Class #47: Interfaces and Events, II

Software Design I (CS 120): D. Mathias, 29 April 20

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>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>MouseEvent &amp; KeyEvent</td>
<td>java.awt.Component &amp; descendants</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>MouseEvent &amp; KeyEvent</td>
<td>java.awt.Component &amp; descendants</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

<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>MouseEvent &amp; KeyEvent</td>
<td>java.awt.Component &amp; descendants</td>
</tr>
</tbody>
</table>

29 April 2020

Software Design I (CS 120)
KeyListener Methods

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

```java
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().

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:
```java
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

```
public void actionPerformed(ActionEvent e) {
    if ( e.getSource() == timer ) {
        move();
    } else if ( e.getSource() == button ) {
        startOrStop();
        this.requestFocus();
    }
}
```

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

```
public interface MouseListener {
    public void mouseClicked(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
```

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 predefined public static constants that identify the distinct buttons on a mouse (there is also a `BUTTON3`).

For some situations, as in `mouseEntered()` and `mouseExited()`, we don’t care about the buttons at all, since the event isn’t caused by the button, but by the motion of the mouse pointer over some GUI element
In such cases, we can simply respond to the event directly:

```
public void mouseEntered(MouseEvent e) {
    doSomething();
}
```

Using MouseEvents
Using MouseEvents with Graphical Objects

Any object that is built from the `java.awt.Component` class has the ability to generate `KeyEvent`s or `MouseListener`s. 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.
}
```

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.

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.

---

This Week & Next

**Meetings this week:**
- Monday: Recorded Lecture
- Tuesday: Program 06 work
- Wednesday: Recorded lecture
- Friday: Closed lab

**Closed Lab 16:** Friday May 1 by 5:00 PM

**Reading 07:** Ch. 9 April 30 at 6:00 PM (Partic. Exercises)

**Program 06:** Wednesday April 29 at 11:59 PM

**Office Hours:** via the interwebs
- Monday/Tuesday/Wednesday/Friday: 9:00 AM–11:00 AM
- https://kube-0.cs.uwlax.edu:8443/ZombieApocalypseOfficeHours