CSC-14400 Programming Project #1
DUE: Monday, October 30, 2017, 11:59 PM

1 Objectives

- writing a Greenfoot application on your own
- working in an object-oriented environment
- designing and coding your own methods
- utilizing selective execution

2 Problem Statement

You will be completing two Greenfoot (sub)classes (`MyWorld` and `GolfBall`) as well as writing one of your own classes (`Hole`). You are also being provided with three completed classes (`MessageBox`, `RectangleMeter`, and `Selection`). Together, these classes will enable you to play a single hole of golf (provided you implement the parts required).

In order to accomplish this, you are being provided with a (mostly) empty "shell" of a corresponding Greenfoot project. You are being provided with the completed classes listed above, which include documentation that you will likely find useful. Each class that you need to complete is described below; *you may (or may not) find it useful to add instance variables and/or additional public methods to each class.*

MyWorld (subclass of World)

This class should include:

1. a (no-argument, default) constructor that does the following:
   - makes the world 800x400 "boxes", with 1 box being 1 pixel wide and 1 pixel high.
   - adds a `MessageBox` centered at the top of the screen.
   - adds 5 `Selection` boxes underneath the `MessageBox` from the last step. Each of the `Selection` boxes should be horizontally aligned with each other. The boxes should be labeled `"1W"`, `"3W"`, `"1I"`, `"5I"`, and `"P"` (from left to right).
   - adds a (fully visible) `RectangleMeter` in the lower left corner of the screen.
   - adds a `Hole` on the bottom of the screen (making sure the entire `Hole` image is visible). The `Hole` should be randomly placed somewhere between 150 pixels from the left edge and 40 pixels from the right edge of the world. Each run of the scenario should likely give a different hole placement (as should each Greenfoot reset).
   - adds a `GolfBall` on the bottom of the screen, 30 pixels from the left edge of the world.
2. a method called `getMessageBox` that takes no arguments and returns the `MessageBox` object that was added by the constructor.

Remember, you may (more likely *will*) want to add additional instance variable(s) and method(s) to this class. Such includes potentially adding additional code to the constructor described above.

GolfBall (subclass of Actor)

This class should include:

1. a (no-argument, default) constructor that ensures that this `GolfBall` has an appropriate image associated with it. You may want to add other code here as well, depending on what you code elsewhere.

2. an `act` method that takes no arguments and returns nothing. This method should:

   - when the space bar has been pressed on the keyboard and this `GolfBall` is not already moving, does the following to "launch" this `GolfBall`:
     - asks the world what club was selected by the user (hint: check the `MyWorld` public interface). If no club was selected, then a message telling the user to select a club first should be printed in the `MessageBox` and nothing further should be done during this call to `act`.
     - based on the following table of club selections, determines the launch angle of the golf ball:

Club Name	Angle (in degrees)
1W	60
3W	65
1I	70
5I	80
P	1

     - Using the world's `RectangleMeter`, calculates the *power* with witch the golf ball is hit as:

$$power = ratio \cdot 10$$

       where *ratio* is the current ratio value represented by the `RectangleMeter` (see the public interface for that class). A message should be displayed (in the `MessageBox`) indicating how much power the golf ball was just hit with.
     - Converts the angle to radians (see `Math.toRadians()` in the Java API if you have no idea how to do this.), storing that result into Θ. The initial speeds in the x direction (v_x) and y direction (v_y) can then be calculated as follows:

$$v_x = power \cdot cos(\Theta)$$

$$v_y = power \cdot sin(\Theta)$$

       Noting that there are methods `Math.sin` and `Math.cos` in the Java API. Also, note that these calculations result in floating point values.
       However, there's one catch: if you've already hit the ball past the right side of the hole, you'll want to negate v_x before continuing (for example, if the ball is to the right of the hole and you calculated v_x to be 3.3, you should instead set v_x to be -3.3).
       At this point, it would be advisable to remember that you have "launched" this `GolfBall` so that future calls to act know that such has happened.

   - if this `GolfBall` has been "launched" (regardless of whether or not the space bar is being pressed at present) the following should happen:

- if this `GolfBall` is beyond the left or right edge of the world, its image should be replaced by a red arrow (again, see the public interface of the provided methods), making sure that this arrow is rotated to point toward the ball. A message indicating the ball is "Out of Bounds" should also appear in the `MessageBox`.
- if this `GolfBall` is now in bounds, make sure to set its image back to a golf ball. You also should display a message indicating that the ball is back in bounds.
- if the ball is in the air (i.e. its y value is not at the bottom of the screen), then the y speed of the ball (v_y) should have 0.1 subtracted from itself (noting that this represents the impact of gravity). Next, wind resistance should be taken into account by reducing both x and y speeds (v_x and v_y) by 0.5% (not 50%!!!!).
- if the ball is on the ground (i.e. its y value is at the bottom of the screen), friction from the ground should reduce its x speed (v_x) by 30%. The ball will, of course, bounce off of the ground; this should negate its y speed (v_y) and reduce that value by 60%.
- this `GolfBall`'s updated actual location (x_{actual}, y_{actual}) can now be calculated as follows:

$$x_{actual} = x_{actual} + v_x$$

$$y_{actual} = y_{actual} - v_y$$

  Some notes:
  - the resulting values will be floating point values
  - you will need to have initialized the location variables appropriately somewhere (0 is likely not a good value for either)
  - you will likely want to tell the Greenfoot system to update where to locate this `GolfBall`
  - a `GolfBall` cannot tunnel through the earth (so you'll want to make sure that y_{actual} never represents a value below the bottom of the world).
- If the ball slows down so that its speed is less than 0.1 and it is at the bottom of the world, it should stop moving and should wait for the user to press the space bar again. Speed can be calculated as:

$$speed = \sqrt{(v_x)^2 + (v_y)^2}$$

- If the ball is near the hole (i.e. it is touching the image for the hole), display a message like "Wow! That was close".
- If the ball is near the hole and is within 5 pixels of the center of the hole and has a speed of less than 0.1, then display a message like "It's in the hole!" and stop the ball from moving any further.

Remember, you may (more likely *will*) want to add additional instance variable(s) and method(s) to this class. Such includes potentially adding additional code to the constructor described above.

Hole (subclass of Actor)

This class really just needs to make sure an image is set for it. It is not required to do anything in its act method - although you are welcome to add code if you wish to (a fluttering flag can be kind of cool, but is certainly not required):

3 What To Hand In

You will be submitting a zip file containing **ALL** of your Greenfoot project to Canvas. Make sure that you zip up the project into a single folder (which may contain sub-folders). For those that do not know how to create a zip file, we will discuss such in class. *Note that not knowing how to use Canvas will not be considered a valid excuse for not submitting this project. If you have questions on how to use Canvas, ask - do not wait until just before the project is due to discover that you do not know how to use Canvas to submit project(s).*

4 Grading Breakdown

Correct Submission	15%
Classes Compile	20%
Correct Execution	25%
Following Directions	20%
Code Formatting/Comments	20%
Early Submission Bonus	*5%*

5 Notes & Warnings

- For many people, this is not the kind of project that you will be able to start the day before it is due and successfully complete. My recommendation is to start *now*.
- If you have any questions about anything involved with this project, ask me. If you don't ask for help, I will not know that you want it. Do not hesitate to stop by my office with questions or come in during my "lab" hours.
- *Start now*!
- Projects may *not* be worked on in groups or be copied (either in whole *or in part*) from *anyone OR ANYWHERE*. Failure to abide by this policy will result in disciplinary action(s). See the course syllabus for details.
- *START NOW*!
- Do *not* try to "fix" any math that you think is wrong. It will work for the purposes of this project exactly as specified!
- Have you started working on this project yet? If not, then *START NOW !!!!!*