Programming Languages and Techniques (CIS1200)

Lecture 20

GUI library: Events, Notifiers, and Controllers

Chapters 18

Looking Ahead…

- HW05: GUI Programming
  - due Tuesday, **March 18** after Spring Break
  - ***START NOW!!***
  - aim to complete by Friday

- Friday March 7th: NO CLASS
- No classes/recitations/TA office hours during Spring Break!

- HW06: Pennstagram (soft launch)
  - Due Tuesday, **March 25th**
  - If you finish HW06 and want to refresh Java over break, this project is now available.

20: How far along are you in HW05: GUI Programming?

0

Not started yet — 0%

Task 0 finished — 0%

Working on tasks 1-4 — 0%

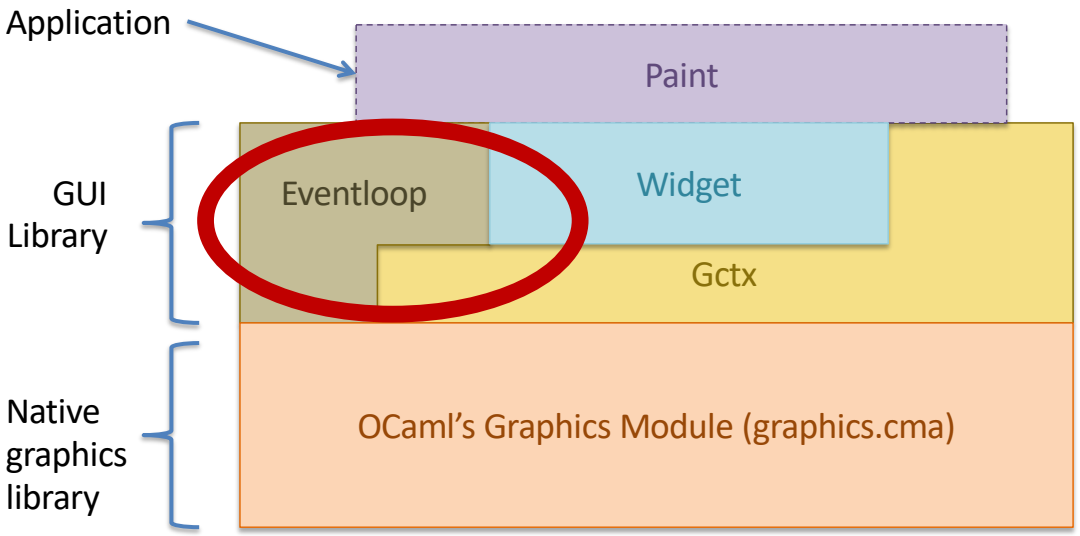
Working on Task 5 — 0%

Working on Task 6 — 0%

All done! — 0%

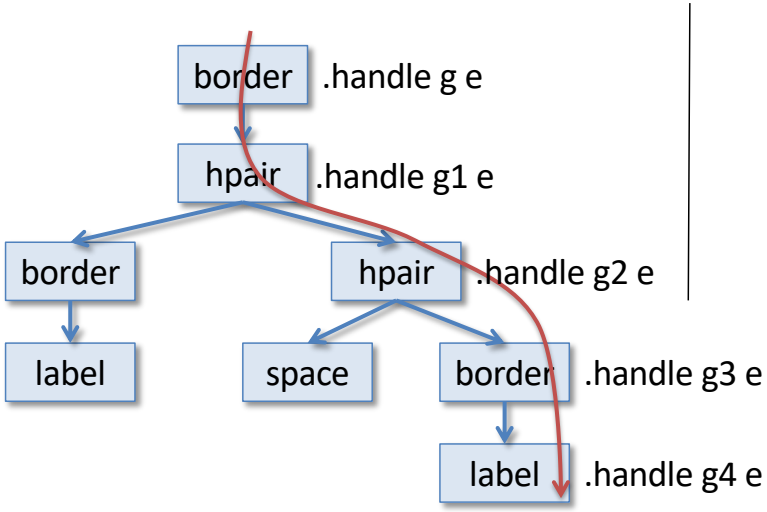
Review: Events and Event Handling

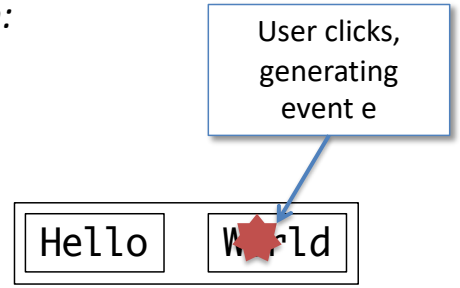
Project Architecture



Event-handling: Containers

Container widgets propagate events to their children:





Widget tree

```
g1 = Gctx.translate g (2,2)
g2 = Gctx.translate g1 (hello_width,0)
g3 = Gctx.translate g2 (space_width,0)
g4 = Gctx.translate g3 (2,2)
```

On the screen

Event Listeners

See notifierdemo.ml

(distributed with the lecture demos in Codio)

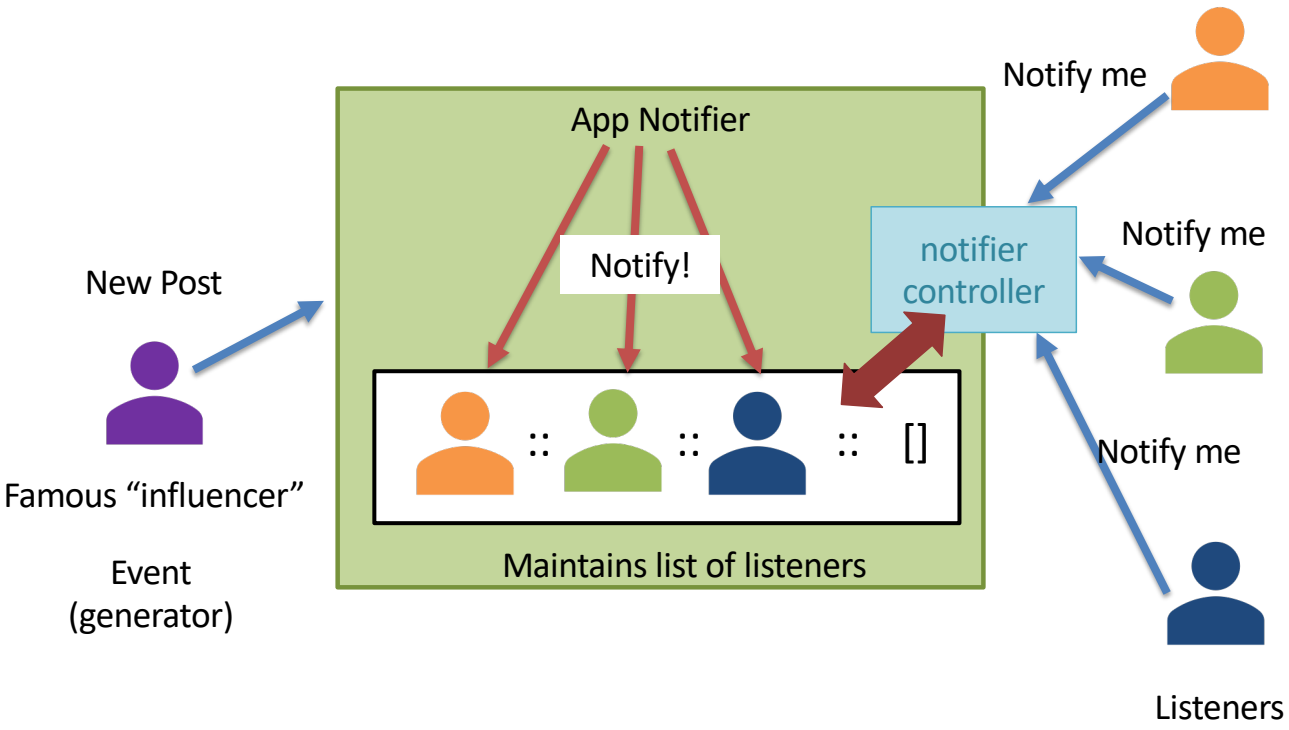
Handling multiple event types

- Problem: *Widgets may want to react to many different events*
- Example: Button
  - mouseclick: activates the button, primary reaction
  - mouse movement: tooltip?
  - key press: keyboard access to the button functionality?
- These reactions should be independent
  - Each event handled by a different *event listener* (i.e. first-class function)
  - Widgets may have *several* listeners to handle a triggered event
  - Listeners react in sequence; all are notified about the event
- Many different kinds of widgets react to events
  - Don't want to repeat the code for buttons in other widgets in the library
- Solution: notifier!

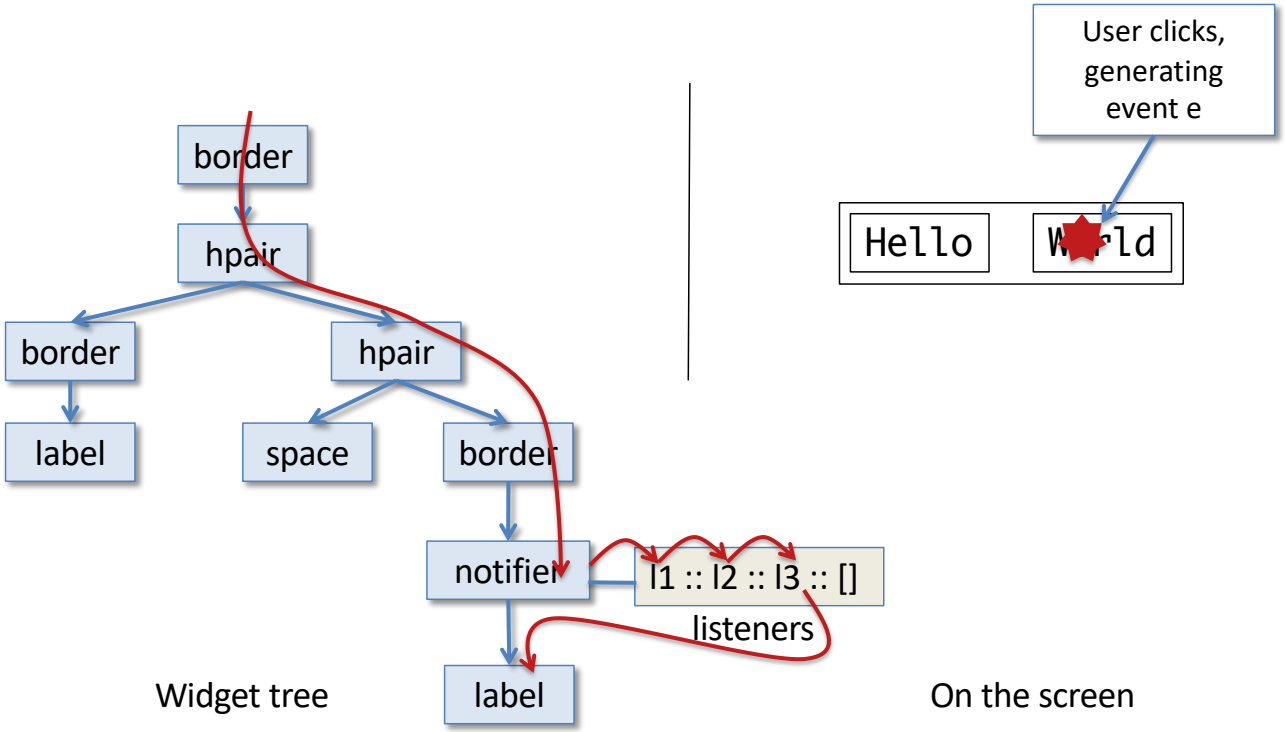
Analogy: Handling multiple event types

- Problem: Imagine a photo/video sharing app where you want to react to when your friend shares a new post
- Option 1 – Manual (Terrible idea!)
  - Keep refreshing the page every minute to see if there's new content
- Option 2 – Push Notifications
  - You can sign up to be *notified* when there is new content
  - Other people can sign up for the same notification too
  - If there is new content, you might "react" in a different way depending on the content – if it's a picture, you want to reshare it; if it's a video, you want to comment on it; ...
  - Your (and other people's) reactions should be independent!

Analogy: Listeners and Notifiers Pictorially



Listeners and Notifiers Pictorially



Notifiers

- A *notifier* is a container widget that adds event listeners to a node in the widget hierarchy

```
val notifier : widget -> widget * notifier_controller
```

  - Note: this way of structuring event listeners is based on Java's Swing Library (we use Swing terminology).

- *Event listeners* "eavesdrop" on the events flowing through the notifier

```
type event_listener = Gctx.gctx -> Gctx.event -> unit
```

  - The event listeners are stored in a list
  - They react in order
  - Then the event is passed down to the contained widget

- Event listeners can be added by using a `notifier_controller`

```
type notifier_controller = { add_listener : event_listener -> unit }
```

Listeners

widget.ml

```
type event_listener = Gctx.gctx -> Gctx.event -> unit

(* Performs an action upon receiving a mouse click. *)
let mouseclick_listener (action: unit -> unit) : event_listener =
  fun (g:Gctx.gctx) (e: Gctx.event) ->
    if Gctx.event_type e = Gctx.MouseDown
    then action ()
```

Note: the type `event_listener` *is* the type of the `handle` method from the `widget` type.

widget.mli

```
type widget = {
  repaint : Gctx.gctx -> unit;
  size    : unit -> Gctx.dimension;
  handle  : Gctx.gctx -> Gctx.event -> unit
}
```

Notifiers and Notifier Controllers

widget.ml

```
type notifier_controller =
     { add_listener : event_listener -> unit }

let notifier (w: widget) : widget * notifier_controller =
  let listeners = { contents = [] } in
  { repaint = w.repaint;
    size    = w.size
    handle  =
      (fun (g: Gctx.gctx) (e: Gctx.event) ->
          List.iter (fun h -> h g e) listeners.contents;
          w.handle g e);
  },
  { add_event_listener =
      fun (newl: event_listener) ->
          listeners.contents <-
                 newl :: listeners.contents
  }
```

Loop through the list of listeners, allowing each one to process the event. Then pass the event to the child.

The notifier_controller allows new listeners to be added to the list.

Buttons (at last!)

widget.ml

```
(* A text button *)
let button (s: string) : widget
                       * label_controller
                       * notifier_controller =
  let (w, lc)  = label s in
  let (w', nc) = notifier w in
    (w', lc, nc)
```

- A button widget is just a label wrapped in a notifier
- Add a mouseclick_listener to the button using the notifier_controller
- (For aesthetic purposes, we could also put a border around the label widget.)

onoff.ml — changing state on a button click

DEMO: ONOFF

Event Handling Summary

- An *event* is a signal: a mouse click or release, mouse motion, or keypress
  - Events carry data, such as e.g., state of the mouse button, the coordinates of the mouse, the key pressed
- An event can be *handled* by a widget
  - The top-level loop waits for an event and then gives it to the root widget, the widgets forward the event down the tree
  - e.g., the container widgets propagate a mouse click event to the button that should handles it
- Typically, the widget that handles an event *updates some state* of the GUI
  - e.g., to record whether the light is on or the label of the button
  - state is usual updated via a *controller,* e.g., a label_controller
- A *listener* associates an action with a particular type of event
  - e.g., a mouseclick_listener does something on a mouse click
  - listeners are triggered when a *notifier* widget handles an event
- User sees the reaction to the event when the GUI repaints itself
  - e.g., button has new label, canvas is a new color