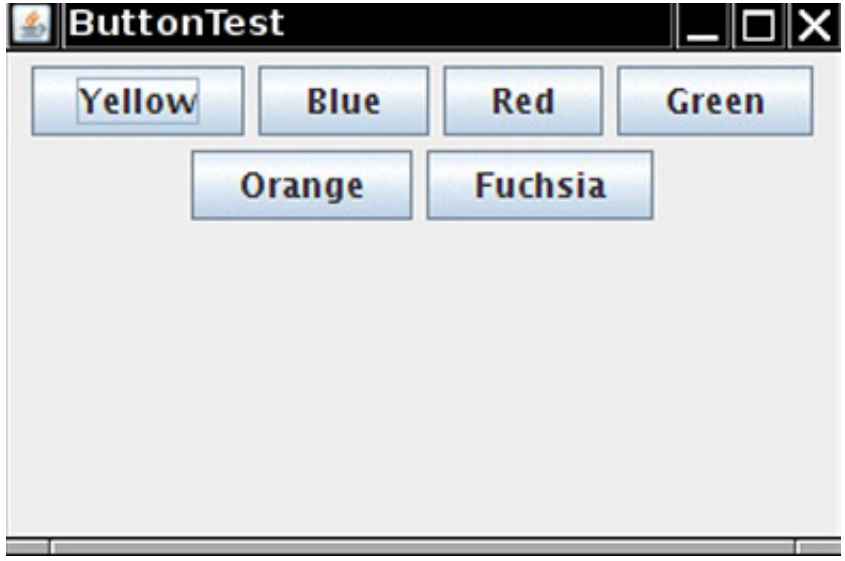
Layout Management

- Layout Managers
  - The buttons are contained in a JPanel object and are managed by the flow layout manager, the default layout manager for a panel
- the buttons stay centered in the panel, even when the user resizes the frame
- components are placed inside containers, and a layout manager determines the positions and sizes of components in a container.

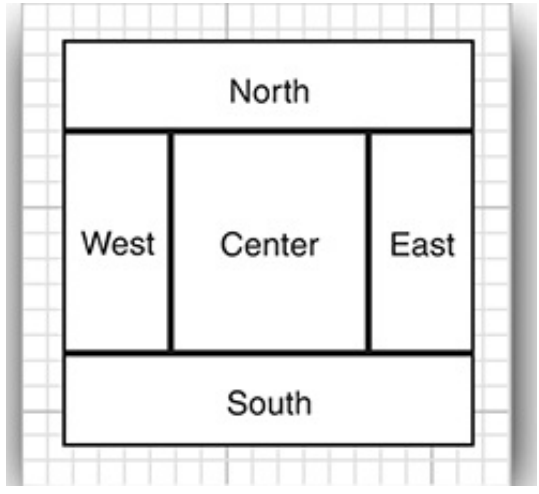
- void setLayout(LayoutManager m)
  - sets the layout manager for this container.
- Component add(Component c)
- Component add(Component c, Object constraints)
  - adds a component to this container and returns the component reference.

- 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.


ButtonTest
Yellow
Blue
Red
Green
Orange
Fuchsia

Border Layout

- The border layout manager is the default layout manager of the content pane of every Jframe
- Unlike the flow layout manager, which completely controls the position of each component
- Border layout manager lets you choose where you want to place each component.
- You can choose to place the component in the center, north, south, east, or west of the content pane


North
West
Center
East
South

- frame.add(yellowButton, BorderLayout.SOUTH);
- var panel = new JPanel();
- panel.add(yellowButton);
- panel.add(blueButton);
- panel.add(redButton);
- frame.add(panel, BorderLayout.SOUTH);


ButtonTest
Yellow
Blue
Red

- BorderLayout()
- BorderLayout(int hgap, int vgap)
  - constructs a new BorderLayout.

Grid Layout

- The grid layout arranges all components in rows and columns like a spreadsheet.
- All components are given the same size.
- The calculator program uses a grid layout to arrange the calculator buttons.
- When you resize the window, the buttons grow and shrink, but all buttons have identical sizes.
- 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

- panel.setLayout(new GridLayout(4, 4));
- panel.add(new JButton("1"));
- panel.add(new JButton("2"));

Text Input

- Components that let a user input and edit text.
- You can use the JTextField and JTextArea components for text input.
- A text field can accept only one line of text;
- a text area can accept multiple lines of text.
- A JPasswordField accepts one line of text without showing the contents.

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 Fields

- JTextField(int cols)
  - constructs an empty JTextField with the specified number of columns.
- JTextField(String text, int cols)
  - constructs a new JTextField with an initial string and the specified number of columns.
- int getColumns()
- void setColumns(int cols)
  - gets or sets the number of columns that this text field should use.

Labels and Labeling Components

- Labels are components that hold text. They have no decorations (for example, no boundaries).
- They also do not react to user input. You can use a label to identify components.
- To label a component that does not itself come with an identifier:
  - 1. Construct a JLabel component with the correct text.
  - 2. Place it close enough to the component you want to identify so that the user can see that the label identifies the correct component.

- That interface defines a number of useful constants such as LEFT, RIGHT, CENTER, NORTH, EAST
- var label = new JLabel("User name: ", SwingConstants.RIGHT);
- var label = new JLabel("User name: ", JLabel.RIGHT);

- 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.

Password Fields

- each typed character is represented by an *echo character,*such as a bullet (•).
- Swing supplies a JPasswordField class that implements such a text field.
- 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

- When you place a text area component in your program,
- a user can enter any number of lines of text, using the Enter key to separate them. Each line ends with a '\n'.
- textArea = new JTextArea(8, 40); // 8 lines of 40 columns each
- You can also use the setColumns method to change the number of columns
- setRows method to change the number of rows.
- textArea.setLineWrap(true); // long lines are wrapped

- 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.
- 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.

Choice Components

- Using a set of buttons or a list of items tells your users what choices they have.
  - Checkboxes
  - radio buttons
  - lists of choices
  - sliders.

Checkboxes

- Checkboxes automatically come with labels that identify them.
- The user can check the box by clicking inside it and turn off the checkmark by clicking inside the box again.
- Pressing the space bar when the focus is in the checkbox also toggles the checkmark

- With *JCheckBox* it is possible to use an *ActionListener* or an *ItemListener*.
- *ItemListener* is the interface for receiving item events.
- The class that is interested in processing an item event, e.g. the observer, implements this interface.
- The observer object is registered with a component using the component's *addItemListener()* method.
- When an item selection event occurs, the observer's *itemStateChanged()* method is invoked.

- **addActionListener(ItemListener l)**: adds item listener to the component
- **itemStateChanged(ItemEvent e)** : abstract function invoked when the state of the item to which listener is applied changes
- **getItem()** : Returns the component-specific object associated with the item whose state changed
- **getStateChange()** : Returns the new state of the item. The ItemEvent class defines two states: SELECTED and DESELECTED.
- **getSource()** : Returns the component that fired the item event.

Constructors and Methods

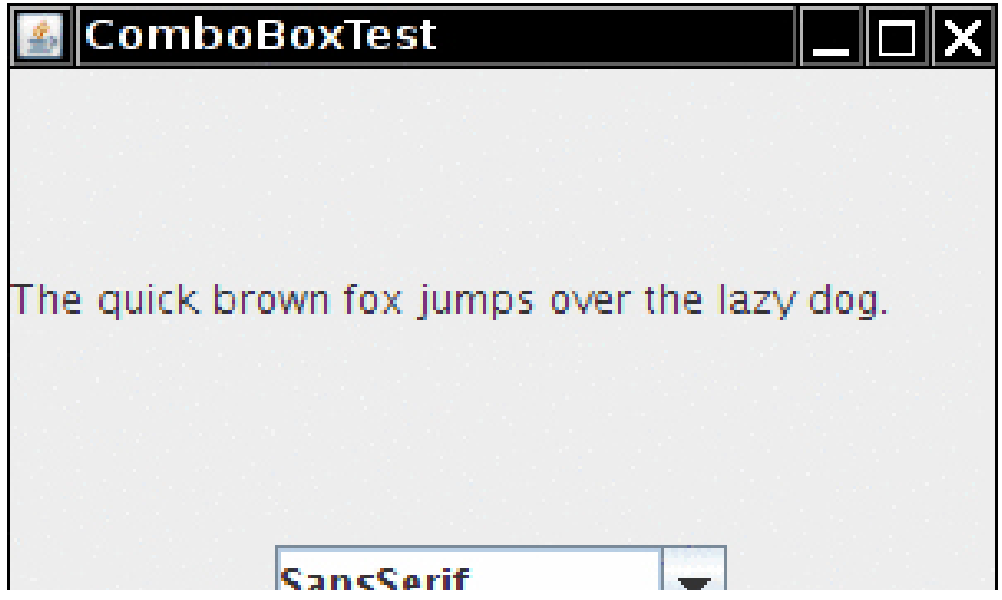
- JCheckBox(String label)
- JCheckBox(String label, Icon icon)
  - constructs a checkbox that is initially unselected.
- JCheckBox(String label, boolean state)
  - constructs a checkbox with the given label and initial state.
- boolean isSelected()
- void setSelected(boolean state)
  - gets or sets the selection state of the checkbox.

Radio Buttons

- to check only one of several boxes.
- When another box is checked, the previous box is automatically unchecked
- JRadioButton(String label, Icon icon)
  - constructs a radio button that is initially unselected.
- JRadioButton(String label, boolean state)
  - constructs a radio button with the given label and initial state.

Combo Boxes

- you can use a combo box.
- When the user clicks on this component, a list of choices drops down, and the user can then select one of them



- JComboBox()
  - Creates a JComboBox with a default data model.
- JComboBox(ComboBoxModel<E> aModel)
  - Creates a JComboBox that takes its items from an existing ComboBoxModel.
- JComboBox(E[] items)
  - Creates a JComboBox that contains the elements in the specified array.
- JComboBox(Vector<E> items)
  - Creates a JComboBox that contains the elements in the specified Vector.

```
import javax.swing.*;
public class ComboBoxExample {
   JFrame f;
   ComboBoxExample(){
      f = new JFrame("ComboBox Example");
      String country[]={“Apple”, “Guava”, “Grapes”, “Mango”, “Orange”};
      JComboBox cb=new JComboBox(country);
      cb.setBounds(50, 50,90,20);
      f.add(cb);
      f.setLayout(null);
      f.setSize(400,500);
      f.setVisible(true);
   }
   public static void main(String[] args) {
      new ComboBoxExample();
   }
}
```

- 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.

Grouping JRadioButton

- JRadioButton rad1 = new JRadioButton("Radio 1");
- JRadioButton rad2 = new JRadioButton("Radio 2");
- JRadioButton rad3 = new JRadioButton("Radio 3");
- ButtonGroup bg1 = new ButtonGroup();
- bg1.add(rad1);
- bg1.add(rad2);
- bg1.add(rad3);
- rad1 .addItemListener(this);
- rad2.addItemListener(this);
- rad3.addItemListener(this);

```
public void itemStateChanged(ItemEvent e) {
        int sel = e.getStateChange();
        if (sel == ItemEvent.SELECTED) {
            JRadioButton button = (JRadioButton)
e.getSource();
            String text = button.getText();
            StringBuilder sb = new
StringBuilder("Selected: ");
            sb.append(text);
            sbar.setText(sb.toString());      }   }
```

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello ");
}
public static void main(String[] args) {
    new OptionPaneExample();
}
}
```

JOptionPane

```
import javax.swing.*;
public class OptionPaneExample {
JFrame f;
OptionPaneExample(){
   f=new JFrame();
   String name=JOptionPane.showInputDialog(f,"Enter Text");
   System.out.println(name);
}
public static void main(String[] args) {
   new OptionPaneExample();
}
}
```

```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
JFrame f;
OptionPaneExample(){
    f=new JFrame();
    f.addWindowListener(this);
    f.setSize(300, 300);
    f.setLayout(null);
    f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    f.setVisible(true);  }
public void windowClosing(WindowEvent e) {
    int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
//showConfirmDialog returns int YES_OPTION,NO_OPTION and CANCEL_OPTION
if(a==JOptionPane.YES_OPTION){
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  }  }
public static void main(String[] args) {
    new  OptionPaneExample();  }    }
```

JMenus

- JMenu(String label)
  - constructs a menu with the given label.
- JMenuItem add(JMenuItem item)
  - adds a menu item (or a menu).
- JMenuItem add(String label)
  - adds a menu item with the given label to this menu and returns the item.
- JMenuItem add(Action a)
  - adds a menu item with the given action to this menu and returns the item.
- void addSeparator()
  - adds a separator line to the menu.
- JMenuItem insert(JMenuItem menu, int index)
  - adds a new menu item (or submenu) to the menu at a specific index.
- JMenuItem insert(Action a, int index)
  - adds a new menu item with the given action at a specific index.
- void insertSeparator(int index)
  - adds a separator to the menu.
- void remove(int index)
- void remove(JMenuItem item)
  - removes a specific item from the menu.

Menus

- var menuBar = new JMenuBar();
- frame.setJMenuBar(menuBar);
- var editMenu = new JMenu("Edit")
- var pasteItem = new JMenuItem("Paste");
- menuBar.add(editMenu);
  editMenu.add(pasteItem);
  editMenu.addSeparator(); JMenu
  optionsMenu = . . .; // a submenu
  editMenu.add(optionsMenu);

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