Java uses a “MouseListener” model

- The user asks the virtual machine to tell it when mouse events occur
  - Mouse movements
  - Mouse button press, release, click
    - button 1,2,3
  - Mouse wheel movements
Java uses a "MouseListener"

- "Observer" design pattern
- Example:
  - [http://java.sun.com/docs/books/tutorialJWS/uiswing/events/ex6/MouseEventDemo.jnlp](http://java.sun.com/docs/books/tutorialJWS/uiswing/events/ex6/MouseEventDemo.jnlp)
The “Observer” gets callbacks from the “Subject”

Same pattern we saw in AJAX

Common pattern in event-driven User Interface software

“MouseListener” maps to the “Observer”

public class MouseEventDemo ... implements MouseListener {
    // Where initialization occurs:
    // Register for mouse events on blankArea and the panel.
    blankArea.addMouseListener(this);
    panel.addMouseListener(this);
    ...

    public void mousePressed(MouseEvent e) {
        saySomething("Mouse pressed; # of clicks: "
            + e.getClickCount(), e);
    }

    public void mouseReleased(MouseEvent e) {
        saySomething("Mouse released; # of clicks: "
            + e.getClickCount(), e);
    }

    public void mouseEntered(MouseEvent e) {
        saySomething("Mouse entered", e);
    }

    public void mouseExited(MouseEvent e) {
        saySomething("Mouse exited", e);
    }

    public void mouseClicked(MouseEvent e) {
        saySomething("Mouse clicked (# of clicks: "
            + e.getClickCount() + ")", e);
    }

    void saySomething(String eventDescription, MouseEvent e) {
        SwingUtilities.invokeLater(()
            + " detected on "
            + e.getComponent().getName() + "." + newline);
    }
}
The mouse just entered me

The mouse just exited me

public class MouseEventDemo implements MouseListener {
    // where initialization occurs:
    // Register for mouse events on blankArea and the panel.
    blankArea.addMouseListener(this);
    addMouseListener(this);
    
    public void mousePressed(MouseEvent e) {
        saySomething("Mouse pressed; # of clicks: "
            + e.getClickCount(), e);
    }
    public void mouseReleased(MouseEvent e) {
        saySomething("Mouse released; # of clicks: "
            + e.getClickCount(), e);
    }
    public void mouseEntered(MouseEvent e) {
        saySomething("Mouse entered", e);
    }
    public void mouseExited(MouseEvent e) {
        saySomething("Mouse exited", e);
    }
    public void mouseClicked(MouseEvent e) {
        saySomething("Mouse clicked (# of clicks: "
            + e.getClickCount() + ")", e);
    }
    void saySomething(String eventDescription, MouseEvent e) {
        textArea.append(eventDescription + " detected on "
            + e.getComponent().getClass().getName()
            + "." + newline);
    }
}
Mouse Event

- When your program is told that something happened, you get extra info with the event
  - Single or double click?
  - (X,Y) of event
    - global and local coordinates
  - which button was pushed (1,2,3)
  - Modifier keys
    - "Shift" click
Mouse Event (cont)

- When your program is told that something happened, you get extra info with the event.
- Which UI component is reporting
  - “blankArea”
- timestamp
- and a few more things
## Method Summary

### Methods

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>getButton() Returns which, if any, of the mouse buttons has changed state.</td>
</tr>
<tr>
<td>int</td>
<td>getClickCount() Returns the number of mouse clicks associated with this event.</td>
</tr>
<tr>
<td>Point</td>
<td>getLocationOnScreen() Returns the absolute x, y position of the event.</td>
</tr>
<tr>
<td>int</td>
<td>getModifiersEx() Returns the extended modifier mask for this event.</td>
</tr>
<tr>
<td>static String</td>
<td>getMouseModifiersText(int modifiers) Returns a String instance describing the modifier keys and mouse buttons that were down during the event, such as &quot;Shift&quot;, or &quot;Ctrl+Shift&quot;.</td>
</tr>
<tr>
<td>Point</td>
<td>getPoint() Returns the x,y position of the event relative to the source component.</td>
</tr>
<tr>
<td>int</td>
<td>getX() Returns the horizontal x position of the event relative to the source component.</td>
</tr>
<tr>
<td>int</td>
<td>getXOnScreen() Returns the absolute horizontal x position of the event.</td>
</tr>
<tr>
<td>int</td>
<td>getY() Returns the vertical y position of the event relative to the source component.</td>
</tr>
<tr>
<td>int</td>
<td>getYOnScreen() Returns the absolute vertical y position of the event.</td>
</tr>
<tr>
<td>boolean</td>
<td>isPopupTrigger() Returns whether or not this mouse event is the popup menu trigger event for the platform.</td>
</tr>
<tr>
<td>String</td>
<td>paramString() Returns a parameter string identifying this event.</td>
</tr>
<tr>
<td>void</td>
<td>translatePoint(int x, int y) Translates the event's coordinates to a new position by adding specified x (horizontal) and y (vertical) offsets.</td>
</tr>
</tbody>
</table>

http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseEvent.html
Different types of input devices

- What about trackpads?
- What about tablets?
- What about rollerballs?
Different types of input devices

- As long as the OS can translate the hardware interaction into the same events then programs are compatible.
- A stylus on a tablet can “click”
- A rollerball “enters” and “exits”
- A finger on a trackpad has an (X,Y)
Multi-touch is different

http://www.youtube.com/watch?v=EiS-W9aeG0s
Multi-touch is different

http://www.youtube.com/embed/Cog8b8ojji0?rel=0
Multi-touch creates new interactions

- This breaks old programs
- unless the OS makes the multi-touch look like a mouse to the program
Multi-touch creates new interactions

- Watch Android 3D widget video
- What is different from working with a mouse?
Multi-touch creates new interactions

- pointer is gone
  - all interaction is active
- hover is gone
- you can’t see what you are clicking
- “clicking” isn’t [as] natural
- “swiping” is natural
Multi-touch creates new interactions

- Software has to be (re)written to be
  - “multi-touch” aware
- The OS can give some support
  - exposing new events like
    - “pinch” (tell me when a pinch occurs)
    - “rotate” (tell me when a rotate occurs)
    - “two finger swipe”
    - “three finger swipe”
Multi-touch creates new interactions

- But multi-touch is really computer vision

Where is the mouse clicking?

What abstractions will the OS expose?
Multi-touch creates new interactions

- Watch 10/GUI video
- http://10gui.com/video/
Multi-touch terminology

- **Multi-touch** – An interactive technique that allows single or multiple users to control graphical displays with more than one simultaneous finger.

- **Multi-point** – An interactive technique that makes use of points of contact rather than movement. A multi-point kiosk with buttons would be an example.

- **Multi-user** – A multi-touch device that accepts more than one user. Larger multi-touch devices are said to be inherently multi-user.

- **Multi-modal** – A form of interaction using multiple modes of interfacing with a system.
Multi-touch terminology

- **Tabletop Computing** – Interactive computer displays that take place in the form of tabletops.
- **Direct Manipulation** – The ability to use the body itself (hands, fingers, etc) to directly manage digital workspaces.
- **Blob tracking** - Assigning each blob an ID (identifier). Each frame we try to determine which blob is which by comparing each with the previous frame.
- **Blob detection** - Process of picking out bright areas of a camera image and somehow relaying them to a computer as a touch.
Multi-Touch Approach #1 - rebuild the Observer Pattern

- Design specific multi-touch/gesture events that you can register for:
  - Pinching movements (in or out)
    - meaning zoom out or zoom in
  - Rotate: Two fingers moving in opposite semicircles is a gesture meaning rotate.
  - Swipe: Three fingers brushing across the trackpad surface in a common direction.
  - Scroll: Two fingers moving vertically or horizontally is a scroll gesture.
Multi-Touch Approach #1 - rebuild the Observer Pattern

- **Advantages:**
  - Simple to code
  - Library/OS does all the work

- **Disadvantages**
  - No flexibility
  - Limited to supported events
Multi-Touch Approach #1 - rebuild the Observer Pattern

- Examples (demo):
  - Document browsing in Preview
    - Zoom
    - Scale
    - Swipe