Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

Event Handling in JDK 1.1, Item Events

Java Programming, Lecture Notes # 99, Revised 02/24/99.

Preface

Students in Prof. Baldwin's **Intermediate Java Programming** classes at ACC will be responsible for knowing and understanding all of the material in this lesson beginning with the Spring semester of 1999.

On 2/24/99, the sample program in this lesson was confirmed to operate properly under JDK 1.2 and Win95.

Introduction

This lesson was originally written on September 22, 1998, using the JDK 1.1.6 download package. The purpose of this lesson is to illustrate the use of *item* events.

Overview

If you instantiate an object of type **ItemListener** and register that object on an object that has an **addItemListener()** method, the **itemStateChanged()** method of the listener object will be invoked whenever the state of the source object changes.

An item event is a semantic event, and several different sources can multicast the event. I will explain the use of the **ItemListener** interface and **ItemEvent** class using radio buttons and checkboxes.

Information regarding the event is passed into the **itemStateChanged()** method in the form of an object of type **ItemEvent**.

Sample Program

The program creates three radio buttons in a group. When any one of the three buttons is selected, the other two are automatically deselected.

An **ItemListener** object is registered on all three buttons. When a button is selected, an **ItemEvent** occurs.

The program also creates an ordinary **Checkbox** object. The **ItemListener** object is also registered on the checkbox. When the checkbox is either selected or deselected an **ItemEvent** occurs.

When an **ItemEvent** occurs, the **itemStateChanged()** method of the **ItemListener** object is invoked on the listener object. Code in this method invokes methods of the **ItemEvent** object (received as a parameter) to display the following information about the event:

1. the *label* of the item,
2. the *"StateChange"* of the item, and
3. the new *state* of the item.

The labels on the three buttons are **AButton**, **BButton**, and **CButton**. The label on the checkbox is **Check Box**.

The following is the output produced by selecting BButton, selecting Check Box, and then deselecting Check Box, in that order.

```
Item: BButton
State Change: 1
State: true

Item: Check Box
State Change: 1
State: true

Item: Check Box
State Change: 2
State: false
```

Several things are worthy of note here. First, a state change from deselected to selected produces a state change value of 1, while a change from selected to deselected produces a value of 2. At least that is true on the checkbox. So far, I have found no documentation to explain the significance of these values.

Second, the checkbox generates an item event when it is selected, and also when it is deselected.

Third, whenever one of the radio buttons is selected, the others are automatically deselected. However, only one item event occurs, and it is attributed to the button that is selected. Automatic deselection of the other buttons does not cause them to generate item events.

Tested using JDK1.1.6 under Win95.

Interesting Code Fragments

We will begin with the first line of the definition of the controlling class, which simply shows that this class extends **Frame** and implements **ItemListener**. Because it implements **ItemListener**, an object of the controlling class is a listener object.

```
public class Event35 extends Frame implements
ItemListener{
```

I will skip the **main** method (that does nothing but instantiate an instance of the controlling class) and move on to the constructor.

The next fragment shows the instantiation of a set of three radio buttons in a group. Another lesson discusses radio buttons in detail. Since the objective of this lesson is to understand item events, and not radio buttons, I won't discuss this code further here.

```
  Event35(){//constructor
    //Create a CheckboxGroup object
    CheckboxGroup cbGrp = new CheckboxGroup();

    //Create three radio buttons in the group
    Checkbox aButton = new
Checkbox("AButton",true, cbGrp);
    Checkbox bButton = new
Checkbox("BButton",false,cbGrp);
    Checkbox cButton = new
Checkbox("CButton",false,cbGrp);
```

The next fragment uses a typical registration method to register **this** object (which is a listener object because it is of a class that implements the **ItemListener** interface) on each of the radio buttons.

```
    //Register item listener object on each
radio button
    aButton.addItemListener(this);
    bButton.addItemListener(this);
    cButton.addItemListener(this);
```

The next fragment creates an ordinary **Checkbox** object and registers the item listener on it also.

```
    Checkbox theCheckbox = new Checkbox("Check
Box");
    theCheckbox.addItemListener(this);
```

Following this, I add the radio buttons and the checkbox to the frame, adjust the parameters of the frame, and make the whole thing visible. That is very familiar code, so I didn't highlight it here. I also instantiated an anonymous **WindowListener** to terminate the program when the user clicks on the *close* button on the frame. I didn't highlight that here either because you should already be familiar with it. You can view all of this code in the programming listing that follows later.

The next fragment defines the **itemStateChanged()** method of the listener interface. This method is invoked whenever an **ItemEvent** occurs on one of the source objects on which the listener is registered. As you can see, the code in this method extracts information from the incoming **ItemEvent** object and displays that information as shown earlier.

Note that this is a method of the controlling class, causing an object of the controlling class to also be a listener.

```
  //Define the method of the ItemListener
interface
  public void itemStateChanged(ItemEvent e){
    System.out.println("Item: " + e.getItem());
    System.out.println("State Change: "
                                 +
e.getStateChange());

    //Note the cast in the following
statement
    System.out.println("State: "
                   +
((Checkbox)e.getSource()).getState());
    System.out.println();//blank line
  }//end itemStateChanged()

}//end class Event35 definition
```

The code in the program that was not highlighted in the above fragments can be viewed in the complete listing of the program that follows in the next section.

Program Listing

This section contains program listings.

```
/*File Event35.java, Copyright 1998, R.G.Baldwin
Illustrates Item events.

This program creates three radio buttons in a group.  When
any of the three buttons is selected, the other two are
automatically deselected.

An ItemListener object is registered on all three buttons.
When a button is selected, an ItemEvent occurs.

The program also creates an ordinary checkbox. The
ItemListener object is also registered on the checkbox.
When the checkbox is either selected or deselected an
ItemEvent occurs.

When an ItemEvent occurs, the itemStateChanged() method
of the ItemListener object is invoked.  Code in this method
invokes methods of the ItemEvent object received as a
parameter to display

1. the label of the item,
2. the "StateChange" of the item, and
3. the new state of the item.

The labels on the three buttons are AButton, BButton, and
CButton.  The label on the checkbox is Check Box.

The following is the output produced by selecting BButton,
selecting Check Box, and then deselecting Check Box, in
that order.

Item: BButton
State Change: 1
State: true

Item: Check Box
State Change: 1
State: true

Item: Check Box
State Change: 2
State: false

Several things are worthy of note here.  First, a state
change from deselected to selected produces a state
change value of 1, while a change from selected to
deselected produces a value of 2.  So far, I have found no
```

```
documentation to explain the significance of these values.

Second, the checkbox generates an item event when it is
selected, and also when it is deselected.

Third, whenever one of the radio buttons is selected, the
others are automatically deselected.  However, only one
item event results, and it is attributed to the button
that is selected.  Automatic deselection of the other
buttons does not cause them to generate item events.

Tested using JDK1.1.6 under Win95.
**********************************************************/

import java.awt.*;
import java.awt.event.*;
import java.util.*;
//=======================================================//
public class Event35 extends Frame implements ItemListener{
  public static void main(String[] args){
    new Event35();
  }//end main
  //-----------------------------------------------------//

  Event35(){//constructor
    //Create a CheckboxGroup object
    CheckboxGroup cbGrp = new CheckboxGroup();

    //Create three radio buttons in the group
    Checkbox aButton = new Checkbox("AButton",true, cbGrp);
    Checkbox bButton = new Checkbox("BButton",false,cbGrp);
    Checkbox cButton = new Checkbox("CButton",false,cbGrp);

    //Register item listener object on each radio button
    aButton.addItemListener(this);
    bButton.addItemListener(this);
    cButton.addItemListener(this);

    //Create an ordinary checkbox and register the item
    // listener on it.
    Checkbox theCheckbox = new Checkbox("Check Box");
    theCheckbox.addItemListener(this);

    //Add the radio buttons and the checkbox to the Frame
    this.add(aButton);
    this.add(bButton);
    this.add(cButton);
    this.add(theCheckbox);

    //Adjust Frame parameters and make it visible
    this.setLayout(new FlowLayout());
    this.setSize(350,100);
    this.setTitle("Copyright 1998, R.G.Baldwin");
    this.setVisible(true);

    // Anonymous inner class to terminate program.
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);}});//end WindowListener
  }//end constructor
  //-----------------------------------------------------//

  //Define the method of the ItemListener interface
  public void itemStateChanged(ItemEvent e){
    System.out.println("Item: " + e.getItem());
    System.out.println("State Change: "
                                 + e.getStateChange());

    //Note the cast in the following statement
    System.out.println("State: "
                   + ((Checkbox)e.getSource()).getState());
    System.out.println();//blank line
  }//end itemStateChanged()

}//end class Event35 definition
//=======================================================//
```

-end-