Other Events

Slides adapted from slides of Marty Stepp based on materials by M. Ernst, S. Reges, D. Notkin, R. Mercer, Wikipedia
http://www.cs.washington.edu/331/
And Bob Myers Florida State University https://www.cs.fsu.edu/~myers/cgs3416/

Focus events

```
public interface FocusListener {
    public void focusGained(FocusEvent event);
    public void focusLost(FocusEvent event);
}
```

- **focus**: The current target of keyboard input.
  - A *focus event* occurs when the keyboard cursor enters or exits a component, signifying the start/end of typing on that control.

- Many AWT/Swing components have this method:
  - `public void addFocusListener(FocusListener kl)`

- The abstract class `FocusAdapter` implements all `FocusListener` methods with empty bodies.

Focus Event example

```
import java.awt.*;
import java.awt.event.*;
class TextFieldFocusEvent extends Frame implements FocusListener
{
    TextField t1,t2;

    public TextFieldFocusEvent()
    {
        createAndShowGUI();
    }

    private void createAndShowGUI()
    {
        setTitle("FocusListener for TextField");
        setLayout(new FlowLayout());

        // Create 2 textfields
        t1=new TextField(20);
        t2=new TextField(20);

        // Add them
        add(t1);
        add(t2);

        // Add FocusListeners
        t1.addFocusListener(this);
        t2.addFocusListener(this);

        setSize(400,400);
        setVisible(true);
    }
```

```
    public void focusGained(FocusEvent fe)
    {
        // Get what textfield got focus
        TextField t=(TextField)fe.getSource();
        t.setBackground(Color.LIGHT_GRAY);
    }

    public void focusLost(FocusEvent fe)
    {
        // Get what textfield lost focus
        TextField t=(TextField)fe.getSource();
        t.setBackground(Color.WHITE);
    }

    public static void main(String args[])
    {
        new TextFieldFocusEvent();
    }
}
```

Component events

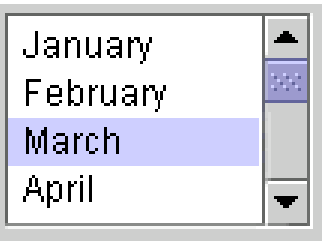
```
public interface ComponentListener {
    public void componentHidden(ComponentEvent event);
    public void componentMoved(ComponentEvent event);
    public void componentResized(ComponentEvent event);
    public void componentShown(ComponentEvent event);
}
```

- These events occur when a layout manager reshapes a component or when it is set to be visible or invisible.
- Many AWT/Swing components have this method:
  - `public void addComponentListener(ComponentListener cl)`
- The abstract class `ComponentAdapter` implements all `ComponentListener` methods with empty bodies.

JList select event

```
public interface ListSelectionListener {
    public void valueChanged(ListSelectionEvent event);
}
```

- These events occur when a user changes the element(s) selected within a `JList`.
- The `JList` component has this method:
  - `public void addListSelectionListener( ListSelectionListener lsl)`



Document events

```
public interface DocumentListener {
    public void changedUpdate(DocumentEvent event);
    public void insertUpdate(DocumentEvent event);
    public void removeUpdate(DocumentEvent event);
}
```

- These events occur when the contents of a text component (e.g. JTextField or JTextArea) change.
- Such components have this method:
  - `public Document getDocument()`
- And a `Document` object has this method:
  - `public void addDocumentListener(DocumentListener cl)`
- And yes, there is a `DocumentAdapter`.

Item events

```
public interface ItemListener {
    public void itemStateChanged(ItemEvent event);
}
```

- These events occur when an item has been selected or deselected by the user.
  - `public void addItemListener (ItemListener il)`
- Used with JCheckBox, JComboBox.

Item events

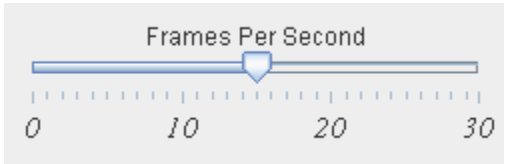
Object	**getItem**()Returns the item affected by the event.
int	**getStateChange**()Returns the type of state change (selected or deselected).
object	**getSource**() Returns the object on which the event occurred

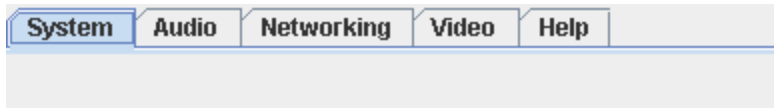
static int	**DESELECTED**This state-change-value indicates that a selected item was deselected.
static int	**SELECTED**This state-change value indicates that an item was selected.

Change events

```
public interface ChangeListener {
    public void stateChanged(ChangeEvent event);
}
```

- These events occur when some kind of state of a given component changes. Not used by all components, but essential for some.
- `JSpinner`, `JSlider`, `JTabbedPane`, `JColorChooser`, `JViewPort`, and other components have this method:
  - `public void addChangeListener(ChangeListener cl)`





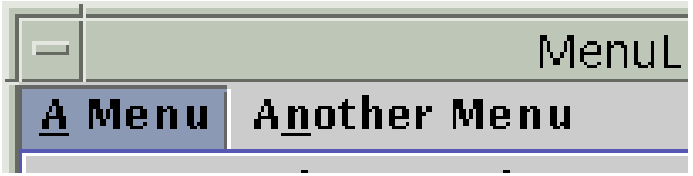
Other Components

JMenu

- Allows for performing actions without cluttering GUI
- Can be used for `JFrame` or `JApplet` objects
- Contained by menu bar
  - `JMenuBar` (requires `setJMenuBar`)
- Comprised of menu items
  - `JMenuItem`
  - `JCheckBoxMenuItem`
  - `JRadioButtonMenuItem`
- When menu is clicked, expands to show menu items
- `JMenuItem` can be a `JMenu` to have a sub-menu
- `addActionListener` for each menu item

JMenuBar

a drop-down menu of commands



- `public JMenuBar()`
- `public void add(JMenu menu)`

Usage: in `JFrame`, the following method exists:

- `public void setJMenuBar(JMenuBar bar)`

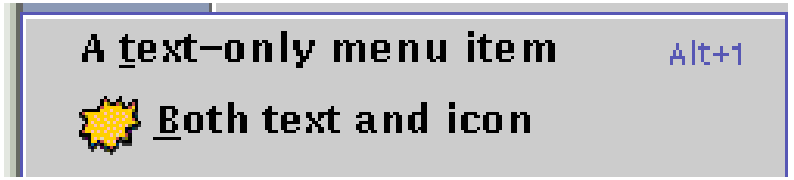
JMenu

a sub-menu of commands with a JMenuBar

- `public JMenu(String text)`
- `public void add(JMenuItem item)`
- `public void addSeparator()`
- `public void setMnemonic(int key)`

JMenuItem

an entry within a JMenu that can be clicked to execute a command



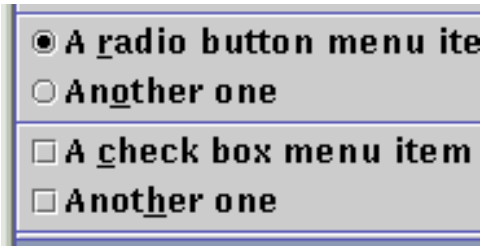
- `public JMenuItem(String text)`
- `public JMenuItem(String text, Icon icon)`
- `public JMenuItem(String text, int mnemonic)`

- `public void setAccelerator(KeyStroke ks)`
- `public void setEnabled(boolean b)`
- `public void setMnemonic(int mnemonic)`

- `public void addActionListener(ActionListener al)`

J(CheckBox|RadioButton)MenuItem

a JMenuItem with a check box or radio circle

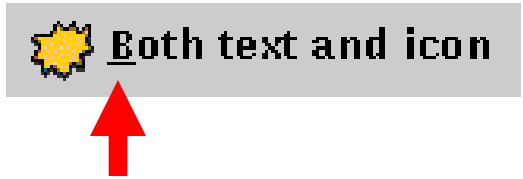


- `public J_____MenuItem(String text)`
- `public J_____MenuItem(String text, boolean selected)`
- `public J_____MenuItem(String text, Icon icon)`
- `public J_____MenuItem(String text, Icon icon, boolean selected)`
- `public void addActionListener(ActionListener al)`
- `public boolean isSelected()`
- `public void setSelected(boolean b)`

Recall: in a `ButtonGroup`, the following method exists:

- `public void add(AbstractButton button)`
- These two classes extend `AbstractButton`.

Mnemonics



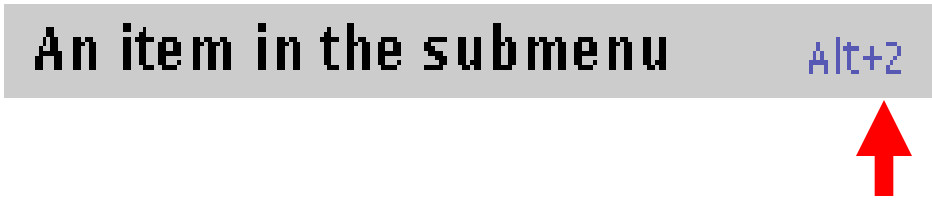
- **mnemonic:** A context-sensitive menu hotkey assigned to a specific button or other graphical component.
  - Usually visible as an underlined key, activated by pressing Alt+*key*.
  - Only works when input focus is on the appropriate component.

  - *usage:* call `setMnemonic(char)` method
    - Menu items also have a constructor that takes a mnemonic.

```
myQuitButton.setMnemonic('Q');
JMenuItem myNewItem = new JMenuItem("New", 'N');
// or: myNewItem.setMnemonic('N');
```

Accelerators

- **accelerator**: A global hotkey that performs an action (ex: Alt-X to exit the program) even on components that aren't in focus / visible.



  - Can be run at any time in the application.
  - Can optionally include modifiers like Shift, Alt.
  - To create an accelerator:
    - Call the static `getKeyStroke` factory method of the `KeyStroke` class.
    - Pass its result to the `setAccelerator` method of the component.

```
menuItem.setAccelerator(
    KeyStroke.getKeyStroke('T',KeyEvent.ALT_MASK));
```

JSlider

- Enable users to select from range of integer values
- Several features
  - Tick marks (major and minor)
  - Snap-to ticks
  - Orientation (horizontal and vertical)
- `paint` draws on subclasses of `JFrame` and `Japplet`
- `paintComponent` draws on subclasses of `JComponent`