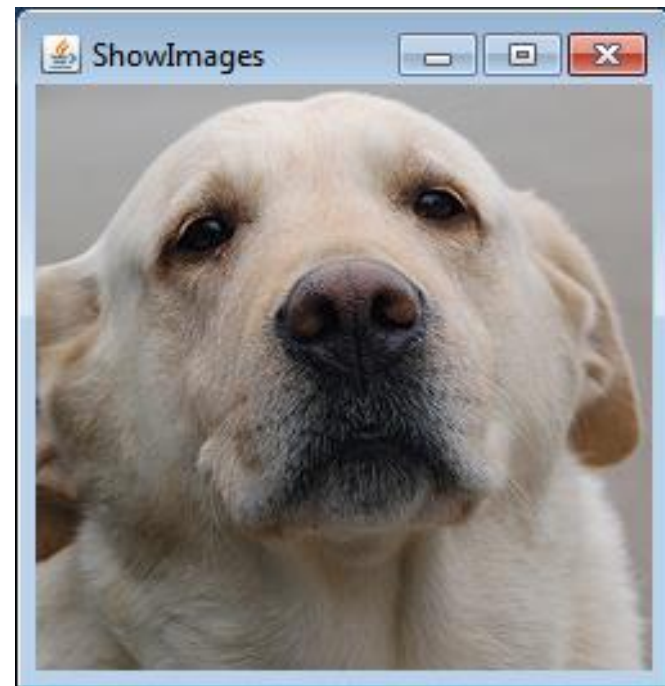
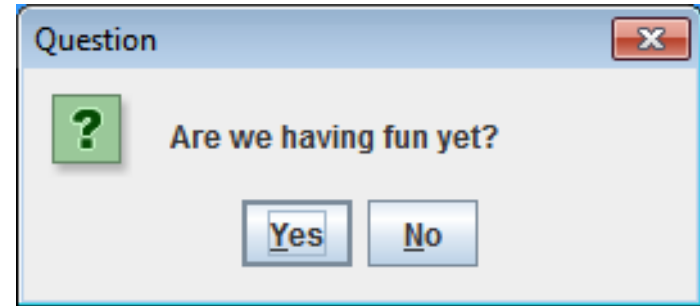
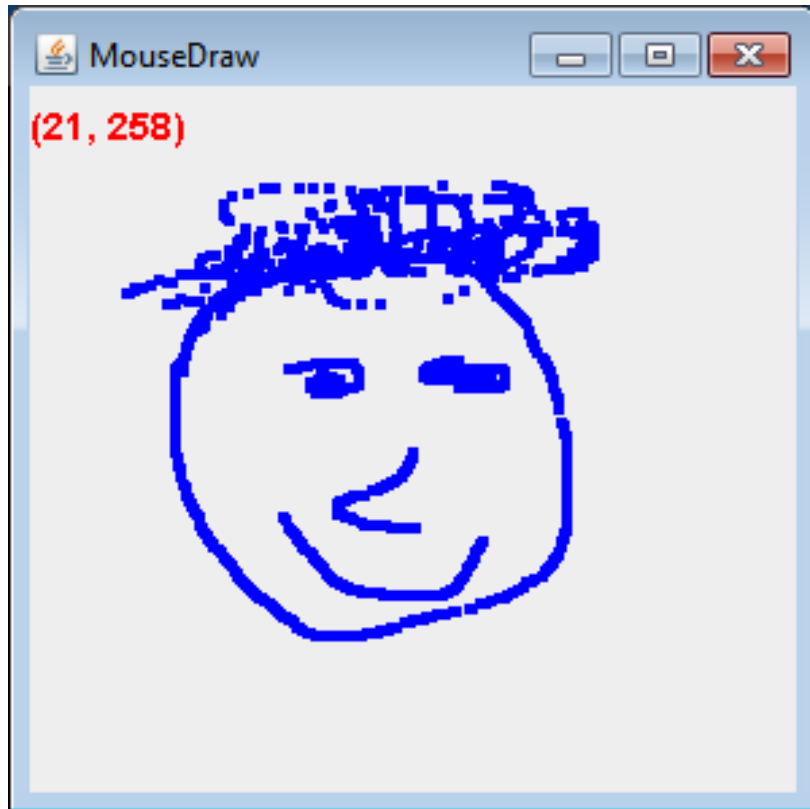
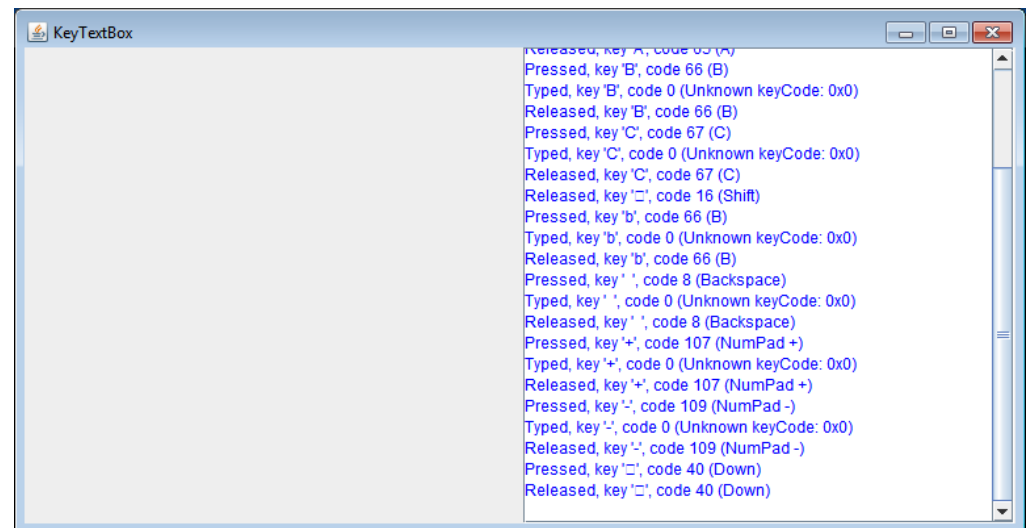
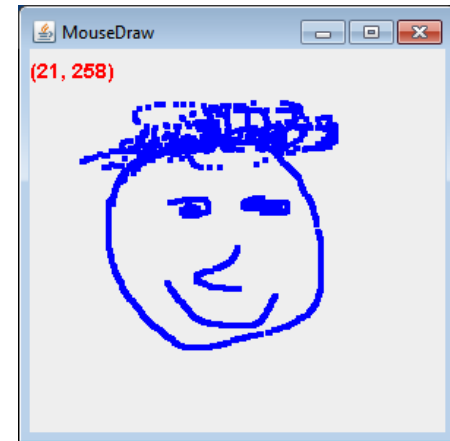
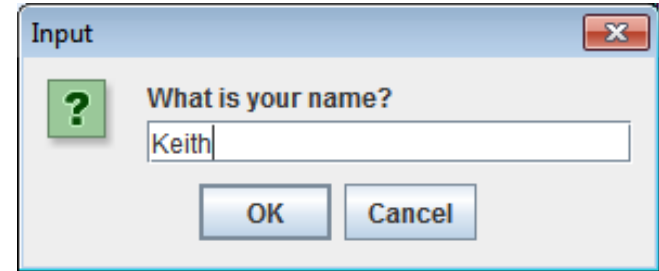


# Graphical User Interfaces 2



# Overview

- Extending JFrame
- Dialog boxes
  - Getting user input
  - Displaying message or error
- Drawing shapes and images
  - JPanel
- Listening for input
  - Mouse
  - Keyboard



# Extending JFrame

- **Approach 1:** (last lecture)
  - `main()` creates instance of the class
  - Runs instance method, e.g. `go()`
    - Creates a `JFrame` and associated GUI elements
  - How Head First Java does it
  - Preferred method
- **Approach 2:**
  - Create a class that **extends `JFrame`**
  - Constructor handles GUI setup
  - **No need to create a `JFrame`**
  - 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();
    }
}
```

**Approach 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();
    }
}

```









**Approach 2:**  
**Make a class that**  
**extends JFrame**

Calls JFrame constructor  
that takes the window  
title as a parameter.

These are instance methods of JFrame,  
but we are a JFrame, so no need to put  
anything before method name.

# Dialog boxes

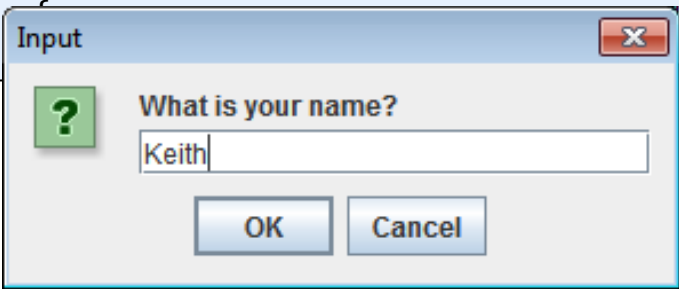
- Dialog boxes
  - Asks a **question**
  - Or gives an **error, information**, etc.
  - Typically *modal*
    - **Blocks rest of GUI until closed**
  - Displays different icons depending on parameter

Constant	Java look and feel	Windows look and feel
<code>JOptionPane.ERROR_MESSAGE</code>		
<code>JOptionPane.INFORMATION_MESSAGE</code>		
<code>JOptionPane.WARNING_MESSAGE</code>		
<code>JOptionPane.QUESTION_MESSAGE</code>		
<code>JOptionPane.PLAIN_MESSAGE</code>		

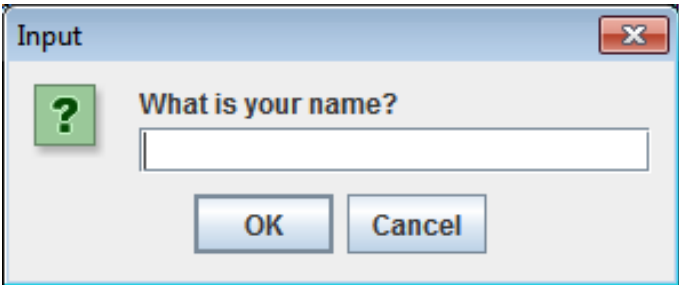
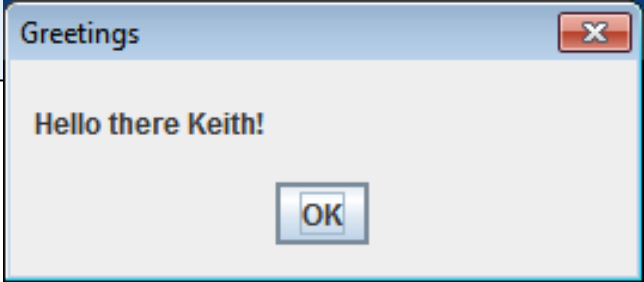
```
public class NameDialog
{
    public static void main(String [] args)
    {
        String name = JOptionPane.showInputDialog("What is your name?");

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

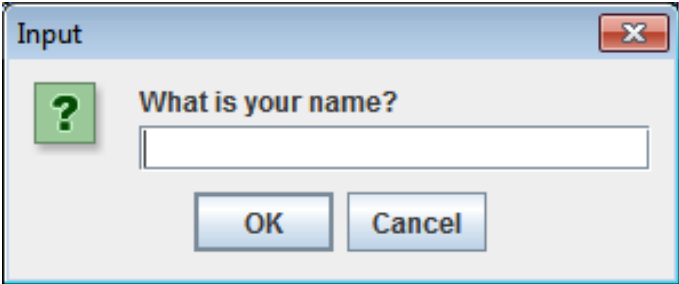
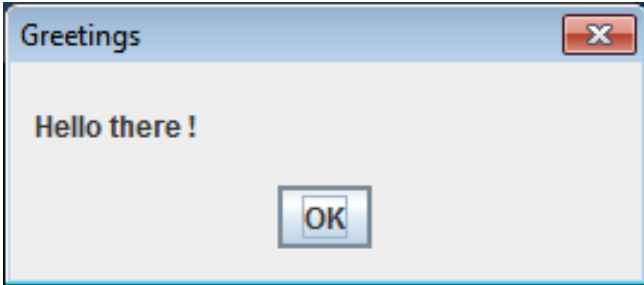
Normally a reference to the JFrame object



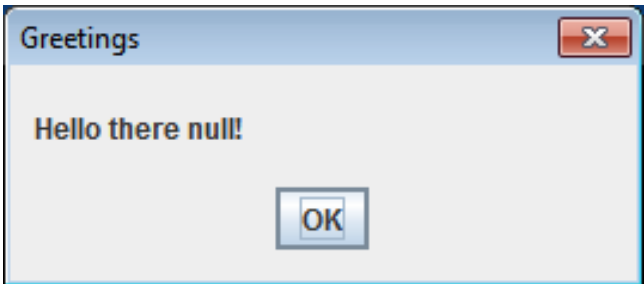
OK  
(text entered)



OK  
(no text entered)



Cancel

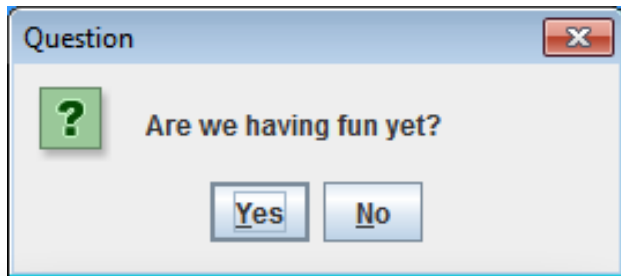


```

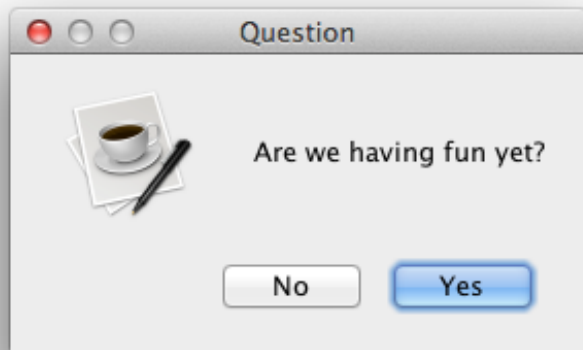
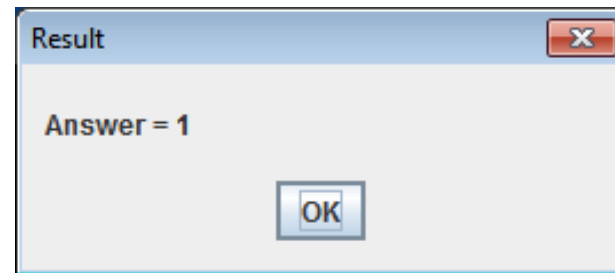
public class YesNoDialog
{
    public static void main(String [] args)
    {
        int result = JOptionPane.showConfirmDialog(null,
                                                    "Are we having fun yet?",
                                                    "Question",
                        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

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

```



**No**  
(Windows 7)



Lots of other dialog  
related options, see:

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



# Panels

- **JPanel**
  - Purpose 1: **Container** for other widgets
    - Allows more control of layout
  - Purpose 2: **Place to draw** lines, circles, images, etc.
    - Like StdDraw
  - Needs to be added to a JFrame
  - Class that **extends JPanel**, drawing done by:
    - `public void paintComponent(Graphics g)`
    - **Called automatically** when needed
      - e.g. window resized
    - Or by **calling `repaint()`** on JFrame

```

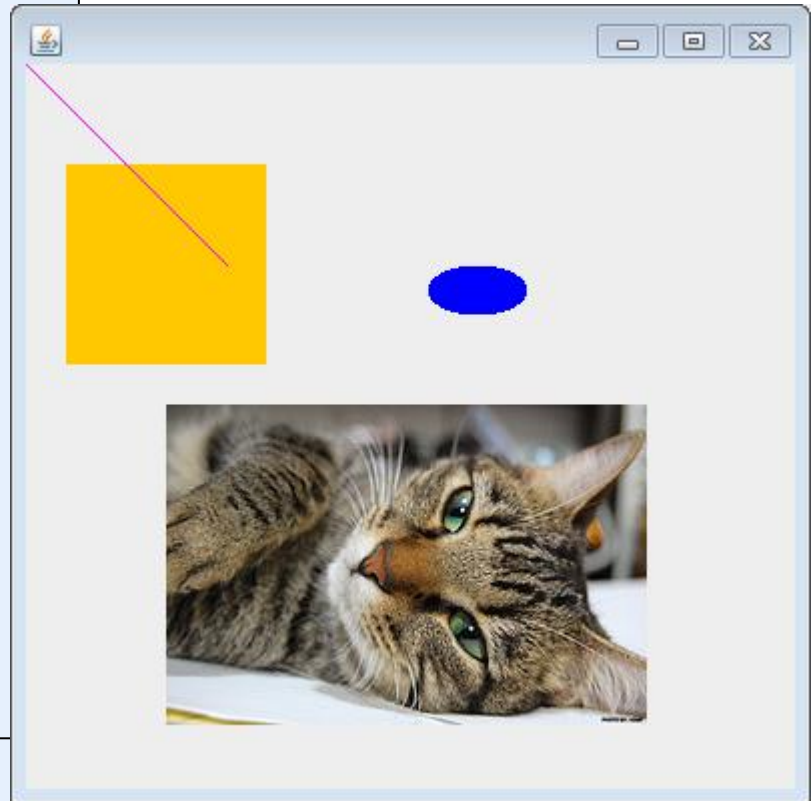
public class MyDrawPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        g.setColor(Color.ORANGE);
        g.fillRect(20,50,100,100);

        g.setColor(new Color(1.0f, 0.0f, 1.0f));
        g.drawLine(0, 0, 100, 100);

        g.setColor(Color.BLUE);
        g.fillOval(200, 100, 50, 25);

        BufferedImage image = null;
        try
        {
            image = ImageIO.read(new File("cat.jpg"));
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        g.drawImage(image, 70, 170, null);
    }
}

```



```

public class Panel
{
    public static void main(String [] args)
    {
        JFrame frame = new JFrame();

        MyDrawPanel panel = new MyDrawPanel();
        frame.getContentPane().add(BorderLayout.CENTER, panel);

        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

```

# Drawing images

- Loading a JPG, PNG, GIF:
  - Construct BufferedImage using static method
  - Pass it a File object constructed using filename
  - Will be null on error
  - ImageIO.read can throw IOException

Special static method that constructs an object of type BufferedImage.

```
BufferedImage image = ImageIO.read(new File("cat.jpg"));  
if (image != null)  
{  
    int width = image.getWidth();  
    int height = image.getHeight();  
}
```

# Drawing images

- Drawing on a panel

- In the `paintComponent(Graphics g)` method
  - `g.drawImage(Image image, int x, int y, ImageObserver obs)`
- NOTE:  $(x, y)$  is the **upper-left corner of image**
- Keep the `BufferedImage` object around
  - Avoid loading from disk each time you need it

```
g.drawImage(image, 0, 0, null);
```

The component (e.g. `JFrame`, `JPanel`) that gets notified if image was not completely loaded when `drawImage` was first called. We can just say `null` since we always load image from disk before calling.

# Mouse input

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

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

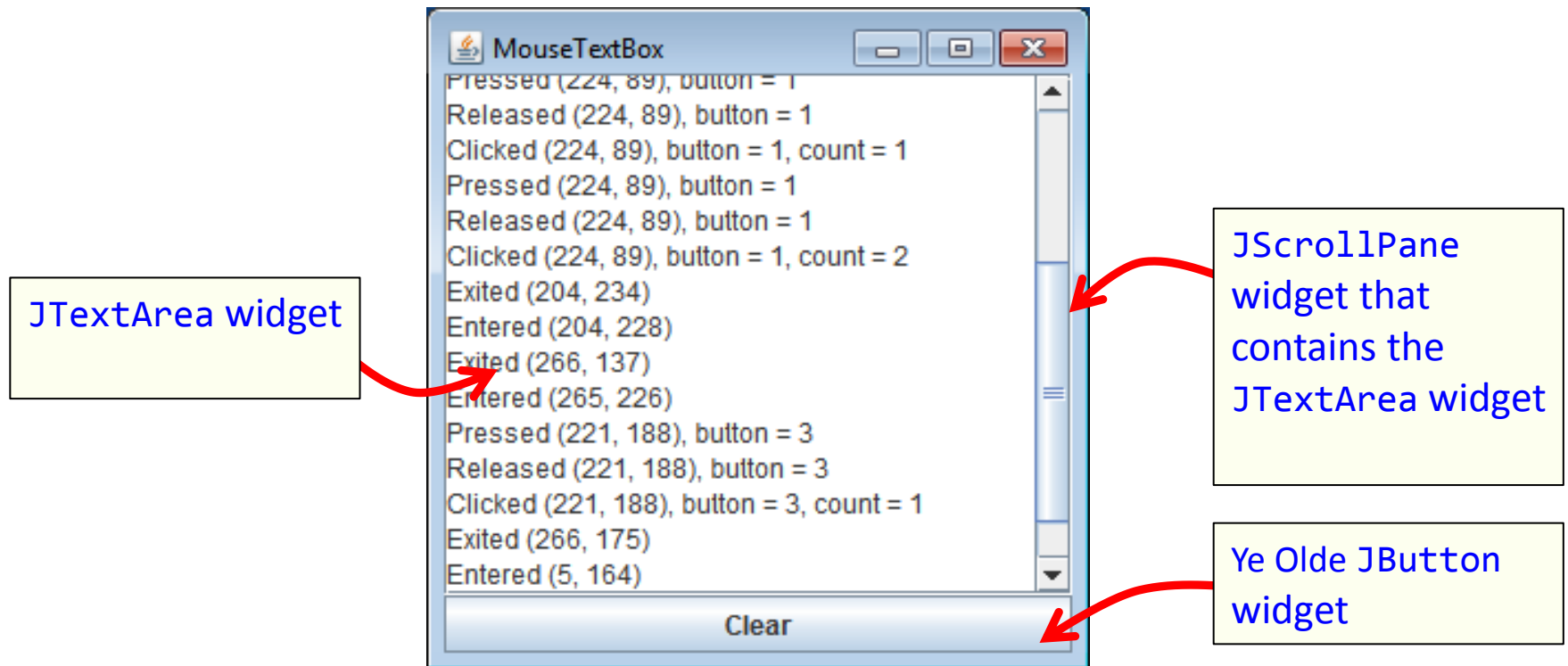
# Mouse input

- **MouseEvent**
  - **(x, y) pixel coordinate**: (0,0) is upper-left
  - Number of **consecutive clicks**
  - **Button that changed** state (pushed, released, clicked)

Method	Purpose
<code>int getClickCount()</code>	Number of quick, consecutive clicks (including this event). For example, returns 2 for a double click.
<code>int getX()</code>	Get the x-coordinate at which event occurred
<code>int getY()</code>	Get the y-coordinate at which event occurred
<code>Point getPoint()</code>	Return a Point object containing event location
<code>int getButton()</code>	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)
  - Events only triggered in JTextArea not JButton



*MouseTextBox.java*

# Mouse motion

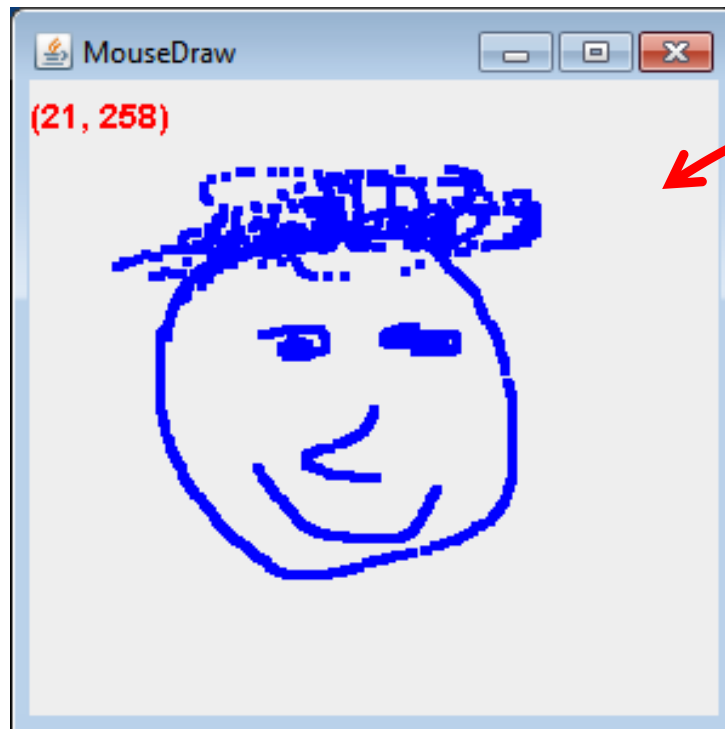
- **MouseEventListener**
  - Detects movement of mouse inside a component
  - With or without the mouse pressed

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



# Mouse motion example 2

- Simple drawing application
  - During **MouseDragged event**, add Point objects
  - Requires a custom JPanel that draws all the points
    - Override `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
<code>keyTyped(KeyEvent)</code>	Called just after user types a Unicode character
<code>keyPressed(KeyEvent)</code>	Called just after the user presses a key
<code>keyReleased(KeyEvent)</code>	Called just after the user releases a key

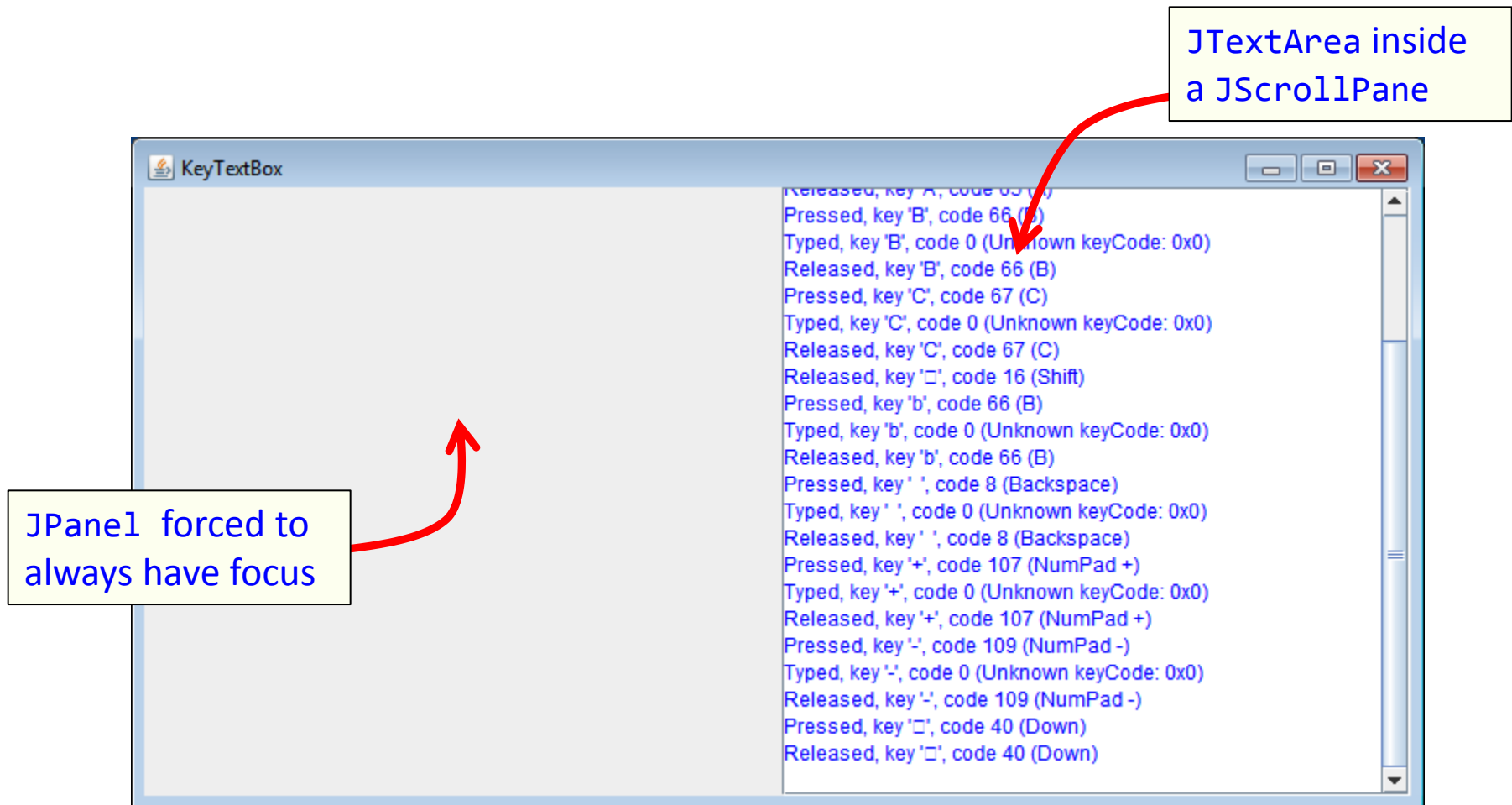
# Keyboard input

- **KeyEvent**
  - Figure out what was typed or pressed
  - Actual **character for typed events**
  - Only **key code for pressed/released events**

Method	Purpose
<code>int getKeyChar()</code>	Return Unicode character of event, only use for key typed events.
<code>int getKeyCode()</code>	Return the key code associated with event. For example, <code>VK_A</code> = letter A, <code>VK_DOWN</code> = down arrow key.
<code>int getModifiersEx()</code>	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



# Summary

- **Extending JFrame**
  - Constructor sets up the GUI widgets
- **Dialog boxes**
  - Collect a response, provide info or error
- **Drawing shapes and images**
  - Requires a JPanel
- **Responding to mouse and keyboard events**
  - `MouseListener` for click related events
  - `MouseMotionListener` for tracking mouse
  - `KeyListener` for keyboard events