Honors Data Structures & Algorithms: Summer Assignment
Part A: Understanding The N-Body Problem

The goal of the summer assignment is to:

- Establish (or Reestablish) your Eclipse programming environment on your home computer
- Become familiar with the "Standard" libraries which are employed in the our textbook, Introduction To Programming in Java, by Robert Sedgewick and Kevin Wayne
- Learn some physics, which you may be currently studying or may have studied last year
- Write about 70 lines of Java code to solve a problem which the world's foremost mathematicians and physicists, including Isaac Newton, were unable to solve for 300 years.

In 1687 Sir Isaac Newton formulated the principles governing the motion of two particles under the influence of their mutual gravitational attraction (aka the "Two-body Problem") in his famous book, the *Principia*. Common examples of the two body problem include a satellite orbiting a planet, a planet orbiting a star, two stars orbiting each other (a binary star), and a classical electron orbiting an atomic nucleus. However, Newton was unable to solve the problem for three particles (aka the "Three-body Problem") or more particles (aka the "N-body Problem"). In general, solutions to systems of three or more particles *cannot be solved analytically*. In other words, these problems cannot be solved with the rules of algebra, trigonometry, geometry and calculus. Instead, *they can only be solved numerically*.

Your challenge is to write a program to simulate the motion of N particles in the plane, mutually affected by gravitational forces, and animate the results. Such methods are widely used in cosmology, semiconductors, and fluid dynamics to study complex physical systems. Scientists also apply the same techniques to other pairwise interactions including Coulombic (force of static electricity), Biot-Savart (the magnetic force produced by flowing electricity), and van der Waals (intermolecular forces).

There are some really cool solutions to the N-body problem in which the bodies assume a stable period motion forever. These are called N-body choreographies. Here are a few great animations of N-body choreographies:

- http://www.maths.manchester.ac.uk/~jm/Choreographies/
- http://www.princeton.edu/~rvdb/WebGL/nBody.html
- http://neutraldrifts.blogspot.co.uk/2011/02/youtube-videos-of-n-body-choreographies.html

This assignment is based on the N-Body Case Study of Introduction To Programming in Java, by Robert Sedgewick and Kevin Wayne as well as course material from their COS 126 course.

AP Computer Science Summer Assignment
Part B: Establishing Your Eclipse Development Environment

I. Establish your Java Development Environment and Write a "Hello World" Program

(1) You should download the *Java SE Development Kit* at http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html Click "Accept License Agreement", then download and install the version which is matches your computer. Be sure to select the right version (32-bit or 64-bit (x64)) for your computer. You most likely need the 64-bit version. If your computer is old, you might need the 32-bit version.

(2) Create a new folder on your computer called *APCompSci*

(3) Inside your *APCompSci* folder create a new folder called *MyCode*

(4) Download *Eclipse IDE For Java Developers* at http://www.eclipse.org/downloads/ to your *APCompSci* folder. The download will be a zip (compressed) file. Note that the download is free and you need not make a donation to download Eclipse. After the download has completed right-click on this zip file and select *Extract All* to extract the contents into a folder. This may take a few minutes. After the extract is complete, if the folder has a weird name (like *eclipse=java-luna-SR2-win32-x86_*64) , right-click on the folder and rename it to *eclipse .* After you have extract is complete, delete the zip file you had downloaded.

(5) Navigate to *APCompSci->eclipse->eclipse*. There is an application in this folder called *eclipse* and the icon for this file should look like this:  . Right-click on this file and select *Send to Desktop (create shortcut)*. This will create a shortcut for eclipse on your desktop

(6) Go you're your Desktop and double-click on your newly created Eclipse icon to start Eclipse

(7) Eclipse might take a little while to start/initialize. After Eclipse has initialized, it will present a *Select A Workspace* dialog box. Click on the *Browse* button and navigate to your *APCompSci->MyCode* folder. Check the checkbox for *Use this as the default and do not ask again.* Then click OK.

(8) A Welcome Window is displayed. If you ever need to return to the Welcome window, select *Help->Welcome*

(9) On the Welcome Window, select *Tutorials,* then select *Create a Hello World Application.*

(10) Follow the tutorial instructions and complete all parts of the tutorial.

II. Configure Eclipse to make "smart" suggestions

(1) In Eclipse, select *Window->Preferences ,* then in the tree on the left, expand the tree to select *Java->Editor->Content Assist->Advanced* . On the *Advanced* window, left-click on the *Restore Defaults* button, then left-click on the OK button.

(2) In Eclipse, select *Window->Preferences ,* then in the tree on the left, expand the tree to select *Code Recommenders->Completions* . On the *Intelligent Code Completion* window, left-click on the *Enable intelligent code completion checkbox*, then left-click on the OK button.

If you have any questions, email me at rwertz@veronaschools.org

AP Computer Science Summer Assignment
Part C: Solve the N-Body Problem

I. Download the Standard Library Code

(1) In your *APCompSci* folder, create a new folder called *Jars*
Visit the following link to download and save the file *stdlib.jar* in your *Jars* folder.
http://introcs.cs.princeton.edu/java/stdlib/stdlib.jar
"Jar" stands for "Java archive" and a jar file is essentially a zip file of useful code. The code in *stdlib.jar* will help us draw graphics, make sound and manipulate images. Be sure to just download and save this file in your *Jars* folder and do not try to open it.

II. Set up your Java project and download input files used to test your program

1) In Eclipse, create a new Java Project (*File->New->Java Project*) and name the project *NBody*
2) Download the zip file nbody.zip . View the directory that you downloaded the zip file to. Right-click on the zip file and "Extract All..." which will ask you to choose the extraction location. (Don't double click the zip file, since it browses the contents but does not extract them.) A new subdirectory nbody will be created at the extraction location, with the files inside of it. Control-A to select all files in the directory, then Control-C to copy them. In the Package Explorer of Eclipse, click on the NBody project (not the source folder, but the top-level folder in the project), and Control-V to paste them. If you are prompted with a "File Operation' dialog box, select "Copy Files" and click OK.
3) In the Package Explorer window of Eclipse, right-click on the Folder for the *NBody* project and select *Build Path->Add External Archives .* Then navigate to the *stdlib.jar* you downloaded in your *Jars* folder and select it. This step effectively imports some important code that will help us use the graphics and other capabilities of the computer.
4) In the Package Explorer window of Eclipse, right-click on the Folder for the *NBody* project and select *New->Class.* Then name the new class *NBody.* An empty class will appear in the editor window of Eclipse.

III. Understand the Program Specification

The program specification outlines your programs inputs, outputs, and what exactly your program is supposed to do:

- Your program (class file) will be called `NBody.java`

Your program will:

- Prompt the user for two `double` values, T and Δt
  - T represents the total time of the simulation in years
  - Δt represents the amount of time between two iterations of your simulation
- Reads in the "universe" of bodies from a file using the `In` class, using parallel arrays to store the data
  - To connect an `In` object which reads a file, use the following code:
    `File f = new File("planets.txt");`

```
In fInput = new In(f);
```

- You'll declare and initialize six 1-D arrays to store:
  - x coordindate of the position of the body, in meters
  - y coordindate of the position of the body, in meters
  - Horizontal (x axis) velocity of the of the body, in meters per second
  - Vertical (y axis) velocity of the of the body, in meters per second
  - Mass of the body, in kilograms
  - Filename of the image used to draw the body (e.g. `"earth.gif"`)

- Simulates the universe, starting at time `t = 0.0`, and continuing as long as `t < T`, using the "leapfrog" scheme described below
- Animates the results using the `StdDraw` class
- Prints the state of the universe at the end of the simulation (in the same format as the input file) to standard output (the console) using the `StdOut` class.

IV. Understand the "Universe" input file format

The input format is a text file that contains the information for a particular universe (in SI units). The first value is an integer `N` which represents the number of particles in the universe. The second value is a real number `R` which represents the radius of the universe, which is used to determine the scaling of the drawing window. Finally, there are N rows, and each row contains 6 values. The first two values are the x- and y-coordinates of the initial position; the next pair of values are the x- and y-components of the initial velocity; the fifth value is the mass; the last value is a String that is the name of an image file used to display the particle. As an example, planets.txt contains data for our own solar system (up to Mars):

Here are the contents of an example input file, planets.txt

```
5
2.50e+11
 1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24    earth.gif
 2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04  6.4190e+23     mars.gif
 5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04  3.3020e+23  mercury.gif
 0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  1.9890e+30      sun.gif
 1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04  4.8690e+24    venus.gif
```

V. Understand the physics

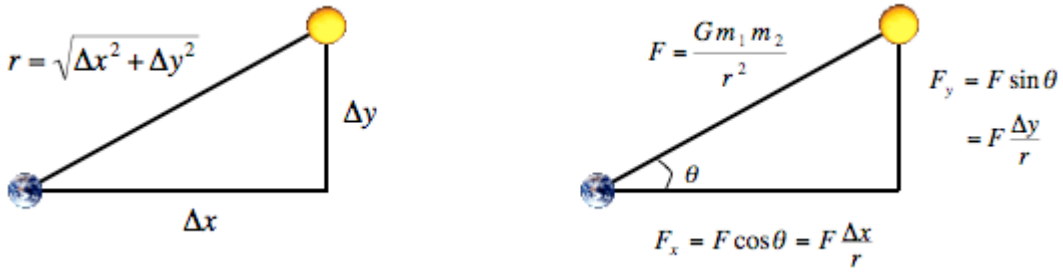
Here we review the equations governing the motion of the particles, according to Newton's laws of motion and gravitation. Don't worry if you haven't yet learned physics or if your physics is a bit rusty; all of the necessary formulas are included below. We'll assume for now that the position (p_x, p_y) and velocity (v_x, v_y) of each particle of each particle is known. In order to model the dynamics of the system, we must know the net force exerted on each particle, in other words, in which direction and how strongly is the particle being "pulled" by the other particles.

- **Compute the "pairwise force" between each unique pair of particles:**

Newton's law of universal gravitation asserts that the strength of the gravitational force between two particles is given by the product of their masses divided by the square of the distance between them, scaled by the gravitational constant $G(6.67 \times 10^{-11} \frac{m^3}{kg \cdot s^2})$.

$$F = \frac{Gm_1m_2}{r^2}$$

The pull of one particle towards another acts on the line between them. Since we are using Cartesian coordinates to represent the position of a particle, it is convenient to break up the force into its x- and y-components (F_x, F_y) as illustrated below.



You should store the pairwise forces in a 2-D array, where the row index represents the index of the "from" body and the column index represents the index of the "to" body. Note that the diagonal elements of this 2-D array should be infinite, as the distance r between a particle and itself is zero.

- **Compute the net force acting on each particle:**
  The principle of superposition says that the net force acting on a particle in the x- or y-direction is the sum of the pairwise forces acting on the particle in that direction. So to get the x component of the net force on particle i, you need to sum the pairwise forces between particle i and all other particles (but not itself).
- **Compute the acceleration of each particle:**
  Newton's second law of motion postulates that the accelerations in the x- and y-directions are given by:

$$a_x = \frac{F_x}{m}$$
$$a_y = \frac{F_y}{m}$$

Store the x components of the acceleration of all particles in a 1-D array. Do the same for the y components of the acceleration.

VI. Simulate the universe

We use the numerical approximation of the "true" solution, called the "leapfrog finite difference approximation" to numerically integrate the above equations: this is the basis for most astrophysical simulations of gravitational systems. In the leapfrog scheme, we discretize time, and update the time variable t in increments of the time quantum Δt (measured in seconds). We maintain the position (p_x, p_y) and velocity (v_x, v_y) of each particle at each time step. In other words, the main loop of our

program has t as its loop variable and t is incremented by Δt each time we go through the loop. The steps below illustrate how to evolve the positions and velocities of the particles.

- **Step 1:** For each particle: Calculate the net force (F_x, F_y) at the current time t acting on that particle using Newton's law of gravitation and the principle of superposition as described in part IV. Note that force is a vector (i.e., it has direction). In particular, be aware that Δx and Δy are signed (positive or negative). In the diagram above, when you compute the force the sun exerts on the earth, the sun is pulling the earth up (Δy positive) and to the right (Δx positive).
- **Step 2:** For each particle:
  a) Calculate its acceleration (a_x, a_y) at time t using the net force computed in Step 1 and Newton's second law of motion: $a_x = \frac{F_x}{m}$, $a_y = \frac{F_y}{m}$
  b) Calculate its new velocity (v_x, v_y) at the *next* time step $(t + \Delta t)$ by using the acceleration computed in Step 2a and the velocity from the *old* time step (t): Assuming the acceleration remains constant in this interval, the new velocity is $(v_x + a_x \cdot \Delta t, v_y + a_y \cdot \Delta t)$
  c) Calculate its new position (p_x, p_y) at the *next* time step $(t + \Delta t)$ by using the velocity computed in Step 2b and its *old* position at time t: Assuming the velocity remains constant in this interval, the new position is $(p_x + v_x \cdot \Delta t, p_y + v_y \cdot \Delta t)$
- **Step 3:** For each particle, draw the particle on the canvas using the position computed in Step 2. Draw each particle at its current position using the StdDraw class, and repeat this process at each time step until a designated stopping time. By displaying this sequence of snapshots (or frames) in rapid succession, you will create the illusion of movement. After each time step (i) draw the background image starfield.jpg, (ii) redraw all the bodies in their new positions, and (iii) control the animation speed (about 40 frames per second looks good). You will use several methods from the StdDraw library; click here to see the API for this library

Note that this simulation is more accurate when Δt is very small, but this comes at the price of more computation.

VII. **Output the final state of your universe:** After the animation stops, your program should output the final state of the universe in the same format as the input, e.g.:

```
5
2.50e11
 1.4925e+11 -1.0467e+10  2.0872e+03  2.9723e+04  5.9740e+24    earth.gif
-1.1055e+11 -1.9868e+11  2.1060e+04 -1.1827e+04  6.4190e+23     mars.gif
-1.1708e+10 -5.7384e+10  4.6276e+04 -9.9541e+03  3.3020e+23  mercury.gif
 2.1709e+05  3.0029e+07  4.5087e-02  5.1823e-02  1.9890e+30      sun.gif
 6.9283e+10  8.2658e+10 -2.6894e+04  2.2585e+04  4.8690e+24    venus.gif
```

VIII. **Backup Up Your Work**

(1) Copy your *MyCode* folder to a USB Flash Drive so that you can bring it to school