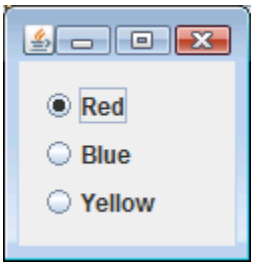
Lesson Three: Option Input

Option controls allow users to select one or more choices from a small group. There are two main types of options – radio buttons and checkboxes.

The JRadioButton Control



The *radio button* control is used when you want the user to select only one option from a small group of items. The user can only select one radio button option at a time. For instance, in this example screen, the user can select either "Red", "Blue" or "Yellow". Each radio button is represented by a **JRadioButton** object.

When you create a **JRadioButton** object, pass into the constructor the text you want displayed next to the selection circle. Don't forget to assign the object to a class-level member variable so you can access it later.

```
JRadioButton redRadio = new JRadioButton("Red");
JRadioButton blueRadio = new JRadioButton("Blue");
JRadioButton yellowRadio = new JRadioButton("Yellow");
```

Radio button controls should work together in a group. When the user selects one button, the other buttons in the group are automatically de-selected. In order to organize a set of **JRadioButton** objects into a related group, you need to create a **ButtonGroup** container and then add each **JRadioButton** to the group.

```
ButtonGroup myGroup = new ButtonGroup();
myGroup.add(redRadio);
myGroup.add(blueRadio);
myGroup.add(yellowRadio);
```

Finally, you need to add each **JRadioButton** object to your panel. Your layout manager will determine how the buttons are arranged on the screen. In the example above we have used a vertical box layout.

```
myPanel.add(redRadio);
myPanel.add(blueRadio);
myPanel.add(yellowRadio);
```

If you want to create multiple groups of radio buttons on the same screen, create one **ButtonGroup** object for each set of radio buttons. The radio buttons within a specific **ButtonGroup** will work as a team, independent of any other radio buttons on the screen.

To figure out which radio button in a group is currently selected, you can call the **isSelected**() method on each control. If method returns **true**, the button is selected. If it returns **false**, the button is not selected.

```
    if (redRadio.isSelected() == true)
    {
        // the user selected the red radio button
    }
```

Your best chance to call **isSelected**() is when some other event happens, such as a button press. Also, each time the user clicks on a radio button, an action event will be sent to an **ActionListener**. To demonstrate, we first implement the **ActionListener** interface and make each **JRadioButton** a class member variable:

```
class MyProgram implements ActionListener
{
    JRadioButton blueRadio = null;
    JRadioButton redRadio = null;
    JRadioButton yellowRadio = null;
```

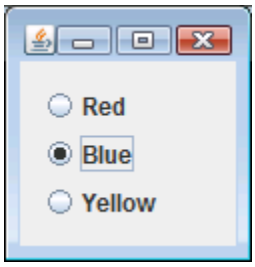
Then, after each variable has been initialized, call the **addActionListener**() method with **this** as a parameter:

```
    redRadio.addActionListener(this);
    blueRadio.addActionListener(this);
    yellowRadio.addActionListener(this);
```

Now in the **actionPerformed**() method you can check the event source to find which button was clicked:

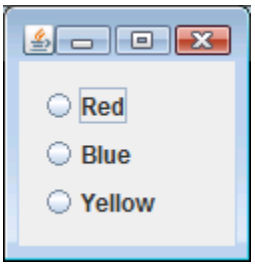
```
    public void actionPerformed(ActionEvent event)
    {
          Object control = event.getSource();
          if (control == redRadio)
          {
                // red radio button was selected
          }
          else if (control == blueRadio)
          {
                // blue radio button was selected
          }
          else if (control == yellowRadio)
          {
                // yellow radio button was selected
          }
    }
```

You can programmatically set the currently selected radio button in any group by using the **JRadioButton setSelected**() method with a **true** parameter:



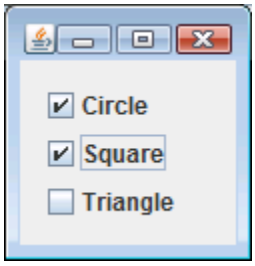

```
    blueRadio.setSelected(true);
```

You can also use this same method to clear a selected radio button with a **false** parameter:




```
    blueRadio.setSelected(false);
```

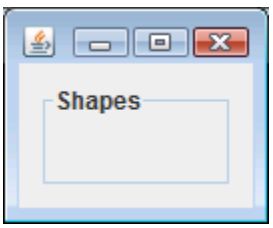
The JCheckBox Control



The *check box* control is very similar to the radio button control. Check boxes are typically grouped together and offer the user a small selection of options. The biggest difference is that a user can select more than one check box at a time (or no check boxes at all)! The check box controls are independent and do not work as a group, even though they are usually arranged in visual groups on the screen.

In Swing, a check box is created with the **JCheckBox** class. Using **JCheckBox** is almost identical to using a **JRadioButton**, except you never need to put **JCheckBoxes** in a **ButtonGroup**. You can construct a **JCheckBox** with the text to display next to the text box. You can call the **isSelected**() and **setSelected**() methods to see if a check box is selected or programmatically set or clear a checkbox. You can also receive an action event through the **ActionListener** interface each time a checkbox is clicked

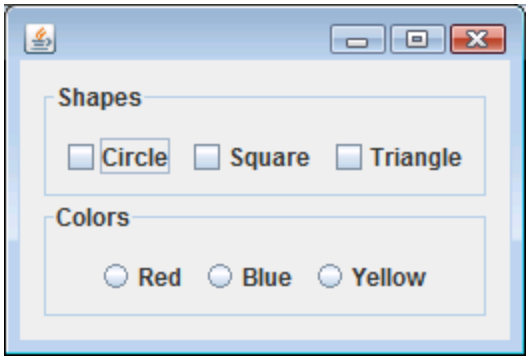
Using Borders



It's often a good idea to also include a visual border around groups of radio buttons and check boxes. You can do this with the same **Border** class we discussed earlier for **JPanels**. Remember the **BorderFactory** is used to create a **Border** object. You can get different kinds of borders by calling different methods on the **BorderFactory**. A handy type of border for radio buttons and check boxes is a "Titled" border, which is a simple frame with a text description at the top. Here is an example:

```
    Border myBorder = BorderFactory.createTitledBorder("Shapes");
```

You still need to assign a border to a **JPanel**, so you would create a new **JPanel** for each group of radio buttons and check boxes. You can then assign a different **Border** to each **JPanel**. Here is an extended example showing two sets of controls on the screen at the same time. For simplicity we are not making class member variables or putting radio buttons into a **ButtonGroup** as you would normally! The code below will produce the example shown to the right.




```
JFrame myFrame = new JFrame();
myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

JPanel myPanel = (JPanel)myFrame.getContentPane();
myPanel.setLayout(new BoxLayout(myPanel,BoxLayout.Y_AXIS));
myPanel.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

JPanel shapePanel = new JPanel();  // create JPanel for check boxes

// add new check boxes to the shape panel
shapePanel.add(new JCheckBox("Circle"));
shapePanel.add(new JCheckBox("Square"));
shapePanel.add(new JCheckBox("Triangle"));

// create a titled border for the shape panel
shapePanel.setBorder(BorderFactory.createTitledBorder("Shapes"));
myPanel.add(shapePanel);  // add the color panel to the main content pane

JPanel colorPanel = new JPanel();  // create JPanel for check boxes

// add new radio buttons to the color panel
colorPanel.add(new JRadioButton("Red"));
colorPanel.add(new JRadioButton("Blue"));
colorPanel.add(new JRadioButton("Yellow"));

// create a titled border for the color panel
colorPanel.setBorder(BorderFactory.createTitledBorder("Colors"));
myPanel.add(colorPanel);   // add the color panel to the main content pane

myFrame.pack();             // pack all of the controls
myFrame.setVisible(true);   // finally, show the screen!
```