Week 7

Event-Driven Programming

- Event-Driven Programming and GUIs
- Buttons and Action Listeners

GUIs—Graphical User Interfaces

Most modern programs use a GUI

GUI (pronounced “gooey”):

- **G**raphical—not just text or characters: windows, menus, buttons, etc.
- **U**ser—person using the program
- **I**nterface—way to interact with the program

Typical graphical elements:

- *Window*—portion of screen that serves as a smaller screen within the screen
- *Menu*—list of alternatives offered to user
- *Button*—looks like a button that can be pressed

Event-Driven Programming

- Programs with GUIs often use *Event-Driven Programming*
- Program waits for events to occur and then responds
- Examples of events:
  - » Clicking a mouse button
  - » Dragging the mouse
  - » Pressing a key on the keyboard
- *Firing an event*—when an object generates an event
- *Listener*—object that waits for events to occur
- *Event handler*—method that responds to an event

A New Approach to Programming

Previous Style of Programming:

- List of instructions performed in order
- Next thing to happen is next thing in list
- Program performed by one agent—the computer

Event-Driven Style of Programming:

- Objects that can fire events and objects that react to events
- Next thing to happen depends on next event
- Program is interaction between user and computer

Buttons and ActionListeners

Basic steps for using a button in a Java applet:

- Create a Button object
- Add the Button object to a container
- Create an `ActionListener` object that has an `actionPerformed` method
- Register the listener for the Button object

The following slides show an example of each step.

Create a Button Object and Add the Button to a Container

```
Button stopButton = new Button("Red");
```

Button is a predefined class for buttons.

String that will appear on the button

```
add(stopButton);
```

The button will be added to the applet.

This example uses the Flow Layout so the add method needs only one parameter.

Create an **`ActionListener`** Object

Make a class into an ActionListener:

- Add the phrase **`implements ActionListener`** to the beginning of the class definition:

```
public class ButtonDemo extends Applet
    implements ActionListener
{   . . .
```

- Define a method named **`actionPerformed`**

```
  public void actionPerformed(ActionEvent e)
  {   . . .
```

The **actionPerformed** Method

- An `actionPerformed` method must have only **one** parameter
- The parameter **must** be of type `ActionEvent`

The parameter can be used to find the command for the `ActionEvent`:

```
public void actionPerformed(ActionEvent e)
{
  if (e.getActionCommand().equals("Red"))
    . . .
}
```

By default, the action command of a button will be the string displayed on the button.

Register the Listener for the Button Object

- If a button has no listener registered for it, there will be no response when the user clicks on the button.
- An example of registering a listener for a button:

```
Button stopButton = new Button("Red");
stopButton.addActionListener(this);
add(stopButton);
```

`this` refers to the object that includes this code in a method. In this example the object is an `Applet` class that implements `ActionListener`.

Interfaces

- Want `ButtonDemo` class to be both an `Applet` and an `ActionListener`
  - » can only derive from one class
  - » derived class of `Applet`
  - » implements `ActionListener` interface
- An ***interface*** is a property of a class that says what methods it must have.
- To ***implement an interface*** a class must do two things:

1. include the phrase implements *`Interface_Name`*
2. implement all the method headings in the interface definition

A class that implements the `ActionListener` interface must implement the `actionPerformed` method.

Java Tip: Code a GUI's Look and Actions Separately

For a complicated GUI, breaking up the work into two parts can help simplify the problem:

- **Code the appearance:**
  - » Use a "do nothing" `actionPerformed` method while getting the appearance right.
  - » Don't have to worry about possible mistakes in action code.
- **Code the actions:**
  - » When appearance is right, add code for actions to `actionPerformed`.
  - » Since appearance code has been tested there is less chance of mistakes in appearance code causing problems.

A temporary "do nothing" version of a method is called a ***stub***. Using stubs is a good programming technique in many situations.