# *Advanced Programming*
# CSE 201

Instructor: Sambuddho

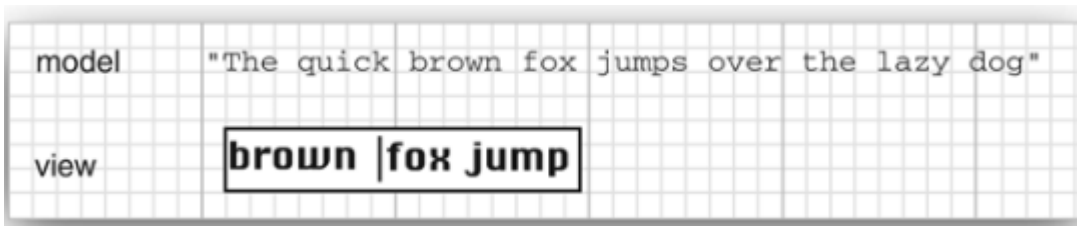
(Semester: Monsoon 2025)

Week 9 - Swing MVC

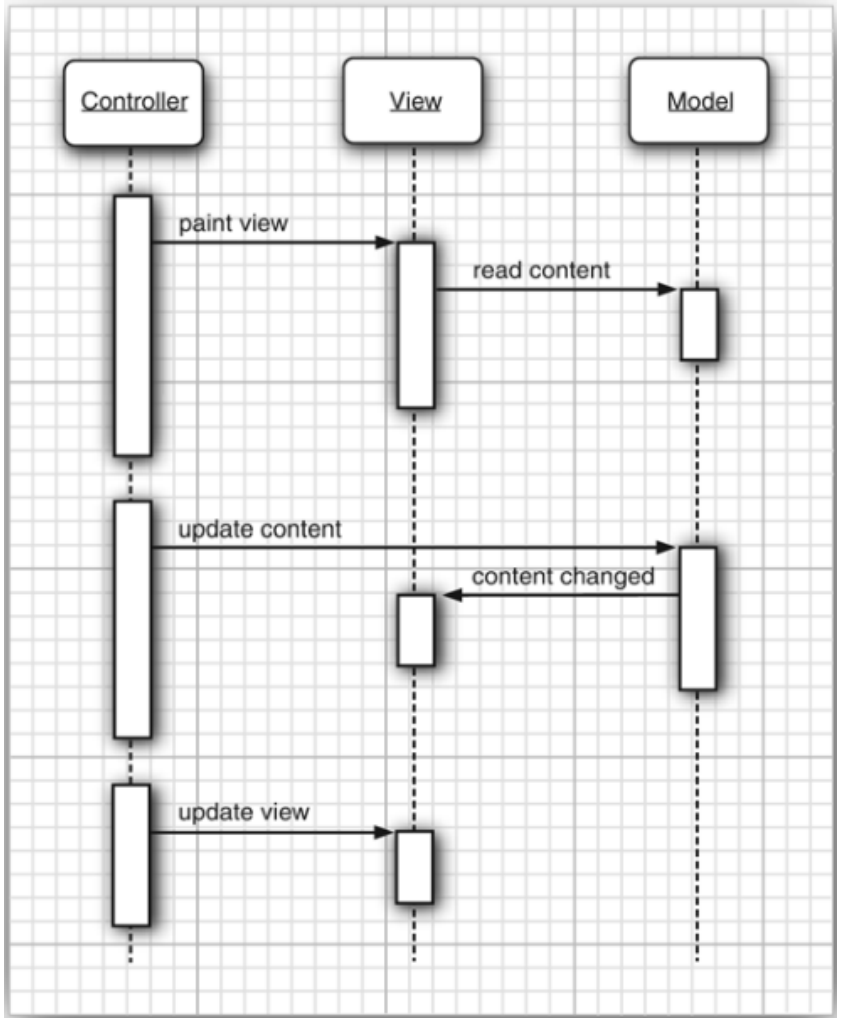
# Model View Controller

- *Design pattern for using AWT/Swing packages*

*– Model: Data to be displayed/used.*
*– View: How the data should appear and be displayed.*
*– Controller: Event handling.*



# Interaction Between Model, View and Controller




```
var button = new JButton("Blue");
ButtonModel model = button.getModel();
```

| Property Name | Value |
|---|---|
| `actionCommand` | The action command string associated with this button |
| `mnemonic` | The keyboard mnemonic for this button |
| `armed` | `true` if the button was pressed and the mouse is still over the button |
| `enabled` | `true` if the button is selectable |
| `pressed` | `true` if the button was pressed but the mouse button hasn't yet been released |
| `rollover` | `true` if the mouse is over the button |
| `selected` | `true` if the button has been toggled on (used for checkboxes and radio buttons) |

# Layout Management


```
panel.setLayout(new GridLayout(4, 4));
```


**java.awt.Container** 1.0

- `void setLayout(LayoutManager m)`

  sets the layout manager for this container.
- `Component add(Component c)`
- `Component add(Component c, Object constraints)` 1.1

  adds a component to this container and returns the component reference.

**java.awt.GridLayout** 1.0

- `GridLayout(int rows, int columns)`
- `GridLayout(int rows, int columns, int hgap, int vgap)`

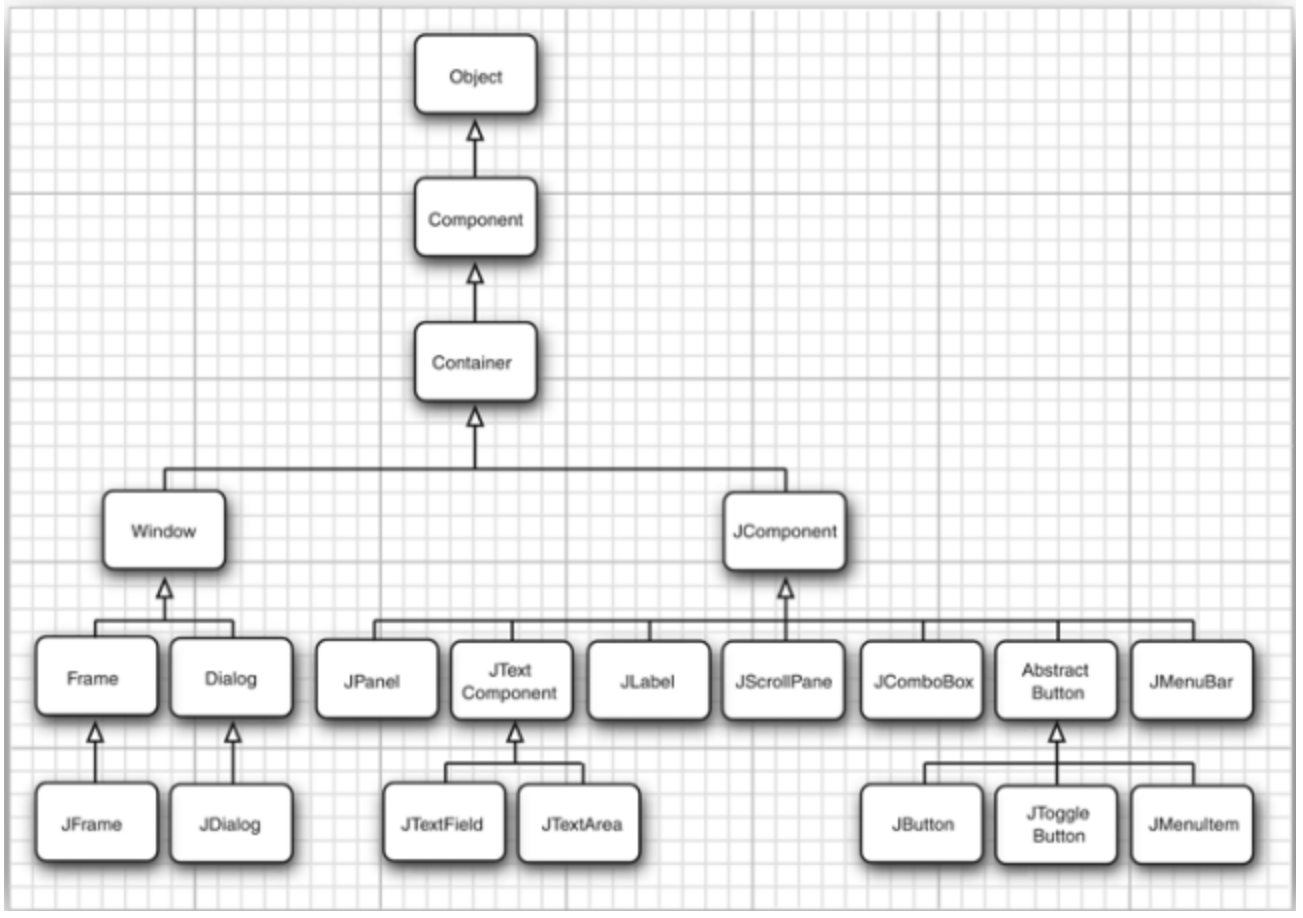
  constructs a new `GridLayout`. One of `rows` and `columns` (but not both) may be zero, denoting an arbitrary number of components per row or column.

**java.awt.FlowLayout** 1.0

- `FlowLayout()`
- `FlowLayout(int align)`
- `FlowLayout(int align, int hgap, int vgap)`

  constructs a new `FlowLayout`. The `align` parameter is one of `LEFT`, `CENTER`, or `RIGHT`.

# Component Inheritances



# Text Input

```
var panel = new JPanel();
var textField = new JTextField("Default input", 20);
panel.add(textField);
```

- Default string
- Width

- JTextField vs JTextArea

Single line vs a block.

**javax.swing.text.JTextComponent** 1.2

- String getText()
- void setText(String text)

  gets or sets the text of this text component.

- boolean isEditable()
- void setEditable(boolean b)

  gets or sets the editable property that determines whether the user can edit the content of this text component.

# *Text Input*

```
var panel = new JPanel();
var textField = new JTextField("Default input", 20);
panel.add(textField);

textField.setColumns(10);
panel.revalidate();
```

- revalidate() method example of controller communicating with the views to change the text field area.

# Text Areas

```
textArea = new JTextArea(8, 40); // 8 lines of 40 columns each

textArea.setLineWrap(true); // long lines are wrapped


textArea = new JTextArea(8, 40);
var scrollPane = new JScrollPane(textArea);
```

**javax.swing.JTextArea** 1.2

- `JTextArea()`
- `JTextArea(int rows, int cols)`
- `JTextArea(String text, int rows, int cols)`

  constructs a new text area.
- `void setColumns(int cols)`

  tells the text area the preferred number of columns it should use.
- `void setRows(int rows)`

  tells the text area the preferred number of rows it should use.
- `void append(String newText)`

  appends the given text to the end of the text already in the text area.
- `void setLineWrap(boolean wrap)`

  turns line wrapping on or off.

**javax.swing.JScrollPane** 1.2

- `JScrollPane(Component c)`

  creates a scroll pane that displays the content of the specified component. Scrollbars are supplied when the component is larger than the view.

**javax.swing.JTextArea** 1.2 *(Continued)*

- `void setWrapStyleWord(boolean word)`

  If `word` is `true`, long lines are wrapped at word boundaries. If it is `false`, long lines are broken without taking word boundaries into account.
- `void setTabSize(int c)`

  sets tab stops every `c` columns. Note that the tabs aren't converted to spaces but cause alignment with the next tab stop.

# Password Fields - Type of Text Fields

**javax.swing.JPasswordField** 1.2

- `JPasswordField(String text, int columns)`

  constructs a new password field.

- `void setEchoChar(char echo)`

  sets the echo character for this password field. This is advisory; a particular look-and-feel may insist on its own choice of echo character. A value of `0` resets the echo character to the default.

- `char[] getPassword()`

  returns the text contained in this password field. For stronger security, you should overwrite the content of the returned array after use. (The password is not returned as a `String` because a string would stay in the virtual machine until it is garbage-collected.)

# Text Areas

```
textArea = new JTextArea(8, 40); // 8 lines of 40 columns each

textArea.setLineWrap(true); // long lines are wrapped


textArea = new JTextArea(8, 40);
var scrollPane = new JScrollPane(textArea);
```

**javax.swing.JTextArea** 1.2

- `JTextArea()`
- `JTextArea(int rows, int cols)`
- `JTextArea(String text, int rows, int cols)`

  constructs a new text area.
- `void setColumns(int cols)`

  tells the text area the preferred number of columns it should use.
- `void setRows(int rows)`

  tells the text area the preferred number of rows it should use.
- `void append(String newText)`

  appends the given text to the end of the text already in the text area.
- `void setLineWrap(boolean wrap)`

  turns line wrapping on or off.

**javax.swing.JScrollPane** 1.2

- `JScrollPane(Component c)`

  creates a scroll pane that displays the content of the specified component. Scrollbars are supplied when the component is larger than the view.

**javax.swing.JTextArea** 1.2 *(Continued)*

- `void setWrapStyleWord(boolean word)`

  If `word` is `true`, long lines are wrapped at word boundaries. If it is `false`, long lines are broken without taking word boundaries into account.
- `void setTabSize(int c)`

  sets tab stops every `c` columns. Note that the tabs aren't converted to spaces but cause alignment with the next tab stop.

# Text Areas

```
1  package text;
2
3  import java.awt.BorderLayout;
4  import java.awt.GridLayout;
5
6  import javax.swing.JButton;
7  import javax.swing.JFrame;
8  import javax.swing.JLabel;
9  import javax.swing.JPanel;
10 import javax.swing.JPasswordField;
11 import javax.swing.JScrollPane;
12 import javax.swing.JTextArea;
13 import javax.swing.JTextField;
14 import javax.swing.SwingConstants;
15
16 /**
17  * A frame with sample text components.
18  */
19 public class TextComponentFrame extends JFrame
20 {
21    public static final int TEXTAREA_ROWS = 8;
22    public static final int TEXTAREA_COLUMNS = 20;
23
24    public TextComponentFrame()
25    {
26       var textField = new JTextField();
27       var passwordField = new JPasswordField();
28
29       var northPanel = new JPanel();
30       northPanel.setLayout(new GridLayout(2, 2));
31       northPanel.add(new JLabel("User name: ", SwingConstants.RIGHT));
32       northPanel.add(textField);
33       northPanel.add(new JLabel("Password: ", SwingConstants.RIGHT));
34       northPanel.add(passwordField);
35
36       add(northPanel, BorderLayout.NORTH);
37
38       var textArea = new JTextArea(TEXTAREA_ROWS, TEXTAREA_COLUMNS);
39       var scrollPane = new JScrollPane(textArea);
40
41       add(scrollPane, BorderLayout.CENTER);
42
43       // add button to append text into the text area
44
45       var southPanel = new JPanel();
46
47       var insertButton = new JButton("Insert");
48       southPanel.add(insertButton);
49       insertButton.addActionListener(event ->
50          textArea.append("User name: " + textField.getText() + " Password: "
51             + new String(passwordField.getPassword()) + "\n"));
52
53       add(southPanel, BorderLayout.SOUTH);
54       pack();
55    }
56 }
```

# Labels

- Labels hold text. No decorations like boundaries. Never react to user input.

**javax.swing.JLabel** 1.2

- `JLabel(String text)`
- `JLabel(Icon icon)`
- `JLabel(String text, int align)`
- `JLabel(String text, Icon icon, int align)`

  constructs a label. The `align` parameter is one of the `SwingConstants` constants `LEFT` (default), `CENTER`, or `RIGHT`.

- `String getText()`
- `void setText(String text)`

  gets or sets the text of this label.

- `Icon getIcon()`
- `void setIcon(Icon icon)`

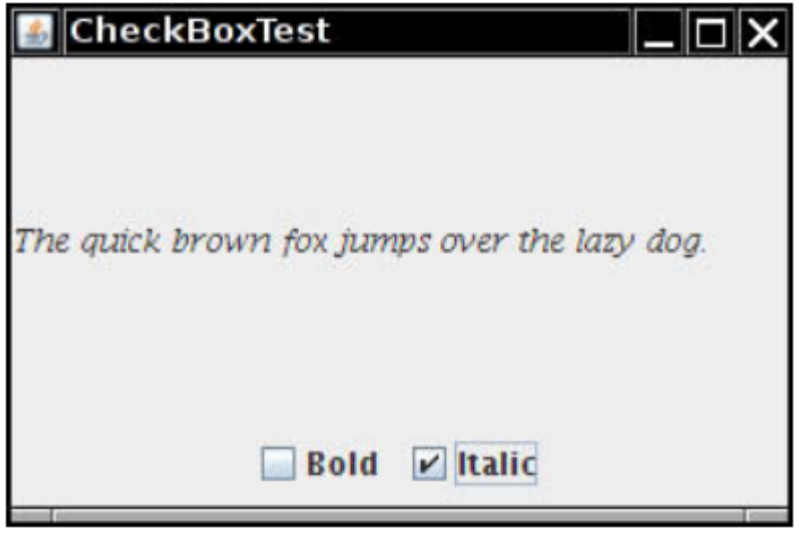
  gets or sets the icon of this label.

```
var label = new JLabel("User name: ", SwingConstants.RIGHT);
```

```
var label = new JLabel("User name: ", JLabel.RIGHT);
```

# Checkboxes

- Used to collect options.
- Come with labels that identify them.
- Upon checking action is treiggered as an ActionEvent.




```
bold = new JCheckBox("Bold");

bold.setSelected(true);

ActionListener listener = . . .;
bold.addActionListener(listener);
italic.addActionListener(listener);
```

```
ActionListener listener = event -> {
   int mode = 0;
   if (bold.isSelected()) mode += Font.BOLD;
   if (italic.isSelected()) mode += Font.ITALIC;
   label.setFont(new Font(Font.SERIF, mode, FONTSIZE));
};
```

# Radio Buttons

- – Used to select one of several options.
- – Event notification on clicking the button.

Button Label

Selected/ Unselected

```
var group = new ButtonGroup();

var smallButton = new JRadioButton("Small", false);
group.add(smallButton);

var mediumButton = new JRadioButton("Medium", true);
group.add(mediumButton);
. . .
```

# Radio Buttons

```
package radioButton;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * A frame with a sample text label and radio buttons for selecting font size
 */
public class RadioButtonFrame extends JFrame
{
   private JPanel buttonPanel;
   private ButtonGroup group;
   private JLabel label;
   private static final int DEFAULT_SIZE = 36;

   public RadioButtonFrame()
   {
      // add the sample text label

      label = new JLabel("The quick brown fox jumps over the lazy dog.");
      label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));
      add(label, BorderLayout.CENTER);

      // add the radio buttons

      buttonPanel = new JPanel();
      group = new ButtonGroup();

      addRadioButton("Small", 8);
      addRadioButton("Medium", 12);
      addRadioButton("Large", 18);
      addRadioButton("Extra large", 36);

      add(buttonPanel, BorderLayout.SOUTH);
      pack();
   }

   /**
    * Adds a radio button that sets the font size of the sample text.
    * @param name the string to appear on the button
    * @param size the font size that this button sets
    */
   public void addRadioButton(String name, int size)
   {
      boolean selected = size == DEFAULT_SIZE;
      var button = new JRadioButton(name, selected);
      group.add(button);
      buttonPanel.add(button);

      // this listener sets the label font size

      ActionListener listener = event -> label.setFont(new Font("Serif", Font.PLAIN, size));

      button.addActionListener(listener);
   }
}
```

# Combo Boxes

- Alternatives to give to users.
- E.g. font selections.
- Event on selection.
- JComboBox


**javax.swing.JComboBox** 1.2

- `boolean isEditable()`
- `void setEditable(boolean b)`

  gets or sets the `editable` property of this combo box.
- `void addItem(Object item)`

  adds an item to the item list.
- `void insertItemAt(Object item, int index)`

  inserts an item into the item list at a given index.
- `void removeItem(Object item)`

  removes an item from the item list.
- `void removeItemAt(int index)`

  removes the item at an index.
- `void removeAllItems()`

  removes all items from the item list.
- `Object getSelectedItem()`

  returns the currently selected item.

# Combo Boxes

```
var faceCombo = new JComboBox<String>();
faceCombo.addItem("Serif");
faceCombo.addItem("SansSerif");
. . .
```

- Create a JComboBox of Strings

- One can add items anywhere in the combobox list.

```
faceCombo.insertItemAt("Monospaced", 0); // add at the beginning
```

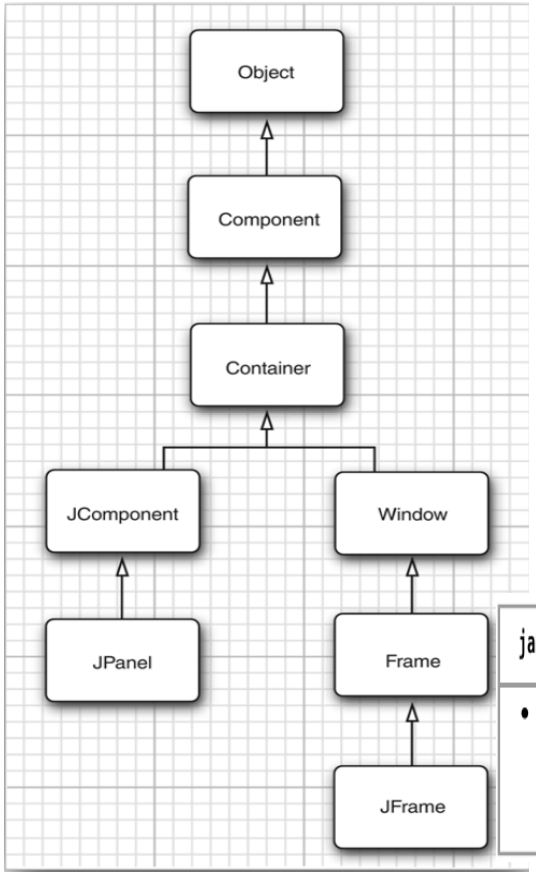
- Remove items from the combobox

```
faceCombo.removeItem("Monospaced");
faceCombo.removeItemAt(0); // remove first item
```

# Combo Boxes

```
1  package comboBox;
2
3  import java.awt.BorderLayout;
4  import java.awt.Font;
5
6  import javax.swing.JComboBox;
7  import javax.swing.JFrame;
8  import javax.swing.JLabel;
9  import javax.swing.JPanel;
10
11 /**
12  * A frame with a sample text label and a combo box for
selecting font faces.
13  */
14 public class ComboBoxFrame extends JFrame
15 {
16    private JComboBox<String> faceCombo;
17    private JLabel label;
18    private static final int DEFAULT_SIZE = 24;
19
20    public ComboBoxFrame()
21    {
22       // add the sample text label
23
24       label = new JLabel("The quick brown fox jumps over the
lazy dog.");
25       label.setFont(new Font("Serif", Font.PLAIN,
DEFAULT_SIZE));
26       add(label, BorderLayout.CENTER);
27
28       // make a combo box and add face names
29
30       faceCombo = new JComboBox<>();
31       faceCombo.addItem("Serif");
32       faceCombo.addItem("SansSerif");
33       faceCombo.addItem("Monospaced");
34       faceCombo.addItem("Dialog");
35       faceCombo.addItem("DialogInput");

39       faceCombo.addActionListener(event ->
40         label.setFont(
41           new
Font(faceCombo.getItemAt(faceCombo.getSelectedIndex()),
42             Font.PLAIN, DEFAULT_SIZE)));
43
44       // add combo box to a panel at the frame's southern
border
45
46       var comboPanel = new JPanel();
47       comboPanel.add(faceCombo);
48       add(comboPanel, BorderLayout.SOUTH);
49       pack();
50    }
51 }
```

# Frame Properties




**java.awt.Component** 1.0

- `boolean isVisible()`
- `void setVisible(boolean b)`

  gets or sets the `visible` property. Components are initially visible, with the exception of top-level components such as `JFrame`.
- `void setSize(int width, int height)` 1.1

  resizes the component to the specified width and height.
- `void setLocation(int x, int y)` 1.1

  moves the component to a new location. The `x` and `y` coordinates use the coordinates of the container if the component is not a top-level component, or the coordinates of the screen if the component is top level (for example, a `JFrame`).
- `void setBounds(int x, int y, int width, int height)` 1.1

  moves and resizes this component.
- `Dimension getSize()` 1.1
- `void setSize(Dimension d)` 1.1

  gets or sets the `size` property of this component.

**java.awt.Window** 1.0

- `void setLocationByPlatform(boolean b)` 5

  gets or sets the `locationByPlatform` property. When the property is set before this window is displayed, the platform picks a suitable location.

**java.awt.Frame** 1.0 *(Continued)*

- `Image getIconImage()`
- `void setIconImage(Image image)`

  gets or sets the `iconImage` property that determines the icon for the frame. The windowing system may display the icon as part of the frame decoration or in other locations.

**java.awt.Frame** 1.0 *(Continued)*

- `Image getIconImage()`
- `void setIconImage(Image image)`

  gets or sets the `iconImage` property that determines the icon for the frame. The windowing system may display the icon as part of the frame decoration or in other locations.

**java.awt.Frame** 1.0

- `boolean isResizable()`
- `void setResizable(boolean b)`

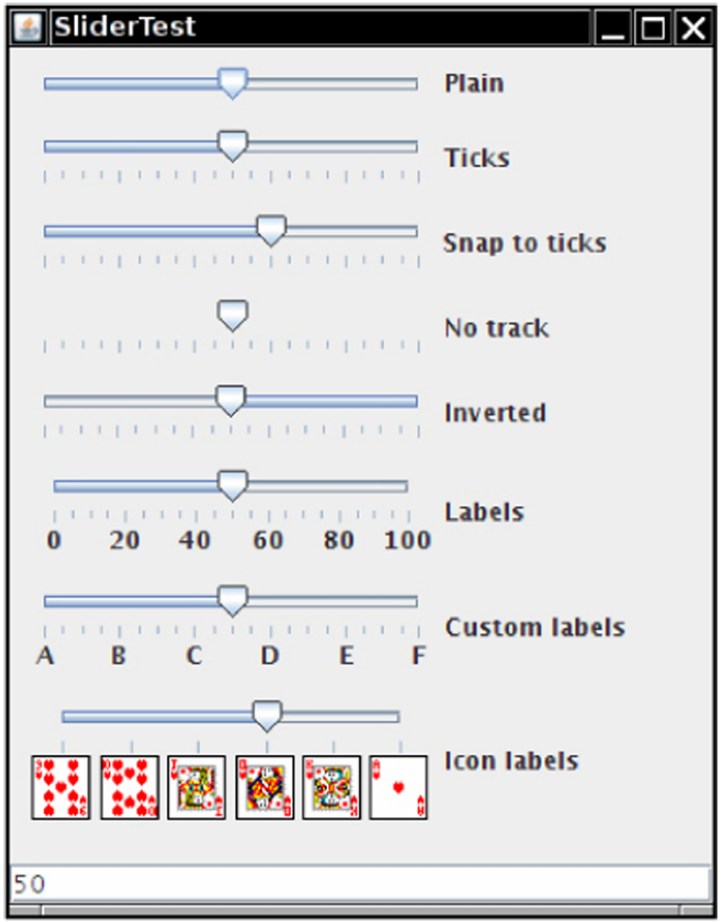
  gets or sets the `resizable` property. When the property is set, the user can resize the frame.
- `String getTitle()`
- `void setTitle(String s)`

  gets or sets the `title` property that determines the text in the title bar for the frame.

# Combo Boxes

```
1  package comboBox;
2
3  import java.awt.BorderLayout;
4  import java.awt.Font;
5
6  import javax.swing.JComboBox;
7  import javax.swing.JFrame;
8  import javax.swing.JLabel;
9  import javax.swing.JPanel;
10
11 /**
12  * A frame with a sample text label and a combo box for selecting font faces.
13  */
14 public class ComboBoxFrame extends JFrame
15 {
16    private JComboBox<String> faceCombo;
17    private JLabel label;
18    private static final int DEFAULT_SIZE = 24;
19
20    public ComboBoxFrame()
21    {
22       // add the sample text label
23
24       label = new JLabel("The quick brown fox jumps over the lazy dog.");
25       label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));
26       add(label, BorderLayout.CENTER);
27
28       // make a combo box and add face names
29
30       faceCombo = new JComboBox<>();
31       faceCombo.addItem("Serif");
32       faceCombo.addItem("SansSerif");
33       faceCombo.addItem("Monospaced");
34       faceCombo.addItem("Dialog");
35
36
37       // the combo box listener changes the label font to the selected face name
38
39       faceCombo.addActionListener(event ->
40          label.setFont(
41             new Font(faceCombo.getItemAt(faceCombo.getSelectedIndex()),
42                Font.PLAIN, DEFAULT_SIZE)));
43
44       // add combo box to a panel at the frame's southern border
45
46       var comboPanel = new JPanel();
47       comboPanel.add(faceCombo);
48       add(comboPanel, BorderLayout.SOUTH);
49       pack();
50    }
51 }
```

# Sliders




```
var slider = new JSlider(min, max, initialValue);
```

- If you omit it then the values become 0 and 100 by default.

```
ChangeListener listener = event ->
   {
     JSlider slider = (JSlider) event.getSource();
     int value = slider.getValue();
     . . .
   };
```

# Sliders

```
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);

slider.setPaintTicks(true);
slider.setPaintLabels(true);
```

- You can add ticks and change their spacings. Display using setPaintTicks() method. setPaintLabels() method displays the labels.

# Sliders

```
1  package slider;
2
3  import java.awt.*;
4  import java.util.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  /**
9   * A frame with many sliders and a text field to show slider
values.
10   */
11  public class SliderFrame extends JFrame
12  {
13     private JPanel sliderPanel;
14     private JTextField textField;
15     private ChangeListener listener;
16
17     public SliderFrame()
18     {
19        sliderPanel = new JPanel();
20        sliderPanel.setLayout(new GridBagLayout());
21
22        // common listener for all sliders
23        listener = event ->
24           {
25              // update text field when the slider value changes
26              JSlider source = (JSlider) event.getSource();
27              textField.setText("" + source.getValue());
28           };
29
30        // add a plain slider
31
32        var slider = new JSlider();
33        addSlider(slider, "Plain");
34
35        // add a slider with major and minor ticks
36
37        slider = new JSlider();
38        slider.setPaintTicks(true);
39        slider.setMajorTickSpacing(20);
40        slider.setMinorTickSpacing(5);
41        addSlider(slider, "Ticks");
42
43        // add a slider that snaps to ticks
44
45        slider = new JSlider();
46        slider.setPaintTicks(true);
47        slider.setSnapToTicks(true);
48        slider.setMajorTickSpacing(20);
49        slider.setMinorTickSpacing(5);
50        addSlider(slider, "Snap to ticks");
51
52        // add a slider with no track
```

# Sliders

```
53
54          slider = new JSlider();
55          slider.setPaintTicks(true);
56          slider.setMajorTickSpacing(20);
57          slider.setMinorTickSpacing(5);
58          slider.setPaintTrack(false);
59          addSlider(slider, "No track");
60
61          // add an inverted slider
62
63          slider = new JSlider();
64          slider.setPaintTicks(true);
65          slider.setMajorTickSpacing(20);
66          slider.setMinorTickSpacing(5);
67          slider.setInverted(true);
68          addSlider(slider, "Inverted");
69
70          // add a slider with numeric labels
71
72          slider = new JSlider();
73          slider.setPaintTicks(true);
74          slider.setPaintLabels(true);
75          slider.setMajorTickSpacing(20);
76          slider.setMinorTickSpacing(5);
77          addSlider(slider, "Labels");
```

```
79          // add a slider with alphabetic labels
80
81          slider = new JSlider();
82          slider.setPaintLabels(true);
83          slider.setPaintTicks(true);
84          slider.setMajorTickSpacing(20);
85          slider.setMinorTickSpacing(5);
86
87          var labelTable = new Hashtable<Integer, Component>();
88          labelTable.put(0, new JLabel("A"));
89          labelTable.put(20, new JLabel("B"));
90          labelTable.put(40, new JLabel("C"));
91          labelTable.put(60, new JLabel("D"));
92          labelTable.put(80, new JLabel("E"));
93          labelTable.put(100, new JLabel("F"));
94
95          slider.setLabelTable(labelTable);
96          addSlider(slider, "Custom labels");
97
98          // add a slider with icon labels
99
100         slider = new JSlider();
101         slider.setPaintTicks(true);
102         slider.setPaintLabels(true);
103         slider.setSnapToTicks(true);
104         slider.setMajorTickSpacing(20);
105         slider.setMinorTickSpacing(20);
```

# Sliders

```
106
107         labelTable = new Hashtable<Integer, Component>();
108
109         // add card images
110
111         labelTable.put(0, new JLabel(new
ImageIcon("nine.gif")));
112         labelTable.put(20, new JLabel(new
ImageIcon("ten.gif")));
113         labelTable.put(40, new JLabel(new
ImageIcon("jack.gif")));
114         labelTable.put(60, new JLabel(new
ImageIcon("queen.gif")));
115         labelTable.put(80, new JLabel(new
ImageIcon("king.gif")));
116         labelTable.put(100, new JLabel(new
ImageIcon("ace.gif")));
117
118         slider.setLabelTable(labelTable);
119         addSlider(slider, "Icon labels");
120
121         // add the text field that displays the slider value
122
123         textField = new JTextField();
124         add(sliderPanel, BorderLayout.CENTER);
125         add(textField, BorderLayout.SOUTH);
126         pack();
127     }
128
129     /**
130      * Adds a slider to the slider panel and hooks up the
listener
131      * @param slider the slider
132      * @param description the slider description
133      */
134     public void addSlider(JSlider slider, String description)
135     {
136         slider.addChangeListener(listener);
137         var panel = new JPanel();
138         panel.add(slider);
139         panel.add(new JLabel(description));
140         panel.setAlignmentX(Component.LEFT_ALIGNMENT);
141         var gbc = new GridBagConstraints();
142         gbc.gridy = sliderPanel.getComponentCount();
143         gbc.anchor = GridBagConstraints.WEST;
144         sliderPanel.add(panel, gbc);
145     }
146 }
```

# Menus

- Menu bars > Menu items > Submenu items

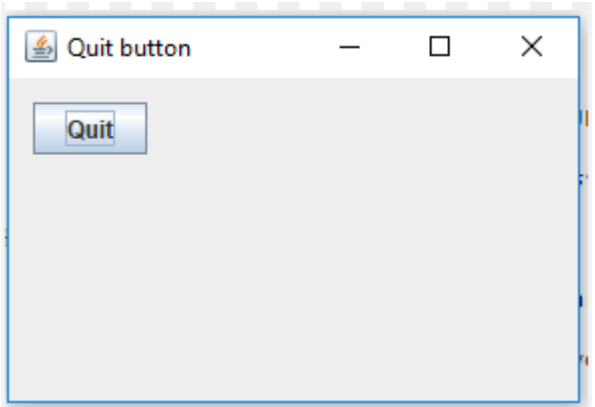

```
var menuBar = new JMenuBar();

var editMenu = new JMenu("Edit");

menuBar.add(editMenu);
```

```
var pasteItem = new JMenuItem("Paste");
editMenu.add(pasteItem);
editMenu.addSeparator();
JMenu optionsMenu = . . .; // a submenu
editMenu.add(optionsMenu);
```

```
ActionListener listener = . . .;
pasteItem.addActionListener(listener);
```

# Menus

- Using menus to trigger exit action.




```
var exitAction = new AbstractAction("Exit") // menu item text
goes here
   {
      public void actionPerformed(ActionEvent event)
      {
         // action code goes here
         System.exit(0);
      }
   };


   var exitItem = new JMenuItem(exitAction);
   fileMenu.add(exitItem);
```

# Menus

```
1  package menu;
2
3  import java.awt.event.*;
4  import javax.swing.*;
5
6  /**
7   * A frame with a sample menu bar.
8   */
9  public class MenuFrame extends JFrame
10 {
11    private static final int DEFAULT_WIDTH = 300;
12    private static final int DEFAULT_HEIGHT = 200;
13    private Action saveAction;
14    private Action saveAsAction;
15    private JCheckBoxMenuItem readonlyItem;
16    private JPopupMenu popup;
17
18    /**
19     * A sample action that prints the action name to
System.out.
20     */
21    class TestAction extends AbstractAction
22    {
23       public TestAction(String name)
24       {
25          super(name);
26       }
27
28       public void actionPerformed(ActionEvent event)
29       {
30          System.out.println(getValue(Action.NAME) + "
selected.");
31       }
32    }
33
34    public MenuFrame()
35    {
36       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
37
38       var fileMenu = new JMenu("File");
39       fileMenu.add(new TestAction("New"));
40
41       // demonstrate accelerators
42
43       var openItem = fileMenu.add(new TestAction("Open"));
44       openItem.setAccelerator(KeyStroke.getKeyStroke("ctrl
O"));
45
46       fileMenu.addSeparator();
47
48       saveAction = new TestAction("Save");
49       JMenuItem saveItem = fileMenu.add(saveAction);
50       saveItem.setAccelerator(KeyStroke.getKeyStroke("ctrl
S"));
51
52       saveAsAction = new TestAction("Save As");
53       fileMenu.add(saveAsAction);
54       fileMenu.addSeparator();
55
56       fileMenu.add(new AbstractAction("Exit")
57          {
58             public void actionPerformed(ActionEvent event)
59             {
60                System.exit(0);
61             }
62          });
63
64        // demonstrate checkbox and radio button menus
65
66        readonlyItem = new JCheckBoxMenuItem("Read-only");
```

# Menus

```
67         readonlyItem.addActionListener(new ActionListener()
68           {
69              public void actionPerformed(ActionEvent event)
70              {
71                 boolean saveOk = !readonlyItem.isSelected();
72                 saveAction.setEnabled(saveOk);
73                 saveAsAction.setEnabled(saveOk);
74              }
75           });
76
77         var group = new ButtonGroup();
78
79         var insertItem = new JRadioButtonMenuItem("Insert");
80         insertItem.setSelected(true);
81         var overtypeItem = new
JRadioButtonMenuItem("Overtype");
82
83         group.add(insertItem);
84         group.add(overtypeItem);
85
86         // demonstrate icons
87
88         var cutAction = new TestAction("Cut");
89         cutAction.putValue(Action.SMALL_ICON, new
ImageIcon("cut.gif"));
90         var copyAction = new TestAction("Copy");
91         copyAction.putValue(Action.SMALL_ICON, new
ImageIcon("copy.gif"));
92         var pasteAction = new TestAction("Paste");
93         pasteAction.putValue(Action.SMALL_ICON, new
ImageIcon("paste.gif"));
94
95         var editMenu = new JMenu("Edit");
96         editMenu.add(cutAction);
97         editMenu.add(copyAction);
98         editMenu.add(pasteAction);
99
100         // demonstrate nested menus
```

```
102         var optionMenu = new JMenu("Options");
103
104         optionMenu.add(readonlyItem);
105         optionMenu.addSeparator();
106         optionMenu.add(insertItem);
107         optionMenu.add(overtypeItem);
108
109         editMenu.addSeparator();
110         editMenu.add(optionMenu);
111
112         // demonstrate mnemonics
113
114         var helpMenu = new JMenu("Help");
115         helpMenu.setMnemonic('H');
116
117         var indexItem = new JMenuItem("Index");
118         indexItem.setMnemonic('I');
119         helpMenu.add(indexItem);
120
121         // you can also add the mnemonic key to an action
122         var aboutAction = new TestAction("About");
123         aboutAction.putValue(Action.MNEMONIC_KEY,
Integer.valueOf('A'));
124         helpMenu.add(aboutAction);
125
126         // add all top-level menus to menu bar
127
128         var menuBar = new JMenuBar();
129         setJMenuBar(menuBar);
130
131         menuBar.add(fileMenu);
132         menuBar.add(editMenu);
133         menuBar.add(helpMenu);
134
135         // demonstrate pop-ups
136
137         popup = new JPopupMenu();
138         popup.add(cutAction);
139         popup.add(copyAction);
```

# Menus

```
140         popup.add(pasteAction);
141
142         var panel = new JPanel();
143         panel.setComponentPopupMenu(popup);
144         add(panel);
145     }
146 }
```

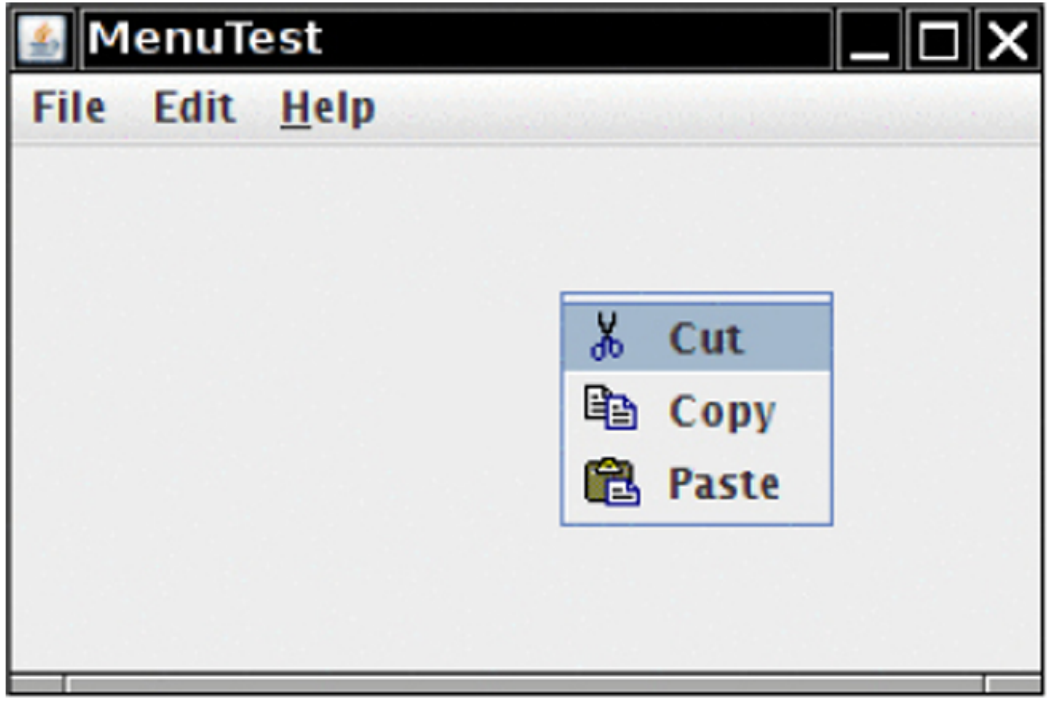
# Checkboxes and Radio Buttons in Menu Items

```
var readonlyItem = new JCheckBoxMenuItem("Read-only");
optionsMenu.add(readonlyItem);
```

```
var group = new ButtonGroup();
var insertItem = new JRadioButtonMenuItem("Insert");
insertItem.setSelected(true);
var overtypeItem = new JRadioButtonMenuItem("Overtype");
group.add(insertItem);
group.add(overtypeItem);
optionsMenu.add(insertItem);
optionsMenu.add(overtypeItem);
```

# Pop-Up Menus

- Floating menus not attached to a menu bar




```
var popup = new JPopupMenu();

var item = new JMenuItem("Cut");
item.addActionListener(listener);
popup.add(item);

popup.show(panel, x, y);
```

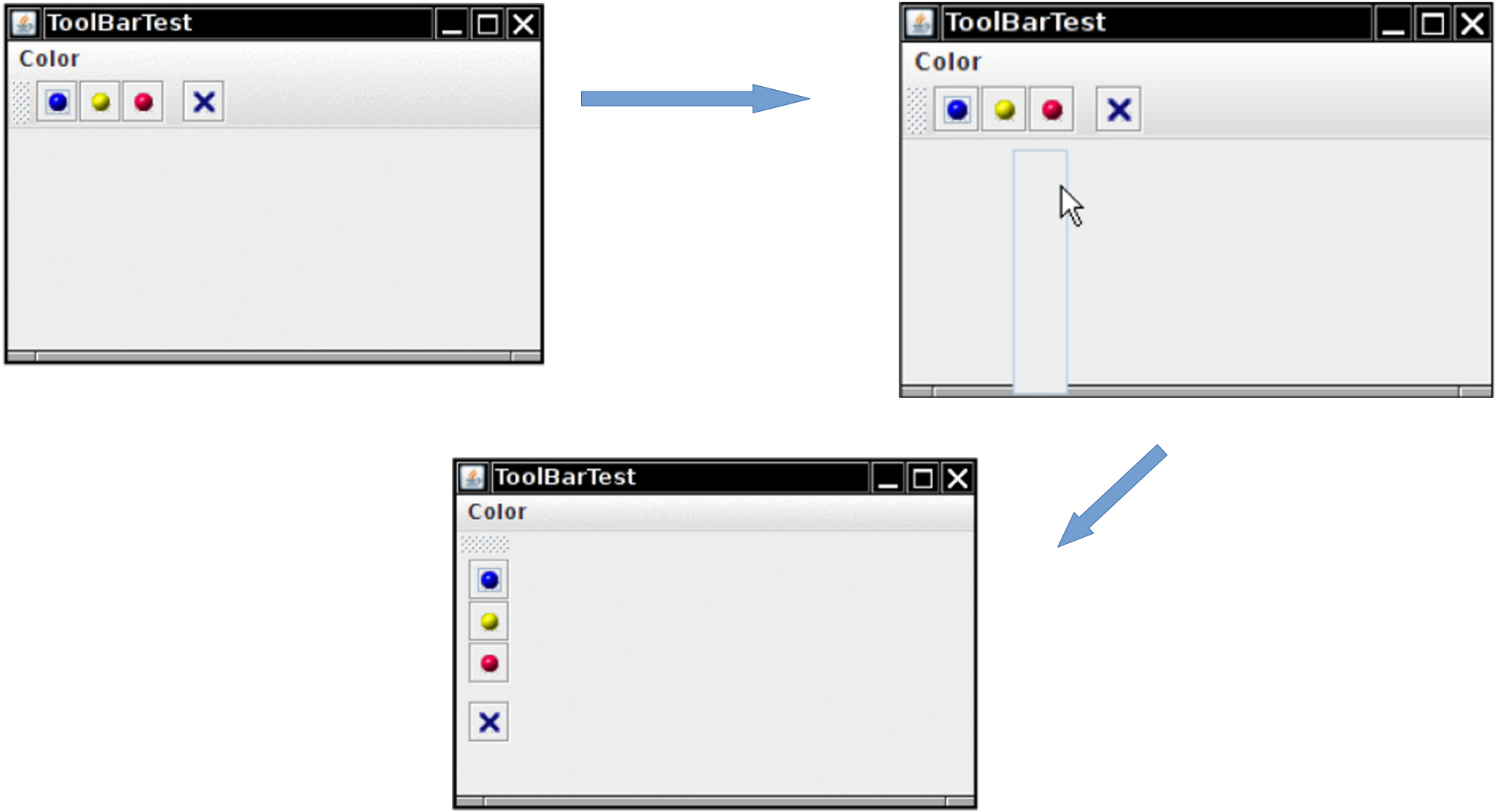
- Shows up when a user clicks on a Component.

```
component.setComponentPopupMenu(popup);
```

# *Enabling and Disabling Menu Items*

```
saveItem.setEnabled(false);
```

# Toolbars



# Toolbars

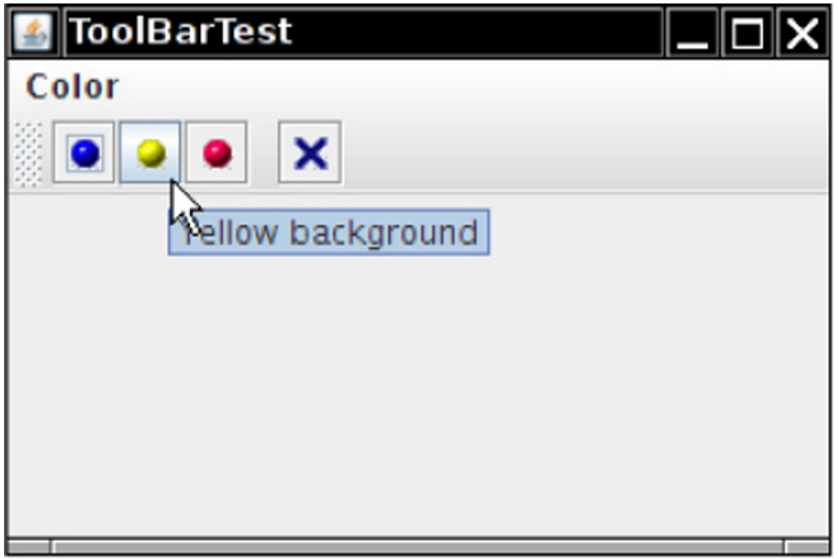
```
var toolbar = new JToolBar();
toolbar.add(blueButton);
```

```
toolbar.add(blueAction);
```

```
toolbar.addSeparator();
```

```
toolbar = new JToolBar(titleString);
```

# Tooltip




- Every JComponent has a built-in tooltip that can be shown using setToolTipText() method.

```
exitButton.setToolTipText("Exit");
```