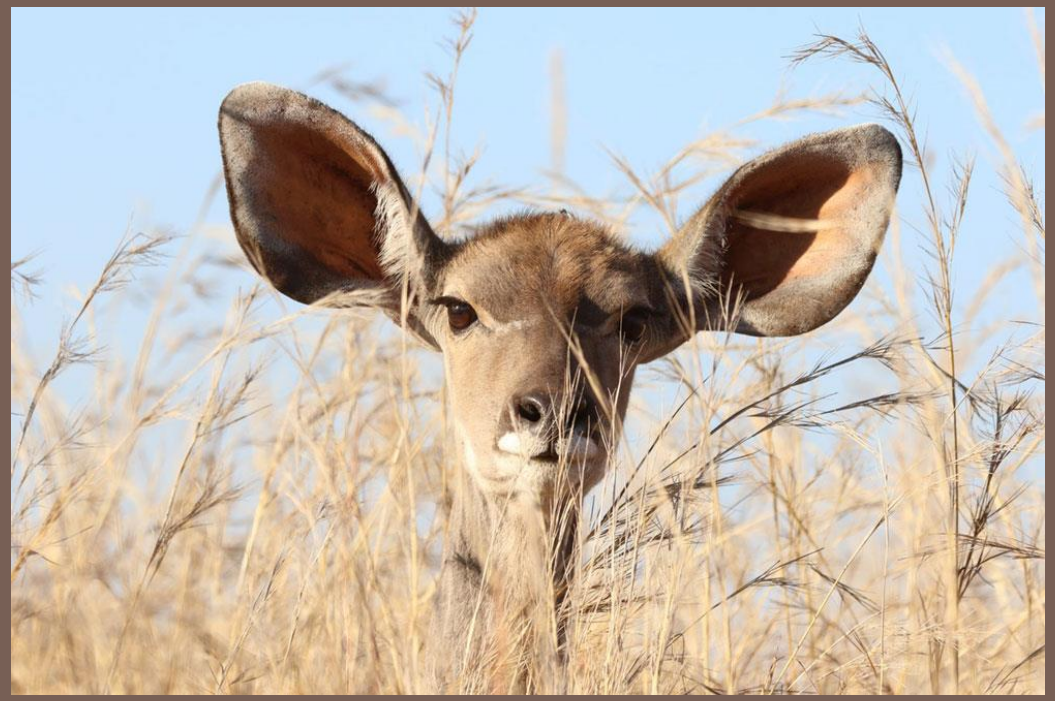
Announcements

- Quiz 9 on Canvas (today-Mon)
- A5 due Tue, Nov 1
- A6 will be a short GUI assignment (Nov 2-10)
- Prelim 2 is Nov 15; conflict survey coming soon



GUIS: HANDLING EVENTS

Lecture 19

CS 2110 Fall 2022

Objectives

Last lecture

- How to open a window
- How to lay out components (aka "widgets")
- Working with a GUI framework (inversion of control)

This lecture

- How to respond to events (e.g. mouse clicks)
- How to draw ("paint") on a component
- How to use dialogs for input/output

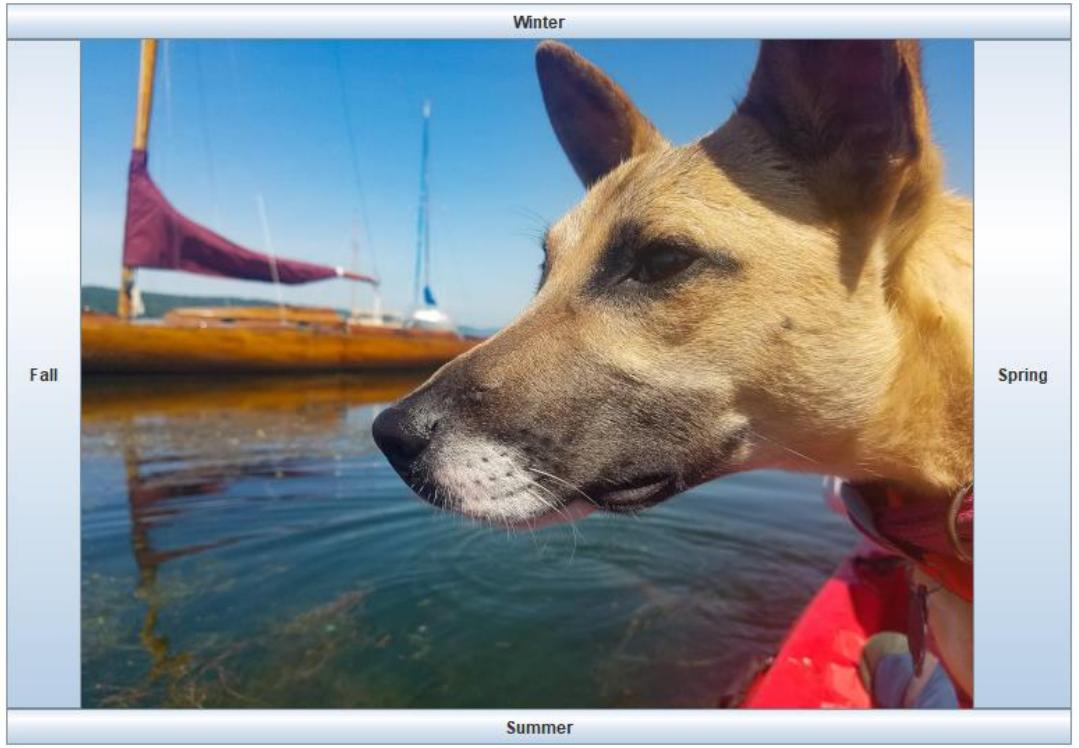
Reference: The Swing Tutorial

Graphical User Interface (GUI)

- Allows friendlier, richer interaction between user and program
- Requires new programming techniques
  - Reactive, event-driven programs; inversion of control
    Code is executed *in response to* mouse clicks, keyboard input, etc.
  - Involves multiple threads of execution
    Can perform background calculation while updating animation
- Separation of concerns: Model, View, Controller (MVC)
  - Model: application state and logic (familiar Java code)
  - View: Show state to users, accept inputs (last lecture)
  - Controller: Respond to events, updating model (this lecture)

Demo

- JFrame
- JLabel
- JButton
- ImageIcon
- invokeLater()



Reactive programming responds to *events*

- Examples of events:
  - Button pressed
  - Menu item selected
  - Key pressed
  - Mouse moved
  - Timer expired
  - Property changed

User input

Code

- When an event occurs, the *source* of the event notifies registered *listeners*
  - A listener is an object with a method appropriate for responding to the event
  - Listeners must be registered with event sources (added to their list of listeners), often separately for each event type

ActionEvents

- Buttons and menu items trigger ActionEvents when selected
- Class ActionEvent
  - getActionCommand()
  - getSource()
  - getWhen()
- Interface ActionListener
  - actionPerformed(ActionEvent e)
- Class JButton
  - addActionListener(ActionListener l)

Old style: Named classes as listeners


```
class App extends JFrame
        implements ActionListener {
  App() {
    JButton b = new JButton("B");
    add(b);
    b.addActionListener(this);
  }
  @Override
  public void actionPerformed(
        ActionEvent e) {
    print("Got " + e.getActionCommand()
        + " from " + e.getSource());
  }
}
```

- Remarks:
  - Can add same listener to many event sources (but distinguishing events by source is not very OO)
  - Defining fine-grained listeners requires defining many classes – tedious! (anonymous classes help some)

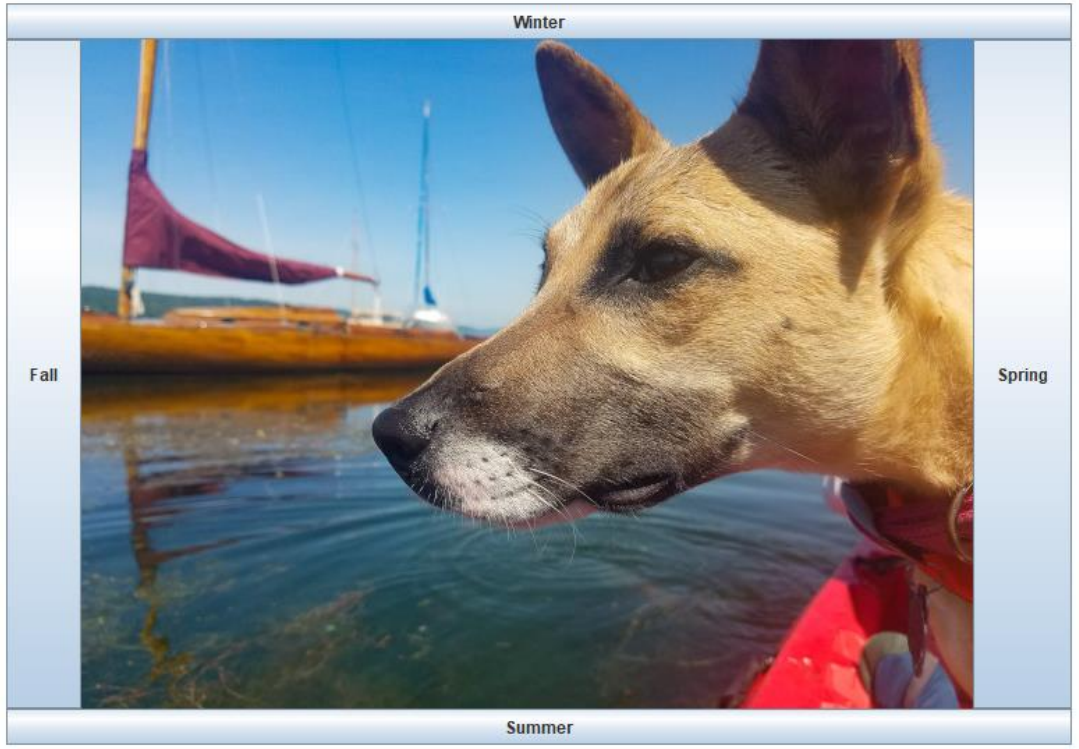
New style: Anonymous functions as listeners

- ActionListener is an Interface with only 1 abstract method
  - Can implement the interface with an anonymous function
- Caveat: some listener interfaces have multiple abstract methods (e.g. MouseListener)
  - Can't use anonymous functions
  - Can use anonymous adapter classes (see window listener in A6)

```
class App extends JFrame {
  App() {
    JButton b = new JButton("B");
    add(b)
    b.addActionListener(e ->
      print("Got " +
        e.getActionCommand())
    );
  }
}
```

Back to demo

- JFrame
- JLabel
- JButton
- ImageIcon
- invokeLater()

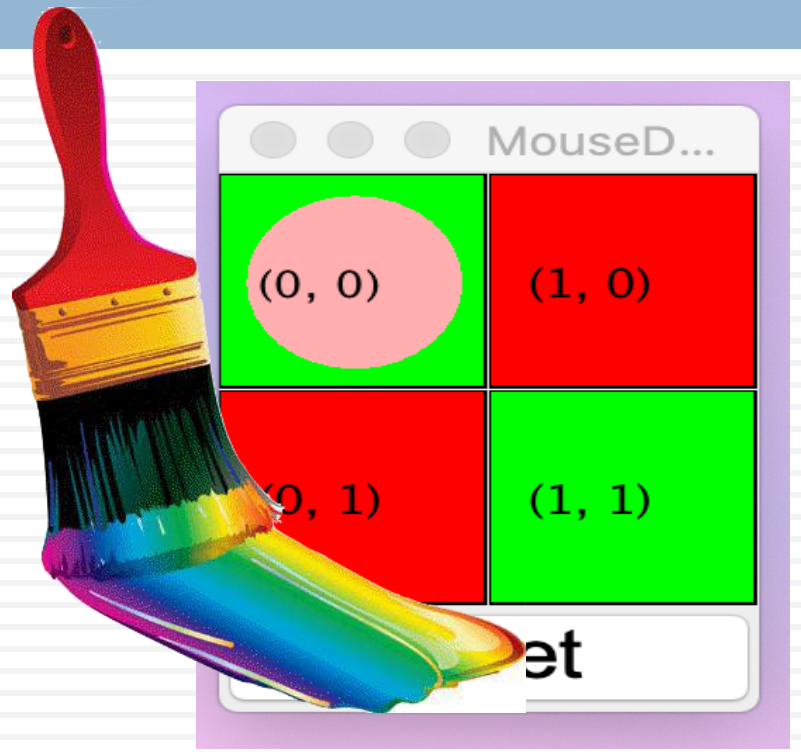


Common events and listeners

- ActionListener (1)
  - Buttons, menu items, …
- MouseListener (5)
  - Mouse clicks and enter/exit
- MouseMotionListener (2)
  - Mouse movement and dragging
- KeyListener (3)
  - Keyboard key presses
- WindowListener (7)
  - Window closing, minimizing, focus
- ChangeListener (1)
  - Sliders
- ItemListener (1)
  - Checkboxes, toggles
- PropertyChangeListener (1)

And many more…

Custom painting



Review: custom components

- Extend JComponent (or JPanel)
- Override paintComponent(Graphics g)
- Attach event listeners

- Why so many paint methods?
  - paint()
  - paintComponent()
  - paintImmediately()
  - repaint()
  - …
- Keep in mind: *inversion of control* and *delegation*

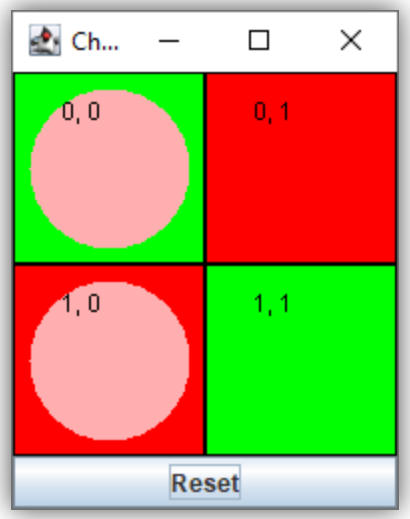
Clarifying paint methods

- repaint(): Request that this component be redrawn at the next opportunity
  - Most of your GUI code runs in event handlers – you are not in control
  - Multiple events might necessitate repainting before the next screen refresh – don't do redundant work
- paint(): "Event handler" for "it's time to draw yourself" events
  - Declared in java.awt.Component
  - Overridden by JComponent:
    - paintComponent()
    - paintBorder()
    - paintChildren()
- paintComponent(): Preferred method to override when extending Swing components
  - Start with super.paintComponent()

Drawing with a Graphics object

- Useful methods:
  - setColor()
  - fillRect()
  - fillOval()
  - drawLine()
  - setFont()
  - drawString()
- Coordinates
  - (0,0) is upper-left corner
  - +y means down
  - Integers: gridlines between pixels
  - "Draw": pixel southeast of gridpoint

Demo: checkerboard



- Custom painting
  - setColor(), drawRect(), fillRect(), fillOval(), drawString()
- Respond to mouse clicks
  - Alternative: extend MouseListenerAdapter
- Respond to button clicks
- Grid layout
- Iterating over child components

Dialogs

- JFileChooser: standard dialogs for opening and saving files
  - "show" methods will **block** until user closes dialog
  - "modal" dialog – cannot interact with *parent frame*
  - Returns whether user "approved" or clicked "cancel"
  - getSelectedFile()
- JOptionPane: standard dialogs for getting quick input or showing messages
  - Static methods for most common dialogs
  - Also modal and blocking

End of GUIs for CS 2110

- Exercise: Build your own interface and respond to events
  - Consider modifying an existing example
  - Lots of details to pay attention to if you want a polished product
    - Keyboard shortcuts, hotkeys, tooltips, right-click menus, localization
- Browse the Swing Tutorial
- Look at demo classes – very short and focused