Practical session 1, Week 5: Line, Rectangle and Oval Drawing

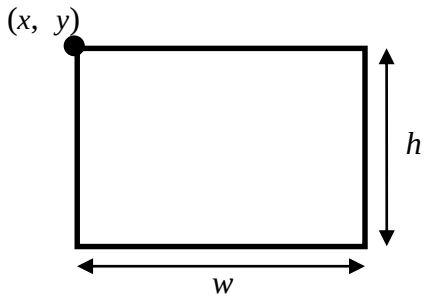
In this practical you will implement the line, rectangle and oval drawing tools in your drawing application.

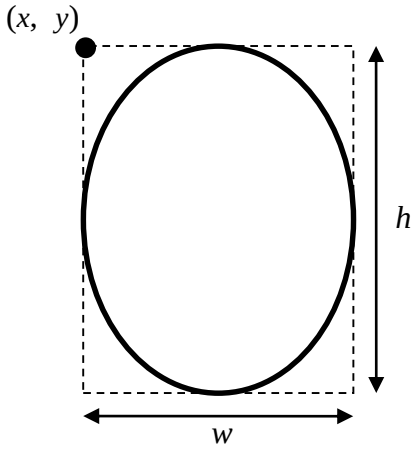
Task 1

From previous practicals you are already familiar filled rectangle drawing (i.e. freehand 'pixels'). Practice drawing unfilled rectangles and ovals (of various colours) on the canvas using the following `Graphics` class methods:

```
...
g.drawRect(x, y, w, h);
g.drawOval(x, y, w, h);
...
```

where *x* and *y* define the coordinates of the top left corner of the drawable object and *w* and *h* define the width and height (respectively) of the object. Note that in the case of an oval, *x* and *y* actually define the top-left corner of the oval's enclosing rectangle:





It is important to realise that the width or the height of the above objects cannot be negative. Verify that a rectangle/oval will not be drawn if **either** *w* or *h* is less than 0. This fact will become important when the user attempts to draw one of the above shapes with the mouse.

Compare the above with line drawing. What are the differences between line drawing and rectangle drawing for example?

Task 2

Recall that in the last practical you defined a *constant* to specify the maximum number of freehand pixels (`MAX_FREEHAND_PIXELS`) and a variable to keep count of the number of pixels being drawn (`freehandPixelsCount`). You have also declared one array to store

the colour of each freehand pixel (`freehandColour`) and another array in order to store the pixel's coordinates and size (`fxy`).

Here, you will need to do the same for each of the remaining three drawing tools. The only small difference lies with the array that will hold the location/size of each drawable (line, rectangle and oval). How big should the size of the second dimension of this array be in each case?

Initialise the maximum number of each of the three drawables to 10 for this practical.

Task 3

Also recall from the last practical that you added some code in the `draw` method in order to draw the freehand traces on the `canvas` object. In a similar manner you need to add code that will draw the lines, rectangles and ovals in their corresponding colour as specified by the contents of the arrays above.

Test your code by manually initialising the arrays with some data.

Task 4

The process by which you will be drawing lines, rectangles and ovals using the mouse is casually called "rubber-banding". This is because the drawables behave like rubber bands while they are in the process of being drawn (i.e. they extend and contract as the mouse is being dragged).

In order to achieve rubber-banding we need to respond to three types of events:

- **mouse button press** (for starting a drawable),
- **mouse drag** (for extending/sizing the drawable – rubber-banding) and
- **mouse button release** (for finishing the drawable).

Recall that the mouse button press and mouse button release events are serviced by the `mousePressed` and `mouseReleased` methods of the inner class `CanvasMouseListener` that implements the `MouseListener` interface.

Also recall that the mouse drag event is serviced by the `mouseDragged` method of the inner class `CanvasMouseMotionListener` that implements the `MouseMotionListener` interface.

An instance of each of the above mentioned classes should already be associated with the `canvas` object so all you need to do is append/implement the code in the methods `mousePressed`, `mouseDragged` and `mouseReleased`.

Try completely implementing one drawing tool at a time. It is recommended that you start with the line drawing tool.

Here are a few pointers to help you:

When the mouse button is *pressed* in the canvas area you need to start drawing a new drawable object. Try to answer the following questions:

- How can you find out which object needs to be drawn?
- Is there enough space remaining in the data structures you have declared to store the position, size and colour of the new object? If not, how can you inform the user about it?
- Where do you need to store the position, size and colour of the new object and where can you find this information?

As the mouse is being *dragged* on the canvas you need to implement rubber-banding for the drawable object. It is important to remember here that every time the mouse moves on the canvas the canvas is cleared and re-drawn. So ask yourself:

- Again, how can you find out which object is in the process of being drawn?
- Do you still need to check whether there is space to store it?
- Which specification of the drawable do you need to change/update with the new mouse location?

When the mouse button is finally *released* on the canvas you need to complete the drawable:

- Do you still need to know which drawable is being completed and whether there is space for it to be saved?
- In the case of rectangles and ovals, how can you ensure that a rectangle or an oval is not created/stored when their width or height becomes negative as a result of the mouse's position relative to its starting position?

Always test and verify your code as soon as you make a change that can be visible when you run and use your application. Having to troubleshoot small increments is far easier than larger pieces of code that are likely to contain more than one error.

Use the help available to you from demonstrators during the practical sessions. Also, if you have doubts, you are encouraged to discuss a possible solution with a demonstrator before you start implementing it.