MULTIPLE EVENT HANDLING

If your GUI contains a number of buttons, each generates an `ActionEvent` object. Also, pressing the Enter key on a text field generates an `ActionEvent` object. Thus, a GUI window can contain many buttons and text fields, all generating `ActionEvent` objects. How to determine which button or text field is the source of a specific event is the subject of this topic.

**Binary Converter Example**

We are going to build an application that converts binary strings (base 2) to decimal (base 10). We start with an application that creates the GUI Window shown to the right. The complete application follows:

```java
import javax.swing.*;
import java.awt.*;

public class BinaryConverter extends JFrame {
    public static JFrame win;

    public static void main ( String[] args ) {
        win = new BinaryConverter();
    }

    Font btnFont = new Font( "Dialog", Font.PLAIN, 18 );
    Font txtFont = new Font( "Courier New", Font.PLAIN, 24 );

    JButton[] digitBtn = new JButton[2];
    JButton backSpaceBtn;
    JTextField entry;

    public BinaryConverter () {
        // window management
        super( "Binary Converter" );
        this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        // configure text field
        entry = new JTextField( 10 );
    }
}
```
Event Handling for the Binary Converter

**Digit Button Handling** We want the digit buttons 0 and 1 to append the associated digit into the text field display.

**Example**
Clicking 1 followed by 0 followed by 1 results in the GUI shown to the right.
Here’s the algorithm for handling these buttons.

Algorithm for Handling the Digit Buttons
grab the entry text
append the appropriate digit (0 or 1) to its right
replace old entry text with the result

The algorithm coded as an `actionPerformed` method is shown below. Note the use of the `getActionCommand` method – which is an instance method of the `ActionEvent` object – to get the digit that labels the button. This allows the same handling code – and therefore only one listener class – to be used for both buttons.

```java
String getActionCommand( )
    // Return the caption string of the GUI component that is the
    // source of the event.

public void actionPerformed( ActionEvent e )
{
    // append button label to right side of entry
    entry.setText( entry.getText( ) + e.getActionCommand( ) );
}
```

**Backspace Button Handling** We want the backspace button ▼ to backspace the text field display.

**Example**
Clicking 1 three times followed by ▼ twice results in the GUI shown to the right.
Following is the algorithm for handling this event. The if statement prevents the user from backspacing when the entry has no characters in it.

```
Algorithm for Handling the Backspace Button
grab the entry text
if the entry length is > 0 then
    remove the rightmost digit
    replace old entry text with the result
end if
```

The algorithm coded as an `actionPerformed` method is shown below.

```java
public void actionPerformed( ActionEvent e )
{
    // remove rightmost bit of entry
    String s = entry.getText( );
    if ( s.length( ) > 0 )
        entry.setText( s.substring( 0, s.length( ) - 1 ) );
}
```

**Text Field Handling** Finally, the user must be able to bypass the digit buttons and type an entry directly into the text field. Unfortunately, this opens the possibility that the user can incorrectly type a character that is not part of a legal binary string, as in 01210. Since the text field only generates an `ActionEvent` object once the user presses the Enter key, the handling code cannot prevent the user from typing a bad character; it can only warn the user that he or she has done so. This utility method checks that a string only contains 0s and 1s.

```java
private boolean isBinary( String s )
{ // Return true if s contains a binary string of 0s and 1s.
    for ( int k = 0; k < s.length( ); k++ )
    {
        if ( s.charAt( k ) != '0' && s.charAt( k ) != '1' )
            return false;
    }
    return true;
}
```
The `actionPerformed` method uses `isBinary` to validate the contents of the text field.

```java
public void actionPerformed( ActionEvent e )
{
    // validate entry
    if ( ! isBinary( entry.getText( ) ) )
    {
        JOptionPane.showMessageDialog
            ( win, "Illegal binary string" );
        entry.setText( "" );
    }
}
```

Given this `actionPerformed` method, if the user enters a legal binary string and presses the Enter key then nothing obviously happens. The method validates the string and, finding it valid, leaves it alone.

If, however, the user enters a bad character, the method displays an error message.
**Listener Implementations for the Binary Converter**
This table summarizes the ways to implement listeners to handle multiple `ActionEvents`. There are two basic strategies – (1) code a separate listener object for each GUI component, or (2) code a single listener that handles all the components. Within each strategy Java allows you some coding shortcuts.

<table>
<thead>
<tr>
<th>Methods of Coding Listeners to Multiple ActionEvents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strategy</strong></td>
</tr>
<tr>
<td>Individual Listeners</td>
</tr>
<tr>
<td>Single Listener</td>
</tr>
</tbody>
</table>
Individual Listeners  The simplest way to handle multiple GUI components is to create a separate listener class for each component as is shown in the topic Single Event Handling. This technique is pictured below for the BinaryConverter example.
**Individual Listener Coding Shortcut** Although the coding solution shown above is simple to understand, it creates several nested classes that used only once in the program. Java allows you to create anonymous subclasses directly within the object creation operator `new`. This technique is shown in the picture below. Realize that this solution is functionally the same as the previous solution. We’ve simply gotten rid of the explicitly named subclasses. Note also that the GUI components can now be declared locally to the method that constructs the window.

```java
import java.awt.event.*;

public class BinaryConverter extends JFrame {
    public static JFrame win;
    public static void main ( String [] args )
    {
        win = new BinaryConverter();
    }

    Font btnFont = new Font( "Dialog", Font.PLAIN, 18 );
    Font txtFont = new Font( "Courier New", Font.PLAIN, 24 );

    public BinaryConverter() {
        // window management
        super( "Binary Converter" );
        this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        // configure text field
        JTextField entry = new JTextField( 10 );
        entry.setHorizontalAlignment( JTextField.RIGHT );
        entry.setFont( txtFont );
        this.add( entry );
        // configure number buttons
        JButton [] digitBtn = new JButton[2];
        for ( int k = 0; k < 2; k++ )
        {
            digitBtn[k] = new JButton( Integer.toString( k ) );
            digitBtn[k].setFont( btnFont );
            this.add( digitBtn[k] );
        }
        // configure Backspace button
        JButton backSpaceBtn = new JButton( "\u2190" );
        backSpaceBtn.setFont( btnFont );
        this.add( backSpaceBtn );
        // configure window
        this.setLayout( new FlowLayout( ) );
        this.setSize( 400, 150 );
        this.setVisible( true );
    }

    private boolean isBinary( String s ) {
        for ( int k = 0; k < s.length(); k++ )
        {
            if ( s.charAt( k ) != '0' && s.charAt( k ) != '1' )
                return false;
        }
        return true;
    }
}
```
**Single Listener** A single listener that handles the event no matter which GUI component initiates it can be constructed using the following instance method in the `ActionEvent` object.

```java
Object getSource()
// Return the GUI component that is the source of the event.
```

Using this method, you can code the listener to execute the appropriate handling code by using a cascading `if-else` statement to determine which GUI component is the event source.
Single Listener Coding Shortcut In the single listener solution, you can eliminate the inner class by making the application itself the listener.

```java
import javax.swing.*;
import java.awt.*;
public class BinaryConverter extends JFrame implements ActionListener {
    public static void main ( String [] args )
    {
        new BinaryConverter();
    }
    Font btnFont = new Font( "Dialog", Font.PLAIN, 18 );
    Font txtFont = new Font( "Courier New", Font.PLAIN, 24 );
    JButton [] digitBtn = new JButton[2];
    JButton backSpaceBtn;
    JTextField entry;
    public BinaryConverter()
    {
        // window management
        super( "Binary Converter" );
        this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        // configure text field
        entry = new JTextField( 10 );
        entry.setHorizontalAlignment( JTextField.RIGHT );
        entry.setFont( txtFont );
        this.add( entry );
        // configure number buttons
        for ( int k = 0; k < 2; k++ )
        {
            digitBtn[k] = new JButton( Integer.toString( k ) );
            digitBtn[k].setFont( btnFont );
            this.add( digitBtn[k] );
        }
        // configure Backspace button
        backSpaceBtn = new JButton( "\u2190" );
        backSpaceBtn.setFont( btnFont );
        this.add( backSpaceBtn );
        // configure window
        this.setLayout( new FlowLayout( ) );
        this.setSize( 400, 150 );
        this.setVisible( true );
    }
    private boolean isBinary( String s )
    {
        // Return true if s contains a binary string of 0s and 1s.
        for ( int k = 0; k < s.length(); k++ )
        {
            if ( s.charAt( k ) != '0' && s.charAt( k ) != '1' )
                return false;
        }
        return true;
    }
    public void actionPerformed( ActionEvent e )
    {
        if ( e.getSource() == entry )
        {
            // code to handle the entry text field
        }
        else if ( e.getSource() == backSpaceBtn )
        {
            // code to handle the backspace button
        }
        else // source is a digit button
        {
            // code to handle the digit buttons
        }
    }
    digitBtn[k].addActionListener( this );
    backSpaceBtn.addActionListener( this );
    entry.addActionListener( this );
    }
}
```
### Exercises

1. Implement event handling for the **BinaryConverter** application shown on pages 1 and 2. Use the handling code developed on pages 3, 4 and 5. Code individual listeners as shown in the picture on page 7.

2. Modify your solution to problem 1 to use the coding shortcuts shown in the picture on page 8.

3. Repeat problem 1, this time coding a single listener as shown in the picture on page 9.

4. Modify your solution to problem 3 to use the coding shortcuts shown in the picture on page 10.

5. This exercise is an extension of Exercise 8 of the topic **GUI Applications**. Create the GUI described below:

   1. It contains four GUI components.
      1.1. A **JButton** object, the UP button.
      1.2. A **JLabel** object containing “X =.”
      1.3. A **JTextField** object.
      1.4. A **JButton** object, the DOWN button.

   Here’s how the GUI must behave:

   2. The text field must accept user input.
      2.1. Its text must be right aligned.
      2.2. It must initially contain 0.
      2.3. When the user presses the Return key, the program checks to make sure the text is a legal non-negative integer (i.e. all decimal digit characters having integer value of zero or more).
         2.3.1. If not, the program displays a **JOptionPane** message dialog indicating the text has illegal characters.
         2.3.2. When the user clicks OK on the message dialog, the program erases the text in the text field.
   3. The UP button must increment the text field value.
      3.1. When clicked, the program first carries out the validation steps of 2.3.
      3.2. If the text is not valid, the program behaves as in 2.3.1 and 2.3.2.
3.3. If the text is valid, the program increments the number in the text field and replaces the text field contents with the result.
4. The DOWN button must decrement the text field value.
   4.1. It behaves as the UP button description in specification 3 except that it decrements the number.
   4.2. Initially, the button must be disabled.
   4.3. The button must be disabled if the text field value becomes 0.

6. This exercise is an extension of Exercise 10 of the topic *GUI Applications*. Create the GUI described below:

1. It contains six GUI components.
   1.1. A `JTextField` object, the “Fahrenheit temperature text field.”
   1.2. A `JLabel` object containing “°F.”
   1.3. A `JButton` object, the “convert to Celsius” button.
   1.4. A `JButton` object, the “convert to Fahrenheit” button.
   1.5. A `JTextField` object, the “Celsius temperature text field.”
   1.6. A `JLabel` object containing “°C.”

Here’s how the GUI must behave:

2. The convert to Celsius button converts the Fahrenheit temperature to Celsius.
   2.1. When clicked, the program converts the Fahrenheit temperature text to a `double` value and places it back into the Fahrenheit text field formatted to two decimal places.
   2.2. It converts the Fahrenheit temperature to Celsius, formats it to two decimal places and places the result into the Celsius temperature text field.
3. The convert to Fahrenheit button converts the Celsius temperature to Fahrenheit.
   3.1. When clicked, the program converts the Celsius temperature text to a `double` value and places it back into the Celsius text field formatted to two decimal places.
   3.2. It converts the Celsius temperature to Fahrenheit, formats it to two decimal places and places the result into the Celsius temperature text field.
4. The user pressing the Enter key on the Fahrenheit temperature text field produces the same action as a mouse click on the convert to Celsius button.
5. The user pressing the Enter key on the Celsius temperature text field produces the same action as a mouse click on the convert to Fahrenheit button.