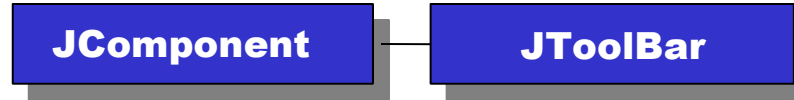


# Toolbars

- Allow users to quickly invoke the most common program features without forcing them to sift through menus
- Common on most modern applications
- Position horizontally across top of window just below the menu bar
- Position sub-task and sub-features bars along sides of window
- Permit the location of the bar to be changed by the user
- Permit the display of the bar to be turned on or off by the user, through standard menus

# Toolbars in Java



- AWT had no toolbar support, you could have built your own, but no images existed for the buttons
- JToolBar *can* contain
  - Graphical buttons
  - tool tips
  - dock/undock
  - Any other JComponent (JComboBox, etc...)
- A typical JToolBar button *should* contain
  - graphics (no text)
  - tooltip (to supply the text)
- Typically a panel containing JButtons, but with dock/undock support

# JToolBar Basics

- Size, as recommended by Microsoft (1999)
  - Use 16x16 or 20x20 pixel graphics
  - Should use GIF or PNG format
    - Background of icon should be transparent
- Location
  - Typically at top of window
  - **Must** use BorderLayout manager in parent container
    - Docking manager assumes BorderLayout configuration no matter what the actual layout is.
    - If docking manager detects East/West dock, it sets the orientation of the toolbar to vertical, then adds it to the container. If you aren't using BorderLayout, effects can be “surprising”

# Actions and JToolBars

- Good use of `AbstractAction(Action)` interface. Able to add the same object to handle events in both `JMenu` and `JToolBar`
  - You could duplicate this functionality by having the container still handle events, and add it to the action listeners of both menus and toolbars
  - Actions let you maintain other state information
  - Uses `AbstractAction.getValue()` with
    - `NAME`: used in menus and buttons
    - `SHORT_DESCRIPTION`: automatically sets tooltip text in toolbar
    - `LONG_DESCRIPTION`
    - `SMALL_ICON`: icon used in toolbars
    - `MNEMONIC_KEY`: key to use for a mnemonic (new in 1.3)
      - Note: Key must be an *Integer* Object, use new `Integer(KeyEvent.VK_A)` for the *a* key
    - `ACCELERATOR_KEY`: key to use for accelerator (new in 1.3)
- Benefits of using Actions
  - Encapsulation
    - Instead of requiring you to create separate menu items and buttons to which you would connect the `ActionListener`, you create a single object
  - State management
    - When you enable/disable an Action, all supporting menu's, toolbars, etc are also enabled/disabled

# JToolBar Methods

- **Constructors**

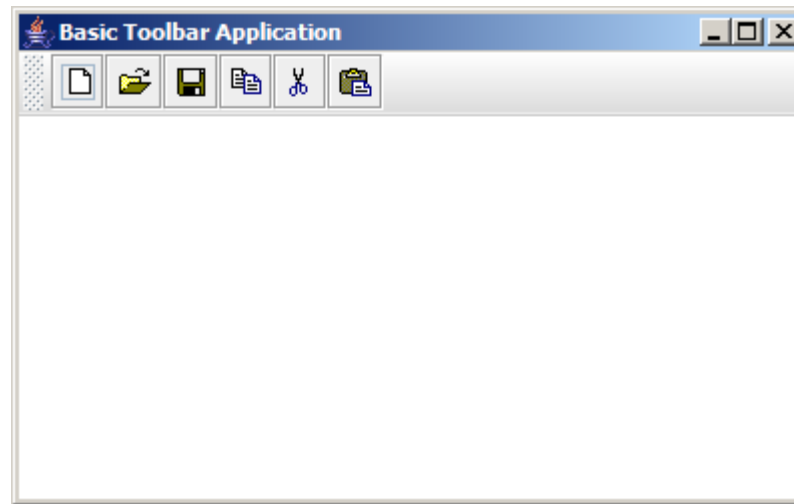
```
public JToolBar()  
public JToolBar(int orientation)  
public JToolBar(String name)  
public JToolBar(String name, int orientation)
```

- **Other significant methods**

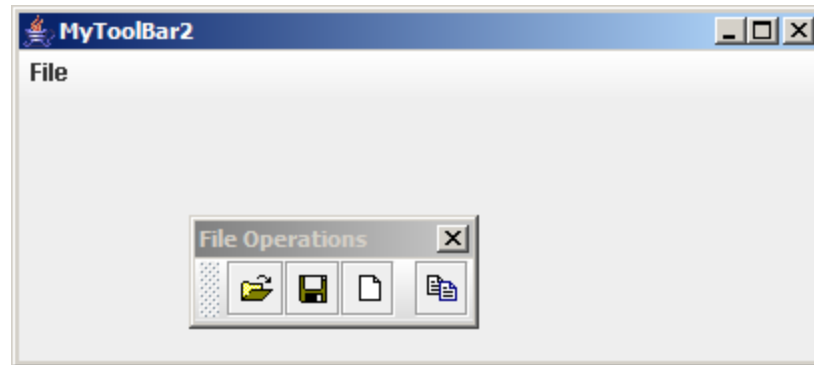
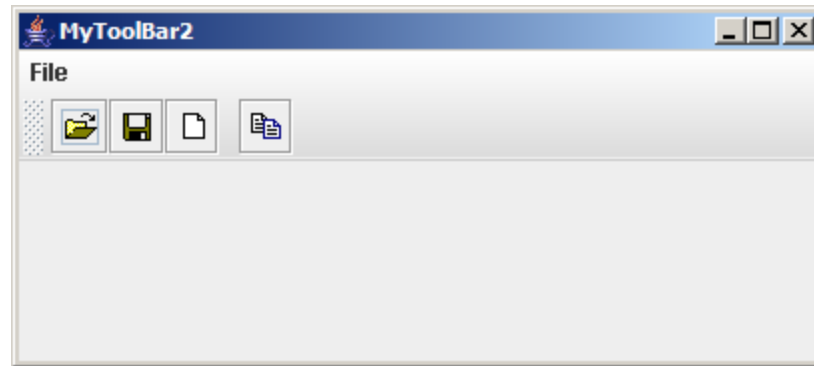
- public void setFloatable(boolean b)
- public boolean isFloatable()
- public JButton add(Action a)
- public void addSeparator()
- public void setMargin(Insets I)
- add/remove methods of JComponent
  - remember, you can add *any* JComponent to the ToolBar

- Since it is a panel, you can use `setBorder()` to change the border of the tool bar.

# Code Example, MyToolBar1.java

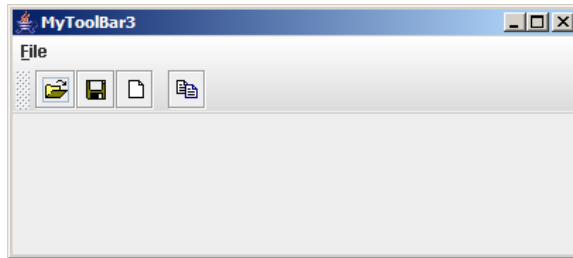


# Code Example, MyToolBar2.java

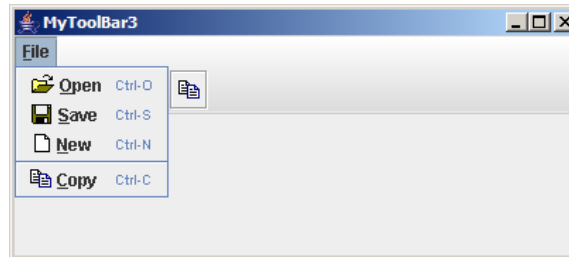


# Code Example, MyToolBar3.java

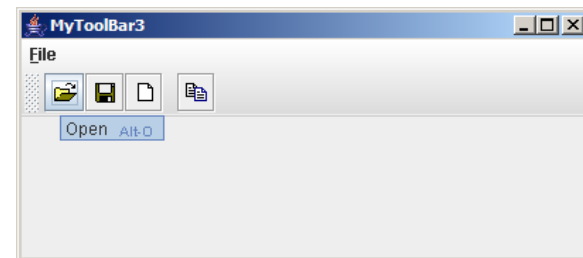
- The works! Abstract actions, Menus, ToolBars, and accelerators all in one!



Menu

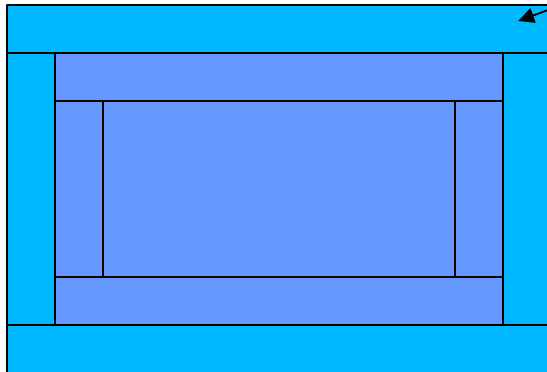


Toolbar Tooltip

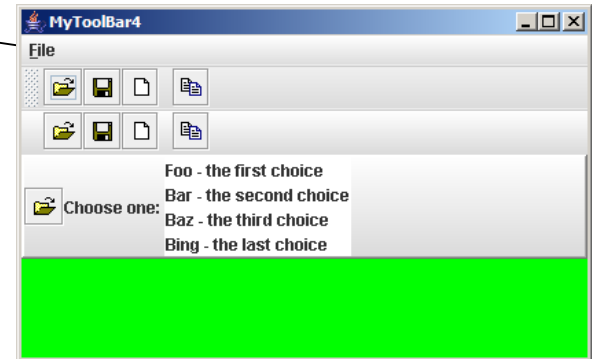


# Code Example, MyToolBar4.java

- You are not restricted to a single Tool Bar
- JToolBar must be used with BorderLayout
- So How do you get multiple toolbars?
  - Nested JPanels with BorderLayouts
  - Each Parent container tracks its Toolbar, and will only allow its own toolbar to dock



Each toolbar  
Goes in its own  
BorderLayout.NORTH



# Placement of JToolBars

- Toolbar colors
  - You can set docking and floating colors for a toolbar with the UI of the toolbar
    - `public void setFloatingColor(Color c)`
    - `public void setDockingColor(Color c)`
- Creating a floating toolbar
  - Toolbar must be floatable in order for it to be detached from its parent container
    - `public void setFloatable(boolean b)`
  - A toolbar detaches when its `setFloating()` method is called
    - `public void setFloating(boolean b, Point p)`
    - `public void setFloatingLocation(int x, int y)`

# Code Example, MyToolBar5.java

