ESS 201 Programming II
Java Lab 4
Due: 21 Nov 2021

GUI with Mouse events

Most Graphical User Interface (GUI) frameworks are designed as a class hierarchy of graphical elements such as panels, buttons, text boxes, labels, whose appearance and behaviour (on mouse or keyboard events) can be customized by the application developer to create the effects needed for an application. We will try to model the functionality of a simple GUI.

We have the following classes to model the UI elements on a typical screen::

- Widget:
    - This is the base class of all UI elements described below. All widgets are rectangular, and have a size (width and height).
    - Their position (position of the lower left corner of the widget) can be set using the method: public void setPos(int x, int y). This positions the widget within the Panel - the main (or root) widget described later.
    - Widget also has other properties such as colour and transparency, which we will not need for this exercise.
- TextBox:
    - Derived class of Widget. Gets ready when the mouse enters the text box region. After that, if any keyboard events are sent to it, the text box displays the text that was typed in. Stops reacting to keyboard input once the mouse moves outside the box. For simplicity, we will not worry about editing commands like delete/backspace etc. Clicking within the TextBox has no effect.
- Label:
    - Derived class of Widget. Displays a static string. No response to mouse or keyboard events
- Button:
    - Derived class of Widget. If the mouse is clicked inside the box, it executes a method associated with this mouse event.
- Panel:
    - Derived class of Widget. This is a special widget - the “root” widget that contains all other widgets. Every program creates exactly one Panel instance. Widgets are positioned within the Panel by specifying their position of the lower-left corner of the widget (using setPos) in the pixel coordinates of the Panel. Position (0,0) corresponds to the lower left corner of the Panel.

Mouse events (move, click etc) are processed by a MouseTracker, which then sends the events to all objects that are interested in mouse events. Similarly, keyboard events are processed by a KeyBoardTracker, which sends keyboard events to all objects that are interested in such events. Each program has exactly one each of MouseTracker and KeyboardTracker.
A class that wants to receive and process mouse or keyboard events, should implement either or both of the following interfaces:

1. **public interface MouseWatcher**, which contains the following methods:
    a. void moveTo(int x, int y) - called when the mouse moves to a new position x,y within the main Panel. Called for each position of the mouse when within the Panel. Only the end position of a mouse movement is reported.
    b. void onClick() - called when the mouse is clicked within the bounds of the Panel. The position of the mouse would be as in the last moveTo() call)
2. **public interface KeyboardWatcher,** which contains:
    a. void onKbdEvent(character x) - called when any character x is typed

A widget (or any other class) that wants either the mouse or keyboard events should be registered with the respective Tracker using the following mechanism:

The MouseTracker contains a method:

- public void addMouseWatcher(MouseWatcher m) { … }

The KeyboardTracker contains a method:

- public void addKeyboardWatcher(KeyboardWatcher w) { … }

Note that a class instance (such as a derived type of widget) can be registered for both types of events, assuming it has implemented the corresponding interfaces, MouseWatcher and KeyboardWatcher. Also note that objects interested in processing mouse or keyboard events should keep track of the position of the mouse so they can decide how they should react to a particular event.

The MouseTracker contains a method (invoked by main) to inform it of a mouse event:

```
public void processEvent(int x, int y, boolean isClicked)
```

where x,y are the coordinates of the mouse position (in Panel coordinates), and isClicked is true if the mouse button is currently clicked/pressed

The KeyboardTracker contains a method (invoked by main) which is called each time a key is pressed:

```
public void processEvent(char x)
```

When the MouseTracker receives a mouse event (through processEvent), it sends that event to all MouseWatcher objects that have been registered with it, by invoking their moveTo or onClick methods. Similarly, the KeyboardTracker receives keyboard events (through processEvent), and sends that to all KeyboardWatchers registered with it by invoking their onKbdEvent method.

Thus, widget sub-types such as Button, TextBox, Label should implement the appropriate Watcher interface and be registered with the appropriate Tracker.

For our example, we have the following:

1. A root Panel, of width 600 and height 800. This contains the following at the specified locations:
    a. A Label of width 200 and height 100 positioned at (50, 500)
    b. A TextBox of width 400 and height 200, positioned at (100, 300)
    c. A Button of width 200 and height 100, positioned at (250, 100)
2. When the textbox receives keyboard inputs (and when the mouse is within the bounds of the textbox), it prints out the complete string typed in so far - for each character input
3. When the button receives a click mouse event within its bounds, it prints out the message: “Selected point: x y”, where x and y are the coordinates of the mouse *within the button widget*.

The main program does the following:

1. Instantiates the root panel, and the MouseTracker and KeyboardTracker
2. Creates each of the widgets defined above, sets their position, and registers them with the appropriate Tracker.
3. Reads the standard input, and for each line of input, invokes the appropriate processEvent method of MouseTracker or KeyboardTracker to process the event. Format of the input is defined below:
    a. The first word is one of MoveTo, MouseClick, KeyPressed or End
    b. if MoveTo, the next two integers are the position (in pixel coordinates of the root panel)
    c. if MouseClick, there is no other data. The clicked point is the same as the end point of the last MoveTo
    d. if KeyPressed, the next character is the character that was typed
    e. if End, that signals the end of input

    Assume the mouse starts at 0,0

Sample Input

```
MoveTo 400 400
MouseClick
KeyPressed H
KeyPressed i
KeyPressed T
KeyPressed h
KeyPressed e
KeyPressed r
MoveTo 50 20
KeyPressed e
MouseClick
MoveTo 350 150
MouseClick
KeyPressed N
MoveTo 150 550
MouseClick
KeyPressed M
MoveTo 250 350
KeyPressed e
End
```

Expected output

```
H
Hi
HiT
HiTh
HiThe
HiTher
Selected point: 100 50
HiThere
```