

## Class 20 Notes

### Event Driven Programming

**OK Now it is time to make the GUIs do something. This is called **Event Driven Programming****

An **event** is an occurrence to which a Component object, such as a JButton, can respond.

When you click a button, an event occurs (or is **generated**). Clicking a checkbox, or radio button generates an event. Selecting an item from a menu also generates an event. In fact, simply moving the mouse generates an event.

A program usually responds to an event with a method call. For example, clicking the X on



generates an event. The response to this event is a call to a method that closes a window. Programs that respond to events are called **event-driven programs**.

Events occur but a program may or may not respond to an event. In any event-driven program, many events can occur but only some are significant. Many events are generated, but only a few are acted on. For example, each mouse click generates an event but only some clicks require a response. If a button is disabled ("greyed-out") you can click on it but there will be no response. An event has occurred but there is no response by the system

#### The Delegation Event Model

When you click a button (or check a box or click the mouse) an **EventObject, e**, is automatically created with information about the event, such as which button was clicked or which box was checked. Remember, objects hold data/information.

This is similar to Exceptions: when an array goes out of bounds an **ArrayIndexOutOfBoundsException** object is created with information about the exception. When you are missing a file an **IOException** object may be created. So clicking a button creates an **EventObject** just as a missing file creates an **IOException**

OK you clicked the button and an **EventObject** was created...now what?

We saw that to handle an exception the **Exception** object was passed to the catch block. And, the catch block handled the exception. The idea here is similar...but not exactly the same.

This **EventObject, e**, is passed to a **"listener" or "handler"** -- kinda like the catch block. The listener handles the event. This is like passing the exception to the catch block. The only difference is that the **listener is an object** with methods that handle the events.

OK let's go back

You click a button...An **EventObject** is created (with information about the event) that object is passed to a listener, an object with methods that handle the event).

The last part is that the button has to be connected to a particular listener. Or we say a button has to **register with a listener**.

This again is similar to Exceptions. A try block is attached to its catch blocks. Now a button (or any other component) must be **registered (connected to, attached to) its listeners**.

#### Summary:

- Whenever an event is generated, an **object** of type **EventObject** **e** is created. This object **e** contains information about the event (what key was pressed, the number of mouse clicks, the source of the event – e.g. a button, the mouse).

The component that generates the event (e.g. button, mouse, key) is the **source object**. So if you click on a button that button is the **source object** and magically an **EventObject** is created. **This magically created object has information about the event**

- The **EventObject** object is passed to one or more **listeners (or handlers)**. The listeners process or handle the event. The listener may change the picture on a label or exit the program or play a game. **A listener is an object** The listener methods do the work. The listener is the servant. The listener methods handle the event. You will create a listener class with methods that handle events.
- **Listeners must register with a source** in order to receive events from that source (e.g. button) In other words, a connection must be established between the source ( e.g. button) and a listener. If no connection is established, the listener will listen forever while the source creates unprocessed events. *Not registering the listener is a common cause of errors in Java event-driven programming.*

At this point it all seems very abstract. The basic idea

- You click on a button and an event object is created
- That event object is passed to the listener—like the catch block but a class
- The listener handles the event i.e. does what the button click is supposed to do
- The source (button) must register with the listener for this to work

Thus the principal actors are:

### 1. Source Object

This is the GUI component on which an event is generated (e.g., a button, a textbox, a list, a mouse button, a checkbox, a radio button).

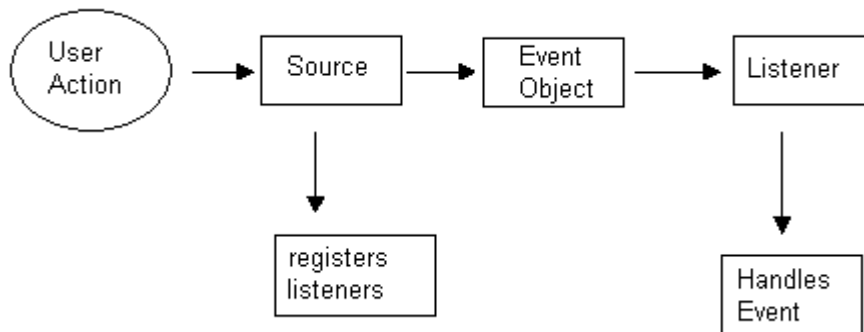
### 2. Event Object

This is the object that contains information about the event. The `EventObject` class contains two important methods: `getSource()` and `toString()`. The `getSource()` method returns the source of the event (a particular button, the mouse etc.), and the `toString()` method returns a string equivalent of the event.

### 3. Listener

A listener **object** waits or listens for an event to occur. A listener is notified when an event occurs. In some sense, the source *phones* the listener. A listener must be *registered* with one or more sources and it must implement methods to receive and respond to an event.

Here is the whole thing in pictures. The user action is , say ,click a button or check a box  
The source is the button.



There is just one more important detail

Remember **interfaces**...well here they are again. Recall: **If a class implements an interface that class is obligated to implement the methods of the interface.**

*Every listener class must implement a particular appropriate listener interface, i.e., a listener has a contract to implement certain methods. A listener is not an independent agent. Every listener implements some specific interface.*

We will just work with buttons and labels for now.

When a button is clicked

- An `ActionEvent` *e* (object) is generated --just as an `IOException` *e* object is created
- The listener class must implement the interface **`ActionListener`**
- The interface `ActionListener` has one method **`ActionPerformed(ActionEvent E)`**

This will become clearer with an example.

To handle any event:

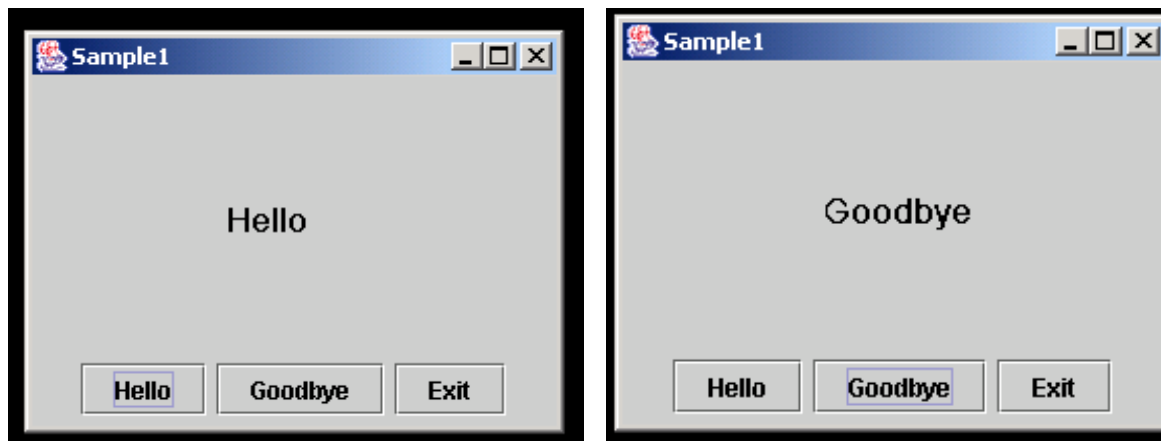
1. Make a class that implements the appropriate listener interface. With a JButton the interface is **ActionEvent**. The methods of the interface must be implemented. In the case of a JButton it is **ActionPerformed(ActionEvent, e)**
2. Register the listener objects with the event source by using the *addXXXListener* method . With a JButton it is `addActionListener(...)`

This is very abstract, and you should understand it only in an abstract sense.

If you understand the basic abstract ideas, an example will show you how to implement them<any programs have th same structure

Example:

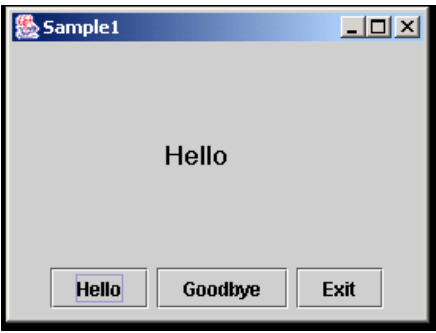
The next very simple program creates the following GUI:



The GUI contains three buttons: Hello, Goodbye, and Exit. There is a JLabel in the CENTER. When the user clicks the Hello button, “Hello” appears on the label. When the user clicks the Goodbye button, “Goodbye” appears on the label. The Exit button ends the program. When the program starts, the string “Hello” is displayed. What could be simpler?

Setting up the GUI is easy. You have already done this with previous programs. Simply arrange three buttons on a panel and place the panel in the SOUTH. Then a label in the CENTER.

This first section of code **just sets up the GUI**. This is no different than before. The buttons do nothing yet. The **buttons are declared globally at the top** and **not in the constructor** so that they are visible for the whole program. The right column gives an explanation of the code

<pre> import java.awt.*; import javax.swing.*; j <b>java.awt.event.* //NOTE</b> public class Sample1 extends JFrame {     <b>private JButton hello;</b>     <b>private JButton goodbye;</b>     <b>private JButton exit;</b>     <b>private JLabel label;</b>      public Sample1()     {         setTitle("Sample1");         setBounds(267,200,267,200);         hello = new JButton("Hello");         goodbye = new JButton("Goodbye");         exit = new JButton("Exit");          JPanel panel = new JPanel();         panel.add(hello);         panel.add(goodbye);         panel.add(exit);          add(panel,BorderLayout.SOUTH);          label = new JLabel("", <b>SwingConstants.CENTER</b>);         label.setFont(new Font("Arial", Font.BOLD,16));         label.setText("Hello");         add(label);         setVisible(true);     }      public static void main(String args[])     {         Sample1 s1 = new Sample1();     } } </pre>	<p><b>You must import java.awt.event.*</b></p> <p><b>NOTES</b></p> <p>Notice I declare the buttons <b>OUTSIDE</b> the constructor. Otherwise they would be unknown outside the constructor</p> <p>The constructor instantiates a JPanel, adds the buttons to the panel. And adds the panel to the SOUTH.</p> <p>It also adds the label to CENTER. Notice the label uses <b>setFont</b> and <b>setText</b></p> <p><b>SwingConstants.CENTER</b> make the text centered in the label</p> <p>The program produces the following frame. The buttons do nothing. It does nothing yet.</p> 
---	--

Step one of our two-step process is:

- Declare a listener class that implements the appropriate listener interface.
- Code all the methods of the listener interface.

OK...Clicking a button *always* generates an `ActionEvent`, e. **The listener class must implement the Java interface `ActionListener`**. And `ActionListener` has one method : **`actionPerformed(ActionEvent e)`**  
Thus to handle a button's `ActionEvent`:

- Declare a class ( the listener) that implements the `ActionListener` interface.
- Implement the listener's single method (`actionPerformed(ActionEvent e)`):  
`actionPerformed(ActionEvent e)`

This part may be a little strange.

The listener class is implemented INSIDE The JFrame class. It is called an **inner class** and it is private.

I am now adding the listener to the frame...It is a PRIVATE INNER CLASS. In RED. The rest of the code is the same as above. I am adding just a few (color coded) lines. Look at the code and read the comments on the right. **YOU MUST IMPORT java.awt.event.\***

<pre> import java.awt.*; import javax.swing.*; <b>import java.awt.event.*;</b> public class Sample1 extends JFrame {     private JButton hello;     private JButton goodbye;     private JButton exit;     private JLabel label;      public Sample1()     {         setTitle("Sample1");         setBounds(267,200,267,200);         hello = new JButton("Hello");         goodbye = new JButton("Goodbye");         exit = new JButton("Exit");          JPanel panel = new JPanel();         panel.add(hello);         panel.add(goodbye);         panel.add(exit);          add(panel,BorderLayout.SOUTH);          label = new JLabel("", SwingConstants.CENTER);         label.setFont(new Font("Arial", Font.BOLD,16));         label.setText("Hello");         add(label);         setVisible(true) ;     }     <b>private class ButtonHandler implements</b> <b>ActionListener</b>     {         <b>public void actionPerformed(ActionEvent e)</b>         { <b>if (e.getSource()== hello) // declared name</b>         {             <b>label.setText("Hello");</b>         } <b>else if (e.getSource() ==goodbye) // declared name</b>         {             <b>label.setText("GoodBye");</b>         } <b>else</b>             <b>System.exit(0);</b>         }     }     public static void main(String args[])     {         Sample1a s1 = new Sample1();     } } </pre>	<p>The listener class ButtonHandler (<b>in red bold</b>) must implement the interface <b>ActionListener</b> <b>Look at it...</b> ButtonHandler is a <b>private inner class</b>. (look at it!!!)</p> <p>The ButtonHandler class, by contract, must implement the method.</p> <p><b>actionPerformed(ActionEvent e)</b></p> <p>The actionPerformed method does the work.</p> <p>It determines the source of the event using the <b>getSource()</b> method of e.</p> <p>If the event source is hello (the hello button) The text on the label is set to "Hello"</p> <p>If the source of the event is goodbye then <b>label.setText("GoodBye")</b></p> <p>If the source of the event is the exit button the program terminates.</p> <ul style="list-style-type: none"> <li>• So far, the program contains buttons that generate events, and</li> <li>• a listener to handle the events</li> </ul> <p>However, no connection has been established between buttons and the listeners.</p> <p>The button is sending the event, the listener is set up to handle the event, <b>but there is no communication between these objects...yet.</b></p> <p>BTW notice the double equals == <b>e.getSource() == hello</b></p>
--	---

Step two is registration, i.e., make a connection between the button and the listener.

**Register the listener objects with event source by using the addActionListener method.**

<pre> import java.awt.*; import javax.swing.*; <b>java.awt.event.*</b> public class Sample1 extends JFrame {     private JButton hello;     private JButton goodbye;     private JButton exit;     private JLabel label;      public Sample1()     {         setTitle("Sample1");         setBounds(267,200,267,200);         hello = new JButton("Hello");         goodbye = new JButton("Goodbye");         exit = new JButton("Exit");          JPanel panel = new JPanel();         panel.add(hello);         panel.add(goodbye);         panel.add(exit);          add(panel,BorderLayout.SOUTH);          label = new JLabel("", SwingConstants.CENTER);         label.setFont(new Font("Arial", Font.BOLD,16));         label.setText("Hello");         add(label);         setVisible(true) ;          // register listener with buttons         <b>hello.addActionListener(new ButtonHandler());</b>         <b>goodbye.addActionListener(new ButtonHandler());</b>         <b>exit.addActionListener(new ButtonHandler());</b>     }     private class ButtonHandler implements         ActionListener     {         public void actionPerformed(ActionEvent e)         {             if (e.getSource()== hello)             {                 label.setText("Hello");             }             else if (e.getSource() ==goodbye)             {                 label.setText("GoodBye");             }             else                 System.exit(0);         }     } } </pre>	<p>To make a connection between the source objects</p> <p>JButton hello, JButton goodbye JButton exit</p> <p>and the Listener</p> <p>ButtonHandler</p> <p>register the Listener object with each JButton:</p> <p>Here is how:</p> <p>Register the listener with the hello button <b>hello.addActionListener(new ButtonHandler());</b></p> <p>Register the listener with the goodbye button <b>goodbye.addActionListener(new ButtonHandler());</b></p> <p>Register the listener with the exit button <b>exit.addActionListener(new ButtonHandler());</b></p> <p>In the example, this registration is <b>done in the constructor.</b> It is in Red</p>
--	--

<pre>public static void main(String args[]) {     Sample1 s1 = new Sample1(); } }</pre>	
---	--

## Here is a summary for the JButton class

**Class:** JButton

**Generates:**(ActionEvent (when clicked)

**Listener:** implements ActionListener (that is the interface the listener must implement)

**Method to implement:** actionPerformed( ActionEvent e)

### Constructors:

- JButton(String s)
- JButton(Icon icon) // JButton button( new ImageIcon("ZAP.gif"));
- JButton(String s, Icon icon)

### Properties: (set and get methods exist for each of the following properties)

- text – e.g. button.setText( "Hello")
- icon -- e.g. button.setIcon( new ImageIcon("ZAP.gif") )
- horizontalAlignment – SwingConstants.LEFT, RIGHT, CENTER etc. How text/icons are placed on button (setHorizontalTextPosition(SwingConstants.RIGHT)
- verticalAlignment -- SwingConstants.TOP, CENTER, or BOTTOM



**Example:** This is almost identical to the last example. You should first set up the GUI. Then make the buttons active by making a private inner class, then register the buttons with the Llistener



Click on a button. When the button is clicked an image of marge, homer or bart appears with the appropriate quote.

When the program begins homer is displayed.

Creates a frame with three buttons in the NORTH section of the frame. The buttons should be labeled with the names: Bart, Homer, and Marge  
Use a JPanel for the buttons

Place a label in the CENTER section of the frame with an initial (Homer) picture on it .

In the SOUTH section of the frame place a label with a quotation from Homer

When you click a button the picture and the quote should change.  
Use

- `label.setText(...)` and
- `label.setIcon(new ImageIcon("Something.jpg"))`

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class Faces extends JFrame
{
    JButton homerButton;
    JButton margeButton;
    JButton bartButton;
    JLabel pictureLabel;
    JLabel quoteLabel;

    public Faces() //default constructor sets up GUI
    {
        super("Faces"); // call constructor of JFrame
        setBounds(50,50, 410,300);
        setBackground(Color.WHITE);

        // make the buttons and put on a panel
        JPanel topPanel = new JPanel();
        homerButton = new JButton("Homer");
        margeButton = new JButton("Marge");
        bartButton = new JButton("Bart");

        topPanel.add(homerButton);
        topPanel.add(margeButton);
        topPanel.add(bartButton);

        add(topPanel, BorderLayout.NORTH);

        //make two labels add to center and south
        pictureLabel = new JLabel (new ImageIcon("hs.jpg")); // initial picture
        quoteLabel = new JLabel("They have the internet on computers now?",SwingConstants.CENTER);

        add(pictureLabel, BorderLayout.CENTER);
        add(quoteLabel, BorderLayout.SOUTH);

        // register buttons with handlers/listeners
        homerButton.addActionListener(new ButtonHandler());
        margeButton.addActionListener(new ButtonHandler());
        bartButton.addActionListener(new ButtonHandler());

        setVisible(true);
    }

    // here is the listener class, notice implements ActionListener, an interface
    private class ButtonHandler implements ActionListener
    {
        public void actionPerformed(ActionEvent e)
        {
            if (e.getSource() == homerButton)
            {
                pictureLabel.setIcon(new ImageIcon("hs.jpg"));
                quoteLabel.setText("They have the internet on computers now?");
            }
        }
    }
}

```

```

if (e.getSource() == margeButton)
{
    pictureLabel.setIcon(new ImageIcon("ms.jpg"));
    quoteLabel.setText("Go out on a Tuesday? Who am I, Charlie Sheen?");
}

if (e.getSource() == bartButton)
{
    pictureLabel.setIcon(new ImageIcon("bs.jpg"));
    quoteLabel.setText("Cowabunga!");
}
}

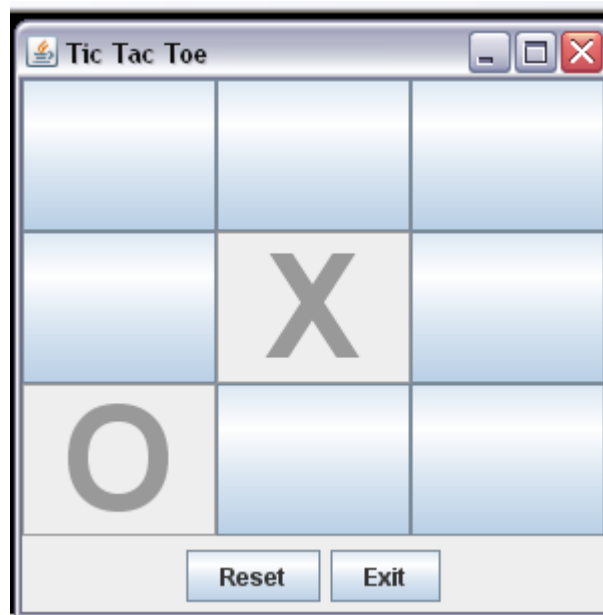
public static void main(String [] args)
{
    JFrame faces = new Faces();
}
}

```

Here is another more complicated example. It uses an **array of nine buttons**

#### **Example – An Interactive Tic-Tac-Toe Board**

Initially the board displays 9 buttons with no text and when a player clicks one of the buttons, the text is set to X or O and the button is disabled (setEnabled(false))



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

```
public class TicTacToeBoard extends JFrame
{
```

```
    private JButton resetButton;           // clear board
    private JButton exitButton ;           // ends game
    private JButton[] board;               // an array of the 9 buttons
    private int turn;                       // 1 for "X" and 0 for "O"
```

```
    public TicTacToeBoard()                // constructor builds the GUI
    {
```

```
        turn = 1;                          // 1 for "X", 0 for "O" -- X goes first
        setTitle("Tic Tac Toe");
        setBounds(0,0,300,300);
        resetButton = new JButton("Reset");
        exitButton = new JButton("Exit");
```

```
        // add exit and reset buttons to a panel and
        //add the panel to the bottom of the frame
        JPanel bottomPanel = new JPanel();
        bottomPanel.add(resetButton);
```

```

bottomPanel.add(exitButton);
add(bottomPanel, BorderLayout.SOUTH);

// instantiate a Panel for the board of 9 buttons
//use the GridLayout layout manager (3 by 3) for the board
JPanel boardPanel = new JPanel();
boardPanel.setLayout(new GridLayout(3,3));

board = new JButton[9];    // this is the array that can hold 9 JButtons

// make each button, set the font and add to the panel—3 lines
for(int i = 0; i < 9; i++)
{
    board[i] = new JButton(); // create the buttons with no text
    board[i].setFont(new Font("Arial", Font.BOLD, 72));
    boardPanel.add(board[i]); // add each to the panel (it uses grid(3 x 3))
}

// add the board panel with the 9 buttons to the frame
add(boardPanel, BorderLayout.CENTER);

// register listener with buttons – the listener class is called ButtonListener()
resetButton.addActionListener(new ButtonListener());
exitButton.addActionListener(new ButtonListener());
for(int i = 0; i < 9; i++) // register all nine center buttons
    board[i].addActionListener(new ButtonListener());

setVisible(true);
}

//Here is the Listener
private class ButtonListener implements ActionListener // must implement this interface
{
    public void actionPerformed(ActionEvent e) // ActionListener Interface method
    {
        if (e.getSource() == resetButton)
            // Reset button? Remove text from all 9 buttons and enable them
            for(int i = 0; i < 9; i++)
            {
                // remove X's and O's from all buttons
                board[i].setText(""); // set the text to the empty string

                // now enable all board buttons (so you can play again)
                board[i].setEnabled(true);
                turn = 1; // set the turn back to X's turn
            }

        else if (e.getSource() == exitButton)
            System.exit(0);

        else // a button was clicked..determine which by going through the array
            for (int i = 0; i < 9; i++) // check each button to find the clicked one
                if (e.getSource() == board[i]) // is it board[i] button???
                    {

```

```

        if (turn == 1)                // if it was X's turn
            board[i].setText("X");    //put an "X" on the board
        else                          // O's turn
            board[i].setText("O");    //put an "O" on the board
            board[i].setEnabled(false); // disable or "gray-out" the button

// now switch the turn
    if (turn == 1)
        turn ==0;
    else
        turn ==1;
    return; // the source button was determined; return
}
}

public static void main(String args[])
{
    TicTacToeBoard frame = new TicTacToeBoard();
}

```

