Chapter 17

Graphical User Interfaces and Applets

GUI versus command line interface

- A GUI requires less keyboard input. For example, with a GUI, a user can input a file name by selecting it from a menu with a single mouse click. In contrast, with terminal I/O, the user has to type in the file name.

- A GUI can easily inform the users of all the choices that are available using the various components on the window.   For example, by displaying four buttons, a GUI communicates to the user that four choices are available.

- By giving users more choices during the execution of a program, it gives users more control a program.

- A GUI can provide a standard user interface across a large variety of programs.

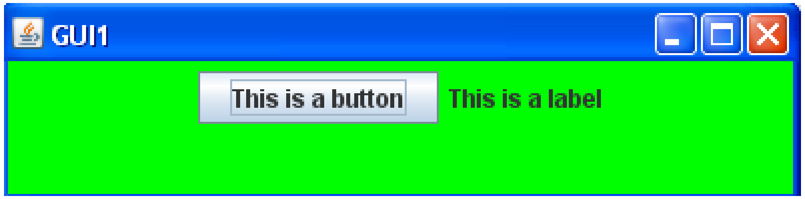
Our first GUI program

1) Create a frame object from the `JFrame` class.

2) Configure the frame. Specifically, we specify its title, size, and the action that occurs when its close button is clicked.

3) Add components to the frame's content pane.

4) Make the frame visible.


GUI1
This is a button
This is a label

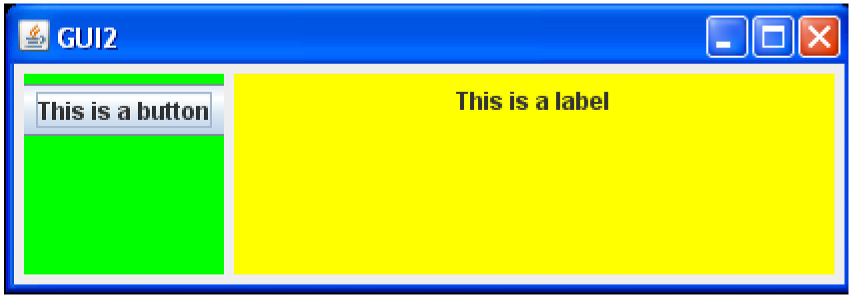
```
import javax.swing.*;
import java.awt.*;
class GUI1 extends JFrame
{
   private Container contentPane;
   private JButton button1;
   private JLabel label1;
   public GUI1()
   {
      setTitle("GUI1");
      setSize(400, 100);
      setDefaultCloseOperation(
                JFrame.EXIT_ON_CLOSE);
```

```
        // get content pane of frame
        contentPane = getContentPane();

        // configure content pane
        contentPane.setLayout(new FlowLayout());
        contentPane.setBackground(Color.GREEN);

        // add button and label to the content pane
        button1 = new JButton("This is a button");
        contentPane.add(button1);
        label1 = new JLabel("This is a label");
        contentPane.add(label1);

        setVisible(true);
    }
    public static void main(String[] args)
    {
       GUI1 window = new GUI1();
    }
}
```

Color Constants

```
Color.BLACK
Color.DARK_GRAY
Color.GRAY
Color.LIGHT_GRAY
Color.WHITE
Color.CYAN
Color.MAGENTA
Color.PINK
Color.RED
Color.ORANGE
Color.YELLOW
Color.GREEN
Color.BLUE
```

Using Panels



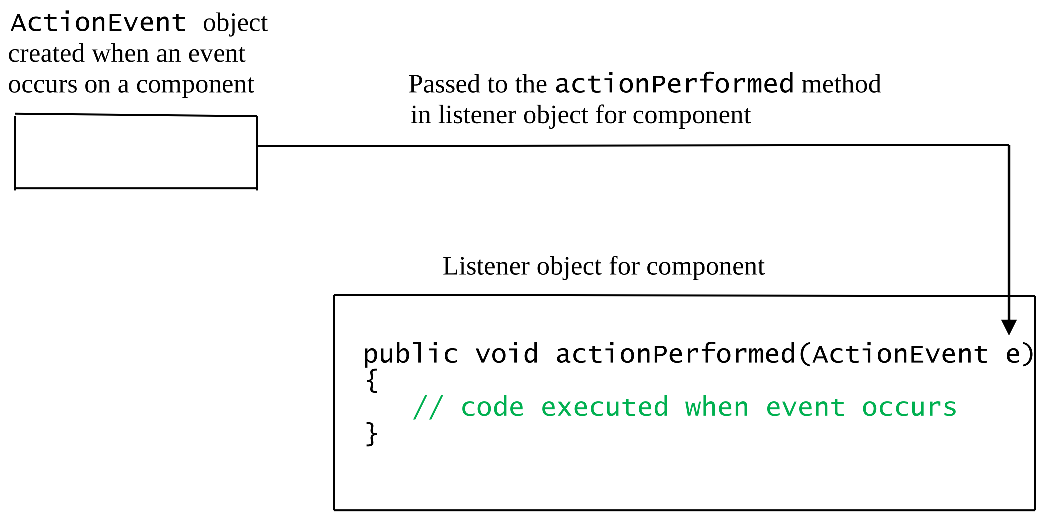
```
panel1 = new JPanel();
panel1.setLayout(new FlowLayout());
panel1.setBackground(Color.GREEN);
panel1.setPreferredSize(new Dimension(100, 100));
button1 = new JButton("This is a button");
panel1.add(button1);

panel2 = new JPanel();
panel1.setLayout(new FlowLayout());
panel2.setBackground(Color.YELLOW);
panel2.setPreferredSize(new Dimension(300, 100));
label1 = new JLabel("This is a label");
panel2.add(label1);
```

```
        // get content pane of frame
        contentPane = getContentPane();

        contentPane.setLayout(new FlowLayout());
        contentPane.add(panel1);
        contentPane.add(panel2);

        pack();  // frame size accommodates panels
        setVisible(true);
    }
    //---------------------------------
    public static void main(String[] args)
    {
        GUI2 window = new GUI2();

    }
}
```

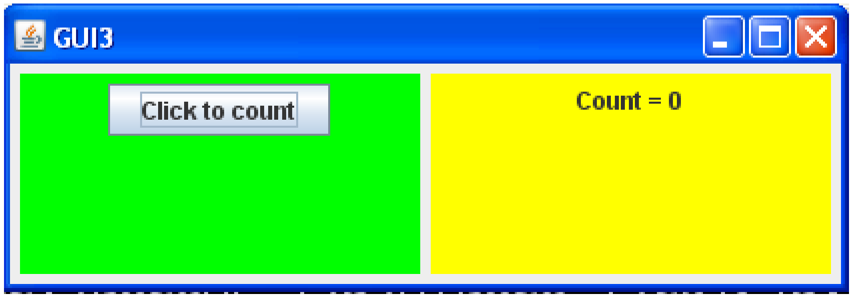
Events and action listeners

1) An `ActionEvent` object is created that represents that event.

2) If the component is associated with a listener object, the `ActionEvent` object is passed to the `actionPerformed` method in the listener object. The code in the `actionPerformed` method is then executed.


ActionEvent object created when an event occurs on a component
Passed to the actionPerformed method in listener object for component
Listener object for component

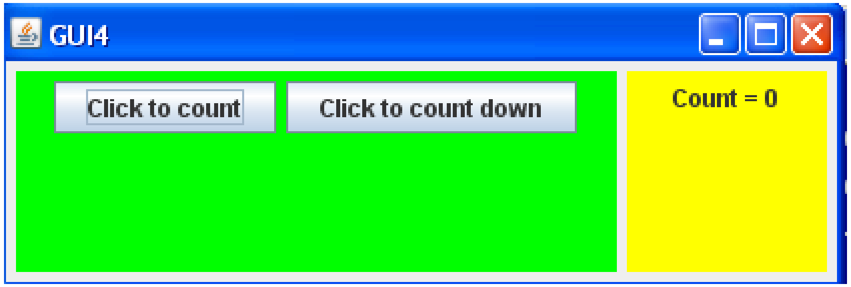
```
public void actionPerformed(ActionEvent e)
{
   // code executed when event occurs
}
```


GUI3
Click to count
Count = 0

```java
        panel1 = new JPanel();
        panel1.setLayout(new FlowLayout());
        panel1.setBackground(Color.GREEN);
        panel1.setPreferredSize(new Dimension(200, 100));
        button1 = new JButton("Click to count");
        button1.addActionListener(new Listener());
        panel1.add(button1);
```

```
   private class Listener implements ActionListener
   {
      public void actionPerformed(ActionEvent e)
      {
         label1.setText("Count = " +  ++count);
      }
   }
}
```

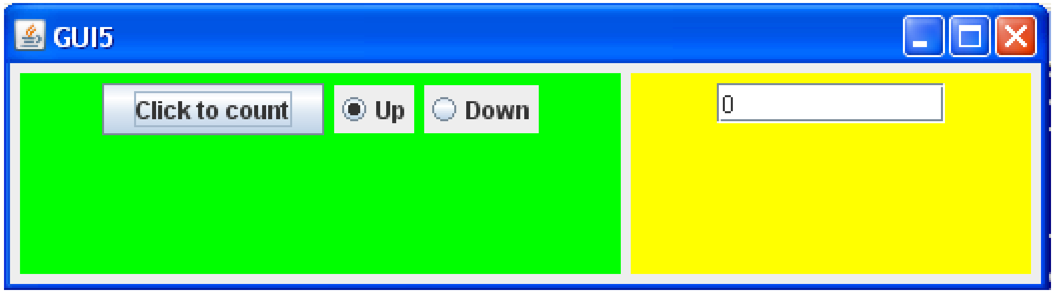
Determining which component triggers an event



```
        panel1 = new JPanel();    // create panel
        panel1.setBackground(Color.GREEN);
        panel1.setPreferredSize(new Dimension(300, 100));
        listener = new Listener();
        button1 = new JButton("Click to count");
        button1.addActionListener(listener);
        panel1.add(button1);
        button2 = new JButton("Click to count down");
        button2.addActionListener(listener);
        panel1.add(button2);
```

```
private class Listener implements ActionListener
{
   public void actionPerformed(ActionEvent e)
   {
      // e.getSource() return triggering component
      if (e.getSource() == button1)
         label1.setText("Count = " +  ++count);
      else
      if (e.getSource() == button2)
         label1.setText("Count = " +  --count);
   }
}
```

Using radio buttons



```java
radio1 = new JRadioButton("Up", true);
radio2 = new JRadioButton("Down");
group = new ButtonGroup();
group.add(radio1);
group.add(radio2);
panel1.add(radio1);
panel1.add(radio2);
```

```
   private class Listener implements ActionListener
   {
      public void actionPerformed(ActionEvent e)
      {
         if (e.getSource() == text1)
            count = Integer.parseInt(text1.getText());
         else
         if (e.getSource() == button)
            if (radio1.isSelected())
               text1.setText("" + ++count);
            else
            if (radio2.isSelected())
               text1.setText("" + --count);
      }
   }
}
```

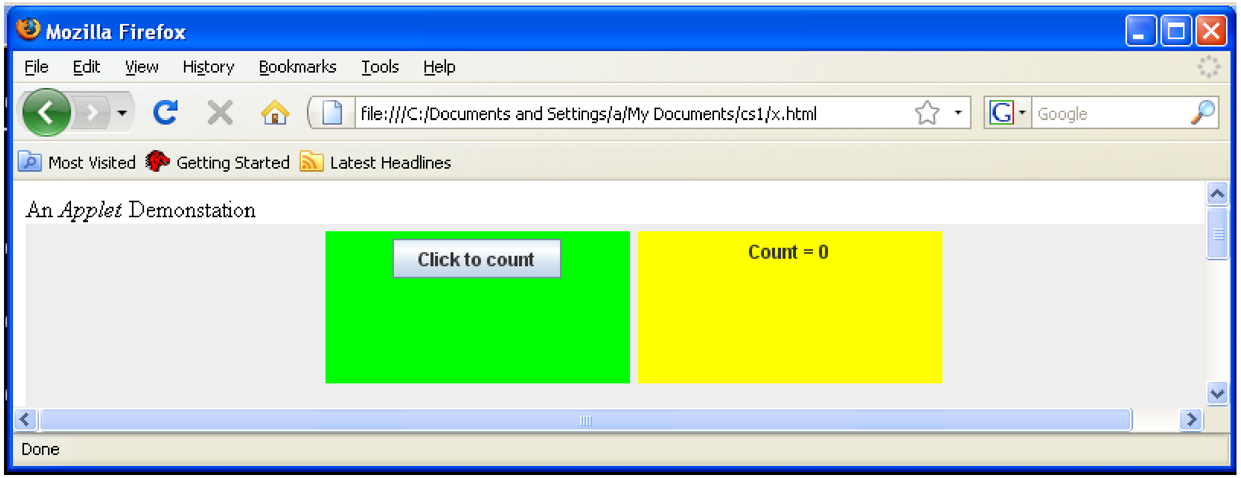
Layout managers

```
c.setLayOut(new FlowLayout(FlowLayout.LEFT));
c.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 10));

c.setLayout(new BorderLayout());
c.add(button1, BorderLayout.NORTH);
c.add(button2, BorderLayout.SOUTH);
c.add(button3, BorderLayout.EAST);
c.add(button4, BorderLayout.WEST);
c.add(button5, BorderLayout.CENTER);


c.setLayout(new GridLayout(3, 2));
c.add(button1);
c.add(button2);
c.add(button3);
c.add(button4);
c.add(button5);
c.add(button6);
```

Applet

index.html

```
An <I> Applet </I> Demonstration
<BR>
<APPLET code="Applet1.class" width=800 height=300>
</APPLET>
```


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents and Settings/a/My Documents/cs1/x.html
Google
Most Visited Getting Started Latest Headlines
Click to count
Done

An *Applet* Demonstration

Count = 0

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Applet1 extends JApplet
{
   private Container contentPane;
   private JPanel panel1, panel2;
   private JButton button1;
   private JLabel label1;
   private int count;
```

```
public void init()
{
   panel1 = new JPanel();
   panel1.setLayout(new FlowLayout());
   panel1.setBackground(Color.GREEN);
   panel1.setPreferredSize(new Dimension(300, 50));
   button1 = new JButton("Click to count");
   button1.addActionListener(new Listener());
   panel1.add(button1);

   panel2 = new JPanel();
   panel2.setLayout(new FlowLayout());
   panel2.setBackground(Color.YELLOW);
   panel2.setPreferredSize(new Dimension(300, 50));
   count = 0;
   label1 = new JLabel("Count = " + count);
   panel2.add(label1);

   // get content pane of frame
   contentPane = getContentPane();

   contentPane.setLayout(new FlowLayout());
   contentPane.add(panel1);
   contentPane.add(panel2);
}
```

```
   private class Listener implements ActionListener
   {
      public void actionPerformed(ActionEvent e)
      {
         label1.setText("Count = " +  ++count);
         System.out.println("Orange = " + Color.ORANGE);
      }
   }
}
```

1) The class for the applet must be public .

2) The class for the applet must extend `JApplet`, not `JFrame`.

3) It does not have the calls of the `setTitle`, `setSize`, `setDefaultCloseOperation`, `pack`, and `setVisible` methods. These methods come from the `JFrame` class. But the `Applet1` class in the preceding listing is not a subclass of `JFrame`. Thus, these methods are not available to the `Applet1` class. A Java application uses these methods to configure the window that it displays. However, an applet does not need these methods because it does not have its own window (an applet uses the browser's window).

4) Because the class of the applet is public, the base name of the file that contains it must match the class name. Thus, the file for the applet in the preceding listing must be `Applet1.java`.

5) The `init` method that starts on line 12 in the applet takes on the function of the constructor in a Java application. The browser initiates the execution of the applet by calling the `init` method—not a `main` method. An applet does not contain a `main` method.

appletviewer

```
appletviewer index.html
```

shows applet—not the web page

or

view `index.html` with a browser to see web page.