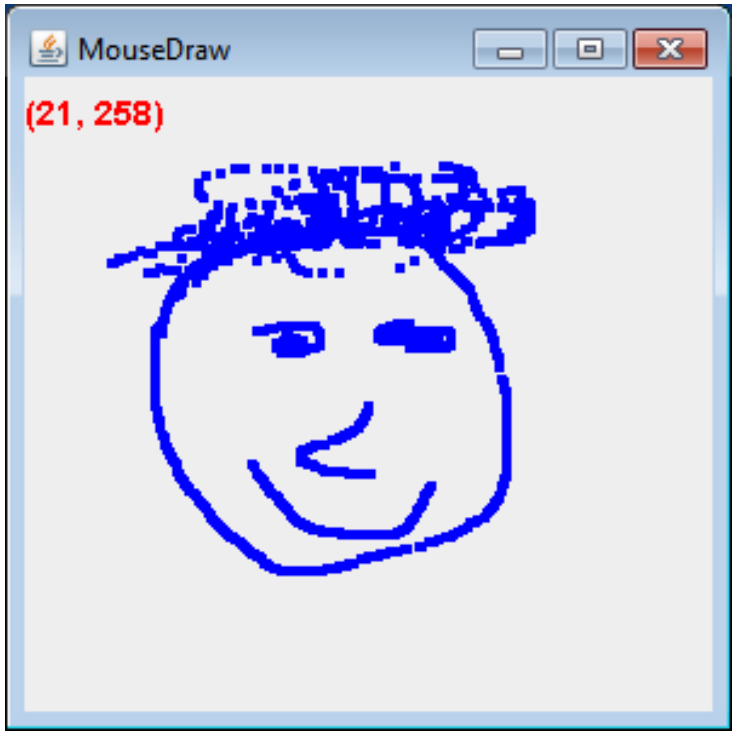
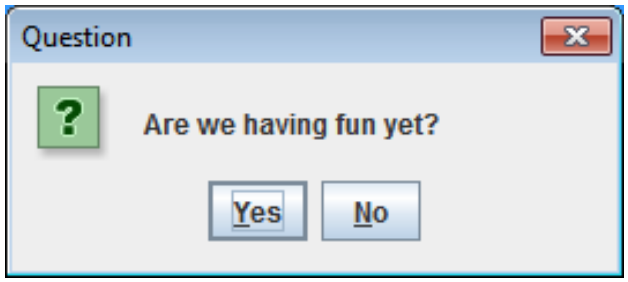
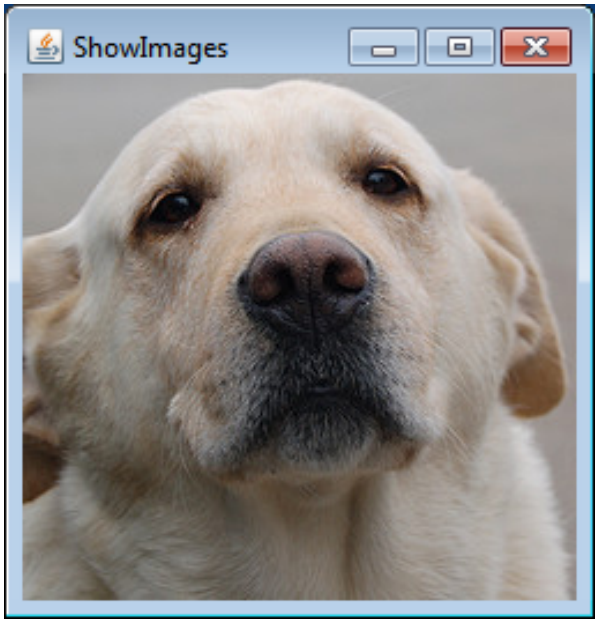
Graphical User Interfaces 2







Overview

- Extending JFrame
- Dialog boxes
  - Getting user input
  - Displaying message or error
- Listening for input
  - Mouse
  - Keyboard
- Java images

Extending JFrame

- Method 1:
  - Class's `main` method creates an instance of the class
  - Runs an instance method (e.g. go) that creates a `JFrame` and associated GUI elements
  - How Head First Java does it
- Method 2:
  - Create a class that `extends JFrame`
  - Constructor of class handles GUI setup
  - Class is itself a `JFrame`, not need to create one
  - Main program class instantiates the class

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

public class ButtonCount implements ActionListener
{
   private int count = 0;
   private JButton button;

   public void actionPerformed(ActionEvent event)
   {
      count++;
      button.setText("count = " + count);
   }

   public void go()
   {
      JFrame frame = new JFrame("ButtonCount");
      button = new JButton("count = " + count);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.getContentPane().add(button);
      frame.setSize(300,300);
      frame.setVisible(true);
      button.addActionListener(this);
   }

   public static void main(String [] args)
   {
      ButtonCount gui = new ButtonCount();
      gui.go();
   }
}
```

Option 1:
Create an object and run a method that explicitly creates JFrame

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

public class ButtonCount2 extends JFrame implements ActionListener
{
   private int count = 0;
   private JButton button;

   public void actionPerformed(ActionEvent event)
   {
      count++;
      button.setText("count = " + count);
   }

   public ButtonCount2()
   {
      super("ButtonCount2");

      button = new JButton("count = " + count);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      getContentPane().add(button);
      setSize(300,300);
      setVisible(true);
      button.addActionListener(this);
   }

   public static void main(String [] args)
   {
      ButtonCount2 gui = new ButtonCount2();
   }
}
```

Option 2:
Make a class that extends JFrame

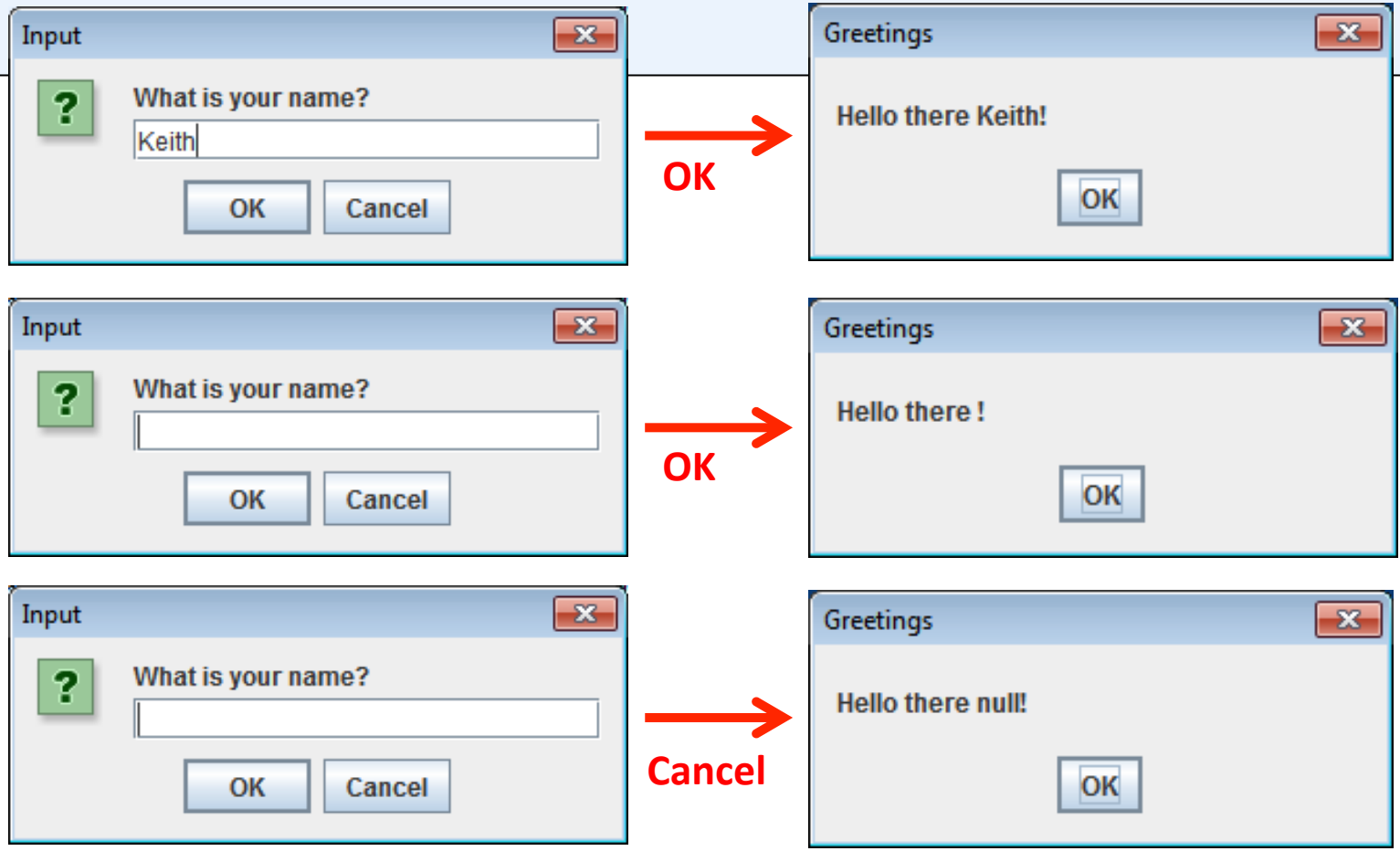
Dialog boxes

- Dialog boxes
  - Ask the user a question
  - Or give the user an error, information, etc.
  - Typically *modal*
    - Blocks rest of GUI until closed
  - Displays different icons depending on parameter

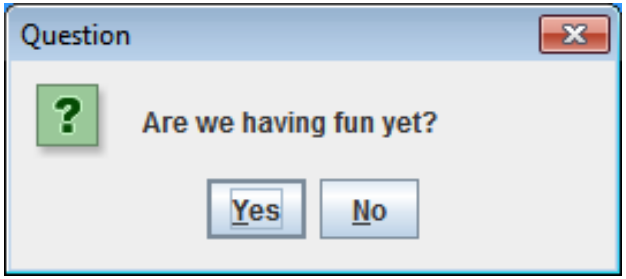
Constant	Java look and feel	Windows look and feel
`JOptionPane.ERROR_MESSAGE`		
`JOptionPane.INFORMATION_MESSAGE`		
`JOptionPane.WARNING_MESSAGE`		
`JOptionPane.QUESTION_MESSAGE`		
`JOptionPane.PLAIN_MESSAGE`		

```
public class NameDialog
{
   public static void main(String [] args)
   {
      String name = JOptionPane.showInputDialog("What is your name?");

      JOptionPane.showMessageDialog(null,
                                    "Hello there " + name + "!",
                                    "Greetings",
                                    JOptionPane.PLAIN_MESSAGE);
   }
}
```

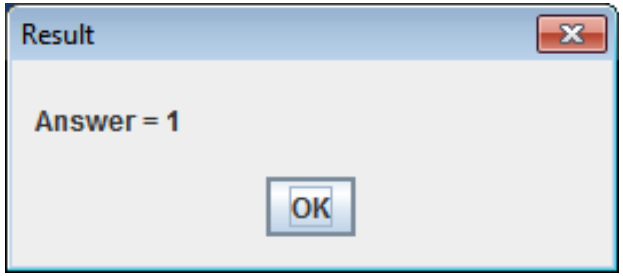
Normally reference to the JFrame object



```
public class YesNoDialog
{
   public static void main(String [] args)
   {
      int result = JOptionPane.showConfirmDialog(null,
                                                 "Are we having fun yet?",
                                                 "Question",
                                                 JOptionPane.YES_NO_OPTION);

      JOptionPane.showMessageDialog(null,
                                    "Answer = " + result,
                                    "Result",
                                    JOptionPane.PLAIN_MESSAGE);
   }
}
```



No



Lots of other dialog related options, see:

http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html#dialogdemo

Mouse input

- `MouseListener`
  - Watches for mouse entry/exit from component
  - Watches for button related actions
  - No events if just moving mouse inside the component
  - Only if inside the listening component!

Method	Purpose
`mousePressed(MouseEvent)`	After the user presses a mouse button while the cursor is over the component.
`mouseReleased(MouseEvent)`	After the user releases a mouse button after a mouse press over the component.
`mouseClicked(MouseEvent)`	After the user clicks the component (after the user has pressed and released).
`mouseEntered(MouseEvent)`	After the cursor enters bounds of the component.
`mouseExited(MouseEvent)`	After the cursor exits bounds of the component.

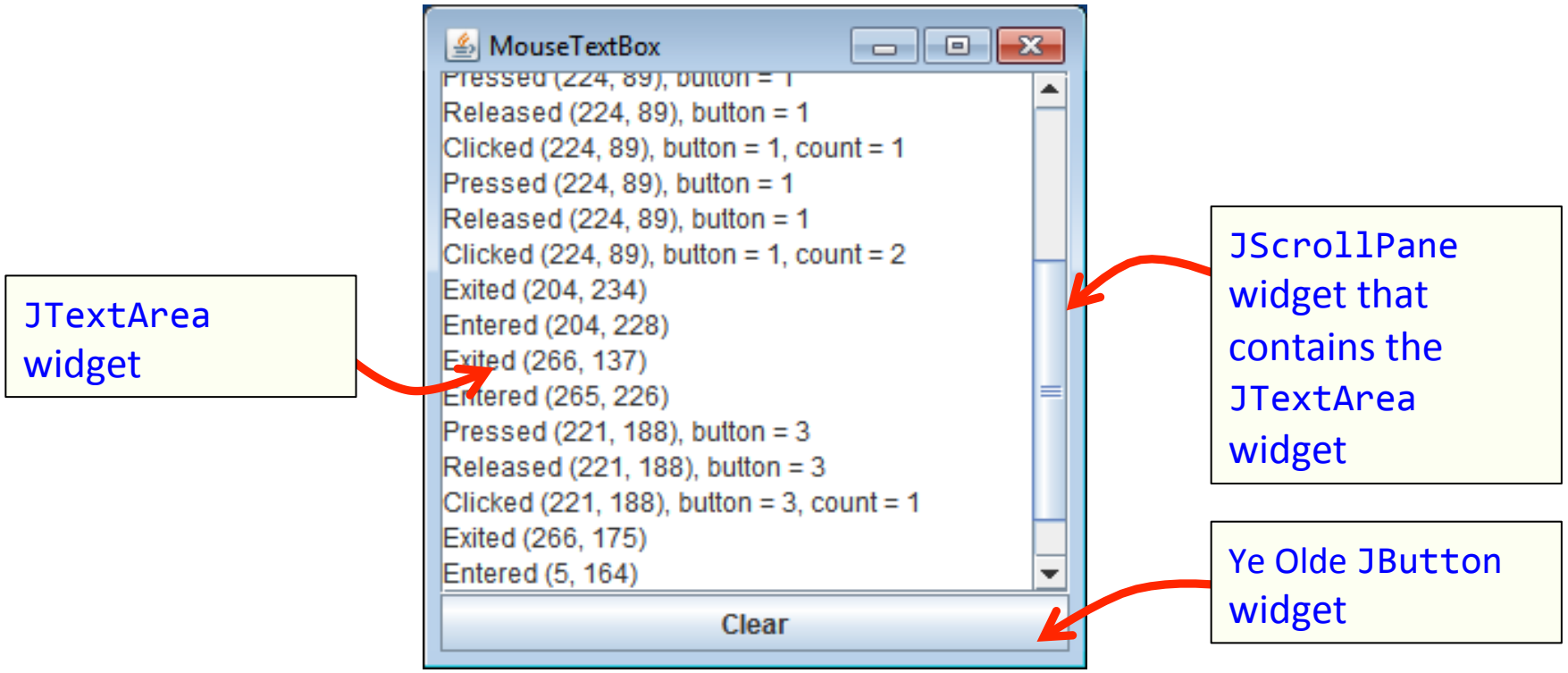
Mouse input

- `MouseEvent`
  - x- and y-coordinate, (0,0) is upper-left
  - Number of quick, consecutive clicks
  - Which button changed state (pushed, released, clicked)

Method	Purpose
`int getClickCount()`	Number of quick, consecutive clicks (including this event). For example, returns 2 for a double click.
`int getX()`	Get the x-coordinate at which event occurred
`int getY()`	Get the y-coordinate at which event occurred
`Point getPoint()`	Return a Point object containing event location
`int getButton()`	Which button changed state: `NOBUTTON`, `BUTTON1`, `BUTTON2`, or `BUTTON3`.

Mouse input example 1

- GUI with a single big text area
  - Add line of text to area on `MouseListener` event
  - Output event type and mouse (x, y)
  - Note: events only triggered in `JTextArea` not `JButton`



MouseTextBox.java

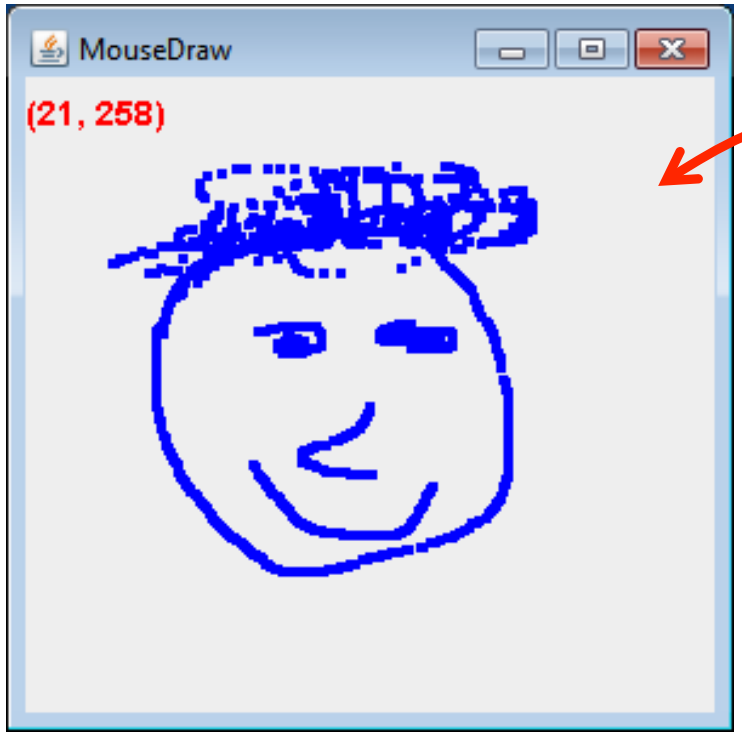
Mouse motion

- `MouseMotionListener`
  - Detects movement of mouse inside a component
  - With or without the mouse pressed

Method	Purpose
`mouseMoved(MouseEvent)`	User is moving the mouse with no mouse button pressed.
`mouseDragged(MouseEvent)`	User is moving the mouse while holding a mouse button down (i.e. a dragging action). Always preceded by call to `mousePressed` event.

Mouse motion example 2

- Simple drawing application
  - During `MouseDragged` event, add `Point` objects
  - Requires a custom `JPanel` that draw all the points
    - Override the `paintComponent(Graphics g)` method
  - Also display current mouse (x, y) in upper-left



`MouseDrawPanel extends JPanel`

MouseDraw.java
MouseDrawPanel.java

Keyboard input

- `KeyListener`
  - When a key is pressed, released, or typed
    - Typed event only for printable characters (not arrow keys, etc.)
    - Numeric key codes for all event types
  - Component must have focus to fire event
    - For custom components (e.g. game drawing panel):
      - Ensure it can accept focus: `setFocusable(true)`
      - `mouseClicked()` handler that calls `requestFocusInWindow()`
      - Or make all other UI widgets not focusable

Method	Purpose
`keyTyped(KeyEvent)`	Called just after user types a Unicode character
`keyPressed(KeyEvent)`	Called just after the user presses a key
`keyReleased(KeyEvent)`	Called just after the user releases a key

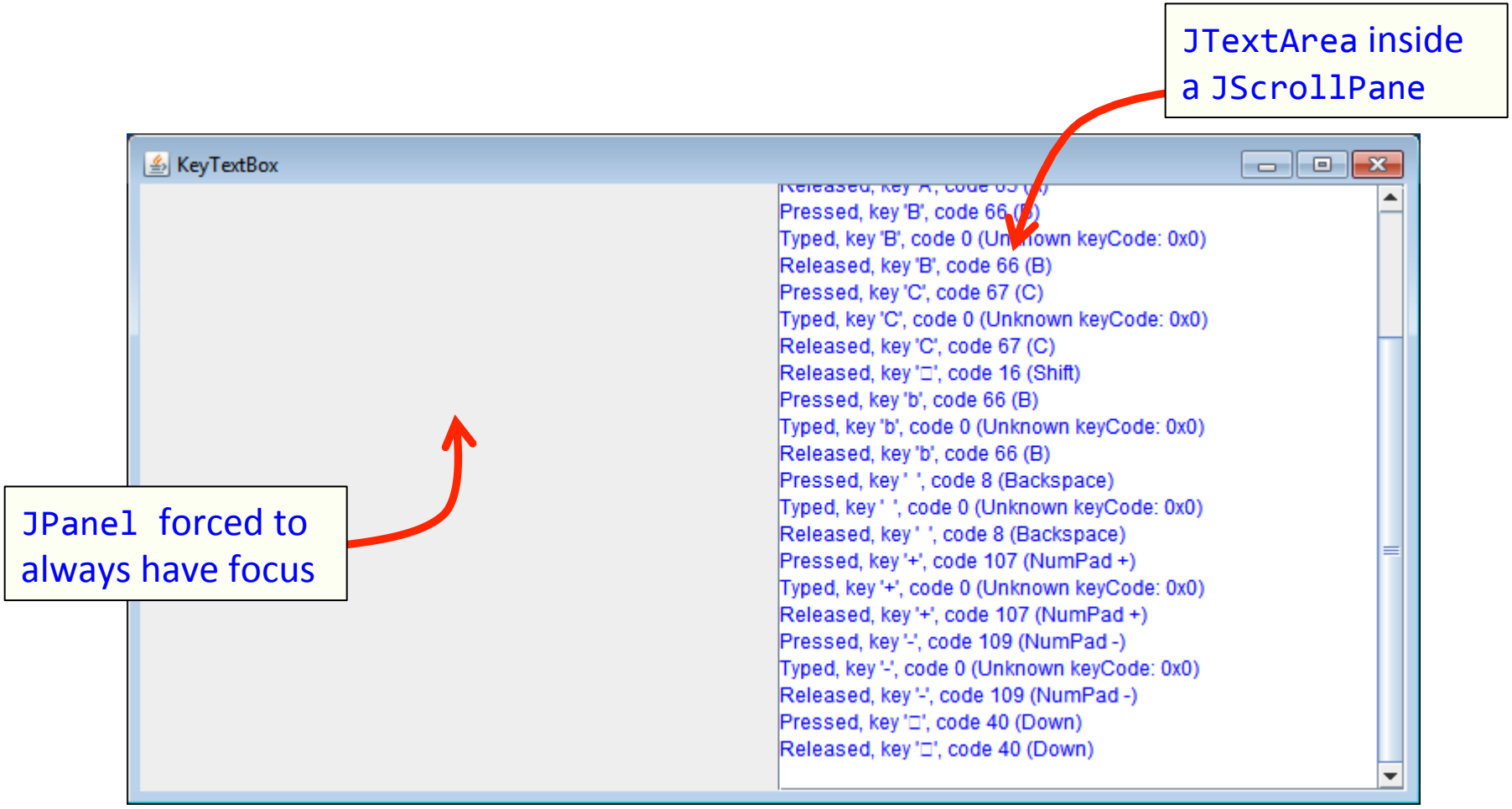
Keyboard input

- `KeyEvent`
  - Figure out what the user typed or pressed
  - Actual character to typed events
  - Only key code for pressed/released events

Method	Purpose
`int getKeyChar()`	Return Unicode character of event, only use for key typed events.
`int getKeyCode()`	Return the key code associated with event. For example, VK_A = letter A, VK_DOWN = down arrow key.
`int getModifiersEx()`	Extended modifier mask for the event, such as whether shift or alt key was down.

Keyboard input example

- Listen for keyboard events
  - Output text about each event



KeyTextBox.java

Drawing images

- Loading a JPG, PNG, GIF:
  - Construct an `ImageIcon` object
  - Pass it the filename
  - Convert to an `Image` object for drawing
  - Width and height will be -1 on error

```
Image image = new ImageIcon("cat.jpg").getImage();
int width   = image.getWidth(this);
int height  = image.getHeight(this);
```

The component (e.g. JFrame, JPanel) that image is being drawn on.

Drawing images

- Drawing on a panel
  - In the `paintComponent(Graphics g)` method
  - `g.drawImage(Image img, int x, int y, ImageObserver obs)`
  - NOTE: (x, y) is the upper-left corner of the image
  - Keep the `Image` object around
    - So you don't have to keep loading from disk using the `ImageIcon` constructor

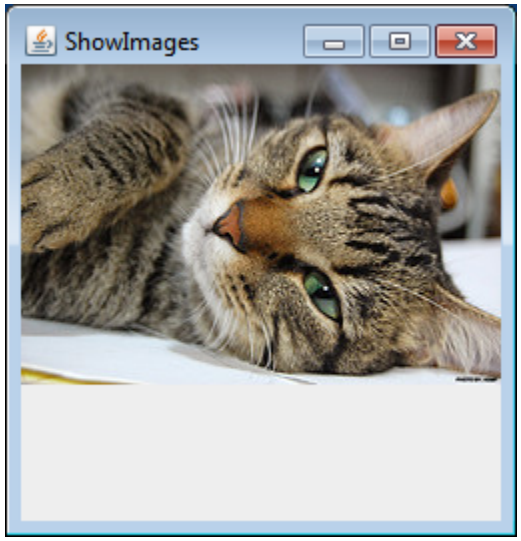
```
g.drawImage(img, 0, 0, this);
```

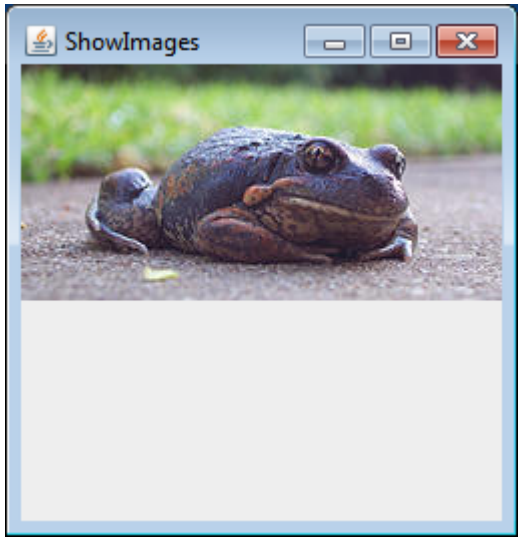
The component (e.g. JFrame, JPanel) that image is being drawn on.

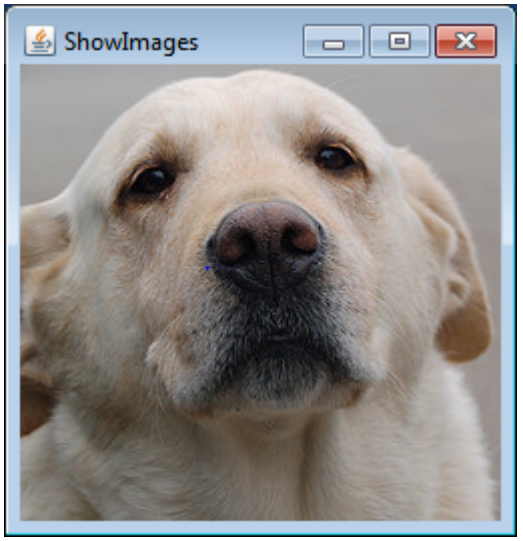
Programming activity

- Create an image slideshow GUI
  - Loads images specified on command line
  - Outputs height/width, ignore bogus files
  - Cycles between images every second

```
% java ShowImages cat.jpg frog.jpg dog.jpg bogus.jpg
cat.jpg [240 x 160]
frog.jpg [240 x 118]
dog.jpg [240 x 228]
```







Summary

- Extending JFrame
  - Constructor sets up the GUI widgets
- Dialog boxes
  - Collect a string or response to several buttons
  - Can just provide information or error message
- Responding to mouse events
  - `MouseListener` handler for click-related events
  - `MouseMotionListener` for tracking mouse coordinates
- Responding to keyboard events
  - Make sure listening component can and does have focus
- Loading and display images