Burrell, David 300209541
NWEN303 Project One Report

Task One:

Model.java manages the logic of the particle simulator, including drawing particles on the GUI, calculating whether particles need to be merged.

Variables:
size: size of the canvas
gravitationalConstant: represents gravity
lightSpeed: controls the size of the simulation universe
timeFrame: controls step length
p: this is the list of all Particle objects in the simulation
pDraw: the drawable version of the particle list.

The model uses Particle objects to simulate particles, these objects handle the calculations involved in how each particle can interact with each other, these calculations rely on the Model.
The Model handles drawing the particles on the GUI with the use of DrawableParticle objects, when called to update the graphical representation the model iterates through the list of particles, and creates DrawableParticle versions, containing only the location of that particle during that step, and its color and size.

The four actions of step():
1. Iterate through the list of all particles and calculates the interactions of each individual particle for one step of time (timeFrame is used)
2. Use mergeParticles() to calculate any merging particles based on updated interactions since the previous step.
3. Iterate through all the particles again, and move them in the x and y axis, based on their newly calculated speed from calling interact() in step 1.
4. updateGraphicalRepresentation() draws the new locations and size of particles to the GUI.
Particle Merging is handled by checking each particle in the particle list for any impacting particles, removing these particles from the main list and adding/combining the particles together, with any other impacting particles that are impacting the original impacting particle.

Task Two:

Gui.java uses a ScheduledExecutorService to handle 2 threads, one for repainting the canvas and the other for handling the Model.

The Model itself is the contended data, needed both to display to the GUI in one thread and run the simulation on the other. With Many Reads, zero writes.

Both threads are scheduled to begin after 500 milliseconds, the repaint() be subsequently called every 25 milliseconds after that.

Task Three:

a) The obvious place to start is with the step() method, as it currently iterates through the entire particle list two separate times, looking at using parallel streams to do these iterations seems like a simple fix, providing I can make sure the particle objects are handled correctly to avoid interference/consistency errors, as well as any contention.
The other place to look at would be the merging of particles, however due to the modification of collections this may be difficult to deal with in a simple manner and may be easier to leave to being dealt with sequentially.

b) It will allow the simulation to more accurately represent the particles, as each particle will be dealt with simultaneously, making the interactions more accurate as each particle will be able to calculate its interactions before any particles are updated (in theory).
c) The data contentions that need to be resolved will involve the calculations of individual particles when interacting with other particles, more accurately the position, speed and mass variables and if they are impacting. More complexly when merging particles as items are removed from the collection during the mergeParticles() method.
d) With correct handling of synchronising and use of atomic calculations, and testing, however there is no guarantee I will avoid all possibilities.

Task Four:

Started by looking for a simple way to introduce some parallelism into the model, the obvious choice was to deal with the constant iterative processes on the Particle List, after figuring out a simplistic implementation of parallelism there I wanted to attempt to do something similar with the merging function. Another Idea I had was to segment the entire simulation into groups and run each group in parallel, however needing to deal with calculations across groups would be a challenge to look at.

Task Five:

Task Six:

Wrote a simple test fuction based on the few variations of models given in the gui class.
used a loop and kept time to see how long each type of model dealt with a set amount of calls to step().