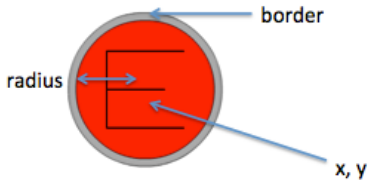
Project 2: That was easy!

due 9:00am Wednesday, October 23, 2013.

Goal

The goal of this assignment is to practice using graphics, events, and states in order to create a usable interactive object. You'll also get practice in writing a program using starter code instead of creating something entirely from scratch.

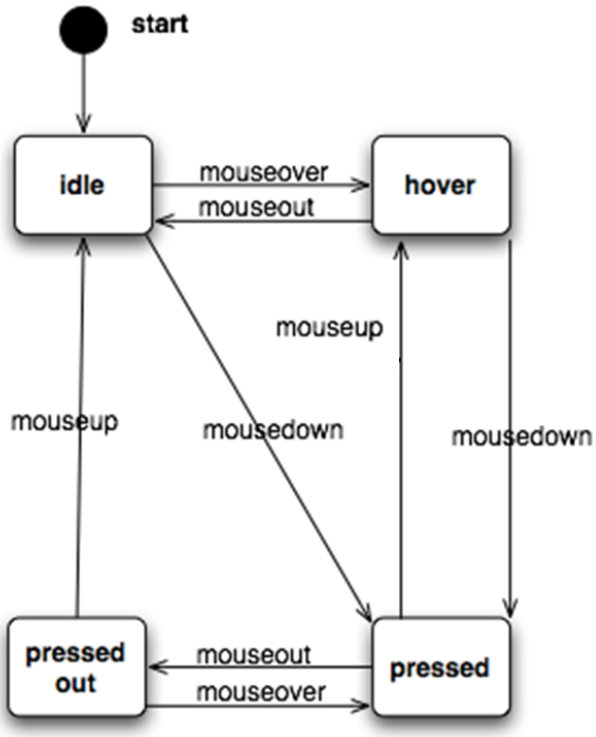


In Part A of this assignment, you'll be implementing the graphics of the Easy Button in order to match the image above. The button will use the **x**, **y**, **radius**, and **border** properties in order to define the shapes that make up the button. You will need to draw:

- The outer grey circle
- The inner red circle
- The letter E (composed of four lines)
    - Hint: in order to position the lines properly, try mapping out the circle onto a coordinate plan, and determine which quadrants the ends of each line fall into.

These graphics should not be drawn using static values- they should be based on the EasyButton's properties, so that it can be resized based on user input.

In Part B of this assignment, you will be modifying the **state** of the EasyButton based on the **events** that happen in the application. You'll need to implement the canonical button state diagram covered in class (shown below). However, since you're creating your own graphics, you will not be able to use the Flex-specific states implementation (defining states in MXML and modifying properties based on them). You'll need to use the provided starter code's states instead, and modify the appearance in draw().



The colors of the middle part of the button for each state should be:

- Idle: 0xFF0000
- Hover: 0xFF6666
- Pressed: 0x990000
- Pressed Out: 0x996666

Finally, most transitions should result in the state being changed appropriately. Note that you will need to call draw() at the end of each event handler to ensure that the appearance of the button updates appropriately! Also, when the button is clicked appropriately, an Alert with the message 'That was easy!' should pop up on the screen. You will need to determine which transition this alert should be associated with.

Remember that you'll need to use the Application to detect mouseup events from the Pressed Out state! This can be done in the appMouseUp function in Hw2a.mxml, provided in the starter code.

Finally, for bonus points, implement *application-level states* that do something interesting based on user events. These states should be implemented using State tags and includeIn/excludeIn, as was shown in class. You will need to implement at least three application states which have different appearances to get bonus credit. Get creative!

And, as always, style counts- make sure to read the feedback from Hw1b and write clean, readable code.