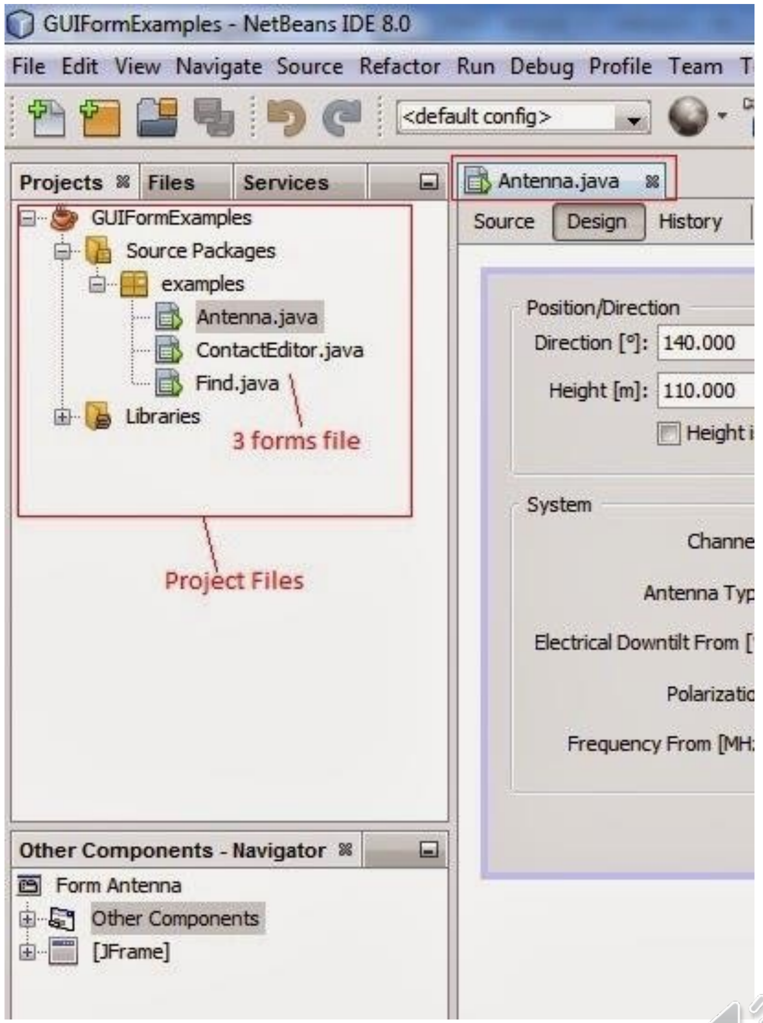
# COMP1618
# Graphical User Interface – Part 2

# Lecture Objectives

- This lecture shows how to write Java programs using JFrame design mode.
  - parse values,
  - combo boxes,
  - radio buttons
  - list

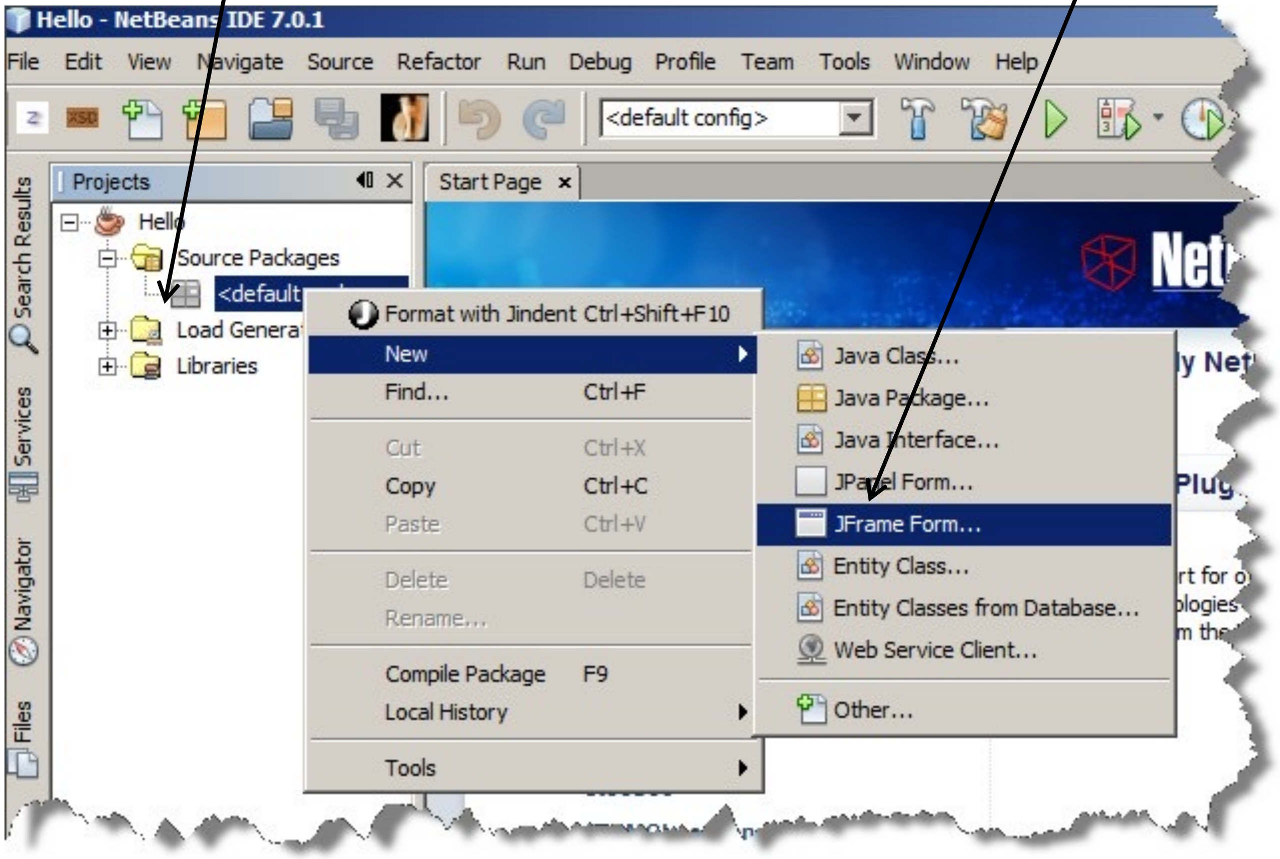
  - File Choosers
  - Colour chooser
  - Menu
  - Look and Feel



# Creating Your JFrame

1. Right click on the default package

2. Select JFrame

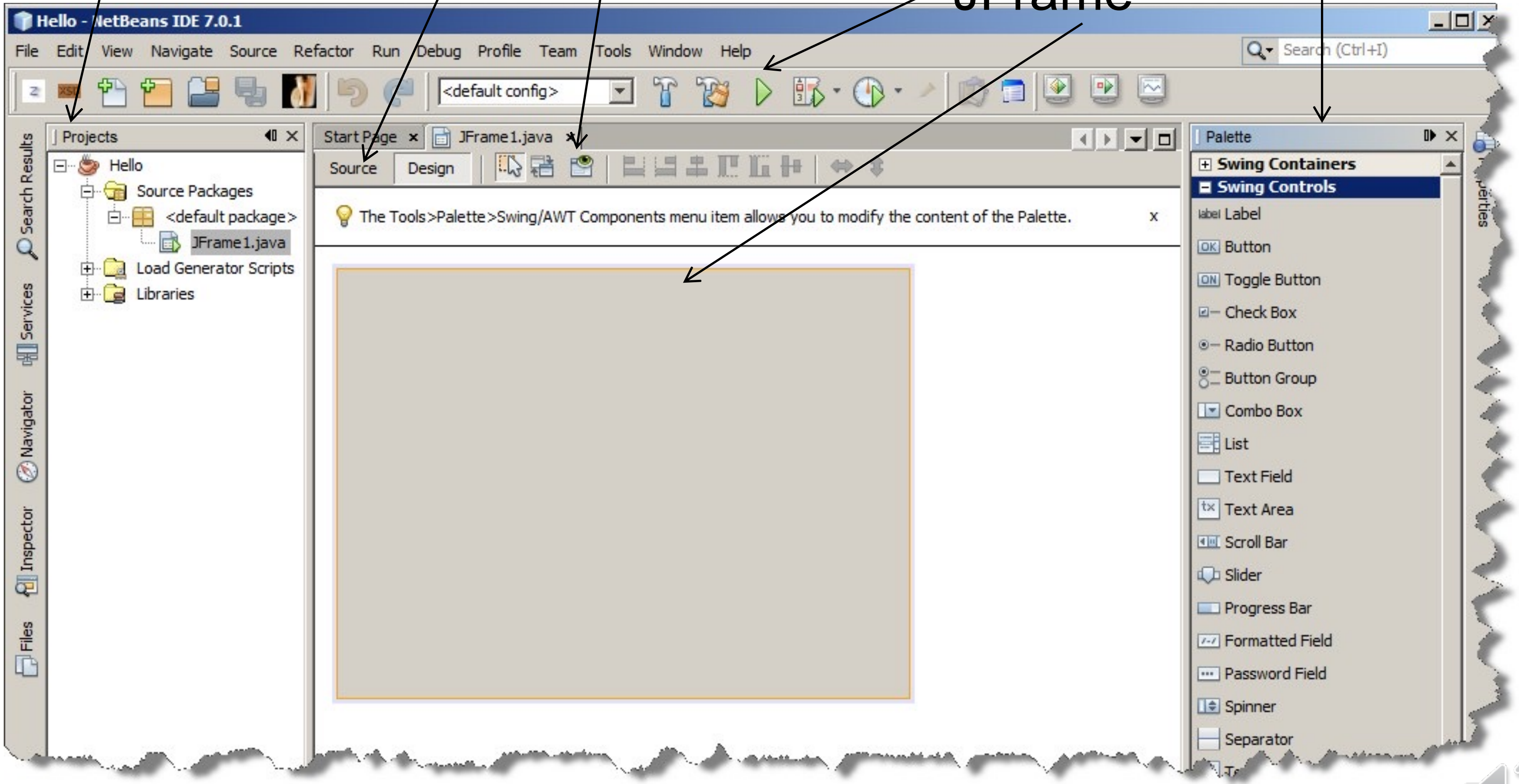


Projects Window Source, to see the background code

Preview your design Run the code

Components

JFrame

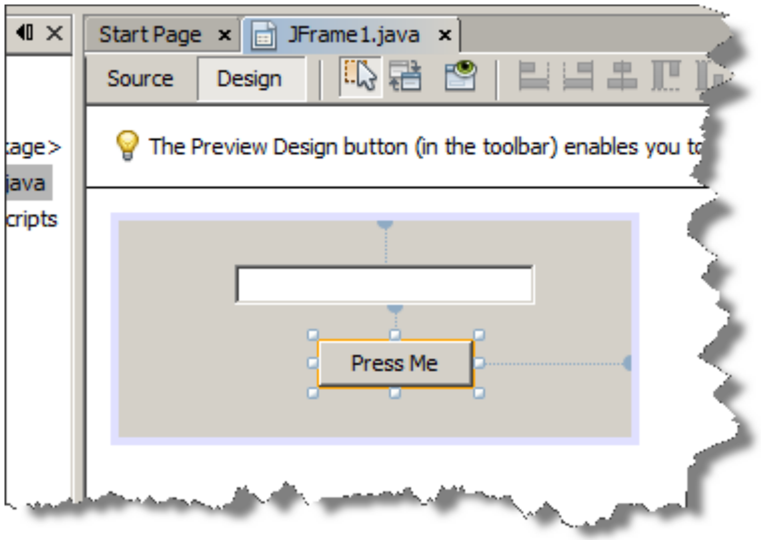


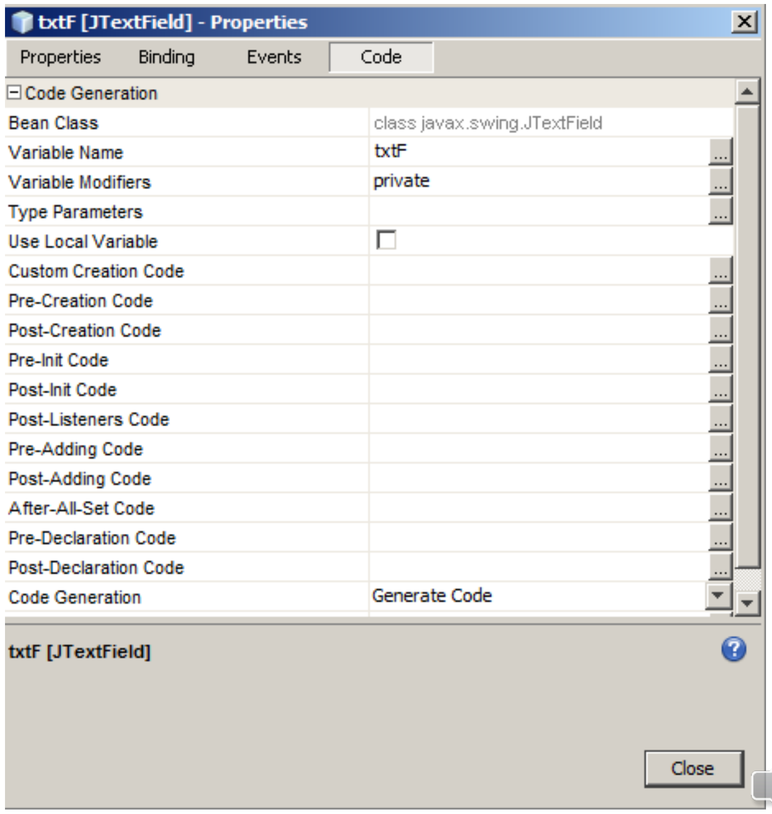
The Tools>Palette>Swing/AWT Components menu item allows you to modify the content of the Palette.

# Components Properties

Right click the components to change their properties from the properties window. This also applies to the JFrame

From the components palette drag components onto the JFrame



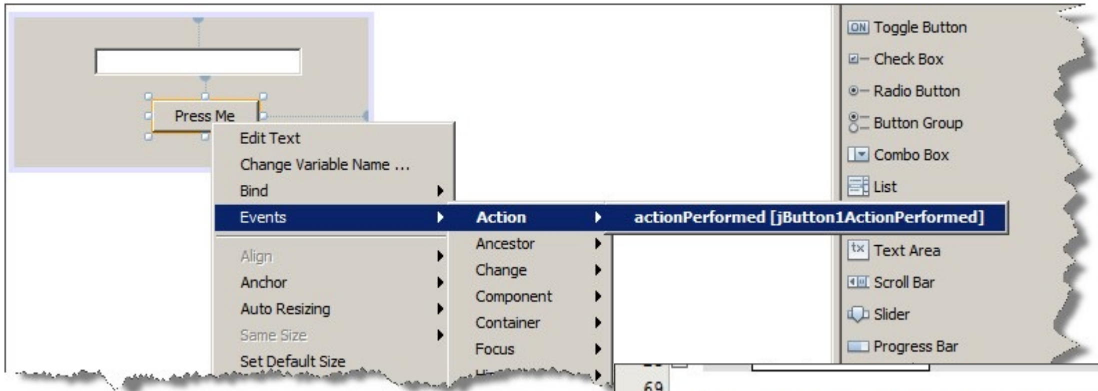


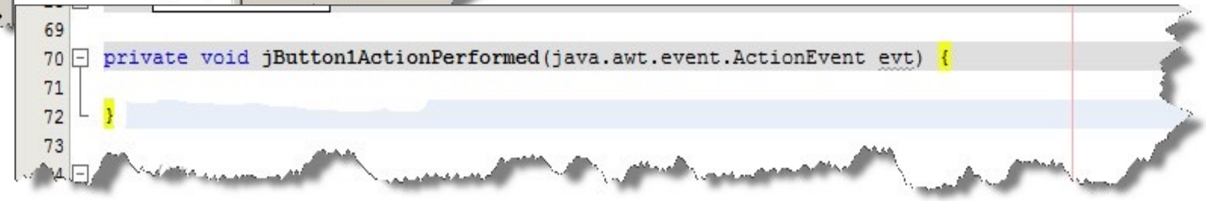
| Code Generation | |
|---|---|
| Bean Class | class javax.swing.JTextField |
| Variable Name | txtF |
| Variable Modifiers | private |
| Type Parameters | |
| Use Local Variable | ☐ |
| Custom Creation Code | |
| Pre-Creation Code | |
| Post-Creation Code | |
| Pre-Init Code | |
| Post-Init Code | |
| Post-Listeners Code | |
| Pre-Adding Code | |
| Post-Adding Code | |
| After-All-Set Code | |
| Pre-Declaration Code | |
| Post-Declaration Code | |
| Code Generation | Generate Code |

txtF [JTextField]

# Handling Action Events

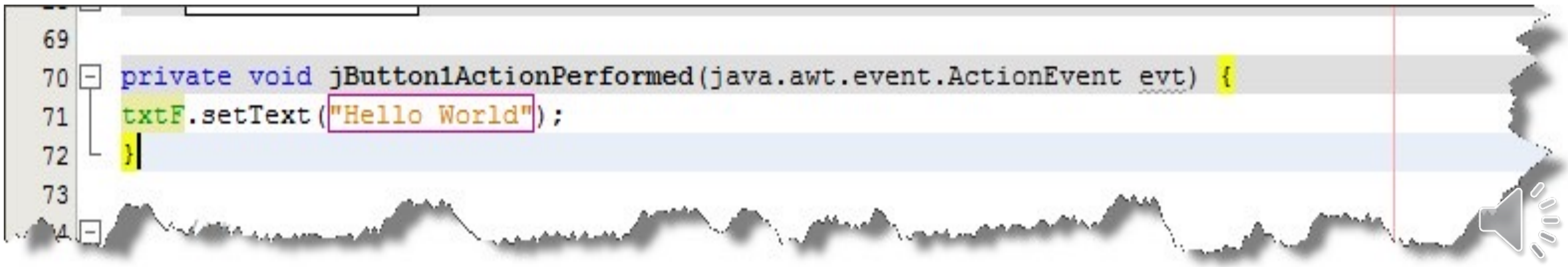
- `JButton` components generate *action events*, which require an *action listener*.
- The button must be "registered" with an *action listener* for it to work.
- To do this right click on the button and select Events -> Action -> actionPerformed



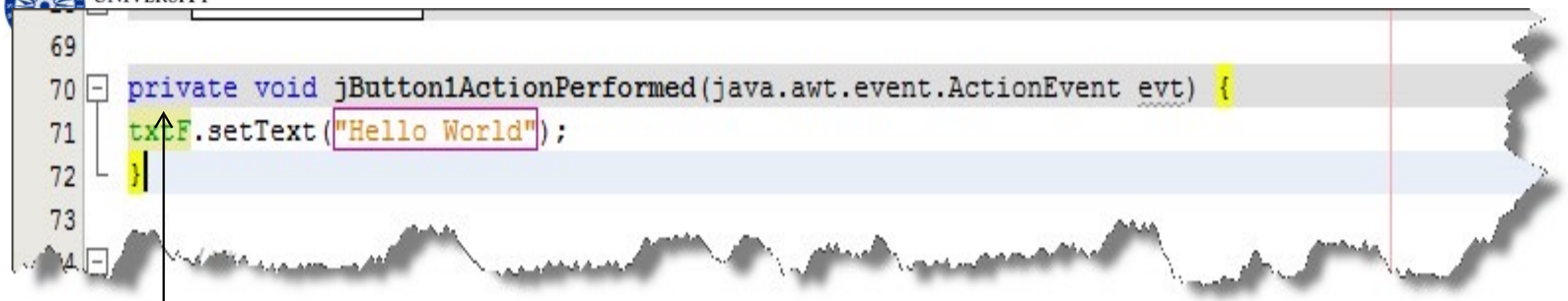


```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

}
```

- This will take you the code:



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
txtF.setText("Hello World");
}
```



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
txtF.setText("Hello World");
}
```

What is this?

txtF is the name we gave the text field.

With every text field there are a number of methods associated with them. The two methods that we will be using are:

*nameofTextField*.setText("*Add your text here*") - inserts text into the text field.

*nameofTextField*.getText() – "gets" the text that's already in the text field.

- Video tutorial: link

## Input into GUIs

- With GUI programming all input is stored as a string.
- Even if the input is a number it will be stored as a string.
- Mathematical operations cannot be done on strings.
- Numerical input has to be first converted into a number before any calculations can be done.

## The Parse Methods

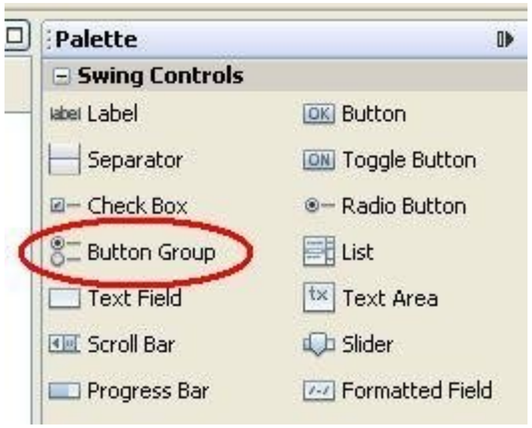
- Each of the numeric classes, has a method that converts a string to a number.
  - The `Integer` class has a method that converts a string to an `int`,
  - The `Double` class has a method that converts a string to a `double`, and
  - etc.
- These methods are known as *parse methods* because their names begin with the word "parse."

# The Parse Methods

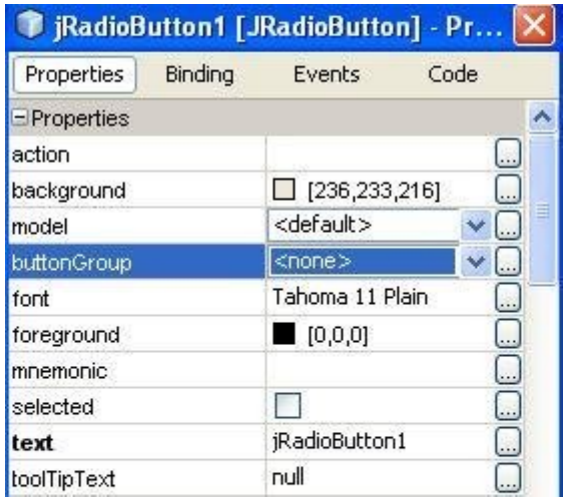
```
// Store 1 in bVar.
byte bVar = Byte.parseByte("1");

// Store 2599 in iVar.
int iVar = Integer.parseInt("2599");

// Store 10 in sVar.
short sVar = Short.parseShort("10");

// Store 15908 in lVar.
long lVar = Long.parseLong("15908");

// Store 12.3 in fVar.
float fVar = Float.parseFloat("12.3");

// Store 7945.6 in dVar.
double dVar = Double.parseDouble("7945.6");
```

# Creating a Radio Button Group

1.Use “Button group” from the components palette and put it into Your form. It will be added as “non-visual”

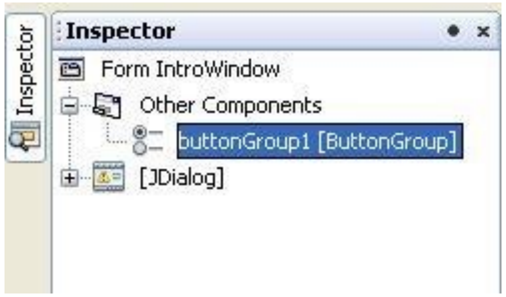


2. Add the radio buttons to the group, changing their ButtonGroup property (combobox in the Properties)



Properties

| | |
|---|---|
| action | |
| background | ☐ [236,233,216] |
| model | <default> |
| buttonGroup | <none> |
| font | Tahoma 11 Plain |
| foreground | ■ [0,0,0] |
| mnemonic | |
| selected | ☐ |
| **text** | jRadioButton1 |
| toolTipText | null |

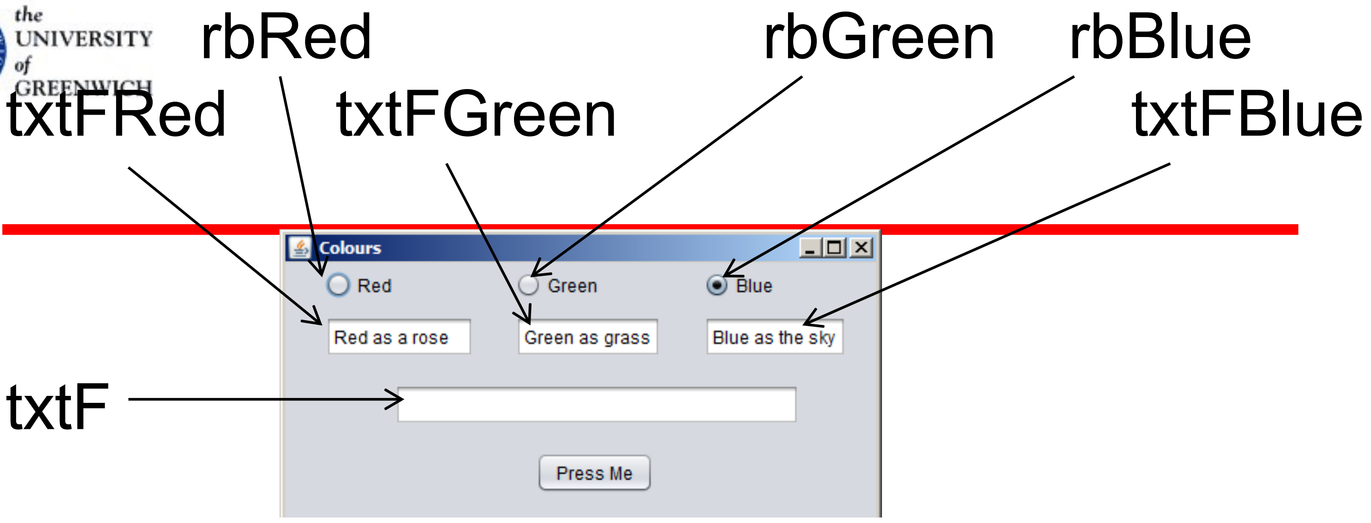


To remove the radio button group, go to the inspector window and remove radio button group there.

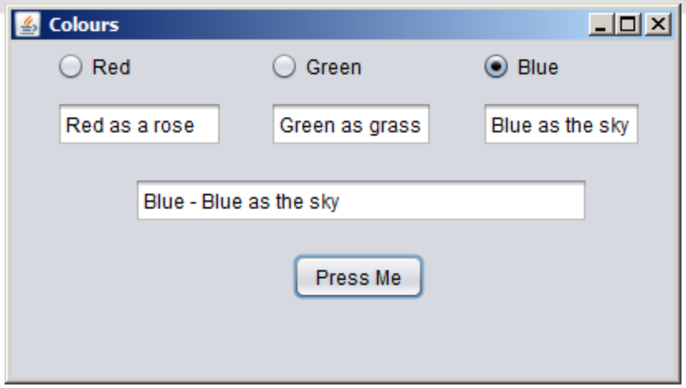
# Determining Selected Radio Buttons

- The `JRadioButton` class's `isSelected` method returns a `boolean` value indicating if the radio button is selected.

```
if (radio.isSelected())
{
   // Code here executes if the radio
   // button is selected.
}
```



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    String colour;
    if (rbRed.isSelected())             colour = "Red - "    + txtFRed.getText();
    else if (rbGreen.isSelected())      colour = "Green - "  + txtFGreen.getText();
    else                                colour = "Blue - "   + txtFBlue.getText();

    txtF.setText(colour);
}
```



See RadioButtons

## Lists

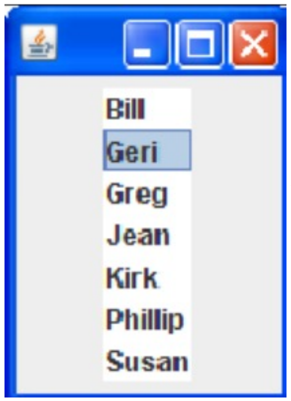
- A *list* is a component that displays a list of items and allows the user to select items from the list.
- The `JList` component is used for creating lists.
- When an instance of the `JList` class is created, an array of objects is passed to the constructor.

```
JList (Object[] array)
```

- The `JList` component uses the array to create the list of items.

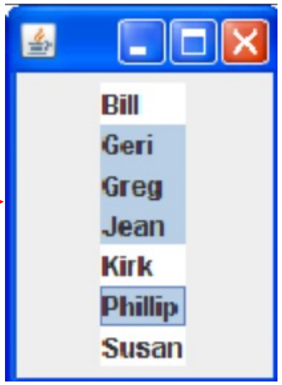
```
String[] names = { "Bill", "Geri", "Greg", "Jean",
  "Kirk", "Phillip", "Susan" };
JList nameList = new JList(names);
```

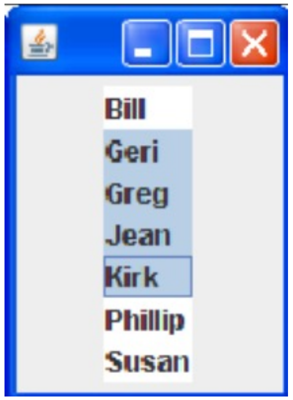
# List Selection Modes



Single selection mode allows only one item to be selected at a time.

Multiple interval selection mode allows multiple items to be selected with no restrictions.





Single interval selection mode allows a single interval of contiguous items to be selected.

## List Selection Modes

- You change a `JList` component’s selection mode with the `setSelectionMode` method.
- The method accepts an `int` argument that determines the selection mode:
  - `ListSelectionModel.SINGLE_SELECTION`
  - `ListSelectionModel.SINGLE_INTERVAL_SELECTION`
  - `ListSelectionModel.MULTIPLE_INTERVAL_SELECTION`
- Example:

```
nameList.setSelectionMode(
            ListSelectionModel.SINGLE_SELECTION);
```

# Retrieving Selected Items

- You may use:
  - `getSelectedValue` or
  - `getSelectedIndex`

  - to determine which item in a list is currently selected.
- `getSelectedValue` returns a reference to the item that is currently selected.

```
String selectedName;
selectedName = (String)nameList.getSelectedValue();
```

- The return value must be cast to `String` is required in order to store it in the `selectedName` variable.
- If no item in the list is selected, the method returns null.

## Retrieving Selected Items

- This code could be used to determine the selected item:

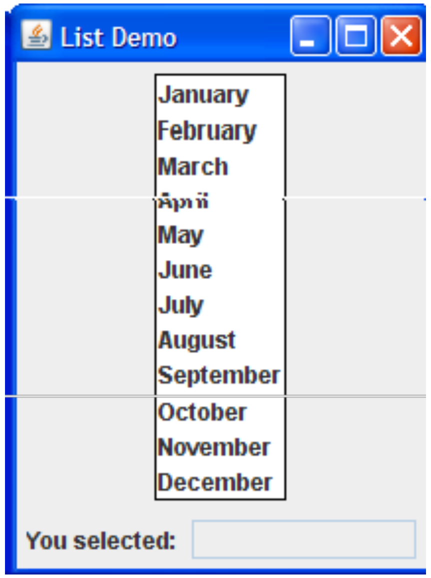
```
int index;
String selectedName;
index = nameList.getSelectedIndex();
if (index != -1)
   selectedName = names[index];
```

- Example: ListWindow.java

# Bordered Lists

- The `setBorder` method can be used to draw a border around a `JList`.

```
monthList.setBorder(
    BorderFactory.createLineBorder(Color.black,1));
```



January
February
March
April
May
June
July
August
September
October
November
December

You selected:

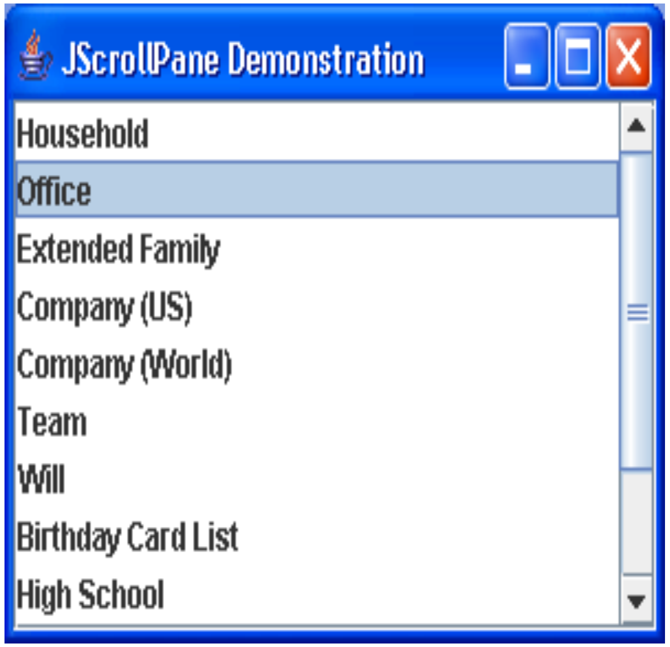
# Adding A Scroll Bar To a List

- Establish the size of the list component.

```
nameList.setVisibleRowCount(3);
```

- Create a scroll pane object and add the list component to it.
- A *scroll pane object* is a container that displays scroll bars on any component it contains.
- The `JScrollPane` class to create a scroll pane object.
- We pass the object that we wish to add to the scroll pane as an argument to the `JScrollPane` constructor.

```
JScrollPane scrollPane = new
  JScrollPane(nameList);
```



## Adding A Scroll Bar To a Panel

- Add the scroll pane object to any other containers that are necessary for our GUI.

```
JPanel panel = new JPanel();
panel.add(scrollPane);
add(panel);
```

- When the list component is displayed, it will appear with:
  - Three items showing at a time and
  - scroll bars:
- Example: ListWindowWithScroll.java

# Adding Items to an Existing List

- The `setListData` method allows the adding of items in an existing `JList` component.

```
void setListData(Object[] data)
```

- Items can be added to the list:

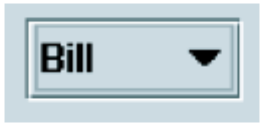
```
String[] names = { "Bill", "Geri", "Greg", "Jean",
 "Kirk", "Phillip", "Susan" };
nameList.setListData(names);
nameList.add("ABC");
nameList.add("DEF");
```

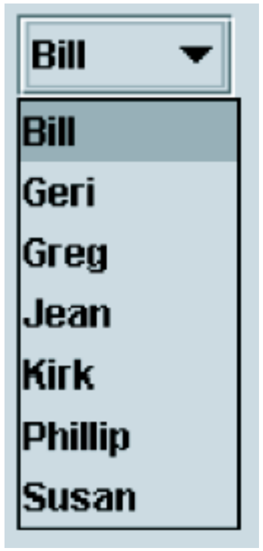
# Combo Boxes

- A combo box presents a drop-down list of items that the user may select from.
- Pass an array of objects that are to be displayed as the items in the drop-down list to the constructor.

```
String[] names = { "Bill", "Geri",
  "Greg", "Jean", "Kirk", "Phillip",
  "Susan" };
JComboBox nameBox = new
  JComboBox(names);
```

- First displayed as the button
- Once clicked, the drop-down list appears and the user may select another item.





## Retrieving Selected Items

- The **`getSelectedIndex, getSelectedItem`**

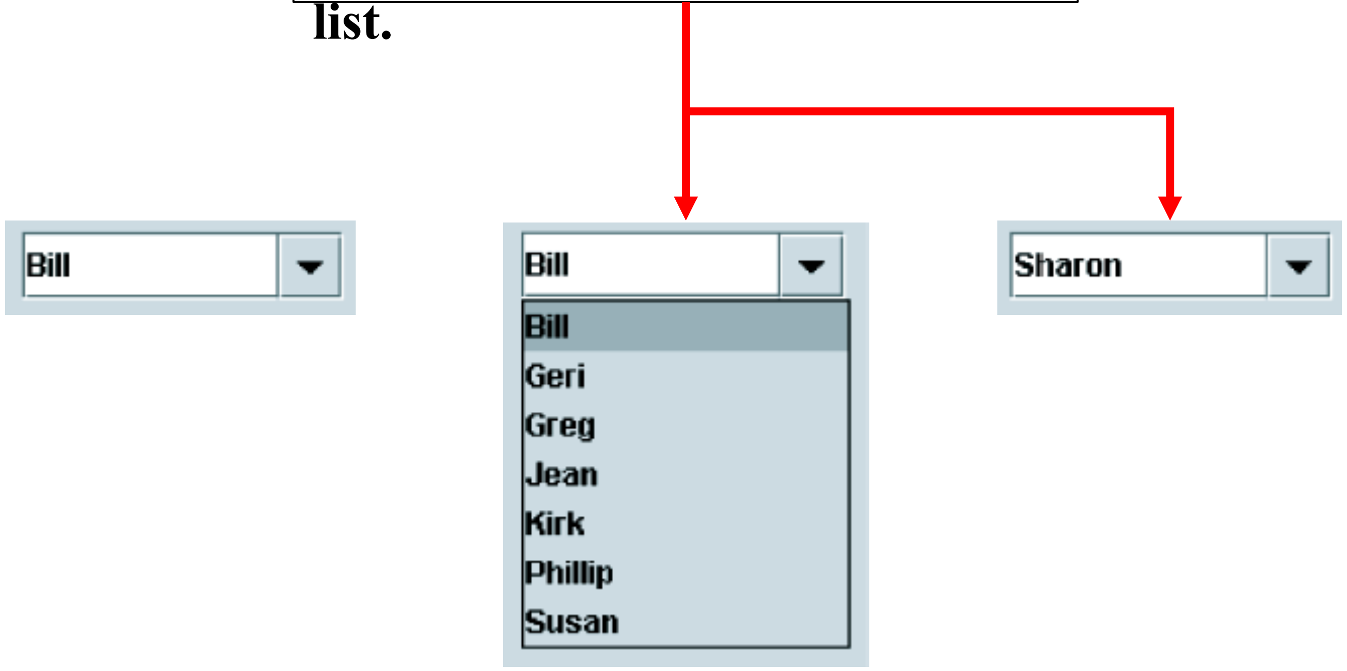
```
   String[] names = { "Bill", "Geri", "Greg",
"Jean", "Kirk", "Phillip", "Susan" };
    JComboBox nameBox = new JComboBox(names);
```

- Get the selected item from the names array:

```
    int index;
    String selectedName;
    index = nameBox.getSelectedIndex();
    selectedName = names[index];
    selectedName = (String)
      nameBox.getSelectedItem();
```

# Editable Combo Boxes

Note that Sharon is not in the list.



## Editable Combo Boxes

- There are two types of combo boxes:
  - uneditable – allows the user to only select items from its list.
  - editable – combines a text field and a list.
    - It allows the selection of items from the list
    - allows the user to type input into the text field
- The `setEditable` method sets the edit mode for the component.

```
String[] names = { "Bill", "Geri", "Greg",
  "Jean", "Kirk", "Phillip", "Susan" };
JComboBox nameBox = new JComboBox(names);
nameBox.setEditable(true);
```

# Displaying Images in Labels and Buttons

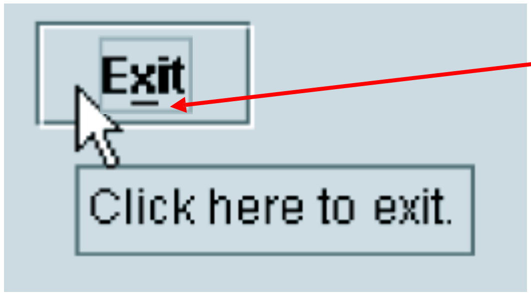
- To add an image to an existing button:

```
JButton button = new JButton(
                  "Have a nice day!");
ImageIcon image = new
  ImageIcon("Smiley.gif");
button.setIcon(image);
```

- You are not limited to small graphical icons when placing images in labels or buttons.
- Example: MyCatImage.java

## Mnemonics

- A *mnemonic* is a key that you press in combination with the Alt key to quickly access a component.
- These are sometimes referred to as hot keys.
- A hot key is assigned to a component through the component's `setMnemonic` method
- The argument passed to the method is an integer code that represents the key you wish to assign.



Note the mnemonic x.

## Mnemonics

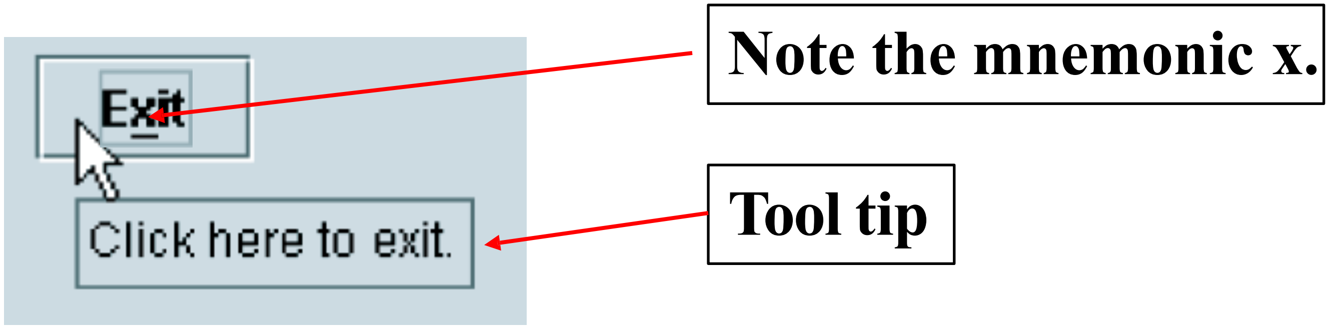
- You can also assign mnemonics to radio buttons and check boxes:

```
JRadioButton rb1 = new
      JRadioButton("Breakfast");
rb1.setMnemonic(KeyEvent.VK_B);
JRadioButton rb2 = new JRadioButton("Lunch");
rb2.setMnemonic(KeyEvent.VK_L);
JCheckBox cb1 = new JCheckBox("Monday");
cb1.setMnemonic(KeyEvent.VK_M);
JCheckBox cb2 = new JCheckBox("Wednesday");
cb2.setMnemonic(KeyEvent.VK_W);
```

## Tool Tips

- Assign a tool tip to a component with the `setToolTipText` method.

```
JButton exitButton = new JButton("Exit");
exitButton.setMnemonic(KeyEvent.VK_X);
exitButton.setToolTipText(
  "Click here to exit.");
```



## File Choosers

- A file chooser is a specialized dialog box that allows the user to browse for a file and select it.

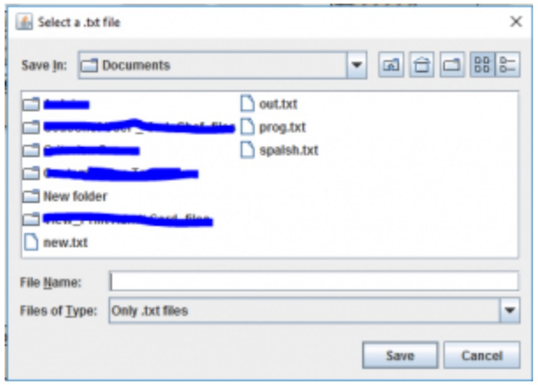
# File Choosers

- To display an open file dialog box, use the `showOpenDialog` method.
- General format:

```
int showOpenDialog(Component parent)
```

- To display a save file dialog box, use the `showSaveDialog` method.
- General format:

```
int showSaveDialog(Component parent)
```


Select a .txt file

Save In: Documents

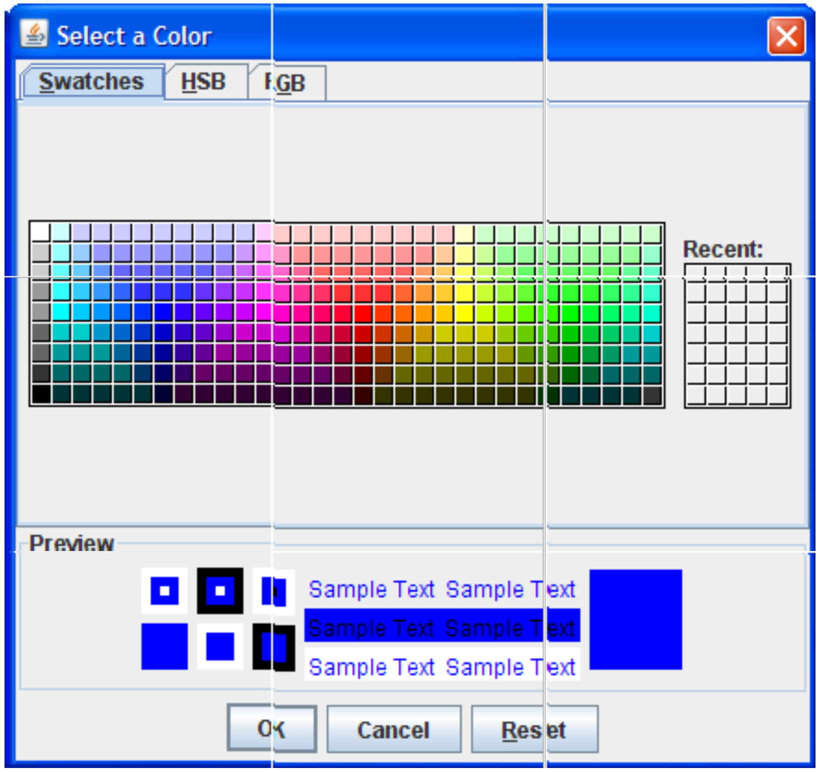
[redacted]
[redacted]
[redacted]
[redacted]
New folder
[redacted]
new.txt
out.txt
prog.txt
spalsh.txt

File Name:
Files of Type: Only .txt files

Save Cancel

# Color Choosers

- A color chooser is a specialized dialog box that allows the user to select a color from a predefined palette of colors.



## Color Choosers

- Example:

```
JPanel panel = new JPanel();
Color selectedColor =
  JColorChooser.showDialog(null,
         "Select a Background Color",
         Color.BLUE);
panel.setBackground(selectedColor);
```

# Menus

- A *menu system* is a collection of commands organized in one or more drop-down menus.

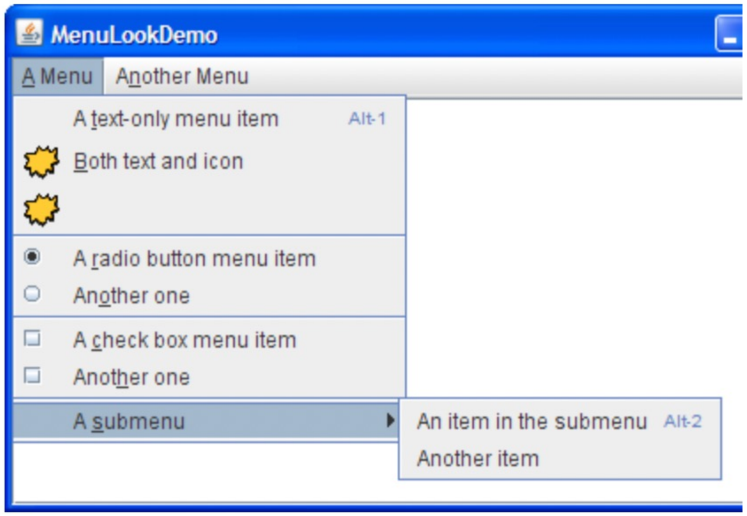
Menubar

Menu

Separator

Check Box Menu Item

Radio Button Menu Item



# Menus

- Example from Code\MenuWindow.java

```
    private JMenuBar menuBar;    // The menu bar
    private JMenu fileMenu;      // The File menu

    menuBar = new JMenuBar();
  // Create the file menus.
    exitItem = new JMenuItem("Exit");
    exitItem.setMnemonic(KeyEvent.VK_X);
    exitItem.addActionListener(new ExitListener());
  // Create a JMenu object for the File menu.
    fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);
  // Add the Exit menu item to the File menu.
    fileMenu.add(exitItem);

  // Add the file and text menus to the menu bar.
    menuBar.add(fileMenu);

  // Set the window's menu bar.
    setJMenuBar(menuBar);
```

# Look and Feel

- Metal look and feel:

```
"javax.swing.plaf.metal.MetalLookAndFeel"
```

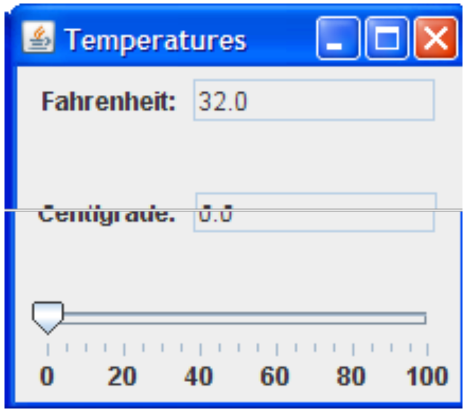
- Motif look and feel:

```
"com.sun.java.swing.plaf.motif.MotifLookAndFeel"
```

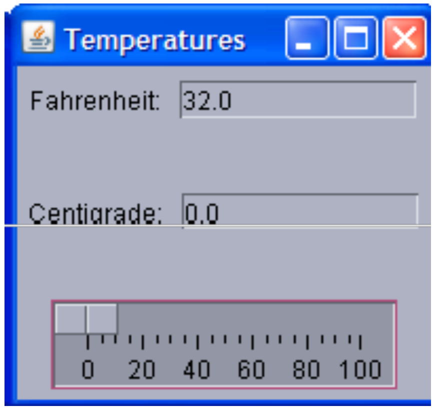
- Windows look and feel:

```
"com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
```

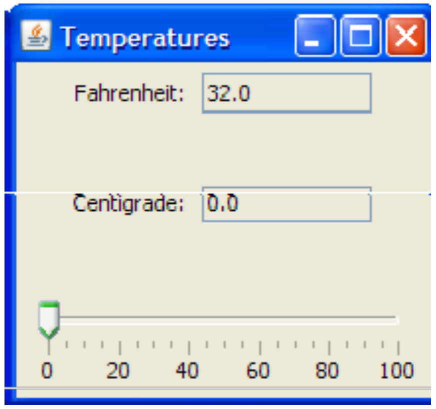
**Metal**



**Motif**



**Windows**



## Look and Feel

- Example (**Motif**):

```
try
{
  UIManager.setLookAndFeel(
  "com.sun.java.swing.plaf.motif.MotifLookAndFeel");
  SwingUtilities.updateComponentTreeUI(this);
}
catch (Exception e)
{
  JOptionPane.showMessageDialog(null,
     "Error setting the look and feel.");
  System.exit(0);
}
```

# Look and Feel

- Example (**Windows**):

```
try
{
  UIManager.setLookAndFeel(
    "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
  SwingUtilities.updateComponentTreeUI(this);
}
catch (Exception e)
{
  JOptionPane.showMessageDialog(null,
       "Error setting the look and feel.");
  System.exit(0);
}
```

## Summary

- We saw Java GUI with drag & drop
- We also saw that how to write Java programs using JFrame design mode.
  - parse values,
  - combo boxes,
  - radio buttons
  - list

  - File Choosers
  - Colour chooser
  - Menu
  - Look and Feel