CSE 11 **START EARLY!**
Fall 2013
Programming/Homework Assignment #4 (revised with typos fixed)

Due: 1 November 2013, 11:00pm (electronic turnin) 100 points

This is a programming assignment and will be turned in electronically. You may develop your code on your own machines, however, the code *must* run on lab machines and *must* be turned in from a lab machine.

NO LATE ASSIGNMENTS WILL BE ACCEPTED

The Program – Cross the Cemetery

Happy Halloween! You will be developing three classes Cemetery`.java, Tombstone.java, and Goblin.java.` The Cemetery is a "game" in which goblins must cross the cemetery without hitting the moving tombstone. If a goblin hits (overlaps) a tombstone, the tombstone randomly decides to either let it pass or vaporize the goblin. The game keeps track of the number of goblins created, the number that were vaporized and the number that passed safely into the afterlife.

You will be using the `objectdraw` library from your previous homework, explore using java interfaces, ActiveObjects, callbacks, random selection, and switch statements. Like the previous assignments, you will continue to learn to develop your program in stages. This is a much more substantial program than BlockAndTackle and you should expect it take you more time. Begin early.

The Goblin Class

The Goblin class is an `ActiveObject` and simulates a goblin crossing the cemetary. In this case, the current canvas is the cemetary. Goblins are visually represented by one of three different graphics images (formatted as a PNG, JPG, or GIF files and are opened as java `Image` objects. These images are placed on the canvas as `VisibleImage` objects defined in objectdraw.

All of the methods described must be implemented with same signatures and functionality. This is so that we can test your Goblin class independently of the other two classes.
The constructor is as follows.

```
public Goblin(int selection, Location initial,
              double velX, double velY,
              Tombstone tombstone, CemeteryController control,
              DrawingCanvas aCanvas)
```

- selection – 0,1,2, Selects which Goblin graphic is to be displayed graphically
- initial -- Initial location of where the Goblin should be placed
- velX – velocity of the Goblin in the horizontal direction in pixels/millisecond
- velY – velocity of the Goblin in the vertical direction in pixels/millisecond
- tombstone -- the tombstone that is moving across the cemetary. Tombstone might be a null object.

- control – a controller class that implements the CemetaryController interface
- aCanvas – the drawing canvas

Functionality Requirements of `Goblin.java`

1. Goblins should use the image files ghost.jpg, pumpkin.jpg, skeleton.gif. These names should be coded as constants.
2. Goblins should update smoothly (using millisecond resolution timers as described in lecture and in the textbook). The length of the pause() should based upon the magnitude of the initial velocity. Faster Goblins should be updated more often, slower spaceships less often.
3. Goblins should "bounce" off the top edge of the canvas Hitting the top of the canvas should cause the current (negative) y-component velocity to reverse (become positive)
4. When Goblins are first created, the images should be re-sized so that their width is 1/SCALE of the *current* canvas width. They should be re-sized in the vertical direction by the same scaling *factor* that properly sets the width. In other words, graphics pictures should retain their aspect ratios. Goblins stay the same size once they have been constructed, even if the canvas size changes
5. Goblins should callback to the Tombstone (if it is not null) when it detects that it has intersected (overlapped) the tombstone. It Goblin should only callback to the tombstone the very first time it hits the tombstone. The callback is performed by invoking the `enter()` method of the Tombstone
6. Goblins must support a method `public void vaporize()`. This is the method the Tombstone calls if it decides to not allow the goblin to pass. If a goblin is vaporized, it should exit the run loop.
7. Goblins must support a method called `public void setVelocity(double vx, double vy).` This should change the velocity of the goblin.
8. When the complete goblin goes off the canvas (left edge, right edge, bottom edge) or is vaporized, it should exit the run method. When it exits the run method it must callback to the Cemetary controller using the `public void record(Goblin wraith, boolean vaporized).` This allows the controller to keep track of how many goblins passed to the afterlife and how many were vaporized.
9. Goblins do NOT detect if they run into each other, they simply pass by.

The Tombstone Class

The Tombstone class is nearly completely implemented for you. This is to save you time. You will have to implement the private methods `scaleAndPlace(), moveIt(),` and `enter().` You must use the double constant `PROBABILITY` defined in `CemeteryController.java` to decide if the goblin that called enter() will be vaporized (or not vaporized).

The Tombstone move only in the horizontal direction. The run() method is fully implemented but depends upon moveIt() being properly implemented. The Tombstone constructor depends upon scaleAndPlace() being properly implemented. See the supplied Tombstone.java file.

The CemeteryController Interface

```
public interface CemeteryController
{
        public void record(Goblin wraith, boolean vaporized);
```

```
}
```

You must create the proper java file to define the CemetaryController interface. This is necessary before of the supplied test and outlines of programs will compile.

Requirements

1. you most define the Constants PROBABILITY, SCALE, SIZE, and MAXV in your CemeteryController interface.
2. Use PROBABILITY = 0.5, SCALE=20, MAXV=0.3 , SIZE=600 as initial values.
3. MAXV is the maximum pixels/ms for x and y components of goblin and tombstone velocities.
4. PROBABILITY is the probability that the tombstone will vaporize a goblin that hits it.
5. SCALE the canvas width:goblin width ratio. 20 means goblins should be created with 1/20th the width of the canvas
6. SIZE is the initial size of the cemetery. It is SIZExSIZE.

The Cemetery

The Cemetery is the master "controller", it creates a tombstone and launches goblins. Whenever a mouse is clicked on the canvas, a new Goblin is created and launched. The cemetery must track the number of goblins that have been created, how many passed, and how many have been vaporized. It has two control "widgets" one to end the game and one to randomly adjust the speed of the most recent goblin created. See the TestCemetery code supplied.

Functionality Requirements of Cemetery:

1. When the mouse is pressed on the canvas, a **green line** (the velocity vector) should be drawn from the coordinate (X location of the mouse press, bottom of the canvas) to the current press point. This is a velocity vector.
2. When the mouse is dragged, the end of the velocity vector should move with the mouse. It should still be rooted the original (X,canvas bottom) location.
3. When the mouse is released, a new goblin should be launched. It's velocity should follow the drawn velocity vector and its speed should be proportional to its length. The simplest way of doing this is to have vx = dx/canvas_width, where dx is the difference between the x coordinates of the velocity vector's endpoints. A similar calculation should be done for vy.
4. The graphical representation of the Goblin (i.e., which image file the Goblin instance will scale and display) should be randomly selected (via the `selection` argument of the Goblin constructor). The selection is 0,1, or 2.
5. A Goblin calls the `record()` method of its controller when it exits the canvas on the left, right or bottom. It also calls record() when it is vaporized.
6. Cemetery (below) must track and display the total number of goblins created, the number of successful passes and the number of goblins that have been vaporized. When no goblins are in the cemetery, the total # goblins = #passed + #vaporized.
7. The canvas can be resized at any time, tnewly constructed Goblins should behave properly (resized) as defined above. Existing goblins do not change size.
8. The initial canvas size should be SIZExSIZE
9. The maximum velocity of a goblins should sqrt(2*MAXV*MAXV)
10. The tombstone must be located at the Y position that is 25% of SIZE. It should move right to

left at MAXV velocity.

Using TestCemetary

You are being provided TestCemetery.java. It is a simple controller that also implements the CemeteryController interface. It has limited functionality. Specifically, it creates goblins when the canvas is clicked. Velocity of the goblin is 0.0 for vx, and is predictable for vy.

How to make the best use of TestCemetery: This allows you to develop and test a good portion of Goblin.java. You should use this to test if goblins move properly (reflecting from the top wall, going off canvas, etc. They should react to canvas resizes properly, and so on. TestCemetery.java is being supplied to you allow you to build in pieces. You can also use TestCemetery as a basis for Cemetery. Note we will test your Goblin class with the distributed version of TestCemetery as just one of the tests during grading.

Note: For TestCemetery to compile you have to properly define the CemeteryController interface in its own .java file.

One of the advantages of Java interfaces is that we can define two different controllers without changing Goblin. java. TestCemetery is simple (< 75 lines total), and allows us to concentrate energies initially on developing and incrementally testing the Goblin.java code.

Developing Goblin.java

You are being provided a very incomplete version of Goblin.java. It codes some of the functionality. It also suggests defining some `private` methods. These private methods can only be used internally in the class, but represent either "complex" calculations that are naturally reused within the Goblin class, or simply wrap a single logical function into a private method to make code more understandable. You might look at the nearly complete Tombstone.java class for some ideas of how to perform some calculations.

The following suggested stages of development of Goblin will likely make your coding go more quickly. The idea is to build increasingly more complex code. You can use TestCemetery as the controller for the first of these stages. Program each step, verify that your code works, then save each working version (in case future development goes wrong), and then modify your code for more functionality. Don't do step 2 before you have coded and tested step 1. Don't do 3 until step 2 is coded and tested.

1. Create Goblins with just a single bitmap image with proper initial velocity, no bouncing off the canvas top, just detect when a goblin leaves the canvas. This allows you to get the image scaling correct, verify that your run() method has the basic performance you want.
2. Create the logic for properly bouncing off the top of the canvas. You should test resizing of the canvas, too to make sure that new goblins are created relative to the new canvas width.
3. Implement a setVelocity method and test by pressing on the square in the TestCemetary
4. Add the capability of selecting different bitmap images for your Goblins.
5. At this point, a great deal of Goblin has been completed, you will need to properly code to determine when to call the enter() method in the Tombstone, react properly if the vaporize() method is called on your Goblin instance and call the record() method of the controller when the goblin is either vaporized or exits the cemetery.(to test those methods, TestCemetary needs

modification) (Hint: use two boolean variables in your run() method to determine when to exit the while loop. Only after your goblin has stopped running (ie. It's off canvas or has been vaporized) should you invoke the record() method in the controller

Developing Cemetery

When you have Goblin mostly or completely developed, you can turn your efforts to developing Cemetery. First, start with TestCemetary, modify it so that it becomes the Cemetery controller. You can choose what order you develop Cemetary but a reasonable way to start might be (make sure you test after you complete each chunk of functionality – incremental development and testing is key to success in ALL programming.)

1. Create a tombstone that is 25% of the distance from the top of the initial canvas and has an initial velocity of 0.3 pixels/millisecond. This 25% location is a requirement.
2. Implement the record() method, print updates when record is invoked.
3. Implement the velocity vector to lauch goblins
4. .....

(Description Continues after the pictures … Keep Reading!)

ILLUSTRATIONS of what your program should look like at various stages.

Illustration 1: TestCemetary with numerous Goblins

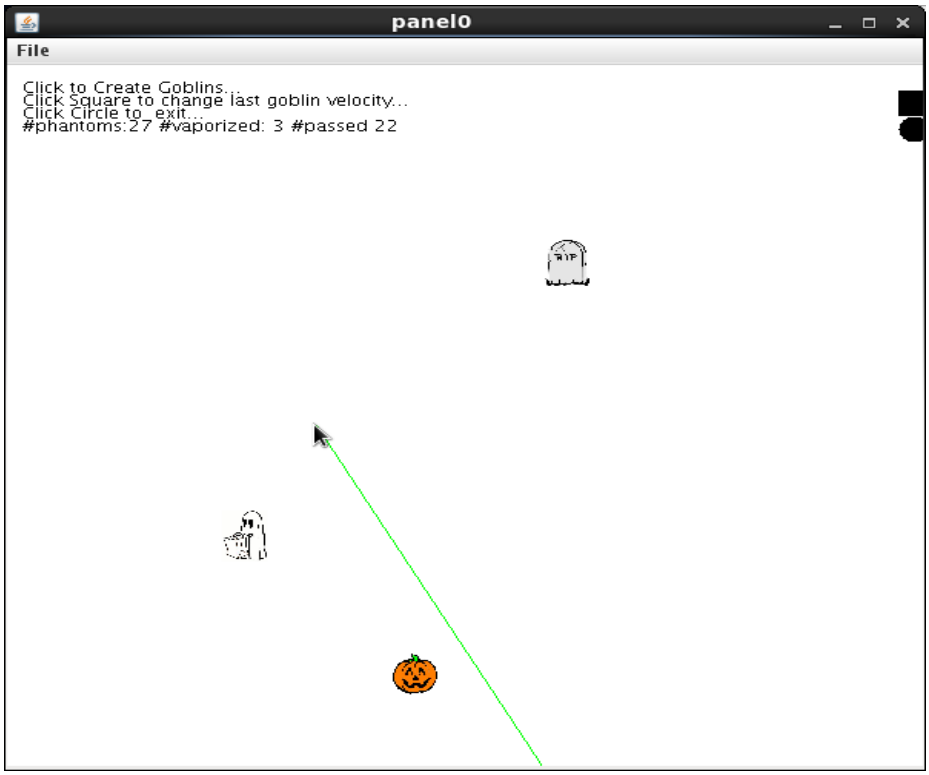

Illustration 2: Cemetary Controller. Ships Phantoms (27), Vaporized (3), Passed(22), There should be 2 goblins on the canvas (counted – vaporized – passed). Notice the green velocity vector

Grading (total of 100 points)

80 points – Does your program compile and run properly. If all aspects are met, then you will receive full credit. Various misfeatures or bugs will lose appropriate points. We will test with different parameters for PROBABILITY, will resize the canvas. and the like.

If your classes do not compile, the best you can receive on the entire project is 50 points. (Half credit).

20 points – Commenting/indentation/style. Please see the Style guide. You Must put your NAME:, LOGIN:, ID: in every file you turn in. Please use ALL CAPS for the NAME, LOGIN, ID labels..

Turning in your Program

YOU MUST BE ON THE LAB MACHINES FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the "bundleP4" program that will turn in the following files

```
Cemetery.java
Goblin.java
CemeteryController.java
Tombstone.java
```

No other files will be turned in and and they **must be named exactly as above.** BundleP4 uses the department's standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

```
$ /home/linux/ieng6/cs11e/public/bundleP4
```

Some of you have had trouble turning in programs that compile properly or have turned in the wrong version. To address this, a A HIGHLY Recommended sequence of commands is the following: (You are not ready to turn in files if you get any errors)

```
$ rm -i *.class
$ javac Cemetery.java Goblin.java Tombstone.java
CemeteryController.java TestCemetery.java
$ java Cemetery
$ /home/linux/ieng6/cs11e/public/bundleP4
```

The above will remove your compiled java classes (asking you first if you want to remove them, say yes to *.class files don't make a mistake and delete .java files!!!), then will compile the files you are about to turn in.

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Suggestion: if you have classes that compile and do some or most of what is specified, turn them in early. When you complete all the other aspects of the assignment, you can turn in newer (better) versions.

Don't forget to turn in your best version of the assignment.

Frequently asked questions

What do I do if my program doesn't compile and I ran out of time? Put comments at the top of your Cemetery.java file that indicates you could not successfully compile your program. We will look at giving partial credit. Turn in what you have. You won't get a great grade, but it is better than a zero.

What if I get runtime exceptions? You shouldn't have any runtime exceptions. Look at the exception and see if you can figure out what is causing it. Some common causes are division by 0 or trying to invoke a method on null instance.

Can I do this as an applet? No. Cemetery should be a java program.

When I make the canvas bigger, and then create new Goblins, they are bigger than than the old ones, is that right?. Yes, the scaling factor for the images is relative to the width of the canvas *at the time* the Goblin was constructed.

Additional Exercises (Not Graded or Turned In)

1. What are the callbacks used in this project?
2. If you wanted to have the Tombstone randomly decide to reverse direction,(.e.g move from right to left, then left to right, then right to left, how would you accomplish that?)
3. Tombstones and Goblins are actually closely related in code. If one were a parent class and the other extended that class, would it be “Tombstone extends Goblin” or “Goblin extends Tombstone”. Why?

START EARLY! ASK QUESTIONS!
HAVE FUN PLAYING YOUR TOMBSTONE GAME!