Programming Exercise:
Boxball

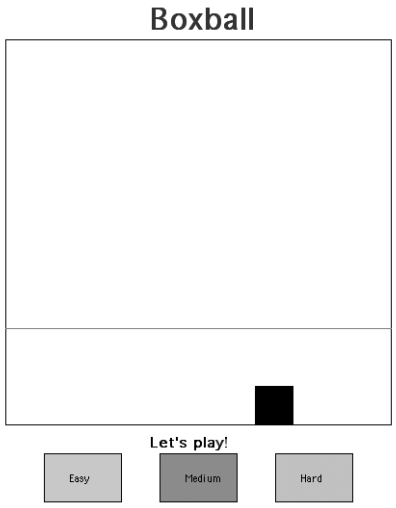
Objective: To gain experience defining constructor and method parameters and using active objects.

The Scenario.

For this lab, we would like you to implement a game called Boxball. This is a simple game in which the player attempts to drop a ball into a box. Boxball has 3 levels of difficulty. With each increasing level, the box gets smaller, and the player is required to drop the ball from a greater height.

When the game begins, the playing area is displayed along with three buttons that allow the player to select a level of difficulty. Selecting a level of difficulty affects the size of the box and the position of the line indicating the minimum height from which the ball must be dropped. The player drops a ball by clicking the mouse in the region of the playing area above the minimum height line. If the ball falls completely within the box, the box is moved to a random location. Otherwise, it remains where it is. In either case, the player may go again, dropping another ball.

The boxball playing area should appear as follows:



The online version of this handout has a demo version of BoxBall.

Design of the program.

For this lab you will need to define three classes: a `Box` class, a `Ball` class, and a `Boxball` class that extends `WindowController`. These will allow you to create the major components of the game, i.e., the box, the balls, and the playing area, respectively.

The `Boxball` class should be responsible for drawing the playing area when the program starts. It should also handle the player's mouse clicks. If the player clicks on an easy, medium, or hard button, the starting line should move to the appropriate height. If the player clicks within the playing area above the starting line, a new ball should be created and dropped from that height.

The `Box` class will allow you to create and manipulate the box at the bottom of the playing area. A box is responsible for knowing how to move to a new random location when the ball lands in the box. It also is responsible for changing size if the player changes the difficulty level.

The `Ball` class should allow you to create a ball that falls at a constant rate. When the ball reaches the bottom of the playing area, the player should be told whether the ball landed in the box. The ball should then disappear. If the ball lands in the box, the box should move to a new location.

Part 1: Setting up Start by setting the layout of your playing area, using the familiar ObjectDraw shapes. Our playing area has both width and height of 400. The code for building the playing area should go in `Boxball.java`.

Once you have set up the playing area, add the Easy, Medium, and Hard buttons to your layout. The buttons are just rectangles that will respond to mouse clicks. After you've displayed the three buttons, add code to the `onMouseClick` method that will adjust the level of the starting line, depending on the button clicked. If the player selects the "Easy" buttton, the line should be relatively low. If the player selects the "Hard" button, the line should be quite high.

Part 2: Adding the box Next, you should add a box to your layout. To do this, you will need to write the `Box` class that will allow you to create and display the box object at the bottom of the playing area.

A box is really just a rectangle, but there is some important information you will need to pass to the `Box` constructor in order to construct it properly. First, you need to tell the box where it should appear on the display. Remember that its horizontal position will change over time, but its vertical position will always be the same. What are the extreme left and right values for its horizontal position?

In order for the rectangle to be drawn, you will also need to tell the box what canvas it should be drawn on. This information will be passed on to the constructor for the rectangle.

The box needs one more piece of information for it to be drawn correctly. It needs to know how wide it should be. Of course, even in its hard setting, the box should still be a little wider than the ball. Since the `Boxball` class will create both the ball and the box, it knows their relative sizes. It must therefore tell the box how big it should be when the box is created.

Once you have written the constructor for the `Box` class, you should go back to the `begin` method of the `Boxball` controller class and create a new `Box`.

The default setting for the game is "Easy". If the player clicks on the "Medium" or "Hard" button, the box should get smaller (or much smaller). The box needs a method, `setSize`, to allow its size to change when the player clicks on Easy, Medium, or Hard. Think carefully about what parameters you need to pass to `setSize` to accomplish this command.

After writing the `setSize` method, test it by modifying the `onMouseClick` method in the `Boxball` controller. Clicking on one of the level selection buttons should now not only raise or lower the bar, but it should also adjust the size of the box.

Part 3: Dropping a ball The `Ball` class is different from other classes with which we have been working. Objects of this class will be active.

The player will create a ball by clicking with the mouse in the playing area above the starting line. When the click is detected, a new `Ball` should be constructed. The constructor for the `Ball` class should draw a ball at the appropriate location on the screen. It should also start it moving down the screen. To do so, it will call a `start` method, which will cause the code in `run` to execute. Make sure that the `start` method call is the last statement in your constructor.

You need to think carefully about what information a ball needs to know to construct itself properly. Recall that `Boxball` knows how big the ball should be (so that the ball and the box can be sized appropriately). `Boxball` also knows where the mouse was clicked. This should be the starting location for the ball. You need to pass this information to the `Ball` constructor so that it can draw itself and fall.

We will use a bit of video game magic to make the ball appear to fall. The ball will appear to move smoothly down the screen, but in fact, it will be implemented by a series of movements. Each movement should move the ball a short distance, wait a short time, and then move again. The `run` method contains a while loop to allow this. On each iteration of the while loop, the ball should move a small distance, say 10 units. Then it should wait a short time. A pause of 50 milliseconds between moves should result in movement that appears to be continuous to the human eye. To complete the while statement, you need to provide the condition that determines when to exit the while loop. Specifically, the ball should stop moving when it reaches the bottom of the playing area.

Once you have written the `Ball` constructor and the `run` method, you should test them. Return to your `Boxball` controller class, and add code to the `onMouseClick` method that will construct a ball if the player clicks above the starting line in the playing area. At this point, do not worry about whether the ball falls in the box. Just check that it is drawn at the right starting location and that it makes its way to the bottom of the playing area.

Part 4: Checking the box Now you are ready to determine whether the ball fell in the box. As the ball reaches the bottom of the playing area, it should compare its location to the box's location. Of course, the ball will need to find out the box's location. The `Box` class needs to provide methods `getLeft` and `getRight` that give the positions of the edges of the box. 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 lands in the box, you should display the message "You got it in!". If the ball misses, you should display "Try again!". Since the `Boxball` 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.

Test the additions that take care of checking whether the ball fell in the box.

Finally, use the random number generator (i.e., `RandomIntGenerator`) to pick a new location for the box when the player gets the ball in the box. The box should be responsible for picking the new location and moving itself. Therefore, you will need to add a method to the `Box` class called `moveBox`.

When the box is narrow its left edge can have a relatively large value while still having the whole box in the playing area. However, when the box is wide, it cannot move quite as far without having its right edge going outside the playing area. You may set up the random number generator to only give values that will result in the widest box staying inside the playing area. For extra credit, you can further refine the program so that even the narrower boxes can end up all the way on the right of the playing area.

Submitting Your Work

Before submitting your work, make sure that each of the .java files includes a comment containing your name. Also, before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations.