CS 203 Homework #4: Box Catcher

Due Dates:

- **HW#4a:** Wednesday, 12 Feb 2014 at 10pm
- **HW#4b:** Wednesday, 19 Feb 2014 at 10pm

This homework will span two weeks and has two due dates. For the first milestone, you will create code to draw two boxes on the screen and detect whether they overlap partially or entirely. To complete the homework you will add a function to move one of the boxes around and "bounce" it off the sides of the window.

Grading

Your assignment grade for both parts of the assignment will be based on:

- Program functionality (80%)
- Code quality (10%)
- Report (10%)

Getting Started

1. Download the starter code (`BoxCatcher.zip`) from the course website.
2. Open the BlueJ project and examine the BoxCatcher class' current source code. At the top of the file are several variables that contain the state of the game. While the program is running, two boxes (squares) will be drawn on the screen: a "big" one and a "lil" one. Several of the variables define where these two squares are at any given time and also how big they are.
3. Scroll down to the section heading titled "ATTENTION STUDENTS". The code below is off limits for changes. To complete the homework you'll only need to implement the methods above this heading.
4. Run the starter code to see what the program already does. You should find that you can move the smaller box around with your mouse. How is this happening? *It's important for you to understand that your `paint` method is getting called many times per second.* Each time you move the mouse, the `lilBoxX` and `lilBoxY` variables are changed and then the `paint` method is called to redraw the screen with the box in its new position. This creates the illusion of motion.

Homework #4a: Specification

This specification takes you step by step through the implementation.

1. Modify your `paint` method so that if the little box is painted red and the big box is black with a black border. These will be the default colors. You'll change what colors the boxes will be in certain circumstances (see below).

2. Now, add code to implement the `isInside` method. The comment above this method in the starter code describes what it is supposed to do.
3. Modify your `paint` method so that when the little box is entirely inside the big box the little box is green and the big box is white with a black border.
4. Now, add code to implement the `isOverlapping` method.
5. Modify your `paint` method so that when the little box overlaps with but is not entirely inside the big box the little box is painted yellow and the big box is gray with a black border. The little box and big box should still be green and white respectively when the little box is entirely inside the big box.
6. Modify the `paint` method to draw the current elapsed time (in seconds) and the user's score at the bottom right. The `seconds` variable is automatically updated for you. The score should stay at zero for now. Use the `WINDOW_WIDTH` and `WINDOW_HEIGHT` variables to calculate where to display this information.
7. Finally, modify the `paint` method to track the user's score. The `score` variable should be incremented each time the user successfully moves the small box so that is entirely contained within the larger box. *Don't increment it more than once each time the user successfully gets the little box inside the big one.* To score again, the user will have to move the little box outside and back in again. You should use the boolean variable named `inside` to record whether the little box is inside or not each time that the `paint` method is called. Only increment the score when value of `inside` changes from false to true.
8. Make sure all the game elements drawn by your `paint` method are drawn in the proper order so that they don't cover each other up when they should not. In particular, the big box should never hide the smaller box.
9. As a final step, modify the `init()` method to allow the user to adjust the difficulty of the game. Use a `JOptionPane` to ask the user to set the difficulty level of the game by entering a number between 1 and 5 (where 5 is hardest). If the user enters an invalid reply, you should notify the user of the error and ask again. Your program should continue to ask the user for an answer until a valid response is given. Once you receive a valid input from the user, adjust the `bigBoxSize` as you see fit based upon the requested difficulty. There should be a different size for each difficulty level.

Homework #4b: Specification

Once you have completed 4a, you're ready to start moving the big box around. Moving objects around the screen is simply a matter of animation. First you draw the object in one position, then wait a split second, then draw it in a slightly different position. In other words, you'll do it just like a movie does by drawing a sequence of frames in rapid succession.

This specification takes you step by step through the implementation:

1. Before you do anything else, you'll need to set the `movement` instance variable to true. This tells Prof. Nux's part of the code to start calling your `moveTheBox` method.
2. Now, you'll need to implement the `moveTheBox` method to actually move the bigger box! Because you will create the illusion of movement by drawing the the big box in a slightly different position for each frame, you will need to track both the box's current position (with `bigBoxX` and `bigBoxY`) and its velocity: how much its position changes for each

frame. The velocity of the box is stored in the variables `bigBoxDeltaX` and `bigBoxDeltaY`.

3. To draw your sequence of frames you will need a while-loop. Each iteration of the loop will draw the next frame in the animation. The while loop should iterate until the user catches the box. Inside the while loop, you'll need to perform three steps:
   a) First, you want to change the box's position. The `bigBoxDeltaX` and `bigBoxDeltaY` variables are automatically set for you. Change the value of `bigBoxX` and `bigBoxY` by the amount specified in the delta variables.
   b) Once the box's position has been changed, you still need to repaint the screen so that the user can see the new position. You can trigger this by calling the `repaint()` method. *Do not call your paint method directly.*
   c) Now that the screen has been redrawn, you need to wait for slow human eyes to register this fact. A method has been written for you called `pause()` that delays for a split second. (Note: The amount of delay decreases as the user's score goes up to keep the user challenged.) Call this method at the bottom of your while loop.
4. Run your program at this point and you'll see that there is a problem: the box moves as you wanted it to, but then it moves right off the screen! You'll have to fix this so that the big box appears to "bounce" off of the sides of the window. Here's how to do it: Modify your `moveTheBox` method so that it detects when the box is about to move off of the screen and changes `bigBoxDeltaX` or `bigBoxDeltaY` to its additive inverse (i.e., multiply it by -1). Which variable you should change will depend upon which side of the window it is bouncing off of.

Additional Enrichment

With this homework complete, you now have many of the "tools" you need to create a simple arcade game. The starter code Prof. Nux wrote for you is fairly simple to follow. One thing you may want to add is code to detect mouse clicks. To find out how to do this, use an Internet search to read about using the `MouseListener` interface. It's easy.

Code Quality (10% each for 4a and 4b)

A good computer program not only performs correctly, it also is easy to read and understand:

- A comment at the top of the program includes the name of the program, a brief statement of its purpose, your name, and the date that you finished the program.
- Variables have names that indicate the meaning of the values they hold.
- Code is indented consistently to show the program's structure.
- The body of `if` and `else` clauses are enclosed in braces and indented consistently, even if they consist of only a single line of code.
- Opening braces are placed consistently, either at the end of the if-statement or directly under the 'i' of `if` or the 'e' of `else`. Closing braces are in the same column as the 'i' of `if` or the 'e' of `else`.
- Within the code, major tasks are separated by a blank line and prefaced by one or more single-line comments identifying the task, e.g., "draw a tree."

- Methods are separated by blank lines and prefaced by a multi-line comment describing what they do and what their parameters mean. (See starter code for examples.)
- Very long statements (such as long print statements or complex boolean expressions) are broken across lines and indented to show their structure.
- *Now that you are familiar with loops and methods, your code should not contain redundant or repeated sections.*

Logistics

- Remember to turn this homework when you complete 4a and then again when you complete 4b.
- Be sure to add your name to the comment header at the top of your .java file.
- Remember to include comments to remind yourself and your reader the functionality of the different parts of your program.
- Turn in your BoxCatcher.java file via the “Turn it in Here” link on the course website. There will be a separate link for 4a and 4b, so be sure to use the correct link.