Lecture 17

Chapter 15 Event-driven Programming

15.3 The `Event` Class

15.3 The Event Class

Event Handlers (`EventHandler` class in our graphics module) handle each individual type of event when triggered.

Recall that the signature of the `handler` method had two parameters: `self`, `event`

```
def handle(self,event)
```

`event` parameter is an instance of `Event` class.

Let's talk more about the Event class.

15.3 The Event Class - methods

`getTrigger(self)`

returns a <u>reference</u> to the object that triggered the event (a canvas or a drawable object)

`getDescription(self)`

returns a <u>text description</u> of the event ('mouse click', 'mouse release', 'mouse drag', 'keyboard', 'timer')

`getMouseLocation(self)`

returns a <u>Point</u> designating the location of the mouse at the time of the event

`getOldMouseLocation(self)`

returns a <u>Point</u> designating the location of the mouse at the <u>start of a mouse drag</u>.

`getKey(self)`

returns a <u>string</u> designating the key pressed for a keyboard event

15.3 The Event Class

Mouse Events – Example 1:

Let's define a handler that will be displaying the type of an event along with the mouse location at that moment (*p. 519, Practice 15.1*):

15.3 The Event Class

Mouse Events – Example 1:

mouse_events1.py sketch:

```
class M_EventHandler(EventHandler):
```
identifies a type of mouse event and notifies the user

```
class ExitButtonHandler(EventHandler):
```
for program termination

```
def main():
    paper=Canvas(700,600, ...)

    text1 = ...
```
first part of the message (no changes)

```
    text2 = ...
```
second part of the message - changes according to the type of a mouse event

```
    mouseEvent = M_EventHandler(text2)
```
create the handler for identification of mouse event

```
    ExitButton = Button(...)
```
create exit button

```
    exitEvent = ExitButtonHandler(paper,text2)
```
create the handler for program termination

```
    paper.addHandler(mouseEvent)
    ExitButton.addHandler(exitEvent)
```
activating the handlers

15.3 The Event Class

Mouse Events – Example 1:

mouse_events1.py sketch:

```
class M_EventHandler(EventHandler):

class ExitButtonHandler(EventHandler):

def main():
    paper=Canvas(700,600, ...)

    text1 = ...
    text2 = ...

    mouseEvent = M_EventHandler(text2)

    ExitButton = Button(...)

    exitEvent = ExitButtonHandler(paper,text2)

    paper.addHandler(mouseEvent)
    ExitButton.addHandler(exitEvent)
```

creation: text to be changed as argument

mouse clicking is "attached" to entire Canvas instance; termination of the program is "attached" to the Exit Button

15.3 The Event Class

Mouse Events – Example 1:

Note that:

1. The order in which handlers created and activated does not affect event handling

2. sys.exit([arg])
Program termination
Implemented by raising the SystemExit exception.
Zero as argument is considered “successful termination”.
See Python's documentation for more:
    27.1. sys — System-specific parameters and functions

See programs:
  mouse_events1.py (no coordinates), and
  mouse_events2.py (with coordinates)

15.3 The Event Class

Mouse Events – Example 2:

Note that these are the only changes in program mouse_events2.py:

in the handle method/function of M_EventHandler class:

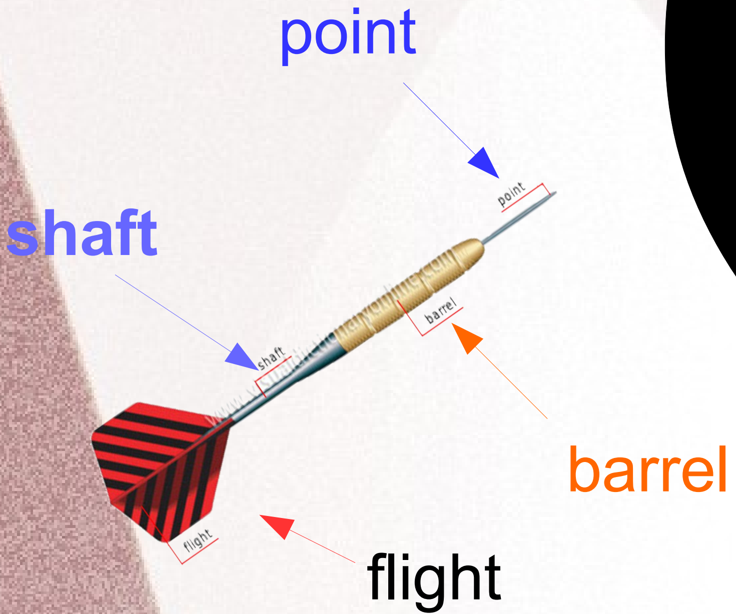
```
def handle(self,event):
    p=event.getMouseLocation() # get location of Mouse - point
    if event.getDescription() == 'mouse click':
        self._text.setMessage('mouse click at ' + str(p))
    elif event.getDescription() == 'mouse release':
        self._text.setMessage('mouse release at ' + str(p))
    elif event.getDescription() == 'mouse drag':
        self._text.setMessage('mouse drag at ' + str(p))
    else:
        self._text.setMessage('not a mouse event')
```

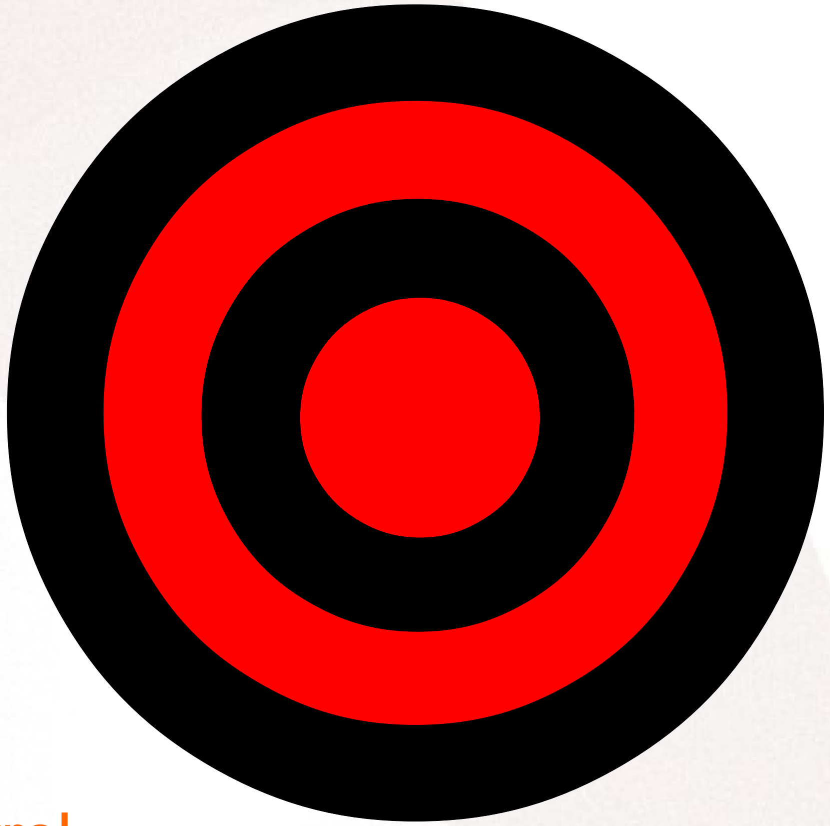
In the main function: `text1=Text('Event:',18,Point(50,550))`

Dartboard Example

Mouse Events – Example 3:

Let's write a program for a dartboard.





Example 3: Dartboard

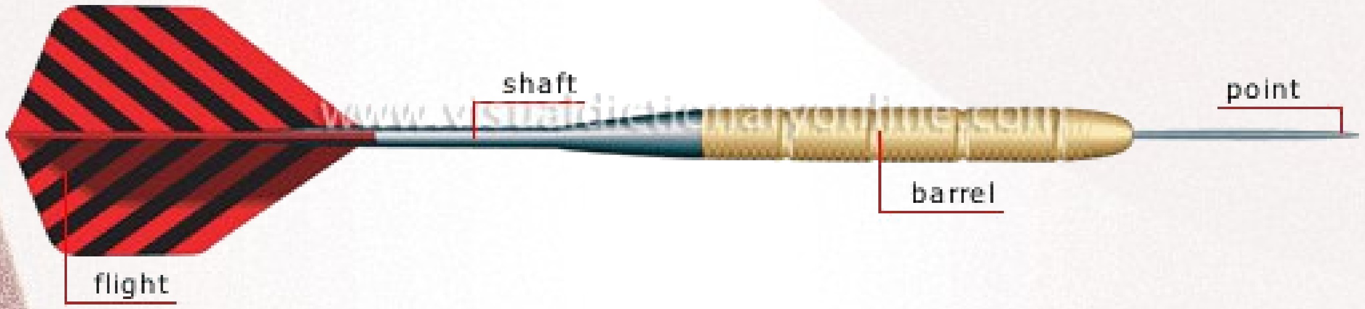
Dart
arrow (or point) shaft flight flightline (or barrel)
_draw()

DartsHandler: EventHandler

__init__(paper)
handle(event)

ExitButtonHandler: EventHandler

__init__(paper, textObj)
handle(event)



Example 3: Dartboard

Separately we'll define method throwDart(paper,point) that will be drawing dart's flight from the bottom left corner to the Point. Darts' handle method will be invoking it.

Example 3: Dartboard

Here is the sketch of the main method:

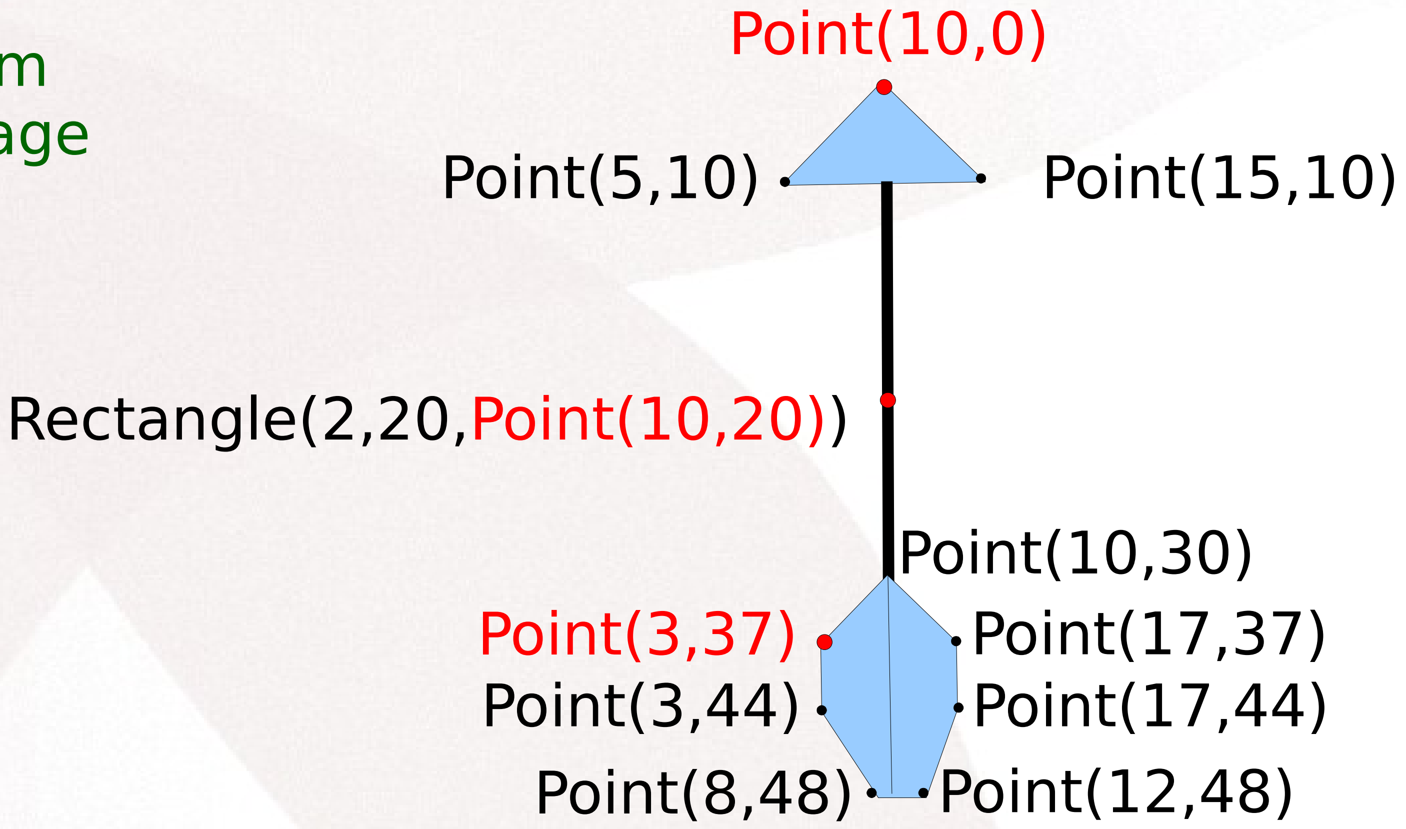
```
def main():

 paper=Canvas(700,600,'light yellow','Dart Board')

 draw Dart Board

 text=Text('Click on the DB to send a dart')
 paper.add(text)

 draw Exit button

 # throwing darts
 darts=DartsHandler(paper) # creating d. handler
 exitB=ExitButtonHandler(text) # and Exit handler

 paper.addHandler(darts) # activating d. handler
 exitButton.addHandler(exitB) # and Exit  handler
```

See program dartboard.py

Example 3: Dartboard

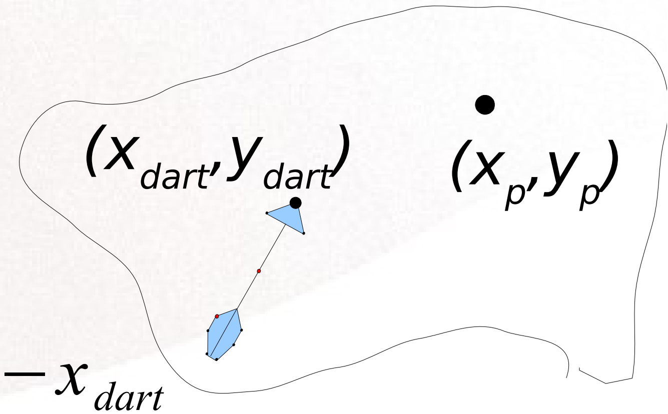
Math in the program
(if not using an image
with a dart):

1. Dart drawing



```
self._arrow=Polygon(Point(10,0),Point(15,10),Point(5,10))
self._shaft=Rectangle(2,20,Point(10,20))
self._flight=Polygon(Point(3,37),Point(10,30),Point(17,37),
Point(17,44), Point(12,48), Point(8,48), Point(3,44))
self._flightLine=Path(Point(10,30),Point(10,48))
```

Example 3: Dartboard



Math in the program:

2. Arrow Flight: at each of 100 iterations,
step in *x*-coordinate is calculated (*run*): $\frac{x_p - x_{dart}}{100-i}$

stepX=(point.getX()-d.getReferencePoint().getX())/float(100-i)

step in *y*-coordinate is calculated (*rise*): $\frac{y_p - y_{dart}}{100-i}$

stepY=(point.getY()-d.getReferencePoint().getY())/float(100-i)

then we move the arrow stepX pixels right, and stepY pixels up
d.move(stepX,stepY)

also, at each fifth iteration (I.e. cases when i is divisible by 5) we rotate our arrow 1 degree clockwise:
d.rotate(1)

See program dartboard.py

Keyboard Events

When a user presses a key on the keyboard, this triggers a keyboard event upon whatever object currently has the “focus”. This type of event is reported as 'keyboard' by getDescription().

If needed, the getMouseLocation() is supported for a keyboard event.

getKey() method returns the single character that was typed on the keyboard to trigger the event.

Example: program that echoes characters in the graphics window until the user clicks a mouse. If the mouse is clicked, everything is erased, and the user can type in a new sentence: echo-keyboard.py

Monitor class

Supports two methods:

`wait()`

when called, control of that flow will not be returned until the monitor is somehow released, presumably by some event handler.

`release()`

releases a monitor

This class can be thought of as “monitoring” some condition and alerting us once that condition is met.

Recall clicks.py and other programs with cue=paper.wait()
- we waited for an event to happen (e.g. a mouse click)

Monitor class

Let's consider another example:

Let's write a program with too shapes (circle and rectangle) and when the user clicks on one of them the graphics window changes its background color.

See program monitor-example.py

Suggested HW assignment

write a program that:
- draws three distinct objects (say a circle, a triangle and a rectangle), then
- user can drag the figure around the screen
- user can scale the figure (re-size)
- user can change the fill color of the figure
- user can remove the figure from the paper

The last three items can be done with keyboard, for example:
if 'm' key is pressed the figure is re-sized with a ratio of 2 (magnified),
if 'z' button is pressed, the figure is re-sized with a ratio of 0.5 (zoomed out),
if 'c' button is pressed, the fill color of the figure is changed,
if 'd' button is pressed, the figure is removed from the paper

All the events should be commented by a text message.