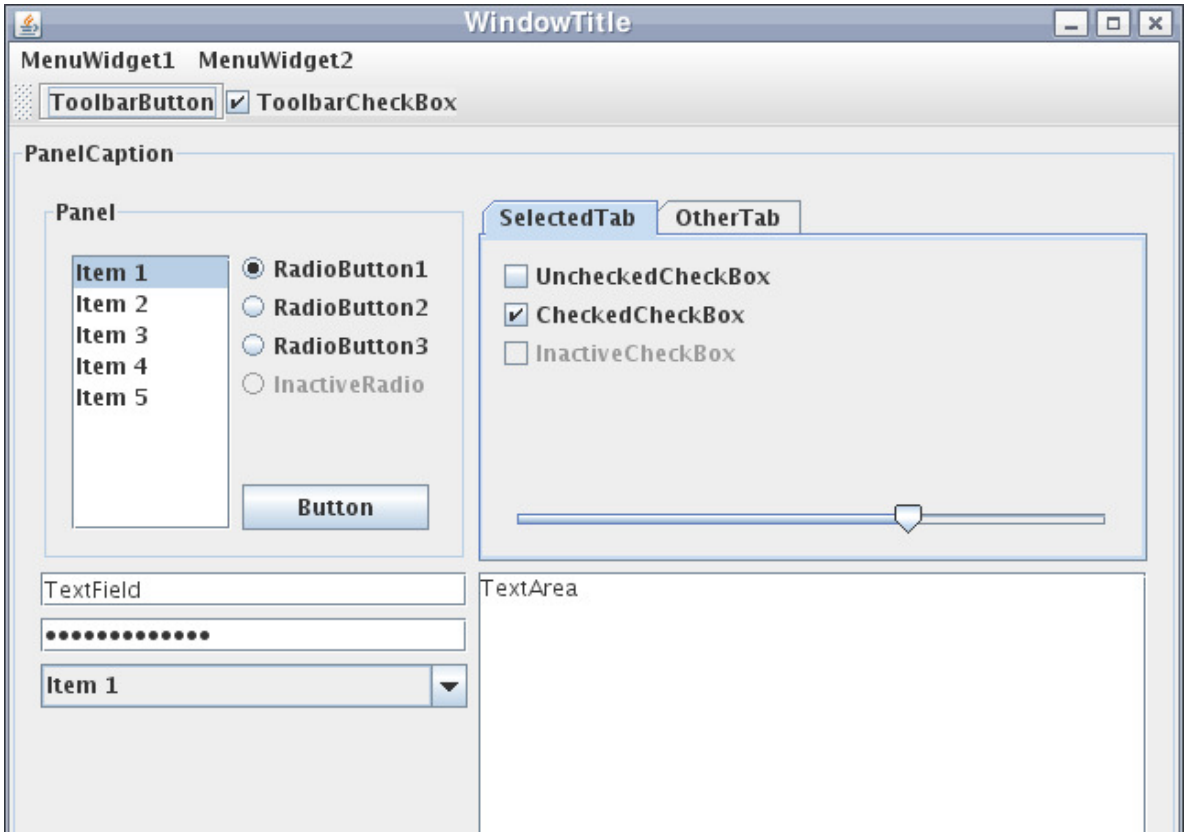
CS 106A, Lecture 23
Graphical User Interfaces (GUIs) with Swing, part 1

reading:

Art & Science of Java, Chapter 10

Lecture at a glance

- Today we will begin learning about how to make a **GUI** (graphical user interface).
  - Most modern applications have graphical user interfaces.
  - We will learn about various GUI components, layout, and events.

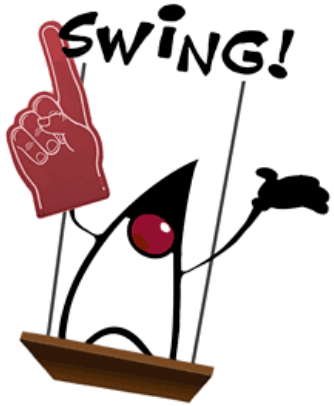


Java GUI History

- **Abstract Windowing Toolkit** (**AWT**): Java's first cross-platform GUI library. *(Java 1.0 - 1.1)*
  - Limited to lowest common denominator; clunky to use.

- **Swing**: Newer GUI library with more powerful features. *(Java 1.2+)*
  - More features; compatibility; OO design.
  - Both AWT and Swing exist now; have to use both in various places.

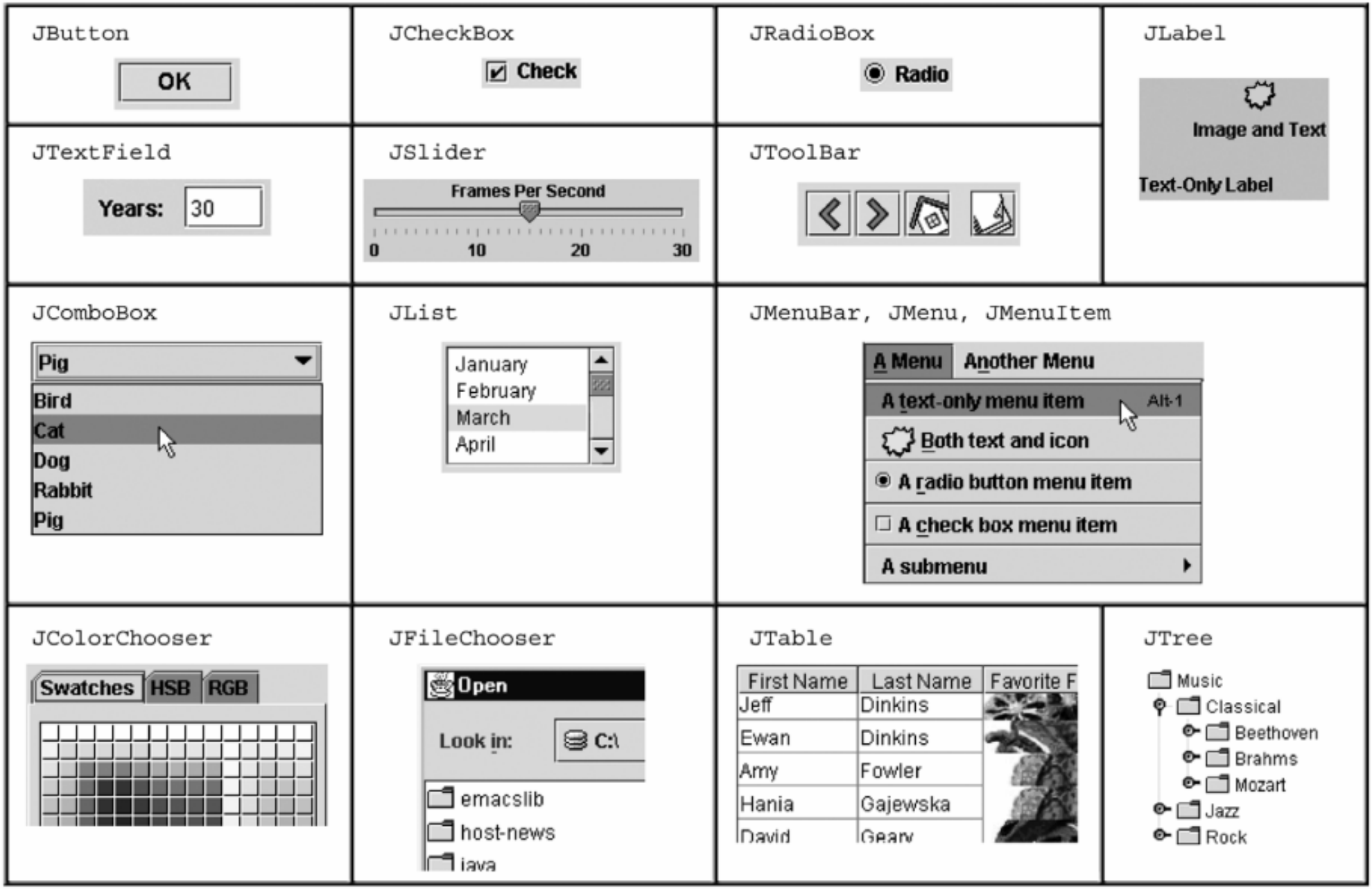
- **Stanford** has built some graphical libraries on top of AWT/Swing.



GUI terminology

- **window**: A first-class citizen of the graphical desktop.
  - examples: frame, dialog box, applet
- **component**: An interactive GUI object that resides in a window.
  - Also called *controls* or *interactors,* or *widgets*.
  - examples: button, text area, checkbox, label, canvas
- **container**: A logical grouping for storing components.
  - examples: panel, table, box, scroll pane

Components


JButton
OK
JCheckBox
Check
JRadioBox
Radio
JLabel
Image and Text
Text-Only Label
JTextField
Years:
30
JSlider
Frames Per Second
0
10
20
30
JToolBar
JComboBox
Pig
Bird
Cat
Dog
Rabbit
Pig
JList
January
February
March
April
JMenuBar, JMenu, JMenuItem
A Menu
Another Menu
A text-only menu item
Alt-1
Both text and icon
A radio button menu item
A check box menu item
A submenu
JColorChooser
Swatches
HSB
RGB
JFileChooser
Open
Look in:
C:\
emacslib
host-news
java
JTable
First Name
Last Name
Favorite F
Jeff
Dinkins
Ewan
Dinkins
Amy
Fowler
Hania
Gajewska
David
Geary
JTree
Music
Classical
Beethoven
Brahms
Mozart
Jazz
Rock

GUI programs

When writing a GUI program, make the following changes:

- **Import** packages: `acm.gui`, `java.awt`, `javax.swing`

- Change your class header.
  - from: `extends GraphicsProgram`
  - to: `extends` **`Program`**

- Change the program's `run` method to **`init`**.
  - `init` is for setting up components and then waiting for events.
  - `run` is for animation loops and resource loading (input files).

- In `init`, create your **components** and add them to the window.

GUI program template

```
import acm.program.*;
import acm.gui.*;          // Stanford graphic components
import java.awt.*;         // Java graphical objects
import javax.swing.*;      // Java graphical objects

public class Name extends Program {
    public void init() {
        // the code to add components to the window
        statements;
    }
}
```

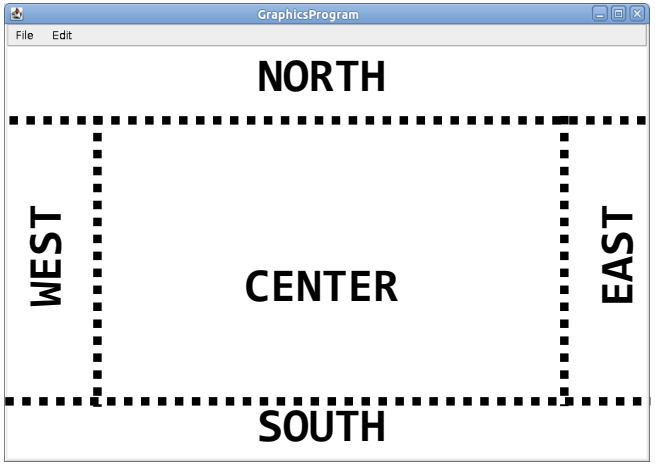
JButton

a clickable region for causing actions to occur

Method	Description
`new JButton("text")`	Creates new button with given text string
`jb.getBackground()` `jb.setBackground(color);`	get or set background color on button
`jb.isEnabled()` `jb.setEnabled(boolean);`	get or set whether button is clickable
`jb.getFont()` `jb.setFont(font);`	get or set text font used for button text
`jb.getForeground()` `jb.setForeground(color);`	get or set text color on button
`jb.getIcon()` `jb.setIcon(icon);`	get or set icon image showing on button
`jb.getText()` `jb.setText("text");`	set or return text showing on the button

Window regions

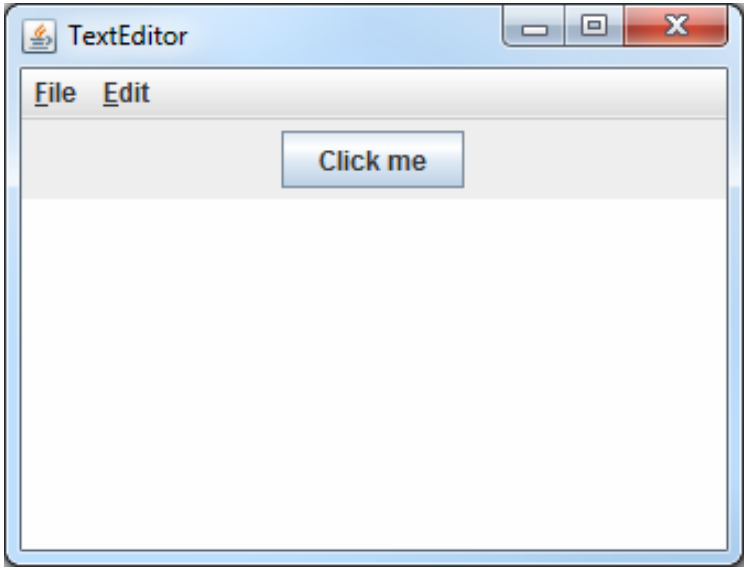
- The content of a graphical program is divided into five regions:
  - NORTH, SOUTH, EAST, WEST, and CENTER



- You can add a component to a region of the graphical window by calling:

  add(***component***, ***REGION***);

```
JButton jb = new JButton("Click me");
add(jb, NORTH);
```



Component properties

- Each has a `get` (or `is`) accessor and a `set` modifier method.
- examples: `getColor`, `setFont`, `setEnabled`, `isVisible`

name	type	description
get/setBackground	`Color`	background color behind component
get/setBorder	`Border`	border line around component
is/setEnabled	`boolean`	whether it can be interacted with
is/setFocusable	`boolean`	whether key text can be typed on it
get/setFont	`Font`	font used for text in component
get/setForeground	`Color`	foreground/text color of component
get/setHeight, Width	`int`	component's current size in pixels
is/setVisible	`boolean`	whether component can be seen
get/setTooltipText	`String`	text shown when hovering mouse
get/setSize, Minimum / Maximum / PreferredSize	`Dimension`	various sizes, size limits, or desired sizes that the component may take

JLabel

a string of text displayed on screen in a graphical program to give information or describe components

Method	Description
`new JLabel("text")`	Create new label with given text
`jl.getFont()` `jl.setFont(font);`	get/set text font used for label text
`jl.getForeground()` `jl.setForeground(color);`	get or set text color on label
`jl.getHorizontalAlignment()` `jl.setHorizontalAlignment(align);`	set or return horizontal alignment of text in the label; pass `JLabel.LEFT`, `CENTER`, or `RIGHT`
`jl.getText()` `jl.setText("text");`	set/return text in the label

Font

Arial
Arial Black
Comic Sans MS
Courier New
Georgia
Impact
Times New Roman
Trebuchet MS
Verdana

a typeface, a style and shape of letters for displaying text on a computer screen

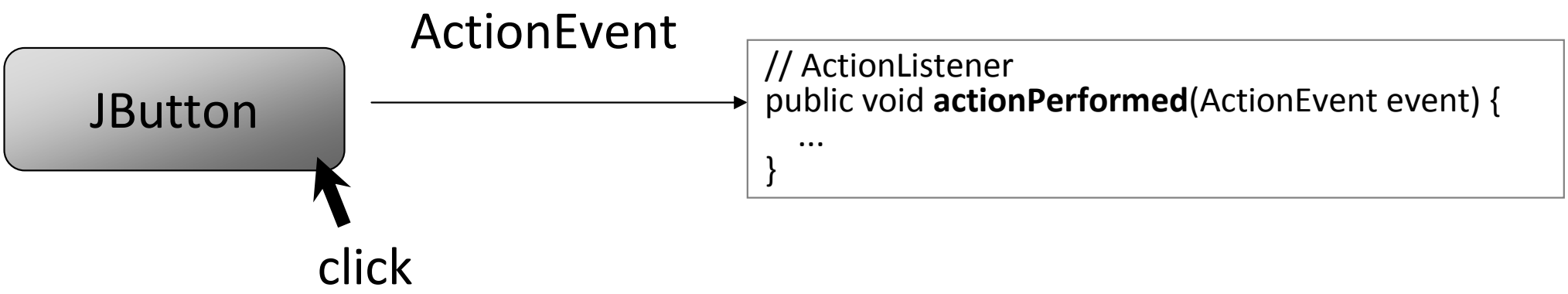
```
Font name = new Font("name", Font.STYLE, size);
```

- *name* can be a specific font such as "Times New Roman" or a general string such as "Monospaced", "Serif", or "SansSerif".
- *STYLE* must be one of `Font.PLAIN`, `BOLD`, `ITALIC`, or a mix.
- *size* is the font size as an integer, such as 24 for 24pt.

Action events

- **action event**: An action that has occurred on a GUI component.
  - user clicks on a button or other component
  - user checks/unchecks a check box or radio button
  - user presses Enter in a text field
  - ...

  - Handled by classes that implement a method `actionPerformed`



Event programs

When writing a program with action events:

- **Import** package: `java.awt.event`
- Write a method **`actionPerformed`**.
  - In here, put the code to run when various buttons are clicked.
- At the <u>end</u> of `init`, call **`addActionListeners();`**
  - Makes it so that `actionPerformed` will be called when buttons are clicked in your program.

Program w/action event

```
import acm.program.*;
import acm.gui.*;          // Stanford graphic components
import java.awt.*;         // Java graphical objects
import java.awt.event.*;   // for Java events
import javax.swing.*;      // Java graphical objects

public class Name extends Program {
    public void init() {
        statements;
        addActionListeners();   // enable action events
    }

    // the code to run when the event occurs
    public void actionPerformed(ActionEvent event) {
        ...
    }
}
```

ActionEvent objects

- The **`ActionEvent`** parameter contains useful event information.
  - Use `getSource` or `getActionCommand` to figure out what button or component was interacted with.

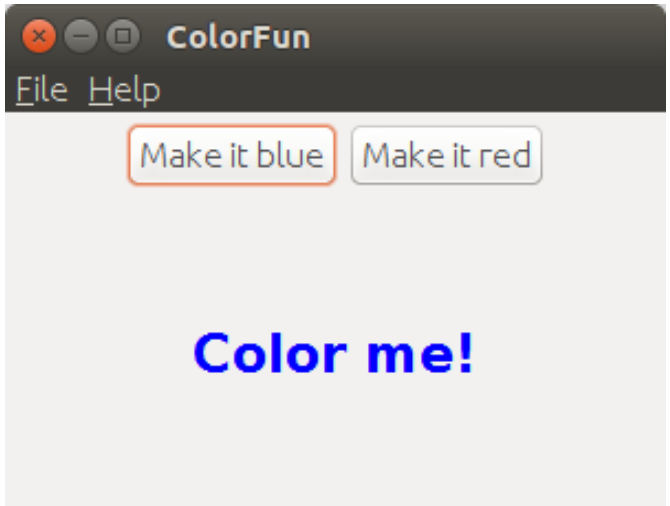
Method	Description
e`.getActionCommand()`	a text description of the event *(e.g. the text of the button clicked)*
e`.getSource()`	the interactor that generated the event

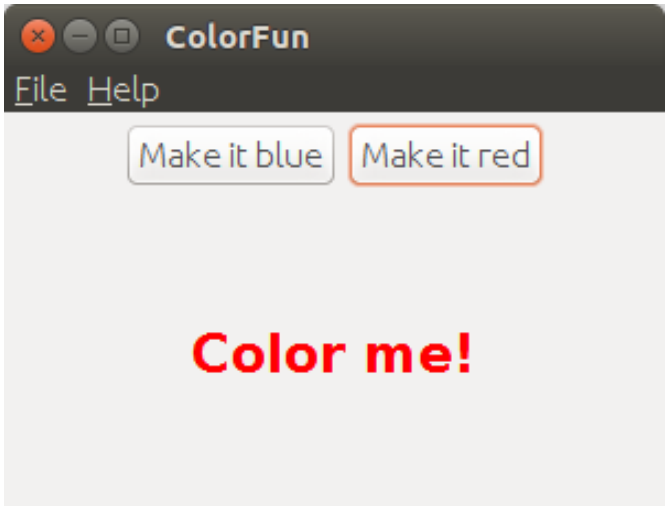
```
public void actionPerformed(ActionEvent event) {
    String command = event.getActionCommand();
    if (command.equals("Save File")) {
        // user clicked the Save File button
        ...
    }
}
```

ColorFun exercise

- Write a GUI program named **`ColorFun`** :
  - window size: 300 x 200
  - window title: "ColorFun"

  - The top of the window contains "blue" and "red" buttons.
  - When each button is clicked, a label's color changes.





ColorFun solution

```
public class ColorFun extends Program {
    private JButton blueButton;
    private JButton redButton;
    private JLabel middle;

    public void init() {
        setSize(300, 200);
        blueButton = new JButton("Make it blue");
        add(blueButton, NORTH);
        redButton = new JButton("Make it red");
        add(redButton, NORTH);
        middle = new JLabel("Color me!");
        middle.setFont(new Font("SansSerif", Font.BOLD, 24));
        middle.setHorizontalAlignment(JLabel.CENTER);
        add(middle);
        addActionListeners();
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == blueButton) {
            middle.setForeground(Color.BLUE);
        } else {
            middle.setForeground(Color.RED);
        }
    }
}
```

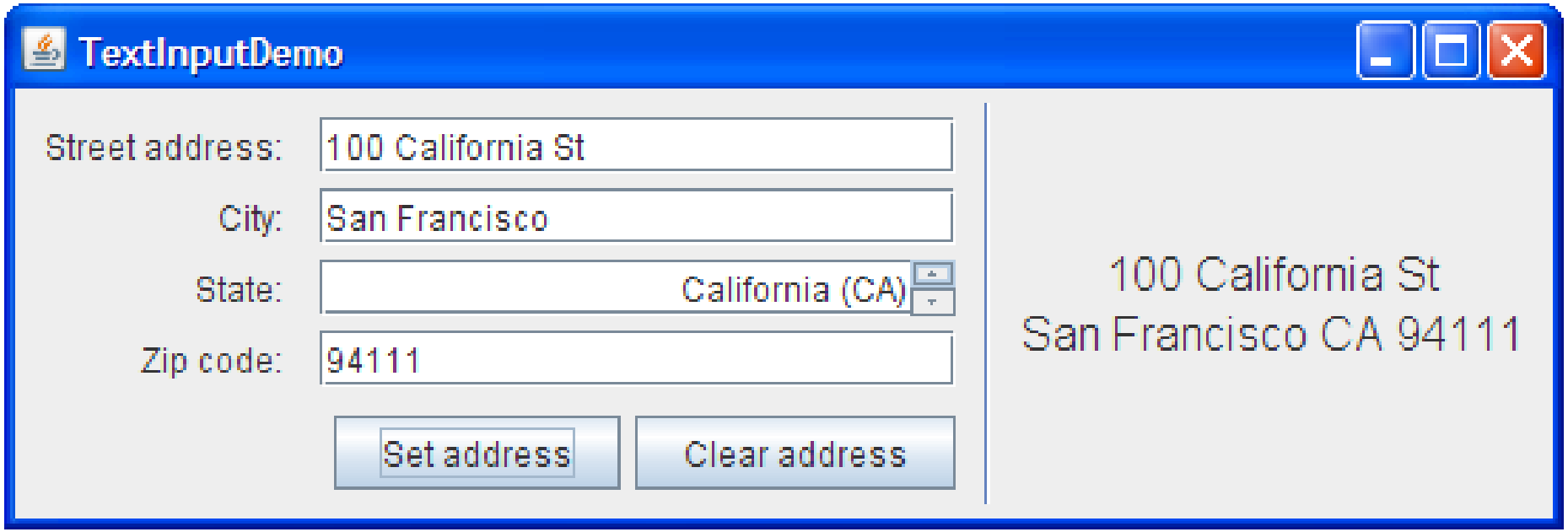
JTextField

a single-line input control for typing text values

Method	Description
`new JTextField("text")` `new JTextField(columns)`	Create new text field of given size
`jtf.addActionListener(this);`	causes action events to occur when the user presses Enter on the field
`jtf.getActionCommand()` `jtf.setActionCommand("cmd");`	set/return a string to identify the action events that will occur in this field
`jtf.getText()` `jtf.setText("text");`	set/return text in the field

Types of text fields

- `JTextField`: Accepts any text as input.
- `IntField`: Only accepts ints; will re-prompt on bad data.
- `DoubleField`: Only accepts doubles; will re-prompt on bad data.
- `JSpinner`: Like an `IntField` with up/down buttons.



JCheckBox, JRadioButton

a toggleable yes/no value (checkbox)
or a way choose between options (radio)

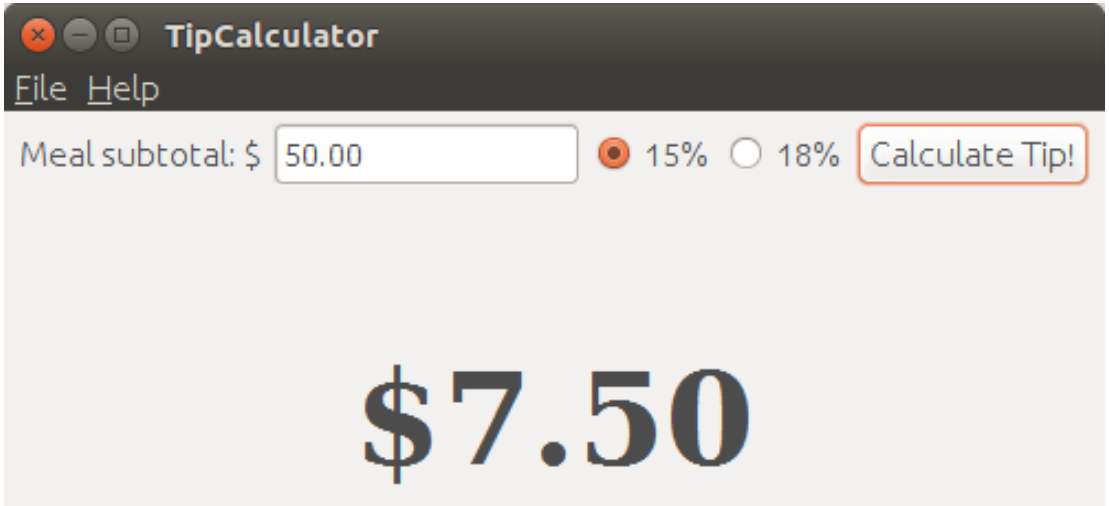
Method	Description

`new JCheckBox("`***text***`")`
`new JCheckBox("`***text***`", `***checked***`)`
`new JRadioButton("`***text***`")`
	Create new check box or radio button

jcb`.isSelected()` ***jcb***`.setSelected(`***boolean***`);`	set/return whether box is checked
jcb`.getText()` ***jcb***`.setText("`***text***`");`	set/return text in the check box

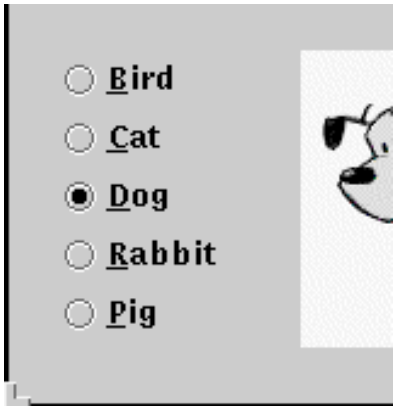
TipCalculator exercise

- Write a GUI program named **`TipCalculator`** :
  - window size: 500 x 200
  - When the "Calculate Tip!" button is clicked, the program computes and displays the tip amount in a large central label.
  - Convert a `String` into a double using,
    `double d = Double.parseDouble(str);`



ButtonGroup

a logical collection to ensure that exactly one radio button from a group is checked at a time



- `public` **`ButtonGroup`**`()`
- `public void` **`add`**`(JRadioButton button)`
  - The `ButtonGroup` is not a graphical component, just a logical group; the `RadioButtons` themselves also need to be added to an onscreen container to be seen.

TipCalculator solution

```
public class TipCalculator extends Program {
    private JTextField subtotal;    // field for user to type meal amount
    private JLabel output;          // displays tip amount
    private JRadioButton fifteen;   // radio buttons for 15%, 18% tip
    private JRadioButton eighteen;

    public void init() {
        setSize(500, 200);

        subtotal = new JTextField(10);
        fifteen = new JRadioButton("15%");
        fifteen.setSelected(true);
        eighteen = new JRadioButton("18%");
        ButtonGroup group = new ButtonGroup();
        group.add(fifteen);
        group.add(eighteen);
        output = new JLabel("$0.00");
        output.setFont(new Font("Serif", Font.BOLD, 58));

        add(new JLabel("Meal subtotal: $"), NORTH);
        add(subtotal, NORTH);
        add(fifteen, NORTH);
        add(eighteen, NORTH);
        add(new JButton("Calculate Tip!"), NORTH);
        add(output, SOUTH);
        addActionListeners();   // enable action events
    }
    ...
```

TipCalculator solution 2

```
    public void actionPerformed(ActionEvent event) {
        try {
            String subtotalText = subtotal.getText();
            double subtotal = Double.parseDouble(subtotalText);
            double percentTip = 15.0;
            if (eighteen.isSelected()) {
                percentTip = 18.0;
            }
            double tip = percentTip / 100.0 * subtotal;
            tip = Math.round(tip * 100.0) / 100.0;  // round to nearest cent

            String tipText = "$" + tip;    // "$234.56"    "123.5"
            if (tipText.charAt(tipText.length() - 2) == '.') {
                tipText += "0";
            }
            output.setText(tipText);
        } catch (NumberFormatException nfe) {
            subtotal.setBackground(Color.YELLOW);
            JOptionPane.showMessageDialog(this, "Illegal number :-(");
        }
    }
}
```

Types of text fields

- `JTextField`: Takes in any text as input.
- `IntField`: Only accepts ints; will re-prompt on bad data.
- `DoubleField`: Only accepts doubles; will re-prompt on bad data.
- `JSpinner`: Like an `IntField` with up/down buttons.

- If the user presses Enter in any of these, no event will occur. To receive events for this situation:

```
textField.setActionCommand("command string");
textField.addActionListener(this);
```

Events on text fields

- By default, no event will occur if you press Enter on a `JTextField`.
- To change this, manually add your program as an action listener to the field and set its action command.

```
myTextField.addActionListener(this);
myTextField.setActionCommand("bingo");
...

public void actionPerformed(ActionEvent event) {
    if (event.getActionCommand().equals("bingo")) {
        ...
    }
}
```

Icon

a picture that can appear inside a component

- `ImageIcon` ***`name`*** `= new ImageIcon("`***`filename`***`");`

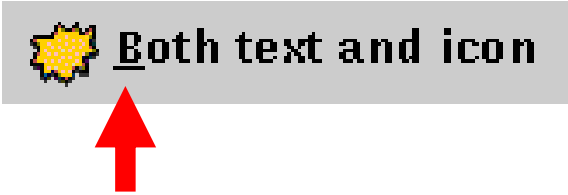
- in `JButton`, `JLabel`, `JRadioButton`, `JCheckBox`, etc…
  - **constructor** that takes an `ImageIcon`
  - or, ***`jb`***`.`**`setIcon(`**`icon`**`);`

- example:

```
ImageIcon icon = new ImageIcon("res/smiley.gif");
myButton.setIcon(icon);
```

Mnemonics

- **mnemonic:** A context-sensitive menu hotkey assigned to a specific button or other graphical component.
  - usually visible as an underlined letter
  - activated by pressing Alt + *letter*



  - *usage:* call ***component***`.setMnemonic(`***char***`)` method.

```
JButton quitButton = new JButton("Quit");
quitButton.setMnemonic('Q');
```