EELU
الجامعة المصرية للتعلم الإلكتروني
Egyptian E-Learning University

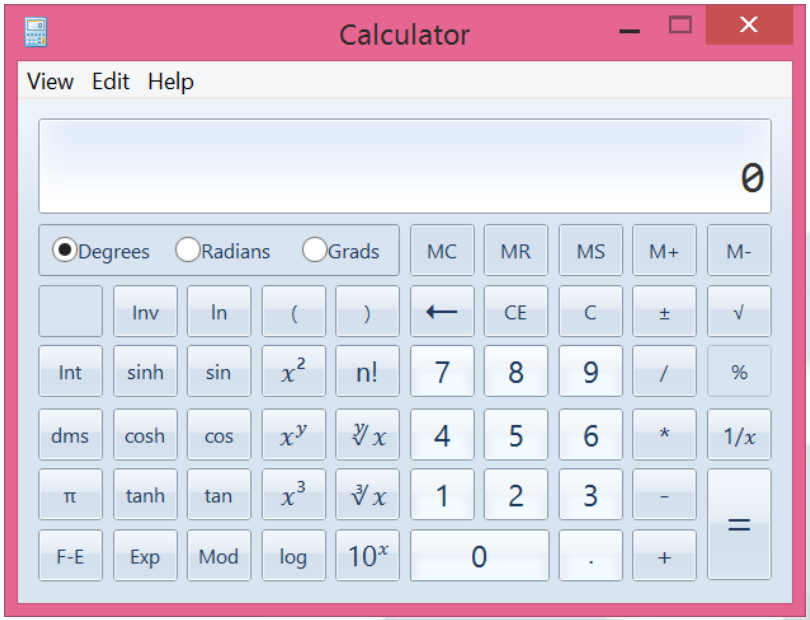
SWE203
Programming Techniques (3)

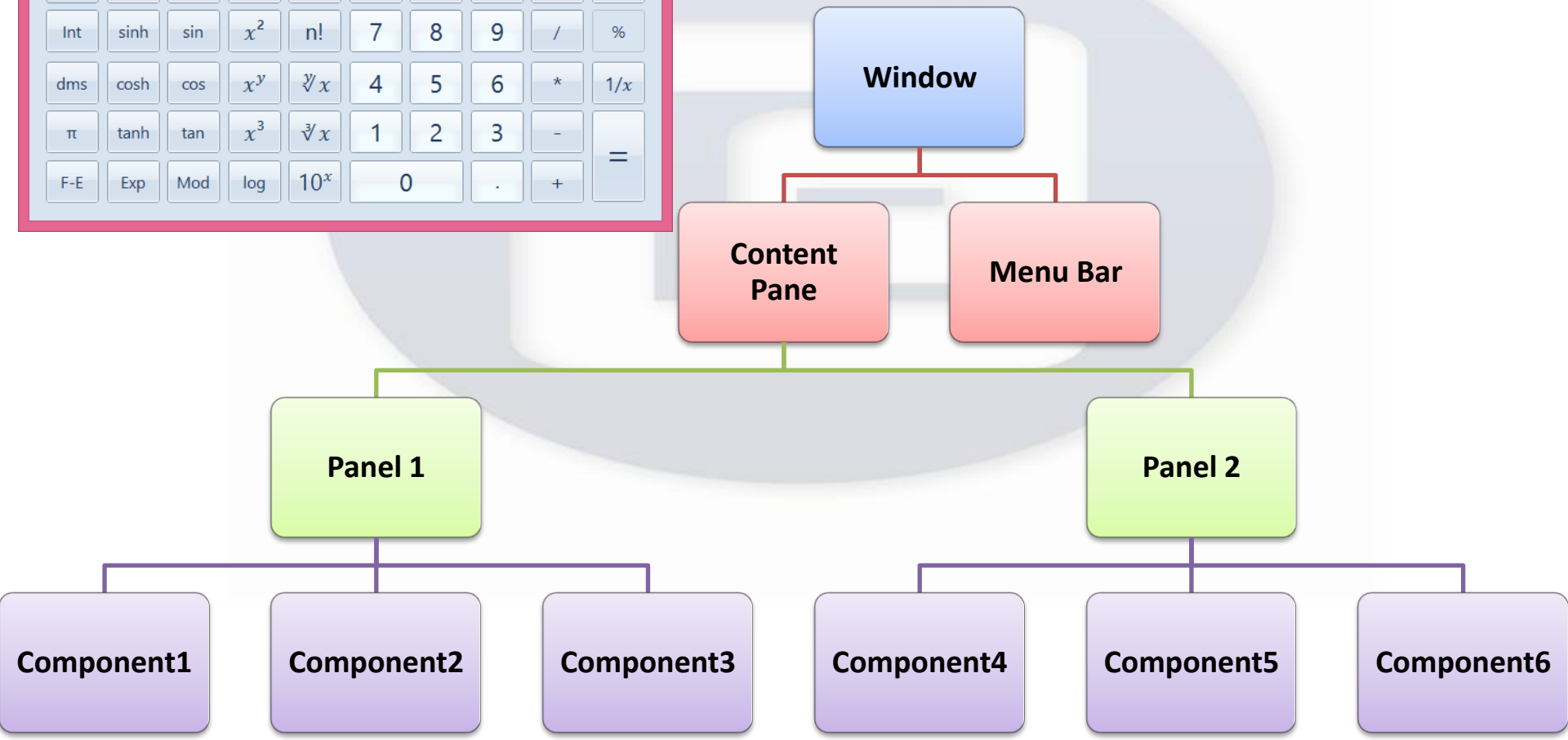
GUI

(Graphical User Interface)

Part II

Reminder - Swing Hierarchy



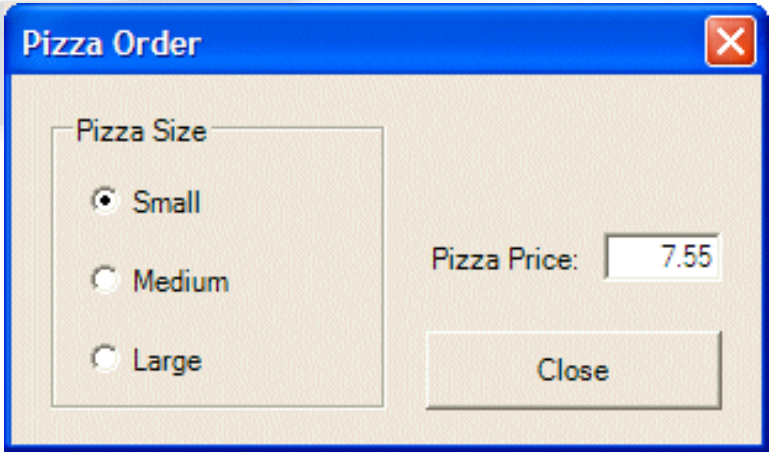


Components

- Radio Buttons
- Check boxes
- Borders
- Lists / Combo Boxes
- Images
- Mnemonics and Tool Tips
- File Choosers and Color Choosers
- Menus
- Sliders

Radio Buttons

- One choice from possible options
  - **Selected**
    - JRadioButton Small=new JRadioButton(“Small”,true)
  - **deselected**
    - JRadioButton Medi=new JRadioButton(“Medium”)
    - JRadioButton Large=new JRadioButton(“Large”)



ButtonGroup

- Groups a set of radio buttons, so that only one radio button is selected at any time.

- ButtonGroup gr=new ButtonGroup();
- gr.add (Small);
- gr.add (Medi);
- gr.add (Large);

Radio Buttons Events

- Responding to a click
  - ActionListener ➔ actionPerformed(ActionEvent e)
- Determining whether a radio button is selected
  - Radio.isSelected()
- Selecting a radio button programmatically
  - Radio.doClick()
- Allowing a radio button to programmatically change the selection status.

  - Radio. setSelected(boolean)

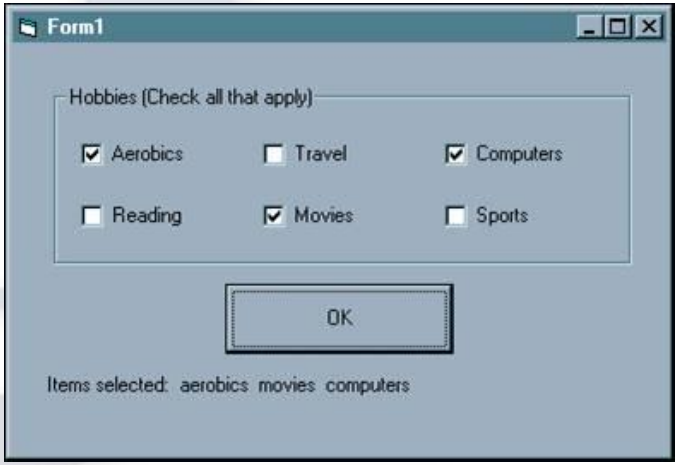
Radio Buttons Events

- Specifying a alt-Key alternative for selecting the RadioButton.
  - Radio. setMnemonic(char)

CheckBoxes

- Creation
  - JCheckBox Check1=new JCheckBox(“Aerobics”,true)
  - JCheckBox Check2=new JCheckBox(“Travel”)

- Handling click events:
  - ItemListerner ➔ itemStateChanged

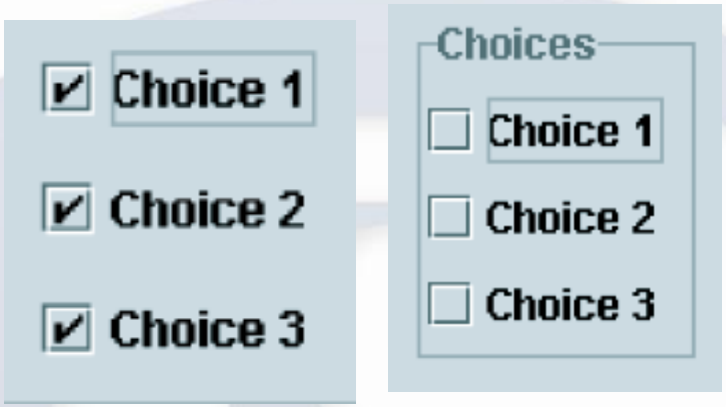


- Determine if a checkbox is selected
  - Check1.isSelected()

- Selecting a checkbox programmatically
  - Check1.doClick()

Borders

- Windows have a more organized look if related components are grouped inside borders.



- You can add a border to any component that is derived from the `JComponent` class.
  - Any component derived from `JComponent` inherits a method named `setBorder`

Borders

- The `setBorder` method is used to add a border to the component.
- The `setBorder` method accepts a `Border` object as its argument.
- A `Border` object contains detailed information describing the appearance of a border.
- The `BorderFactory` class, which is part of the `javax.swing` package, has static methods that return various types of borders.

Lists

- A *list* is a component that displays a list of items and allows the user to select items from the list.

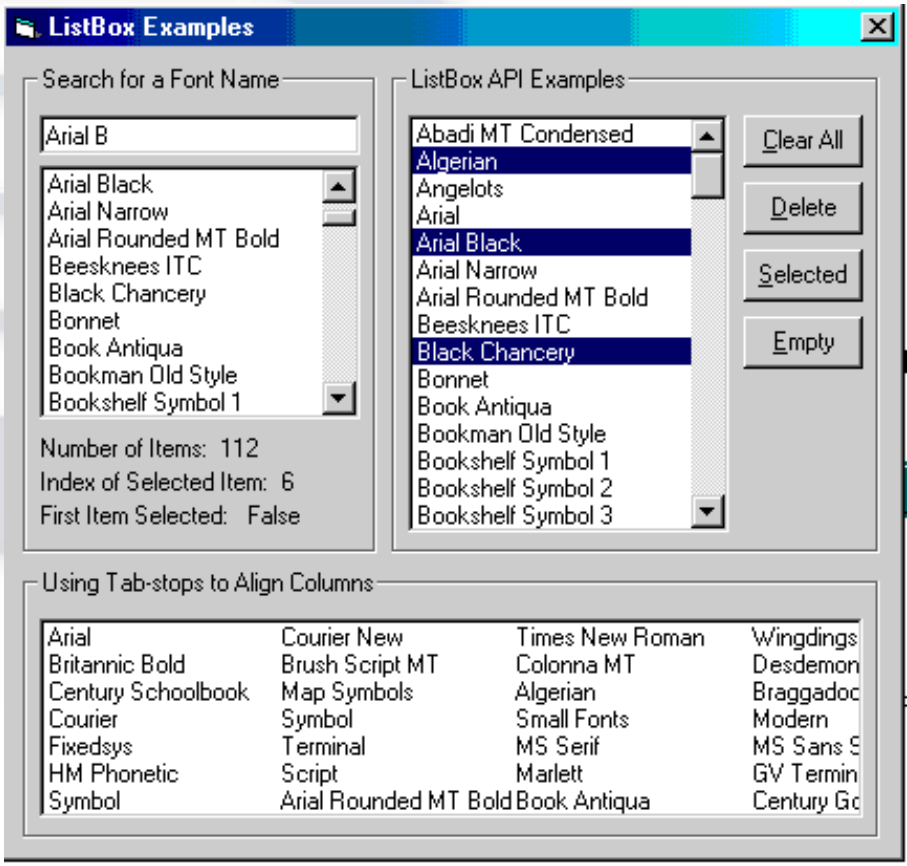
- The `JList` component is used for creating lists.

- 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 Events

- When an item in a `JList` object is selected it generates a *list selection event*.

- The event is handled by an instance of a *list selection listener* class, which must meet the following requirements:
  - It must implement the **`ListSelectionListener`** interface.
  - It must have a method named **`valueChanged`**. This method must take an argument of the `ListSelectionEvent` type.

- Use the `addListSelectionListener` method of the `JList` class to register the instance of the list selection listener class with the list object.

Adding Items to an Existing List

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

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

- This replaces any items that are currently displayed in the component.
- This can be used to add items to an empty list.

Adding Items to an Existing List

- You can create an empty list by using the `JList` component's no-parameter constructor:

```
JList nameList = new JList();
```

- Items can be added to the list:

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

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

- The `getSelectedIndex` method returns the index of the selected item, or `-1` if no item is selected.
- Internally, the items that are stored in a list are numbered (similar to an array).
- Each item's number is called its *index*.
- The first item has the index 0.
- You can use the index of the selected item to retrieve the item from an array.

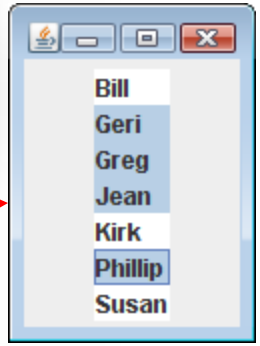
```
String[] names = { "Bill", "Geri",
  "Greg", "Jean", "Kirk", "Phillip",
  "Susan" };
JList nameList = new JList(names);
```

List Selection Modes

- The `JList` component can operate in any of three selection modes

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

```
L1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```



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

```
L1.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
```

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

```
L1.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
```

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);`

Single Interval Selection Mode

- A list is set to single interval selection mode by passing the constant **`ListSelectionModel.SINGLE_INTERVAL_SELECTION`** to the component's `setSelectionMode` method.

- An interval is a set of contiguous items.

- The user selects:
  - The first item in the interval by clicking on it
  - The last item by holding the Shift key while clicking on it.

- All of the items that appear in the list from the first item through the last item are selected.

Single Interval Selection Mode

- The `getSelectedValue` method returns the first item in the selected interval.
- The `getSelectedIndex` method returns the index of the first item in the selected interval.
- To get the entire selected interval, use the `getSelectedValues` method.
  - This method returns an array of objects, which are the items in the selected interval.
- The `getSelectedIndices` method returns an array of int values that are the indices of all the selected items in the list.

Multiple Interval Selection Mode

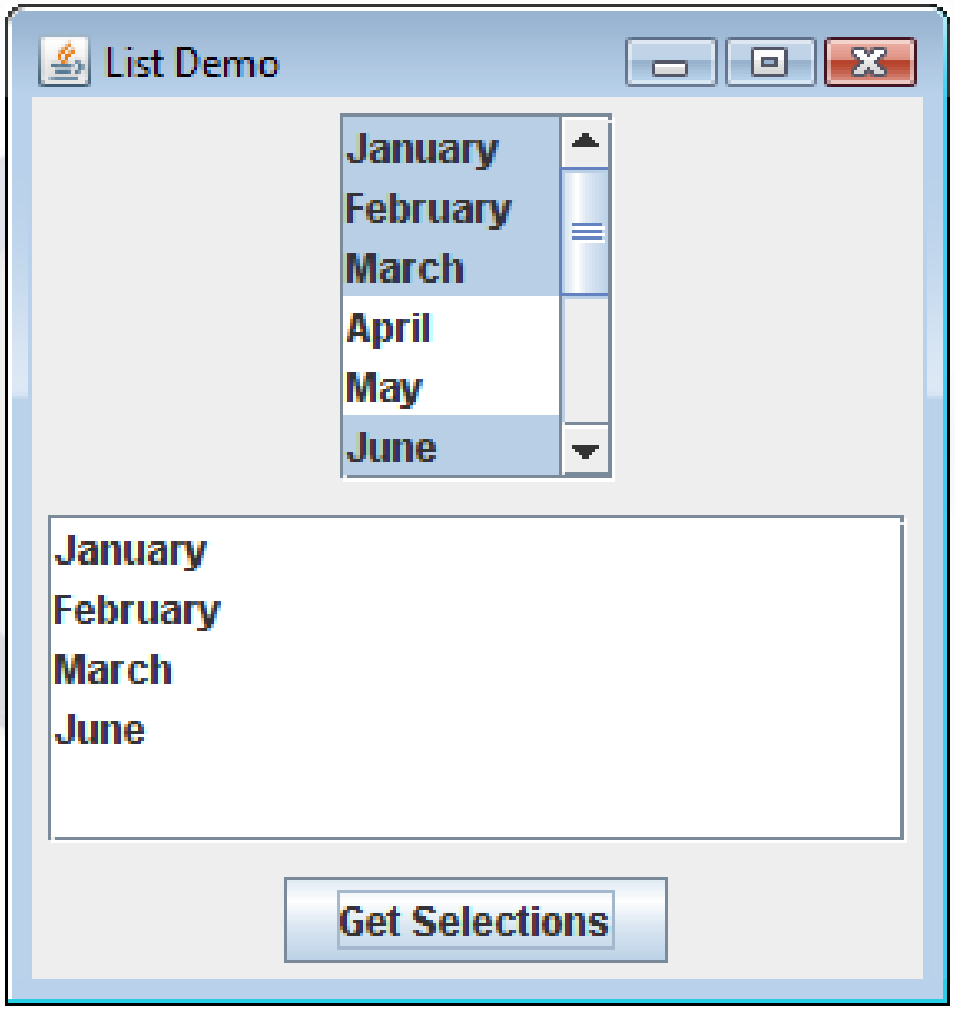
- Set multiple interval selection mode by passing the constant

  `ListSelectionModel.MULTIPLE_INTERVAL_SELECTION` to the component's `setSelectionMode` method.

- In multiple interval selection mode:
  - Multiple items can be selected
  - The items do not have to be in the same interval.

- In multiple interval selection mode the user can select single items or intervals.

Multiple Interval Selection Mode



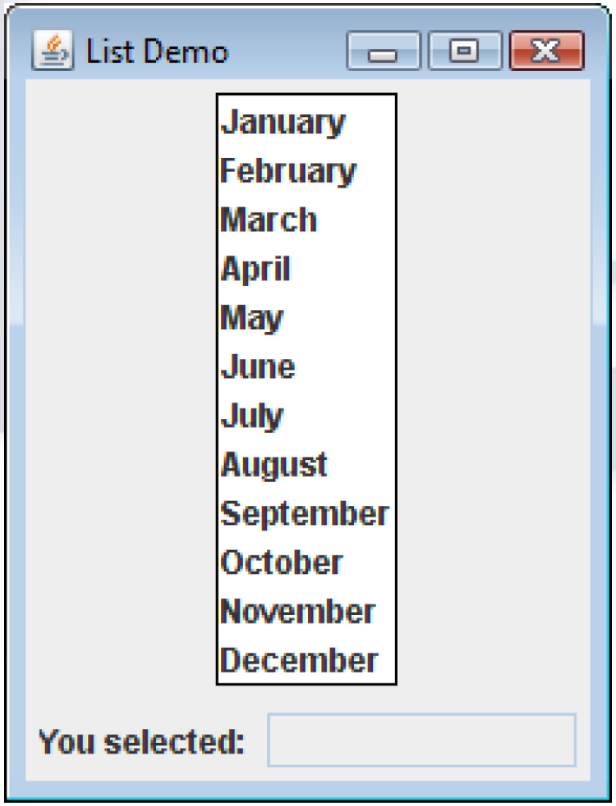
Multiple Interval Selection Mode

- The user holds down the Ctrl key while clicking on an item
  - it selects the item without deselecting other items.
- The `getSelectedValue` method returns the first selected item.
- The `getSelectedIndex` method returns the index of the first selected item.
- The `getSelectedValues` method returns an array of objects containing the items that are selected.
- The `getSelectedIndices` method returns an `int` array containing the indices of the selected items.

Bordered Lists

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

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



Adding a Scroll Bar To a List

- Sometimes a list component contains too many items to be displayed at once.

- To display a scroll bar on a list component, follow these general steps.
  1. Set the number of visible rows for the list component.
  2. Create a scroll pane object and add the list component to it.
  3. Add the scroll pane object to any other containers, such as panels.

Example

1. Establish the size of the list component.
   - `nameList.setVisibleRowCount(3);`

2. 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);`

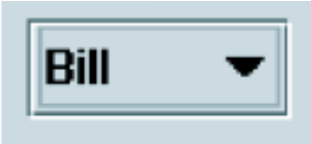
3. Add the scroll pane object to any other containers that are necessary for our GUI.
   - `JPanel panel = new JPanel();`
   - `panel.add(scrollPane);`
   - `add(panel);`

Components

- ✓ Radio Buttons
- ✓ Check boxes
- ✓ Borders
- ✓ List Boxes
- Combo Boxes
- Images
- Mnemonics and Tool Tips
- File Choosers and Color Choosers
- Menus
- Sliders

Combo Boxes

- A combo box presents a drop-down list of items that the user may select from.



- The `JComboBox` class is used to create a combo box.

- 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);
```

- When displayed, the combo box created by this code will initially appear as the button

Combo Boxes

- The button displays the item that is currently selected.
- The first item in the list is automatically selected when the combo box is displayed.

- When the user clicks on the button, the drop-down list appears and the user may select another item.

Combo Box Events

- When an item in a `JComboBox` object is selected, it generates an action event.
- Handle action events with an action event listener class, which must have an `actionPerformed` method.
- When the user selects an item in a combo box, the combo box executes its action event listener's `actionPerformed` method, passing an `ActionEvent` object as an argument.

Retrieving Selected Items

- There are two methods in the `JComboBox` class that can be used to determine which item in a list is currently selected:
  - `getSelectedItem`
  - `getSelectedIndex`

- The `getSelectedItem` method returns a reference to the item that is currently selected.

```
String selectedName;
selectedName = (String)
  nameBox.getSelectedItem();
```

- `getSelectedItem` returns an `Object` reference so we cast the return value to a `String`.

Retrieving Selected Items

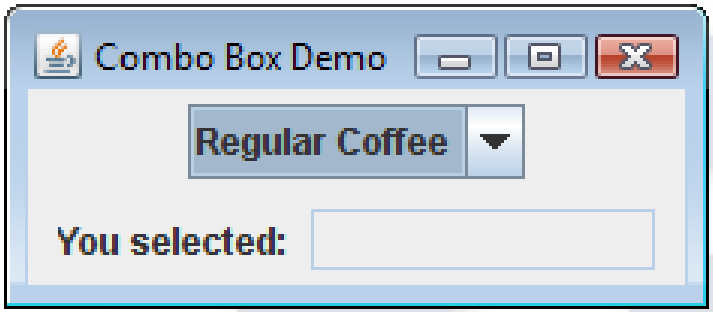
- The `getSelectedIndex` method returns the index of the selected item.

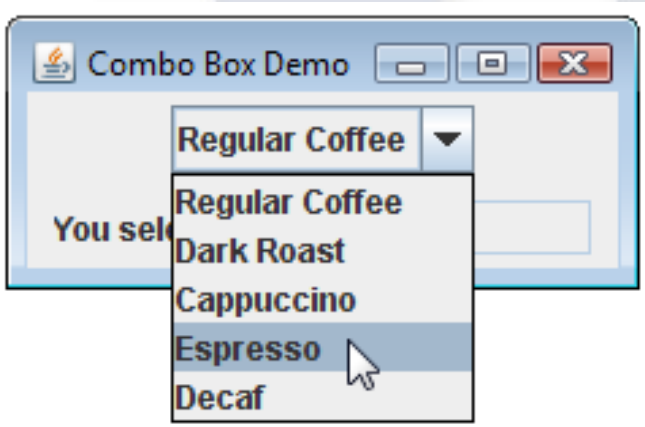
```
String[] names = { "Bill", "Geri", "Greg", "Jean",
  "Kirk", "Phillip", "Susan" };
JComboBox nameBox = new JComboBox(names);
```

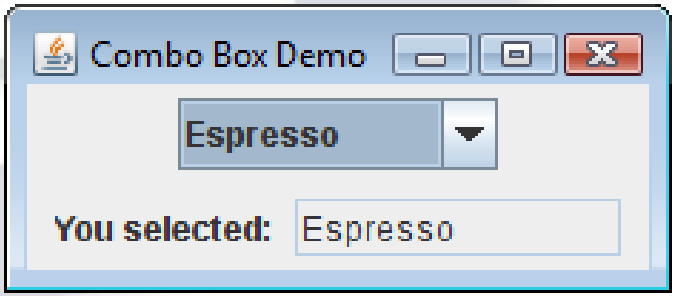
- Get the selected item from the names array:

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

Retrieving Selected Items







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);
```

Editable Combo Boxes

- An editable combo box appears as a text field with a small button displaying an arrow joining it.
- When the user clicks on the button, the drop-down list appears as shown in the center of the figure.
- The user may:
  - select an item from the list.
  - type a value into the text field.
- The user is not restricted to the values that appear in the list, and may type any input into the text field.

Editable Combo Boxes