Homework Assignment #5

Due Monday, May 21, at the start of lecture. Turn in a typed hardcopy of your answers to the first two questions, and a (readable) print-out of one or more file-outs for your new and modified classes for the last 3 questions.

1. Explicitly parenthesize the following Smalltalk expression:

   \[ \text{abc := qwe ert xrd: wed + wer sdf rrt q: qwe + cvb * qwe poi.} \]

   How many messages are sent in the above expression?

2. In the following class hierarchy and methods (where methods are listed below each partial class declaration that they’re part of, and method bodies are indented below each method header), indicate what would be printed onto the Transcript (assuming that the method start is executed first), and explain why.

   Object subclass: #A ...
   start
   C new m1.
   m1
   Transcript show: 'A::m1 '.
   self m2: self.
   m2: arg
   Transcript show: 'A::m2 '.
   arg m3.
   m3
   Transcript show: 'A::m3 '.

   A subclass: #B ...
   m1
   Transcript show: 'B::m1 '.
   super m1.
   m2: arg
   Transcript show: 'B::m2 '.
   arg m3.

   B subclass: #C ...
   m1
   Transcript show: 'C::m1 '.
   super m1.
   m3
   Transcript show: 'C::m3 '.

3. Define a new subclass of MovingParticle, MovingParticleWithRandomMassChanges, whose mass changes randomly within the range 20-100 each time its doOneTimeStep:in: method is called. The particle’s size should reflect its new mass. See the mass: method in Particle and the code in initialize in MovingParticle for code to reuse. Your solution should be just a couple of lines of code, and use super. A nice solution would factor out into its own method.
the code in initialize to set the mass to a random value, and call it from both initialize and from your new code in MovingParticleWithRandomMassChanges.

Change the code in ParticleWorld to create an instance of your new class whenever it receives a mouseDown event with the shift key pressed (send shiftPressed to the event argument of mouseDown: to test whether the shift key was pressed when the mouseDown event happened); on other mouseDown events ParticleWorld should create a regular MovingParticle instance. You may wish to add a class argument to the newRandomParticleAt:, computed by the mouseDown: method in ParticleWorld.

4. Define a new subclass of MovingParticle, BoundedMovingParticle, that bounces off the walls of the ParticleWorld. Whenever a MovingParticle would move off the edge of the world and be deleted, the BoundedMovingParticle would reflect off the wall (using proper reflection!) back into the world, with its velocity and position adjusted. Take care to handle bouncing in the corners of the world. You should modify the current doOneTimeStep:in: method of MovingParticle to invoke a helper method (e.g. updatePositionTo: newPos in: world), which in MovingParticle does the current updating of center and checking for deletion, and in the BoundedMovingParticle subclass does your new code. (This is a good example of inserting sends to self to make a method more reusable.) You might wish to look at the bounceIn: method of AtomMorph for an example algorithm to keep a particle within a world. You might wish to run the BouncingAtomsMorph demo to see how this code works.

BoundedMovingParticles should be created by ParticleWorld whenever the right mouse button is pressed (send yellowButtonPressed to the event argument of mouseDown: to test for a right-mouse-button press).

5. Extra credit: Define a CheckerboardMorph, which contains an 8x8 grid of submorphs of alternating black and white colors. If the mouse is clicked on one of the submorphs, then it should change to a random color.