Listening for Different Events

- Depending upon what we want to deal with, we can implement a range of event-based interfaces to react to a different events

<table>
<thead>
<tr>
<th>implements</th>
<th>Methods</th>
<th>Listener</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionListener</td>
<td>actionPerformed(ActionEvent e)</td>
<td>addActionListener(…)</td>
</tr>
<tr>
<td>MouseListener</td>
<td>mouseClicked(MouseEvent e)</td>
<td>addMouseListener(…)</td>
</tr>
<tr>
<td></td>
<td>mouseExited(MouseEvent e)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mousePressed(MouseEvent e)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mouseReleased(MouseEvent e)</td>
<td></td>
</tr>
<tr>
<td>KeyListener</td>
<td>keyPressed(KeyEvent e)</td>
<td>addKeyListener(…)</td>
</tr>
<tr>
<td></td>
<td>keyReleased(KeyEvent e)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>keyTyped(KeyEvent e)</td>
<td></td>
</tr>
</tbody>
</table>

Where Events Come From

- Different sorts of events come from different places
  - The following is just a partial list

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Classes that Produce the Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionEvent</td>
<td>JTextField, Timer, JFileChooser, JComponent, JButton</td>
</tr>
<tr>
<td></td>
<td>(All of these are from javax.swing.*)</td>
</tr>
<tr>
<td>MouseEvent &amp; KeyEvent</td>
<td>java.awt.Component</td>
</tr>
<tr>
<td></td>
<td>(Includes any classes that are built from it, including JComponent, JLabel, JFrame,...)</td>
</tr>
</tbody>
</table>

Where Events Go (Listeners)

- If a class can produce an event of a certain type, then we can add the same type of listener to it
KeyListener Methods

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

- The `KeyListener` interface has three methods in it:
  1. `keyPressed()` runs as soon as the user presses a key
  2. `keyReleased()` runs as soon as the user lets a key go
  3. `keyTyped()` runs *in between* the two others, but *only if* doing so would generate a valid Unicode printable character

Pressing and releasing the 'A' key on a keyboard will cause all three methods to run (`Pressed ⇒ Typed ⇒ Released`).
Pressing and releasing something like an arrow-key button only causes the first two methods to run (`Pressed ⇒ Released`), since the arrow-key is not a printable character.

Using KeyEvents

```
java.awt.event.KeyEvent
{query: + char getKeyChar() + int getKeyCode()}
```

- Imported from the `java.awt.event` library
- The two `get...()` methods allow us to find out what key(s) were pressed/typed/released during our event

```
public void keyTyped(KeyEvent k) {
    if (k.getKeyChar() == 'a')
        doSomething();
    else if (k.getKeyChar() == 'A')
        doSomethingElse();
}
```

**Note:** for a `keyTyped()` event, we get the character (if there is one) that will be printed out when typing. Even if the event was caused by multiple keys being pressed (like Shift+A), there will be at most one character returned by `getKeyChar()`.

One Last Complication: Focus

- When writing interactive programs that react in various ways to user input and other events, things can get quite complex, and there are some pitfalls to watch out for
- One big one is **program focus**: when we open different windows, and have different graphical elements all interacting, various parts of the program “take control”
- For example, if we have more than one window open, then events happening in a background window may be ignored, and only those happening in the foreground window are processed.

```
java.awt.event.KeyEvent
{query: + char getKeyChar() + int getKeyCode()}
```

Since these are all static `int` variables, each can be accessed from the Class name directly (like the way `Color.black` works)
E.g., we can check whether some `KeyEvent k` was caused by the Space-key using:
```
if (k.getKeyCode() == KeyEvent.VK_SPACE)
```
One Last Complication: Focus

A common issue is the interactions between buttons and keyboard.

On many OS platforms, the keyboard can be used to control on-screen buttons (e.g., hitting “Return” rather than pressing “Enter” button).

Sometimes, we will need to return the focus of the program back to the components that we want it to pay attention to.

Here, a JFrame (this) has other things like a Timer or JButton to deal with events. Whenever it handles those events, it then asks for focus to be returned to itself (the window area) so that things like keyboard actions actually control the window itself, and not the buttons.

MouseListener Methods

The MouseListener interface has five methods in it:

1. mousePressed() runs as soon as the user presses down on a mouse button (or trackpad or other cursor control device)
2. mouseReleased() runs as soon as the user lets button go
3. mouseClicked() runs on a complete press-and-release
4. mouseEntered() runs when the user moves the mouse over top of the GUI object that is listening for events
5. mouseExited() runs when the mouse leaves the area where the listener object exists on-screen

Using MouseEvents

Imported from the java.awt.event.* library

The getButton() method allows us to find out which mouse button was used to generate the event (if we care)

public void mouseClicked( MouseEvent e )
{
    if ( e.getButton() == MouseEvent.BUTTON1 ) {
        doSomething();
    }
    else if ( e.getButton() == MouseEvent.BUTTON2 ) {
        doSomethingElse();
    }
}

Note: MouseEvent.BUTTON1 & MouseEvent.BUTTON2 are pre-defined public static constants that identify the distinct buttons on a mouse (there is also a BUTTON3).
Using MouseEvents with Graphical Objects

- Any object that is built from the `java.awt.Component` class has the ability to generate KeyEvents or MouseEvents.
- This includes the Oval, Rectangle, etc. that we have used many times: each of these can execute `addKeyListener()` or `addMouseListener()` and start generating those events.

```java
import java.awt.Component;
public class Oval extends Component {

    // Code omitted.

} // Oval class
```

The use of `extends` causes the Oval class to be able to do everything that a Component can. This is called inheritance, and is the subject of upcoming classes.

20 November 2019
Software Design I (CS 120)

Using MouseEvents with Graphical Objects

- Component-based objects can have a `MouseListener` added:

```java
Oval o = new Oval(10, 10, 50, 50);
o.addMouseListener(this);
```

- We can then respond to events generated by those objects:

```java
public void mouseClicked(MouseEvent e) {
    Oval source = (Oval) e.getSource();
    source.setBackground(Color.red);
}
```

Note: since `getSource()` returns the source of the event in generic `Object` form, we must cast so that we can use specific class methods like `setBackground()`, which only certain types of graphical objects actually have.

20 November 2019
Software Design I (CS 120)

This Week & Next

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

- **Reading 08**: Ch. 9 due Tuesday, Nov 26 at 12:00 PM (Noon)
  - Only 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
- Next Wednesday is Friday schedule: we meet in lab

20 November 2019
Software Design I (CS 120)