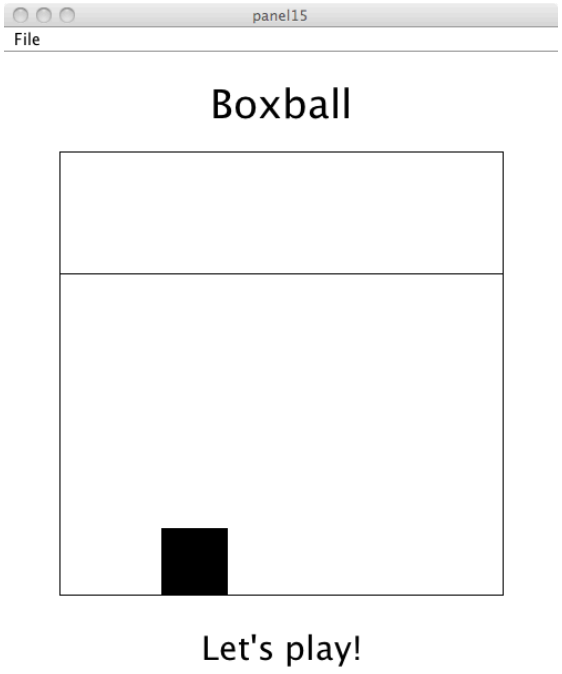
# Lab 5 – Boxball[1]

## CSCI 1101B – Spring 2015

## Due: March 31, 10 pm

**Objective:** To gain experience using loops and active objects.

For this lab, you will implement a game, called Boxball, in which the player attempts to drop a ball into a box. When the game begins, the playing area is displayed. There is a rectangular court with a horizontal "minimum-height" line dividing the court into an upper quarter and a lower three-quarters. A box moves back and forth automatically at the bottom of the court. The player tries to drop a ball into the box by clicking the mouse in the region of the playing area above the minimum height line, which creates a ball where the mouse was pressed that falls in a straight line. If the ball falls completely within the box, the message changes to say "You got it!" and, in one second, changes to "Try again!" The player may now try to get another ball into the box. **Note that you need *not* prevent the player from dropping another ball until the current falling ball has reached the bottom. For example, the user could click four times in quick succession and there would be four balls falling at the same time.** The boxball playing area should start out with the following appearance:




[1]Based on a lab provided with *Java: An Eventful Approach*, K. Bruce, A. Danyluk, and T. Murtagh

## Designing Your Program

You will write your program in three classes that allow you to create the major components of the game: the playing area, the box, and the ball.

- The `Events` class extends the `FrameWindowController` class and should be responsible for drawing the playing area when the program starts. It should also handle the player's mouse presses. If the player clicks within the playing area above the minimum-height line, a new ball should be created and dropped from that height.
- The `Box` class extends the `ActiveObject` class and should allow you to create the moving box at the bottom of the playing area. The box should move at the rate of 2 pixels per 20 milliseconds, but these should, of course, be named constants in your `Box` class and, thus, easily changed. This class will also need to have methods that return information about the box. These methods are indicated in the program skeleton provided.
- The `Ball` class also extends the `ActiveObject` class and should allow you to create a ball that falls at a rate of 2 pixels per 10 milliseconds. When the ball is completely hidden by the box, it should disappear and the player should be told whether the ball landed in the box. If the ball did not get in the box, the message should say "Try again!" If it did fall completely in the box, the message should say "You got it!" and then, in one second, change to "Try again!"

### Part 1: Setup

Start by setting up the playing area, using the familiar `objectdraw` shapes. The playing area should have both width and height of 400 (you will need to resize the main window in your `begin` method to accommodate the playing area using the `resize` method). Put the code for building the playing area in the `Events.java` file. The minimum-height line should be $\frac{1}{4}$ of the distance from the top of the court to the bottom. Note that, except for the named constants controlling the animation, which should be in the `Box` and `Ball` classes, all the named constants governing the appearance and dimensions of the court, texts, box, and balls should be in the `Events` class.

### Part 2: Adding the Box

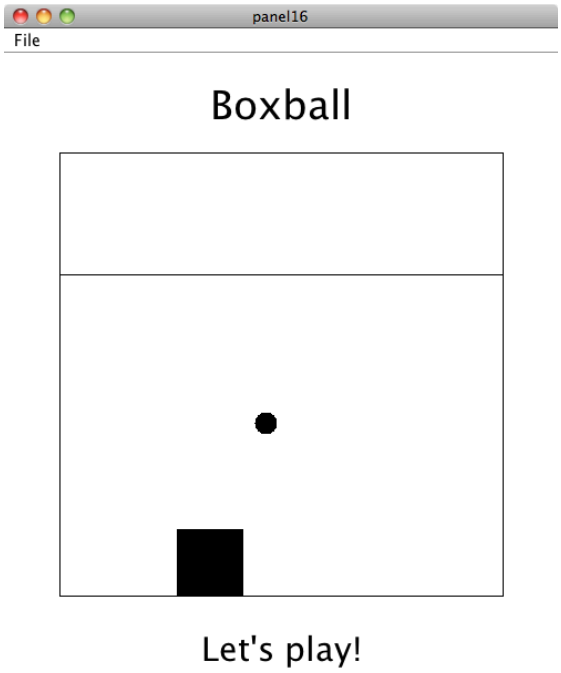
Next, you should add a box to your layout. To do this, you will need to write a `Box` class that extends the `ActiveObject` class and that will allow you to create and animate the box object at the bottom of the playing area.

A box is really just a square, but there is some important information you will need to pass to the `Box` constructor in order to construct it properly, i.e. how big a side of the box should be (it should be three times the width of a ball), the y-coordinate of the top of the box, and the x-coordinates of the edges of the playing court. This information will be passed in to the `Box` constructor through parameters. You should also write a `run` method that makes the box move back and forth across the court, from edge to edge, forever, at

the rate of 2 pixels per 20 milliseconds. There are also methods indicated in the `Box` class skeleton that you will need to call from the `Ball` class to calculate when the ball reaches the bottom of the court and whether it landed completely in the box. These are indicated in the program skeleton.

### Part 3: Dropping a Ball

The `Ball` class will extend the `ActiveObject` class. The player will create a ball by clicking with the mouse in the playing area above the minimum-height line. When the click is detected, a new ball should be constructed at the appropriate location on the screen and start moving down the screen. Here is a ball in the process of moving down the court:




You need to think about what information a ball needs to know to construct itself properly. Recall that the `Events` class knows how big the ball should be (so that the ball and the box can be sized appropriately). The `Events` class also knows where the mouse was pressed. This should be the starting location for the ball. You need to pass this information to the `Ball` constructor so that it can draw a ball of the right size in the right location (centered where the user pressed the mouse) and start it falling.

The ball should stop moving after it is completely hidden by the box and should then disappear. In order to determine this, the ball will need to know the y-coordinate of the top of the box. Also, you will need to compare the ball's location to the box's location to determine whether the call landed in the box. To do this, the ball will need to be able to find out the x-coordinates of the box's left and right edges. The `Box` class needs to provide

methods `getY`, `getLeft`, and `getRight` that give this information. To call those methods, the `Ball` class must know about the box. Go back and modify your `Ball` constructor to pass in the `Box` as an additional parameter.

If the ball did not get in the box, the message should say "Try again!" If it did fall completely in the box, the message should say "You got it!" and then, **in one second**, change to "Try again!". Since the `Events` controller is responsible for the layout of the game, it should construct a `Text` object that displays a greeting message. The `Text` object should be passed to the `Ball` constructor as a parameter so that the ball can change the message appropriately when it hits or misses the box.

Once you have written the `Ball` constructor and the `run` method, you should test them. Return to your `Events` controller class, and add code to the `onMousePress` method that will construct a ball if the player clicks above the starting line in the playing area.

## Optional Extra Credit

Modify the game so that it displays the current "streak" count – i.e., the number of successive attempts where the ball made it into the box without any misses. To do this, change the `Text` object displaying either "Try again!" or "You got it!" so that it also displays the current streak count. Initially, your program should show "Let's play! Current streak: 0". If you then create three balls and all make it into the box, your program should show "You got it! Current streak: 3" (which will change in one second to "Try again! Current streak: 3", assuming that no further points have been scored in the interim). If any balls miss the box, the streak counter should reset to 0.

To make this modification, you will need to keep track of the streak counter somewhere. Unfortunately, you cannot do this in the `Ball` class, since each `Ball` object is separate and does not know about any other balls (which may be making it into the box, or miss the box, etc). Thus, you should modify your `Box` class (since there is only one actual `Box` object) by adding a method to indicate that an attempt was made (and either hit the box or missed), and a second method to get the current streak counter. These methods can then be used by the `Ball` class to update the text object appropriately.

### Lab Logistics

The lab starter files on the course webpage contains a skeleton project for the classes you need to write. You should not create any new class files. Before submitting your work, make sure that you have replaced all of the comments in the supplied class skeletons with the appropriate information. Please submit your entire, compressed project folder in the usual way via Blackboard (remember to name the file using your login ID and lab number).