**Overview**

Group: Jack Frankland- s1404032

Martin Tasker- s1414001

Michael Spence- s1419697

The aim of this project is to simulate the solar system using code that describes N-body systems that interact through Newtonian gravitation using the Verlet Algorithm for time integration. We will at the end compare it with known physical data.

This is an outline of the class structure we will use.

Class Layout

### Vector3D Class:

**Each instance represents a three dimensional vector using 3 doubles as components.**

**Properties:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Notes |
| xx | double | x-coordinate |
| yy | double | y-coordinate |
| zz | double | z-coordinate |

**Constructors:**

|  |  |
| --- | --- |
| **Arguments** | **Notes** |
|  | Default constructor, creates (0,0,0) |
| Vector3D original | Copy constructor to create create copies of Vector3D object instances |
| double xx, double yy, double zz | Constructor method to create vector given explicit components |

**Instance Methods:**

**public double getX():**

Returns the x-component of the vector.

**public double getY():**

Returns the y-component of the vector.

**public double getZ():**

Returns the z-component of the vector.

**Void setX(double xx):**

Sets x-component of the vector.

**Void setY(double yy):**

Sets y-component of the vector.

**Void setZ(double zz):**

Sets x-component of the vector.

**Void setVector(double xx, double yy, double zz):**

Convenient setter method to set all three components.

xx is the x-component, yy is the y-component, and zz is the z-component.

**String toString():**

Returns Vector in for (x-comp, y-comp, z-comp).

**double magVector():**

Returns the magnitude of the vector.

**double magVectorSquared():**

Returns magnitude of the vector squared.

**Vector3D scalarDivision(double a):**

Divides Vector by a.

**Vector3D scalerMult(double a):**

Multiplies Vector by a.

**Static Methods:**

**Vector3D addVector( Vector3D vectorA, Vector3D vectorB):**

Adds two vectors together and returns the result.

**Vector3D subVector(Vector3D vectorA, Vector3D vectorB):**

Subtracts two vectors and returns the result.

**Vector3D crossVector(Vector3D vectorA, Vector3D vectorB):**

Returns the cross product of the two vectors.

**double dotVector(Vector3D vectorA, Vector3D vectorB):**

Returns the dot product of two vectors.

**Boolean eqaulVector(Vector3D vectorA, Vector3D vectorB):**

Method for determining if two vectors are equal.

### Particle3D Class:

Each instance represents a massive particle in three dimensional space, with a scalar mass and a position vector with respect to an origin.

**Properties:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Variable Type** | **Notes** |
| m | mass | The particle's mass |
| p | Vector3D | The particle's position |
| v | Vector3D | The particle's velocity |

**Constructors:**

|  |  |
| --- | --- |
| **Argument** | **Notes** |
|  | Default constructor to set postition and velocity to zero vectors mass to zero double and label to a blank string |
| Particle3D original | A copy constructor to create copies of Particle3d object instnaces |
| Java.util.Scanner myScanner | Constructor method that reads the values of the properties from a scanner object |
| Vector3D p, Vector3D v, double m, java.lang.String l | An explicit constructor to initialize the properties with specific values |

**Instance Methods:**

**public java.lang.String getLabel():**

Get the label of the particle.

**public double getMass():**

Get the mass of the particle

**public Vector3D getPosition():**

Get the position of the particle

**public Vector3D getVelocity():**

Get the velocity of the particle

**public double kineticEnergy():**

Instance method to calculate the kinetic energy of particle Kinetic energy calculated as 1/2\*mv^2

**public void leapPosition( double dt):**

A method to update the particle position for a given timestep and the current particle velocity vector, as r(t+dt) = r(t) + dt\*v(t);

**public void leapPosition( double dt, Vector3D force):**

A method to update the particle position for a given timestep, current particle velocity and force, as r(t+dt) = r(t) + dt\*v(t) +dt^2\*f(t)/2m;

**public void leapVelocity( double dt, Vector3D force):**

A method to update the velocity of the particle for a given timestep and force vector, as v(t+dt) = v(t) + dt\*f(t)/m

**public void setLabel(java.lang.String):**

Set the label of the particle

**public void setMass(double m):**

Set the mass of a particle

**public void setParticle3D(Vector3D p, Vector3D v, double m, java.lang.String l):**

Convenient setter method to set all properties of Particle3D instance simulatnously

**public void setPosition(Vector3D p):**

Set the position of a particle

**public void setVelocity(Vector3D v):**

Set the velocity of a particle

**public java.lang.String toString():**

toString method to print out properties of particle in useful String format

**Static Methods:**

**public static double gpEnergy(Particle3D orbiting particle, Particle3D origin particle):**

Instance method to compute the potential energy of the system as a double

**public static Vector3D gravitationalForce( Particle3D orbiting particle, Particle3D origin particle):**

Instance method to compute the gravitational force as a vector

**public static Vector3D separation(Particle3D particleA, Particle3D particleB):**

We use a static method to compute the relative separation of particles as a vector so we can build complex statements.

### N-body Class:

This class contains the main program used to simulate the evolution of the solar system as a function of time, with certain initial conditions.

MAIN METHOD

The main method will take three command line arguments. The first will be a file contain N-bodies(planets) data (Number of Particles, Label, Mass, Initial Position and Initial Velocity). The second file containing the parameters for the simulation( Number of steps, time step, constants). The third argument will be the name of an output file to which the program will write the trajectory data in a format compatible with VMD.

This main method will be used to simulate our solar system, hence it will be used in conjunction methods specific to calculating certain properties of celestial bodies, these are described below.

Data for the solar system will be taken from NASA/JPL’s HORIZONS system.

Static Methods

Public static void main(String[] argv)

The main method will:

* Read the particle data from the file.
* Read the simulation parameters from a file.
* Hold the particle data in an array of Particle3D objects.
* Use the velocity Verlet time integration algorithm to simulate the evolution of the system.
* Monitor fluctuation of the energies, to estimate the error in the simulation.
* Write the simulation data to a file in a format that is compatible with VMD.

The simulation data will be calculated by calculating the force on each particle due to all of the other particles and updating the velocity, position and energy accordingly using the velocity Verlet time integration algorithm.

Forces will be calculated by newton’s law of gravitation;

We will also calculate the total GPE of the system according to;

We must be careful not to double count the potential, i.e. include the potential energy due to particle B on A and the potential energy of particle A on B.

The program will record fluctuations in the total energy as the system evolves.

These should be minimal if the simulation is to be considered accurate. These values will be

Measured and recorded by the program at every time step.

Total energy will be calculated according to;

Instance methods:

**public Particle3D[] updateVelocity():**

Returns Particle3D array with updated velocities according the velocity Verlet time integration method.

**public Particl3D[] updatePosition():**

Returns particle3D array with updated position according velocity Verlet time integration method.

**public double totalE():**

Returns double representing total energy of the system in its current state, taking care to avoid double counting interactions.

**public double orbitNum();**

returns double representing the number of times a particle has completed an orbit of a centre of mass.

**Public Vector3D aphelion();**

returns Vector3D representing the aphelion of aphelion of a particles orbit(point in the orbit where the particle is furthest away from the centre of mass).

**Public Vector3D perihelion():**

Returns a Vector3D representing the perihelion of a particles orbit(point in the orbit where the particle is closest to the centre mass).