Rapolas Daugintis (Student ID:s1348455, Exam No.: B049522)
Rokas Zemaitis (Student ID: s1307874, Exam No.: B049470)

Design Document

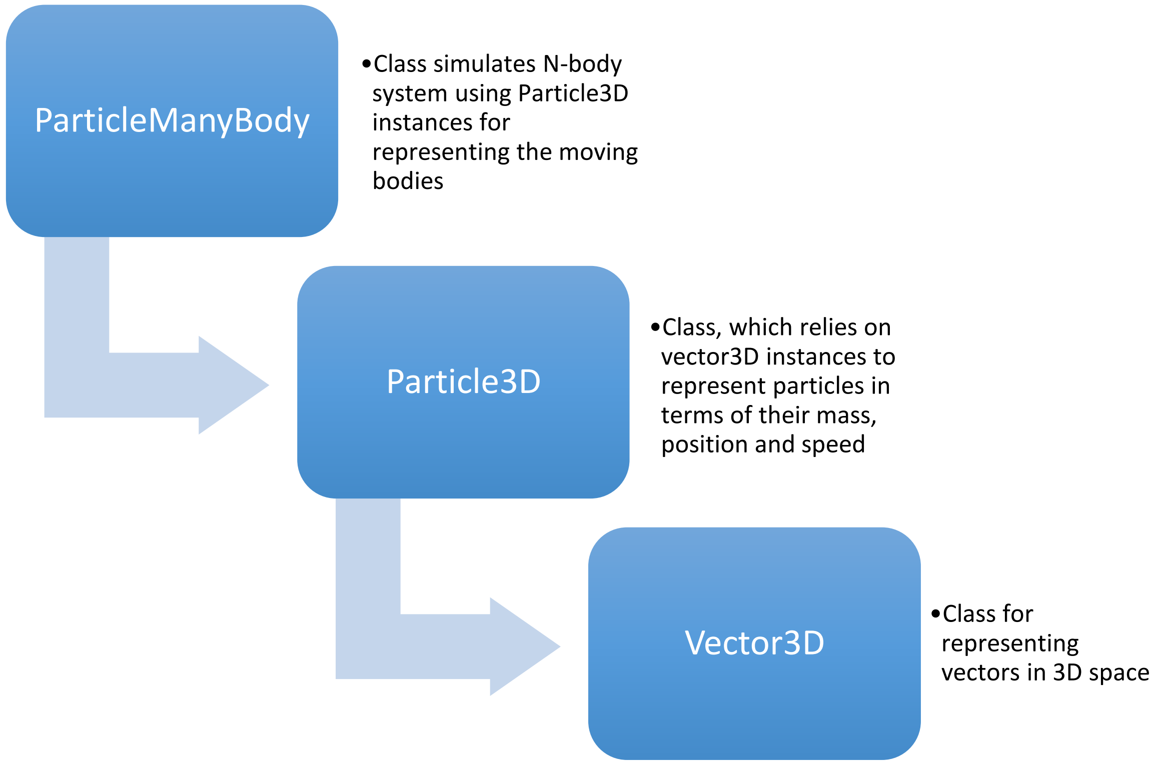
Computer Modelling Project A: N-body simulation

Overview

This program simulates N-body systems, which interact through Newtonian gravity. Velocity Verlet time integration method is used to evolve the trajectories of the bodies. The program will be used to simulate the Solar system, using physical parameters.

The purpose of this document is to describe how the program is implemented in Java. Classes, which are used, are detailed in terms of their methods and their hierarchy.

Class Layout


ParticleManyBody
•Class simulates N-body system using Particle3D instances for representing the moving bodies
Particle3D
•Class, which relies on vector3D instances to represent particles in terms of their mass, position and speed
Vector3D
•Class for representing vectors in 3D space

Vector3D Class

Each instance represents a vector in 3D space.

Properties

Name	Type	Notes
x	double	The x coordinate
y	double	The y coordinate
z	double	The z coordinate

Constructors

Arguments	Type	Notes
		Default constructor creates (0, 0, 0) vector.
original	Vector3D()	Copies the coordinates from the given vector.
xx, yy, zz	double	Creates a vector from explicit coordinates

Instance methods

Setters

`public void setVector(double xx, double yy, double zz)`

Sets x, y and z coordinates of the vector to the arguments specified.

`public void setX(double xx)`

Sets the x coordinate of the vector

`public void setY(double yy)`

Sets the y coordinate of the vector

`public void setZ(double zz)`

Sets the z coordinate of the vector

Getters

`public double getX()`

Returns the x coordinate of the vector

`public double getY()`

Returns the y coordinate of the vector

`public double getZ()`

Returns the z coordinate of the vector

Other Instance Methods

`public String toString()`

Returns String representation of the vector in a form “(x, y, z)”

`public double magSq()`

Returns the magnitude of the vector squared, i.e.
$x^2 + y^2 + z^2$

`public double mag()`
Returns the magnitude of the vector, i.e.
$\sqrt{x^2 + y^2 + z^2}$

`public Vector3D mult(double a)`
Returns the vector, multiplied by the specified double

`public Vector3D div(double a)`
Returns the vector divided by the specified double

Static Methods

`public static Vector3D addVector(Vector3D a, Vector3D b)`
Returns the addition of 2 specified vectors, i.e.
$(x_a + x_b, y_a + y_b, z_a + z_b)$

`public static Vector3D subVector(Vector3D a, Vector3D b)`
Returns the subtraction of the second vector from the first one, i.e.
$(x_a - x_b, y_a - y_b, z_a - z_b)$

`public static double dotVector(Vector3D a, Vector3D b)`
Returns the scalar product of the 2 specified vectors, i.e.
$(x_a x_b, y_a y_b, z_a z_b)$

`public static Vector3D crossVector(Vector3D a, Vector3D b)`
Returns the vector product between the 2 specified vectors, i.e.
$(y_a z_b - y_b z_a, z_a x_b - z_b x_a, x_a y_b - x_b y_a)$

Particle3D Class

Each instance represents a particle in 3D space, with its position, mass and velocity.

Properties

Name	Type	Notes
mass	double	Mass of the particle.
position	Vector3D	Position of the particle.
velocity	Vector3D	Velocity of the particle.
Label	String	Label of the particle.

Constructors

Arguments	**Notes**
	Default constructor sets all double variables to 0 and label to "untitled".
double mass, Vector3D position, Vector3D velocity, String label	Creates a particle with given mass, position, velocity and label.
Scanner scan	Scans values for Particle3D object from an input file

Instance methods

Setters

`public void setPosition(Vector3D p)`

Sets the position of the particle.

`public void setVelocity(Vector3D v)`

Sets the velocity of the particle.

`public void setMass(double m)`

Sets the mass of the particle.

`public void setLabel(String l)`

Sets the label of the particle.

Getters

`public Vector3D getPosition()`

Returns the position of the particle.

`public Vector3D getVelocity()`

Returns the velocity of the particle.

`public double getMass()`

Returns the mass of the particle.

`public String getLabel()`

Returns the label of the particle.

Other Instance Methods

`public String toString()`

Returns a String representing the particle in the form “label x y z”.

`public double kineticEnergy()`

Returns the value of kinetic energy.

`public void leapVelocity(double dt, Vector3D f)`

Time integration support: evolve velocity according to:

$$v(t+dt) = v(t) + \frac{f(t)}{m}dt$$

`public void leapPosition(double dt)`

Time integration support: evolve position according to:

$$r(t+dt) = r(t) + v(t)dt$$

`public void leapPosition(double dt, Vector3D force)`

Time integration support: evolve position according to:

$$r(t+dt) = r(t) + v(t)dt + 0.5\frac{f(t)}{m}dt^2$$

Static methods

`public static Vector3D particleSeparation(Particle3D a, Particle3D b)`

Returns vector separation of two particles as a difference between position vectors:

$\vec{a} - \vec{b}$

ParticleManyBody class

This class contains the main method that simulates a Solar system.

Main Method

`public static void main(String[] args)`

The main method reads in three file names as input from the command line:

- Details of bodies: labels, positions, velocities, masses;
- Simulation parameters: number of steps, length of time step, gravitational constant;
- Output file in .xyz extension to be worked on with VMD (a tool for visualising the motion of the bodies).

An array is made to store all the parameters of bodies which are read from the first file. A 2D array is created to store the old and new forces that the bodies are acted on each particle (old and new force values will be needed for time step integration). Before the simulation, the momentum of centre of mass is calculated:

$$P_{COM} = \sum_{i=1}^{N} m_i v_i$$

In order to have the stationary centre of mass of the solar system, this must be zero. For that, the velocity of centre of mass is computed:

$$v_{COM} = \frac{P_{COM}}{\sum_{i=1}^{N} v_i}$$

This velocity is then subtracted from all velocities of bodies in the system, such that $P_{COM} = 0$ condition is satisfied.

After that, the simulation starts, and then is integrated using the Velocity Verlet method, which at each time step for each particle:

- Updates the position using the current velocity;
- Updates the force using the new position;
- Updates the velocity using the new position and average of old and new force values.

During the simulation, several values are calculated for each planet:

- Fluctuations in total energy of bodies to estimate the error in the simulation (including the Sun);
- The amount of revolutions around the Sun;
- Aphelion and perihelion estimate.

After each time step, the coordinates and the label of each particle are printed to the file in the format which VMD can interpret.

Static methods

```
public static Particle3D[] leapPositionArray(Particle3D[] bodies,
Vector3D[][] forces)
```
Updates all positions of particles using the current velocities.

```
public static Vector3D[][] leapForceArray(Particle3D[] bodies,
Vector3D[][] forces)
```
Updates all forces using the position of particles.

```
public static Particle3D[] leapVelocityVerletArray(Particle3D[]
bodies, Vector3D[][] forces)
```
Updates all velocities using the position of particles and forces by Velocity Verlet method.

```
public static double potentialEnergy(Particle3D a, Particle3D b)
```
Returns the value of particle b's potential energy with respect to particle a.
Note: The static method for calculating potential is written in ParticleManyBody class so as to keep the Particle3D class universal for any particle simulations, because the potential energy may be calculated differently, e.g. intermolecular interactions or astronomical bodies.

```
public static String vmdEntry(Particle3D p)
```
Writes out particle's parameters in format suitable for a VMD trajectory file.