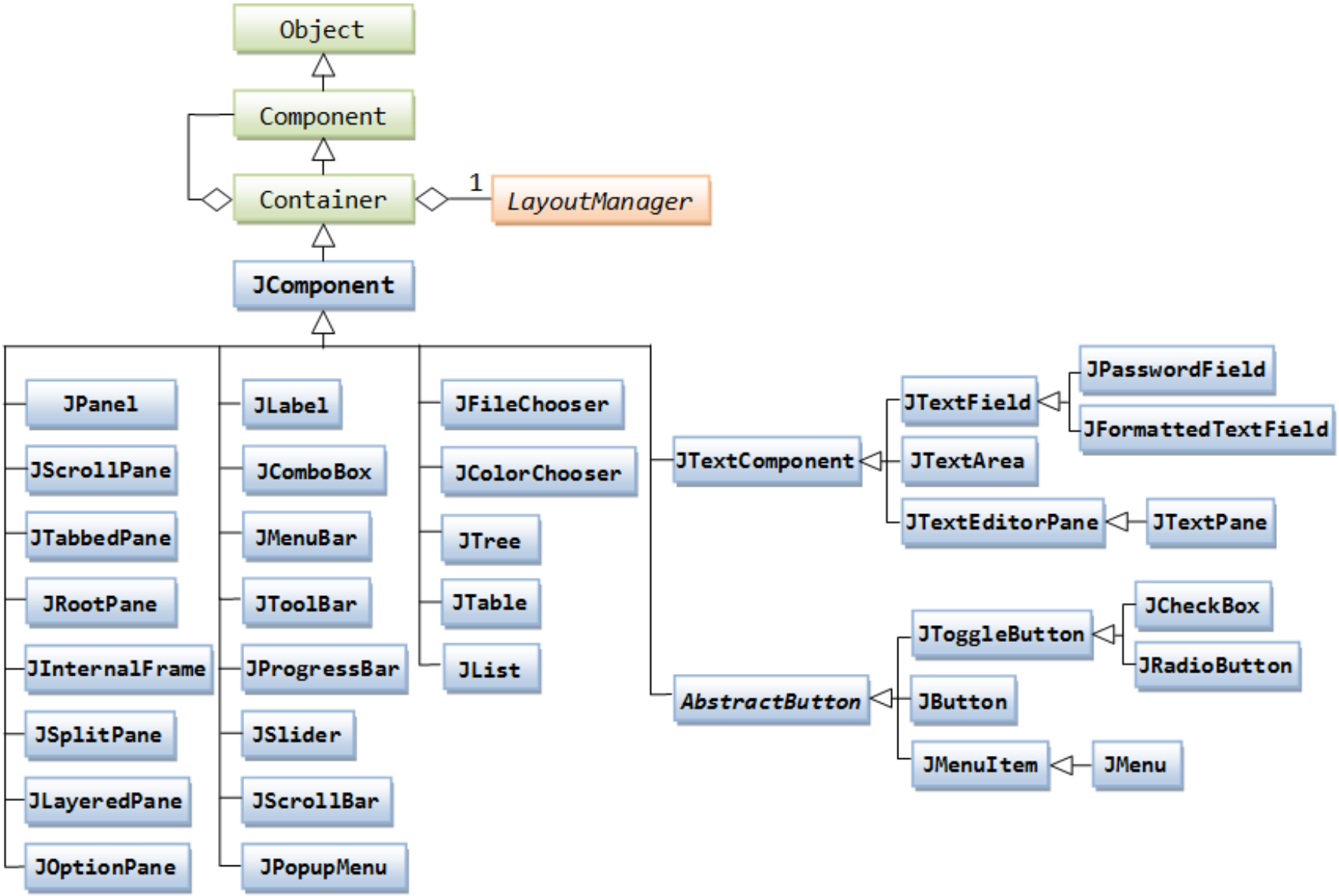
EECS 1720
Building Interactive Systems

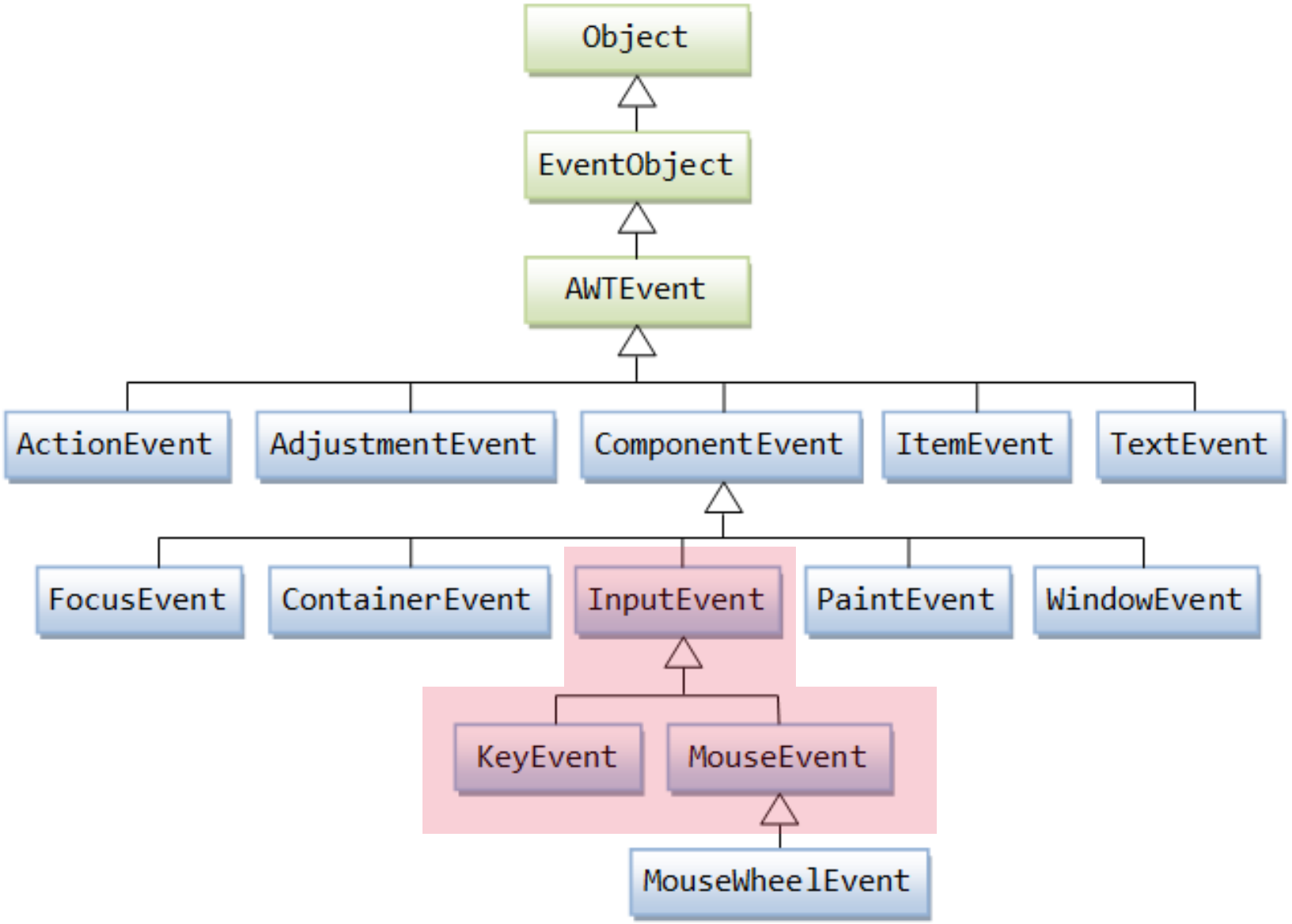
Lecture 17 :: Event Handling [3] →
MouseEvents & KeyEvents

YORK UNIVERSITÉ UNIVERSITY

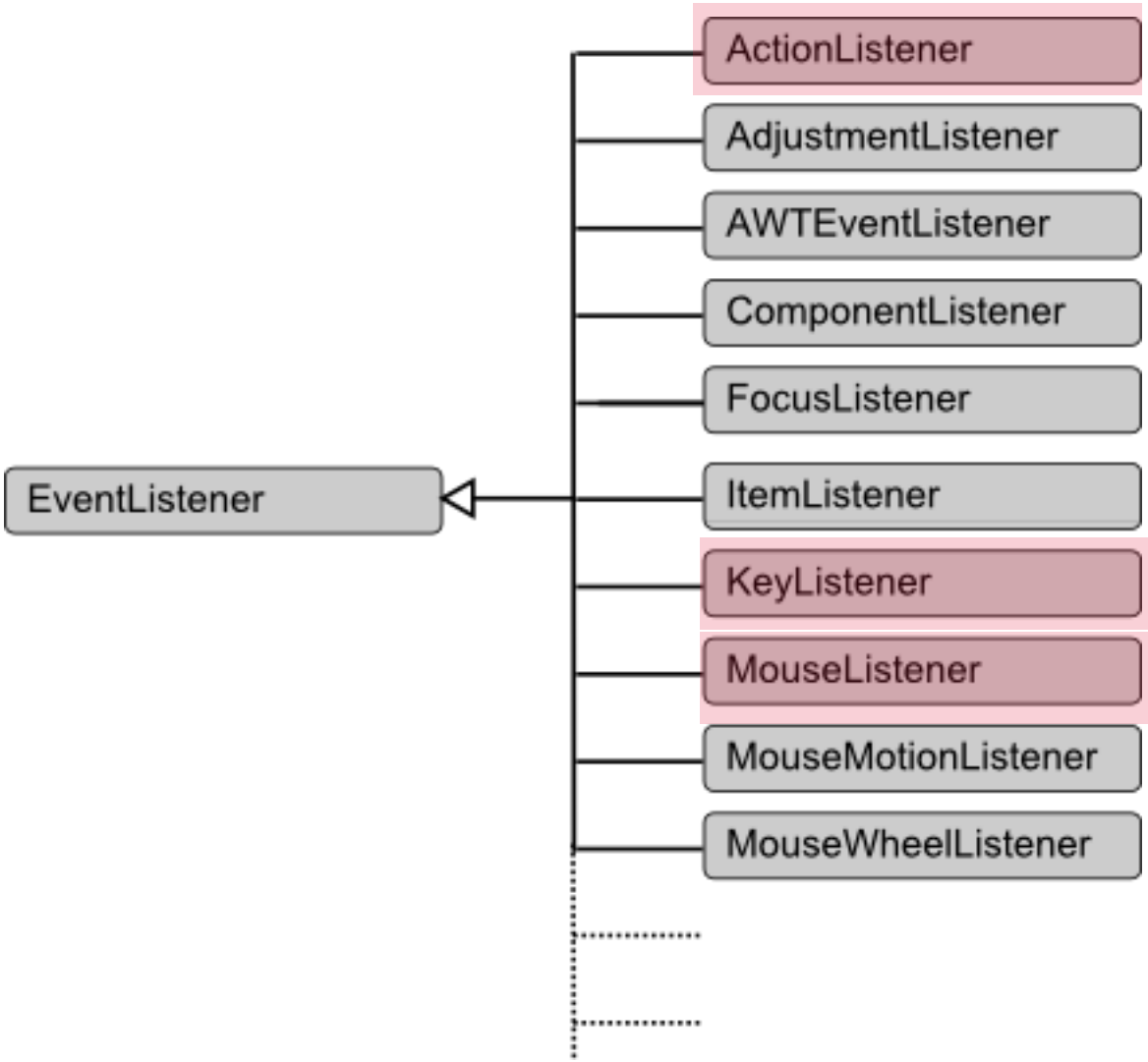
GUI components/containers



Events


Object
EventObject
AWTEvent
ActionEvent
AdjustmentEvent
ComponentEvent
ItemEvent
TextEvent
FocusEvent
ContainerEvent
InputEvent
PaintEvent
WindowEvent
KeyEvent
MouseEvent
MouseWheelEvent

Event Listeners (interfaces)



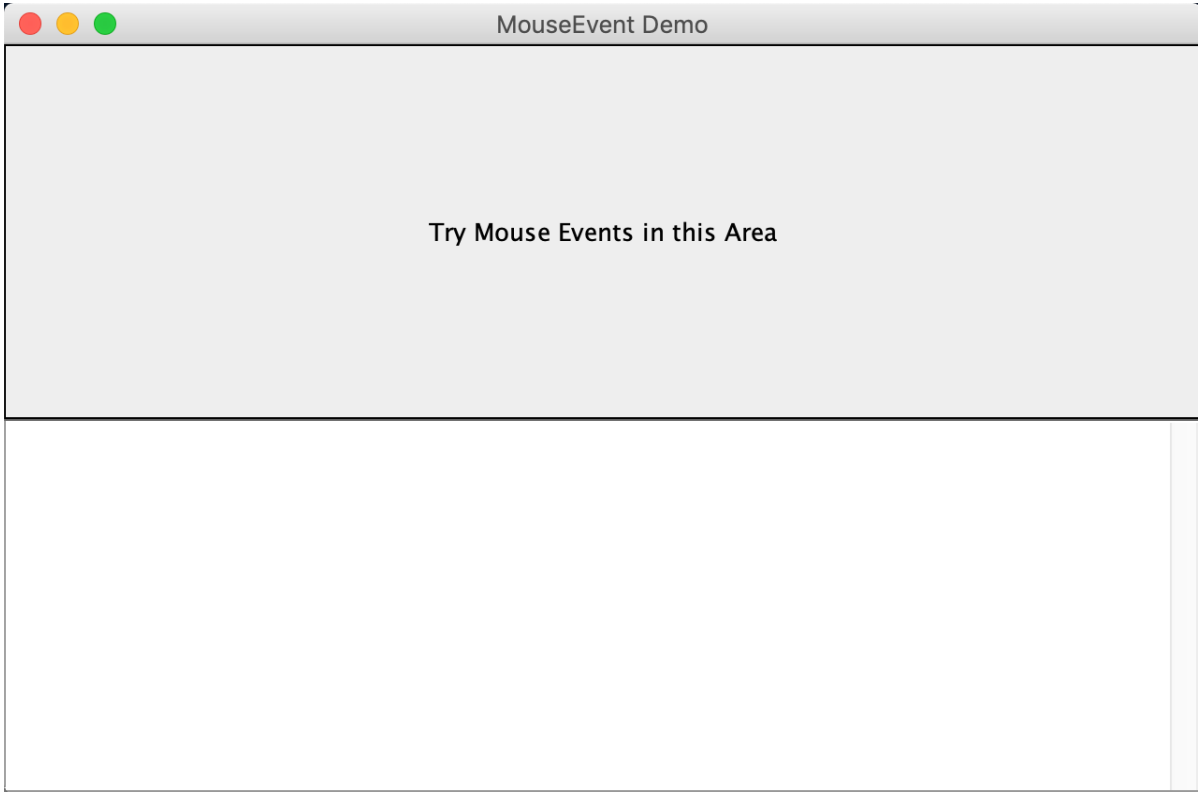
Event Listeners

User Action	Event Triggered	Event Listener interface
Click a `Button, JButton`	`ActionEvent`	`ActionListener`
Open, iconify, close `Frame, JFrame`	`WindowEvent`	`WindowListener`
Click a `Component, JComponent`	**`MouseEvent`**	**`MouseListener`**
Change texts in a `TextField, JTextField`	`TextEvent`	`TextListener`
Type a key	**`KeyEvent`**	**`KeyListener`**
Click/Select an item in a `Choice, JCheckbox, JRadioButton, JComboBox`	`ItemEvent, ActionEvent`	`ItemListener, ActionListener`

MouseEvents & MouseListener

Example 1 (MouseEventDemo)

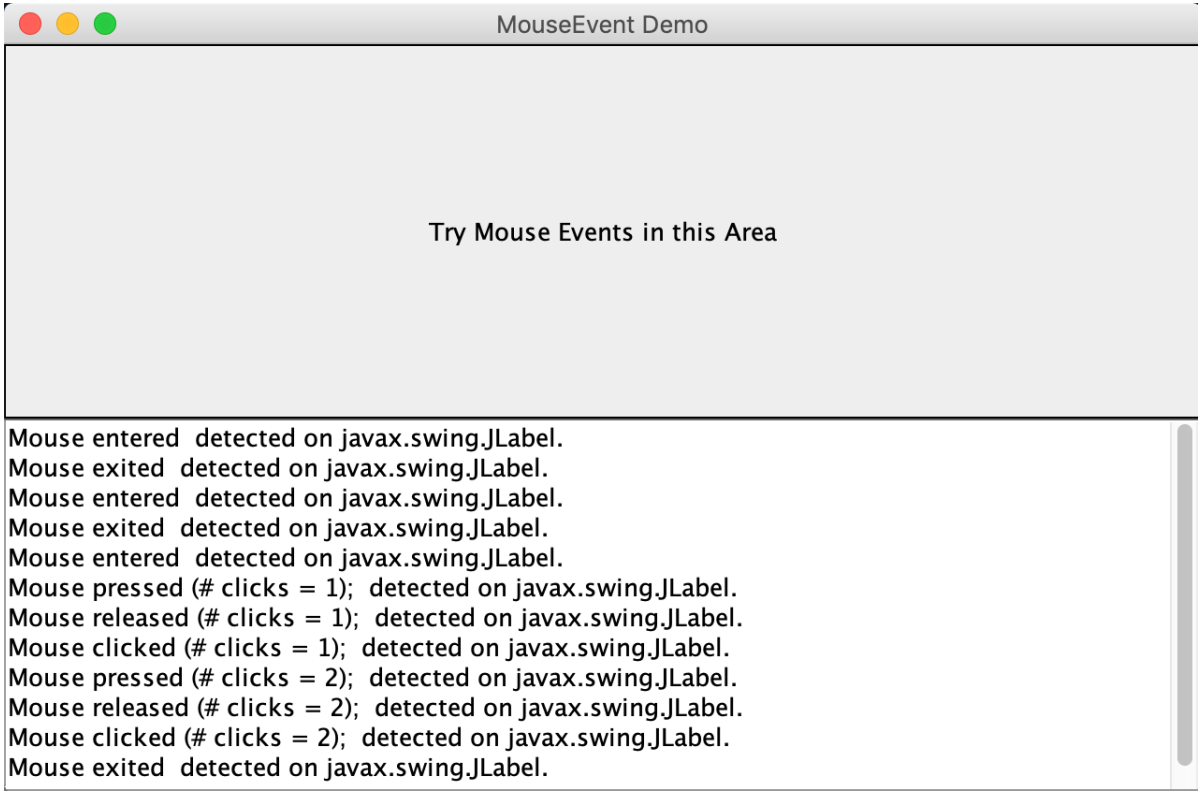
- 2 Areas:
  - one registering mouse events, the other displaying them:



YORK UNIVERSITÉ UNIVERSITY

Example 1 (MouseEventDemo)

- 2 Areas:
  - one registering mouse events, the other displaying them:



```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.*;
import javax.swing.border.Border;


public class MouseEventDemo extends JFrame implements MouseListener {

     private JLabel mouseArea;
     private JTextArea messageArea;

     public MouseEventDemo(String title) {        // ctor (next page)           }

     public static void main(String[] args) {
          MouseEventDemo1 frame = new MouseEventDemo1("MouseEvent Demo");

     }

     // ALL MouseListener methods (to be implemented)

}
```

MouseEvents

- Specifically need to implement MouseListener interface
- MouseListener API:

Method	Purpose
mouseClicked(MouseEvent)	Called just after the user clicks the listened-to component.
mouseEntered(MouseEvent)	Called just after the cursor enters the bounds of the listened-to component.
mouseExited(MouseEvent)	Called just after the cursor exits the bounds of the listened-to component.
mousePressed(MouseEvent)	Called just after the user presses a mouse button while the cursor is over the listened-to component.
mouseReleased(MouseEvent)	Called just after the user releases a mouse button after a mouse press over the listened-to component.

MouseListener Interface

```
public interface MouseListener
extends EventListener
```

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
void	**mouseClicked**(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

```
public MouseEventDemo(String title) {

      super(title);
      Container pane = this.getContentPane();

      // create a mouse interaction component (using a JLabel here)
      this.mouseArea = new JLabel("Try Mouse Events in this Area", JLabel.CENTER);
      this.mouseArea.setPreferredSize(new Dimension(640,200));
      this.mouseArea.setBackground(Color.PINK);
      this.mouseArea.setBorder(BorderFactory.createLineBorder(Color.BLACK));

      // create a message component (scrollable JTextArea)
      // to show mouse events
      this.messageArea = new JTextArea();
      this.messageArea.setEditable(false);


      JScrollPane scrollPane = new JScrollPane(this.messageArea);
      scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
      scrollPane.setPreferredSize(new Dimension(640, 200));


      // register this class as the mouseListener for mouseArea
      this.mouseArea.addMouseListener(this);

      pane.add(this.mouseArea, BorderLayout.NORTH);
      pane.add(scrollPane, BorderLayout.CENTER);

      // normal frame setup (not shown – see sample code)
}
```

```
// MouseListener Event handlers

@Override
public void mouseClicked(MouseEvent e) {

          this.messageArea.append("Mouse clicked (# clicks = " +
                    e.getClickCount() + "); " + " detected on " +
                    e.getComponent().getClass().getName() + ".\n");
}


@Override
public void mousePressed(MouseEvent e) {
          this.messageArea.append("Mouse pressed (# clicks = " +
                    e.getClickCount() + "); " + " detected on " +
                    e.getComponent().getClass().getName() + ".\n");


}

@Override
public void mouseReleased(MouseEvent e) {
          this.messageArea.append("Mouse released (# clicks = " +
                    e.getClickCount() + "); " + " detected on " +
                    e.getComponent().getClass().getName() + ".\n");
}
```

```
@Override
public void mouseEntered(MouseEvent e) {
          this.messageArea.append("Mouse entered detected on " +
          e.getComponent().getClass().getName() + ".\n");
}

@Override
public void mouseExited(MouseEvent e) {
          this.messageArea.append("Mouse exited detected on " +
          e.getComponent().getClass().getName() + ".\n");
}
```

If you don’t want to use one of these event handlers…
Then leave it intact (but empty)!!

i.e. { } (it needs to be implemented, but does not necessarily need to do anything)

Other information in MouseEvent ?

- **MouseEvent** holds specialized mouse info
  - click counts
  - x,y location of mouse at event
  - which button on mouse changed state (if any)
  - any modifier keys active during event
    (e.g. Shift, or CTRL+Shift)

- **MouseEvent** also inherits info from **InputEvent**
  - getID (can check against specific mouse events)
  - getComponent (gets component that fired event)
  - getWhen (gets timestamp of event)

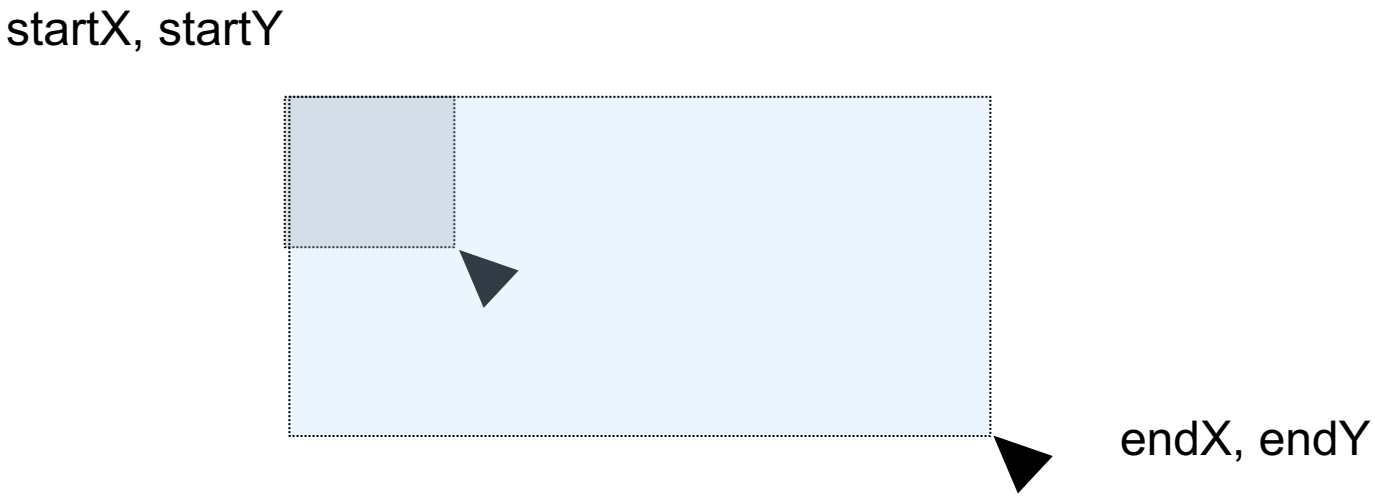
MouseEventDemo2:

```
@Override
public void mousePressed(MouseEvent e) {

     this.messageArea.append("Mouse pressed (# clicks = " + e.getClickCount() + "); "
          + " detected on " + e.getComponent().getClass().getName()
          + " pressed button: " + e.getButton()


     // GET COORDS OF MOUSE PRESS
          + " user: (" + e.getX() + ", " + e.getY() + "); "

          + " screen: (" +    e.getLocationOnScreen().getX() + ", " +
                              e.getLocationOnScreen().getY() + ") .\n");


}
```

YORK UNIVERSITÉ UNIVERSITY

MouseEventDragDetect

- Save x,y position on mouse press
- Save x,y position on mouse release (and output these)



```java
// add some class fields

private double startX=0.0;
private double startY=0.0;
private double endX=0.0;
private double endY=0.0;



@Override
public void mousePressed(MouseEvent e) {
     // ...

     // save start position
     this.startX = e.getX();
     this.startY = e.getY();

}


@Override
public void mouseReleased(MouseEvent e) {
     // ...

     // save end position and output both to console
     this.endX = e.getX();
     this.endY = e.getY();

     System.out.println("start: (" + this.startX + ", " + this.startY + ")" );
     System.out.println("end: (" + this.endX + ", " + this.endY + ")" );
}
```

- More on Mouse Listeners next lecture
- Specifically (MouseMotionListener) + Applications
  - E.g. selecting areas, drawing with mouse, etc

KeyEvents & KeyListener

Focus

- Unlike the Mouse… a key is not immediately connected to an object/position in your GUI!!
  - Generally, when typing, we need to select a window (even a control) in which to type
  - Selecting a window shifts the OS to "**focus**" on that window (to then expect keyboard input)
  - Sometimes focus can shift automatically (based on the cursor/mouse being "over" the window

- In our applications, we can explicitly select components in our GUI that can become the focus for keyboard input
  - This essentially "directs" key events to the listener(s) registered with those objects

KeyEvents

- For a key press to affect a component, the component must have the keyboard ***focus***
- Only one component at a time in the window system can have the keyboard focus

- To check/set focus:
  - isFocusable() should return true if in focus
  - setFocusable(true) will enable keyboard focus on a component

- We can also invoke a method called **`requestFocusInWindow()`** on an individual component for it to gain keyboard focus

KeyEvents

- 2 types of key events
  - Key pressed/released events (pressing a key on keyboard)
  - Typing a Unicode character (not a dedicated key on the keyboard)
  - Keys are also able to be compounded (i.e. multiple keys can be pressed together)
    - Each new key is registered separately, however with each key pressed, a list of the other keys currently being pressed is also available)
    - These other keys are considered “modifiers”

```
public interface KeyListener
extends EventListener
```

The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event either implements this interface (and all the methods it contains) or extends the abstract `KeyAdapter` class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's `addKeyListener` method. A keyboard event is generated when a key is pressed, released, or typed. The relevant method in the listener object is then invoked, and the `KeyEvent` is passed to it.

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
void	**keyPressed**(**KeyEvent** e) Invoked when a key has been pressed.
void	**keyReleased**(**KeyEvent** e) Invoked when a key has been released.
void	**keyTyped**(**KeyEvent** e) Invoked when a key has been typed.

KeyEvents & KeyCodes

- KeyEvent class has many constants (relating to the different kinds of keys that could be pressed)
  - VK_* relates to a particular key code

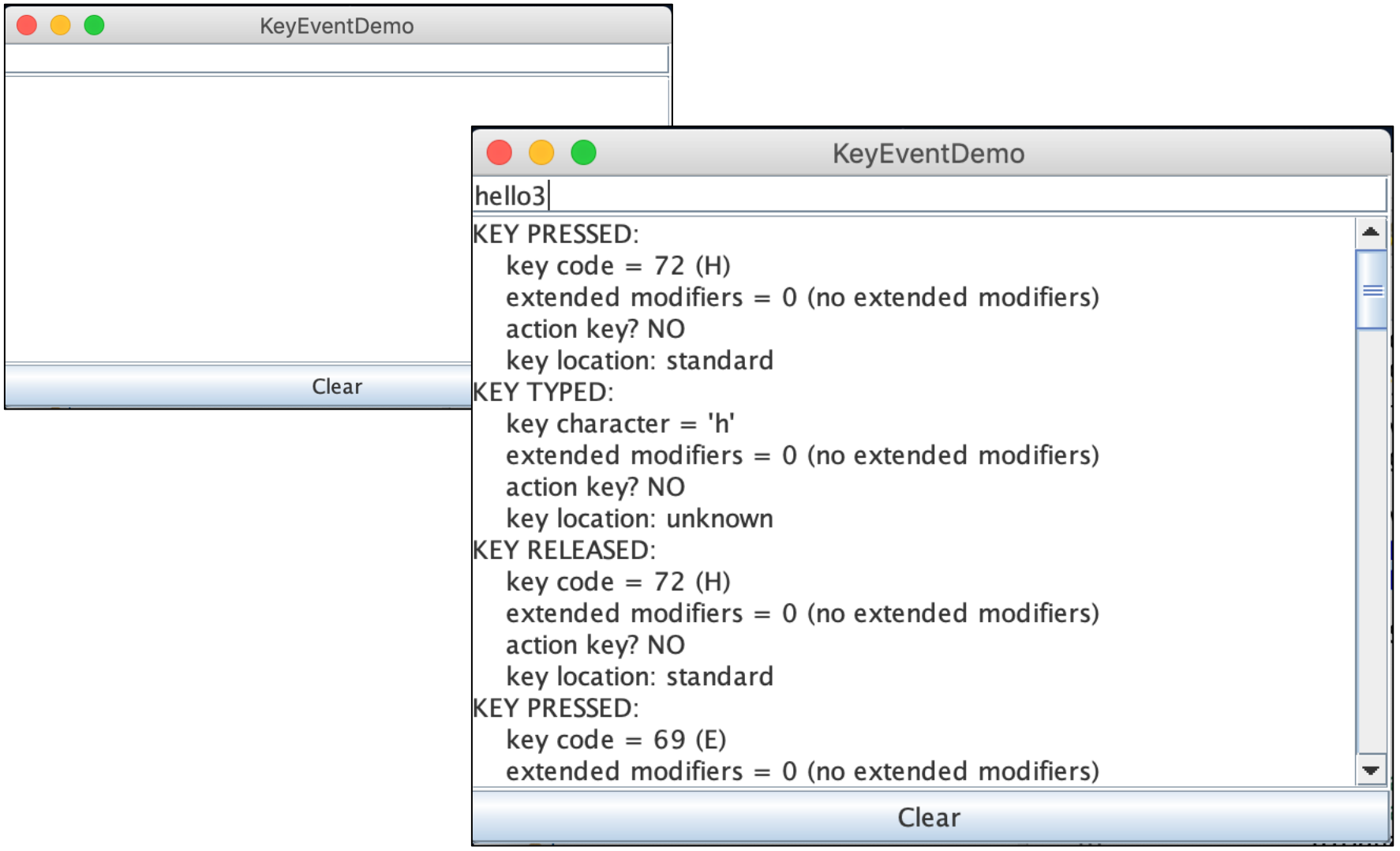
- E.g.
  - `KeyEvent.VK_A → KeyEvent.VK_K (keys 'A' to 'Z')`
  - `KeyEvent.VK_`
  - `KeyEvent.VK_COMMA`
  - `KeyEvent.VK_COLON`

  … plus many MANY more!

Pressing and releasing a Key :

- E.g.
  - pressing (and holding) the Shift key:
    - causes a **KEY_PRESSED** event with a **VK_SHIFT** keycode
  - pressing the 'a' key:
    - causes a **KEY_PRESSED** event with a **VK_A** keycode
  - Separately:
    - a **KEY_TYPED** event with a keyChar value of **'A'** is generated.
  - After the 'a' key is released:
    - a **KEY_RELEASED** event will be fired with VK_A.
  - After the SHIFT key is released:
    - a **KEY_RELEASED** event will be fired with VK_SHIFT.

Example: KeyEventDemo



```
public class KeyEventDemo extends JFrame implements KeyListener, ActionListener {

      JTextField typingArea; // where the key events will be captured (must have focus)
      JTextArea displayArea; // where the key events will be displayed
      static final String newline = System.getProperty("line.separator");

      public KeyEventDemo(String name) {
        super(name);
        Container pane = this.getContentPane();

        JButton button = new JButton("Clear");
        button.addActionListener(this); // clear the display area via a normal action listener

        this.typingArea = new JTextField(20);

        this.typingArea.requestFocusInWindow(); // set this component to get input focus
        this.typingArea.addKeyListener(this);   // key listener

        this.displayArea = new JTextArea();
        this.displayArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(displayArea);
        scrollPane.setPreferredSize(new Dimension(375, 125));
        scrollPane.setAutoscrolls(true);

        pane.add(typingArea, BorderLayout.PAGE_START);
        pane.add(scrollPane, BorderLayout.CENTER);
        pane.add(button, BorderLayout.PAGE_END);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.pack();
        this.setVisible(true);
    }

    public static void main(String[] args) {
    KeyEventDemo frame = new KeyEventDemo("KeyEventDemo");
    }

    // …  need to implement all methods for each interface
```

```java
/** Handle the button click to clear typing and display Area */
public void actionPerformed(ActionEvent e) {

    //Clear the text components.
    displayArea.setText("");
    typingArea.setText("");

    //Return the focus to the typing area.
    typingArea.requestFocusInWindow();
}


/** Handle the key typed event from the text field. */
public void keyTyped(KeyEvent e) {
    displayInfo(e, "KEY TYPED: ");
}

/** Handle the key pressed event from the text field. */
public void keyPressed(KeyEvent e) {
    displayInfo(e, "KEY PRESSED: ");
}

/** Handle the key released event from the text field. */
public void keyReleased(KeyEvent e) {
    displayInfo(e, "KEY RELEASED: ");


    // quit application if 'Q' pressed!!

    if (e.getKeyCode() == KeyEvent.VK_Q)
    System.exit(0);
}
```

See sample code for displayInfo method

Accessing KeyEvent info

- Common Methods:
  - e.getID() → KEY_TYPED | KEY_PRESSED | KEY_RELEASED
  - e.getKeyCode()
  - e.getKeyText(keycode)

  - e.getModifiersEx(); // gets extended modifiers
  - e.getModifiersExText(modifiersEx);

  - e.isActionKey(); // is key
  - e.getKeyLocation(); // is key left/right/standard or on numpad

More Resources

- Demos in todays lecture are adapted from examples in these tutorials:
  - MouseListener
    - https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html
  - KeyListener
    - https://docs.oracle.com/javase/tutorial/uiswing/events/keylistener.html

- More information on other types of listeners available here (if you would like to read ahead about some specific control you want to make a listener for, for your assignments)
  - https://docs.oracle.com/javase/tutorial/uiswing/events/index.html