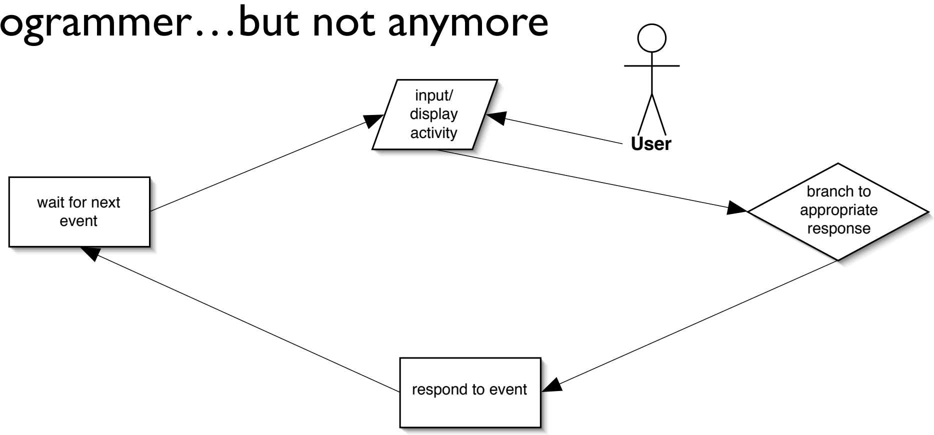
The Java Event Model

- Note how all of the code we have written so far has been 100% *setup* — it executes *before* our panels even appear in front of our users
- What happens when the users see it and start interacting with it? — that's where events come in
- Java, and therefore Swing, provides a well-defined API-level mechanism for triggering, delivering, and, most importantly, receiving *events* from the system — whether generated by the user or by other processes within a Java program

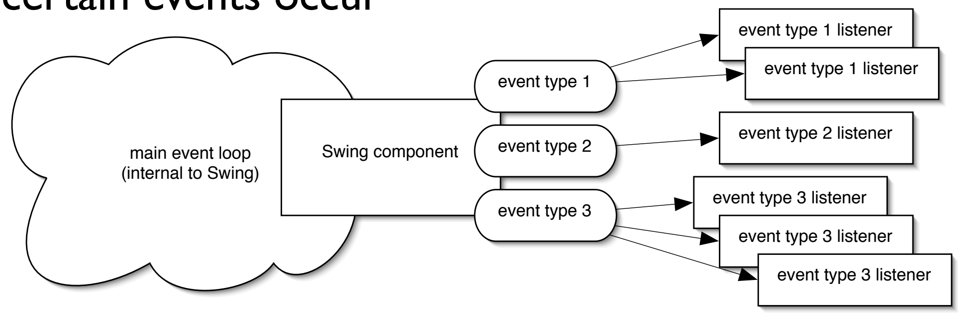
The GUI Event Loop

All graphical user interfaces ultimately follow a control flow called an *event loop*; in the early days of GUI programming, this event loop was visible to the programmer…but not anymore



Event Handlers are Like Plug-Ins to the Event Loop

- Swing defines a wide variety of "things that can happen, and to whom" during the main event loop
- We program our custom GUI behavior by defining objects ("listeners") that wish to be notified when certain events occur



main event loop
(internal to Swing)

Swing component

event type 1

event type 2

event type 3

event type 1 listener

event type 1 listener

event type 2 listener

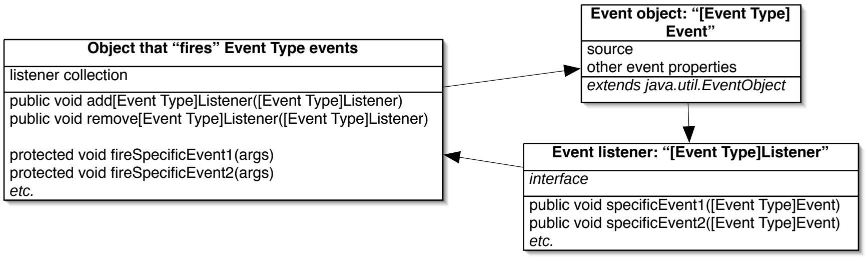
event type 3 listener

event type 3 listener

event type 3 listener

Events in Java

Events and event listeners follow a very specific scheme



Object that "fires" Event Type events

listener collection

public void add[Event Type]Listener([Event Type]Listener)
public void remove[Event Type]Listener([Event Type]Listener)

protected void fireSpecificEvent1(args)
protected void fireSpecificEvent2(args)
etc.

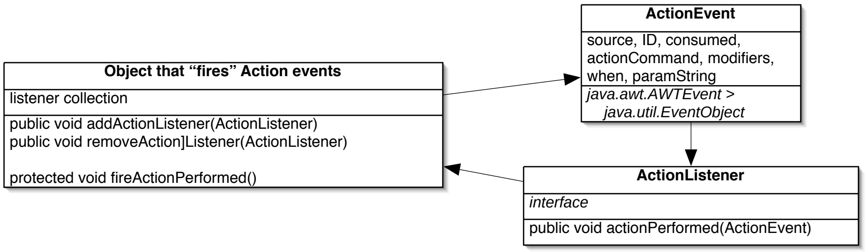
Event object: "[Event Type] Event"

source
other event properties
extends java.util.EventObject

Event listener: "[Event Type]Listener"

interface

public void specificEvent1([Event Type]Event)
public void specificEvent2([Event Type]Event)
etc.



Object that "fires" Action events

listener collection

public void addActionListener(ActionListener)
public void removeAction]Listener(ActionListener)

protected void fireActionPerformed()

ActionEvent

source, ID, consumed,
actionCommand, modifiers,
when, paramString

java.awt.AWTEvent >
java.util.EventObject

ActionListener

interface

public void actionPerformed(ActionEvent)

Events in Swing

- As before, we can't cover absolutely every single possible event type in Swing in class; instead, we look at some highlights and hope you can pick up the rest on your own
- It is useful to think of events in Swing as having two broad categories:
  1. Component- or widget-level events: events that are specific to the functions of Swing components, but independent of the physical, triggering mechanism
  2. Input-level events: events that are bound to physical input devices, such as the mouse or keyboard

Rolling Your Own

- Start by defining the core "event" that you are implementing
- Define the event object class and event listener interface for this event
- Define the methods within the event listener interface that correspond to the events you want to "fire"
- Implement add/remove listener methods at the objects that can generate your event
- Implement protected "fire" events for those objects