Java Interfaces

- An interface is more abstract than a class
  - Almost always completely abstract in specification
  - Not an actual class to be inherited

### KeyListener (interface)

```java
public interface KeyListener
{
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
    public void keyTyped(KeyEvent e);
}
```

No constructor (not a class at all)

All methods are abstract and must all be implemented by the programmer who wants to use them.

### Coding Java Interfaces

Interface implementation is very simple

- A list of method signatures without any code (not even braces in which to place code):
  1. access (public/private, etc.)
  2. return type
  3. method name
  4. parameter type(s)

### ActionListener (interface)

```java
public interface ActionListener
{
    public void actionPerformed(ActionEvent e);
}
```

- Another example for an interface we've used many times (without knowing that's what we were doing)
  1. access (public/private, etc.)
  2. return type
  3. method name
  4. parameter type(s)
Classes to Handle Events

- **Events** in Java are things like button clicks, mouse moves, keyboard inputs.
- Classes can respond to such things by handling these events.

```java
public class TwoButtons implements ActionListener {
    private JButton left, right;
    public void actionPerformed(ActionEvent e) {
        // what to do when actions occur
    }
}
```

The class uses some JButton objects that are sources of the events (when they are clicked).

The class will wait ("listen") for events to occur, and when they do, the method will run, responding to them.

Dealing with Java Events

- **The basic Java event model involves 3 parts:**
  1. Some class of object to create events (button, keyboard, mouse, etc.)
  2. Some class of object to listen for the events
  3. Methods that handle the different events.

The Basic Process

- To respond to any event, the process is basically the same:
  1. Pick class X to implement the type of listener you want
  2. Add the required methods for handling events to X
  3. Find object of class Y that can create events of that type, and add the listener (i.e., X) to it (if X == Y, then you can use this)

  **Note:** for this last step, you need to look up what sort of addActionListener() methods are part of the class Y

  This example assumes X can create ActionEvent objects all by itself.

JButton: Creates ActionEvents

- **JButton** class is one basic Java class for creating events.

Once the events are created, an added ActionListener can respond to them.

ActionListener is the interface that a class can implement to do this.

```java
javax.swing.JButton
+ JButton( String )
+ int getX( )
+ int getY( )
+ int getWidth( )
+ int getHeight( )
+ String getLabel( )
+ void repaint()
+ void setBounds( int, int, int, int)
+ void setSize( int, int )
+ void setLocation( int, int )
+ void setLabel( String )
```
Another Source of ActionEvents: Timer

- Another Java class that handles events is Timer
- Takes input int for its delay (milliseconds)
- Also takes an ActionListener to respond to events
- Generates new event each delay ms, over and over and over...

```java
javax.swing.Timer
+ Timer(int, ActionListener)
+ boolean isRunning()
+ void start()
+ void stop()
...```

Using a Timer for Animation

- Once we have added a Timer object to a class, and then started it running, it will generate events over and over
- With the repetitive timer running, we can now make the class that contains it repeat the same method over and over
- For example, suppose we add a method that moves an object on the screen in some direction by 5 pixels
- Now, when the timer event is handled, we can call this method to move the object
- Since this will happen again and again, the object will “glide” across the screen in some direction

ActionListener Methods

```java
public interface ActionListener {
    public void actionPerformed(ActionEvent e);
}
```

- The ActionListener interface has one basic method, which takes an ActionEvent as input
  1. The event will come from some producer object (and can be produced in a number of different ways)
  2. The method runs if and only if the object that actually implements ActionListener and contains the above method is listening to the producer
- This means that we have run addActionListener() to make sure that this will happen at the right time

Using ActionEvents

- Imported from the java.awt.event.* library
- Usually, all we do is run getSource() in order to tell what object has actually produced the input event

```java
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == oneThing) {
        doSomething();
    } else if (e.getSource() == anotherThing) {
        doSomethingElse();
    }
}
```

Note: if there is only one possible source being listened to, we can just run whatever code we want to happen every time
This Week & Next

- Meetings this week:
  - Monday/Wednesday: regular classroom
  - Tuesday/Friday: in the CS Lab (16 Wing)

- Reading 07: Ch. 8 due Monday, Nov 18 at 11:59 PM
  - Some participation exercises this time

- Program 07: due Thursday November 21 at 11:59 PM

- Office Hours: Wing 212
  - Monday/Friday: 2:15 PM–3:15 PM
  - Tuesday: 1:30 PM–2:30 PM
  - Wednesday: 12:05 PM–1:00 PM

- Next Tuesday is Thursday schedule so this class doesn’t meet