColorSelectionModel to contain and manage the current selection.
- The color selection model fires a change event whenever the user changes the color in the color chooser.
- registers a change listener with the color selection model so that it can update the banner at the top of the window.

ColorChooserDemo.java

ColorChooserDemo2.java

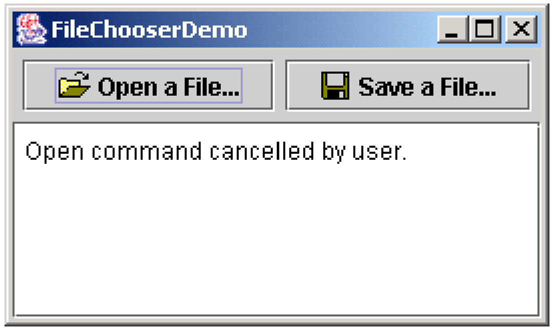




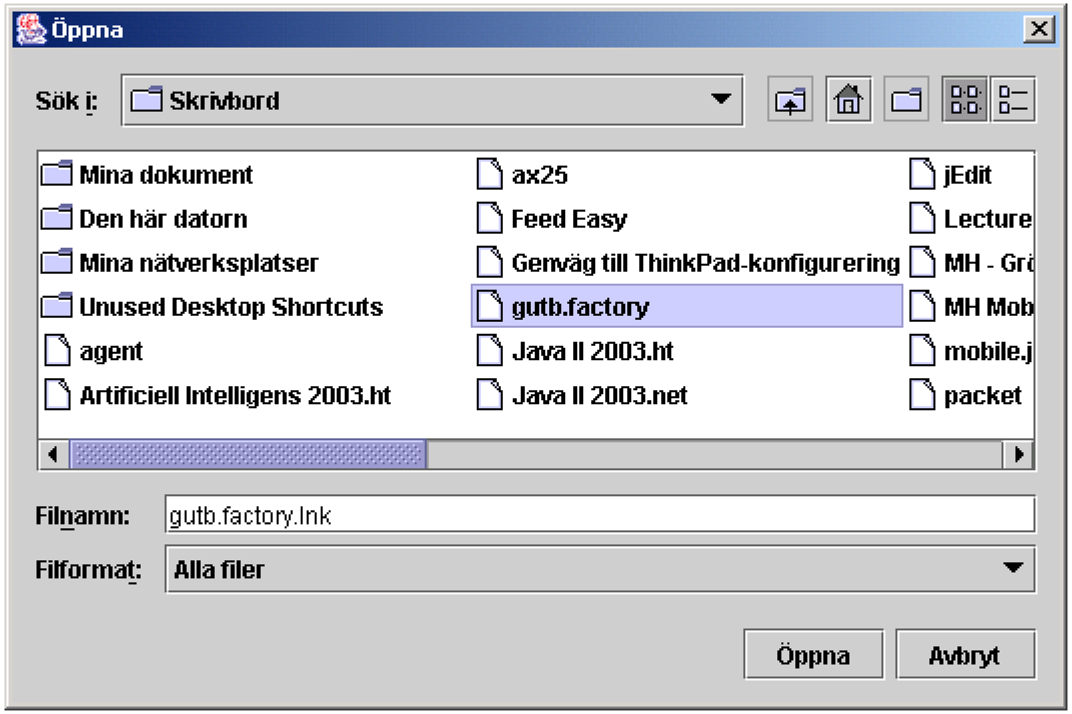
File Choosers

- File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list or entering a file name or directory name.
- A JFileChooser object only presents the GUI for choosing files. Your program is responsible for doing something with the chosen file, such as opening or saving it.
- You can specify the file chooser's initial directory using one of JFileChooser's other constructors, or you can set the directory with the setCurrentDirectory method.
- By default, a file chooser displays all of the files and directories that it detects, except hidden files. A program can apply one or more *file filters* to a file chooser so that the chooser shows only some files.

FileChooserDemo.java



Open command cancelled by user.



Filnamn: gutb.factory.lnk

Filformat: Alla filer

Tables

- With the JTable class you can display tables of data, optionally allowing the user to edit the data. JTable doesn't contain or cache data; it's simply a view of your data.
- If you're using a table without a scroll pane, then you must get the table header component and place it yourself. For example:
- container.setLayout(new BorderLayout()); container.add(table.getTableHeader(), BorderLayout.NORTH); container.add(table, BorderLayout.CENTER);

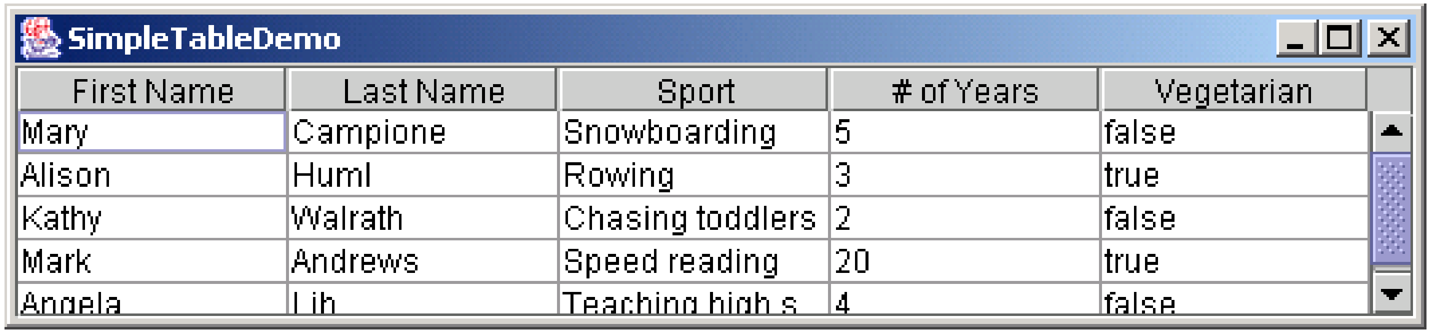
- TableColumn column = null;
- for (int i = 0; i < 5; i++) {
- column = table.getColumnModel().getColumn(i);
- if (i == 2) {
- column.setPreferredWidth(100); //sport column is bigger
- } else {
- column.setPreferredWidth(50); } }
- each column in a table is represented by a TableColumn object. Besides setPreferredWidth, TableColumn also supplies methods for getting and setting the minimum, current, and maximum width of a column.
- When the user explicitly resizes columns, the new sizes become not only the columns' new *current* widths, but also the columns' new *preferred* widths, However, when columns are resized as the result of the table width changing, the columns' preferred widths do not change.

- change a table's resize behavior by invoking the setAutoResizeMode method. The method's argument should have one of these values (defined as constants in JTable):
- AUTO_RESIZE_SUBSEQUENT_COLUMNS
  - The default. In addition to resizing the column to the left of the drag point, adjusts the sizes of all columns to the right of the drag point.
- AUTO_RESIZE_NEXT_COLUMN
  - Adjusts only the columns immediately to the left and right of the drag point.
- AUTO_RESIZE_OFF
  - Adjust the table size instead.

Tables

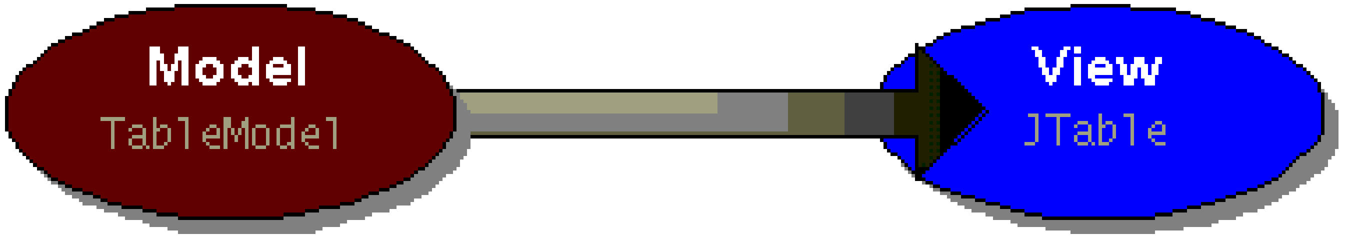
- By default, a table allows the user to select multiple rows -- not columns or individual cells -- and the selected rows need not be next to each other.
- By changing a couple of boolean values, you can make the table allow either column selections or individual cell selections, instead of row selections.

SimpleTableDemo.java

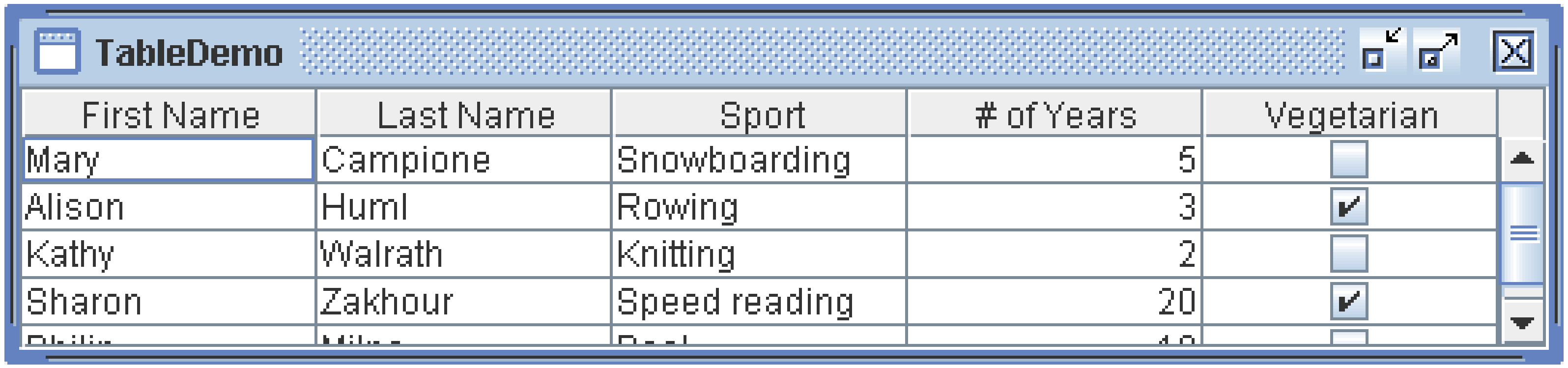


First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	false
Alison	Huml	Rowing	3	true
Kathy	Walrath	Chasing toddlers	2	false
Mark	Andrews	Speed reading	20	true
Angela	Lih	Teaching high s	4	false

Abstract Table Model



TableDemo.java


TableDemo

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	☐
Alison	Huml	Rowing	3	✓
Kathy	Walrath	Knitting	2	☐
Sharon	Zakhour	Speed reading	20	✓

Trees

- With the JTree class, you can display hierarchical data. A JTree object doesn't actually contain your data; it simply provides a view of the data.
- JTree displays its data vertically. Each row displayed by the tree contains exactly one item of data, which is called a *node*. Every tree has a *root* node from which all nodes descend.