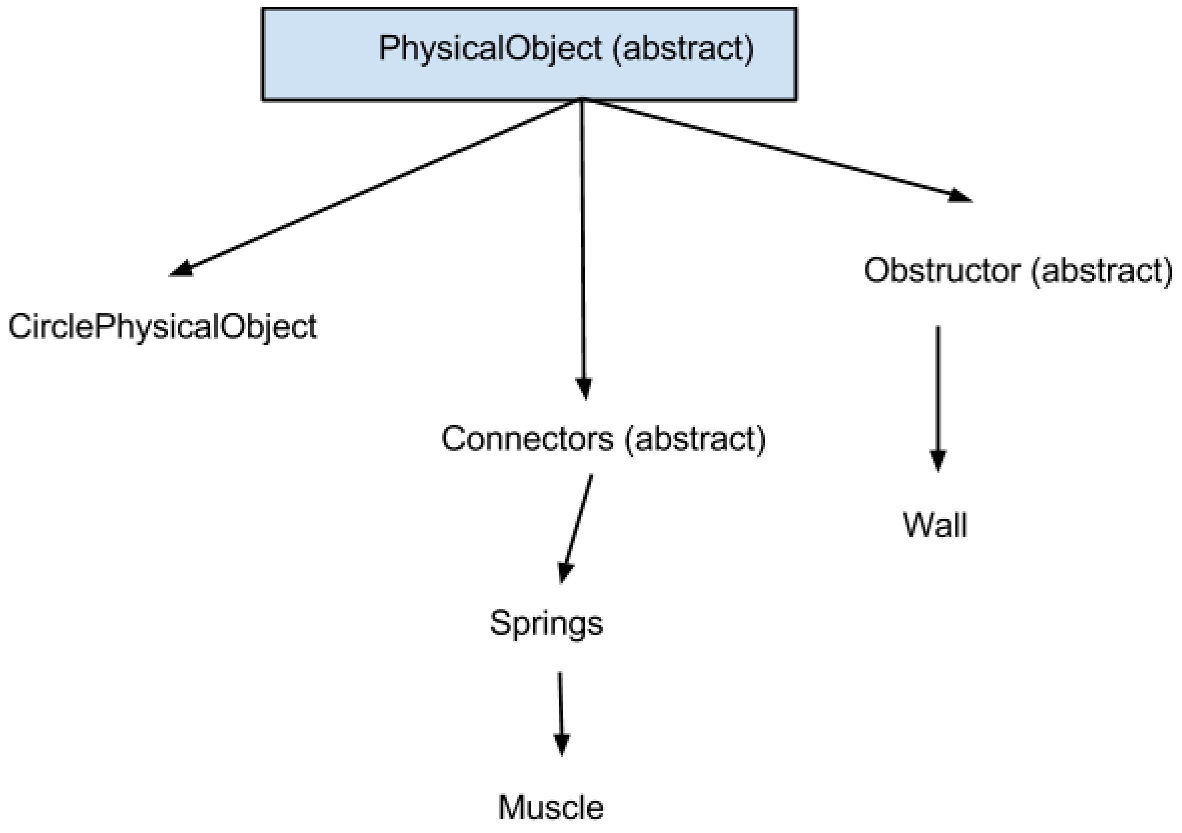
CSC308 Springies Design - Part 1

The overarching purpose is to read through XML files and to create a Physics simulation of gravity and Hooke's law based on the parameters indicated in the databases.

We will have 3 main superclasses: abstract PhysicalObject class, abstract Force class and an Environment class. Objects with masses will extend from PhysicalObject, which in turn extends from JGObject. The class hierarchy for the PhysicalObject classes are as follows:



PhysicalObject itself would have the following instance variables and methods:

Constructor parameters	**Instance Variables**	**Methods**
Constructor 1: String id - representing the id of the object, double x, y - representing the initial position of the object	Body myBody - a variable that stores the object's information in the physics world JGColor myColor - color of object	move() method - this move method gets the x and y position of the object in the physics engine and translates it to the game engine applyForce(Force f) - this

int collision id double xspeed, yspeed - representing the initial speed of the object Constructor 2: String id - representing the id of the object, double x, y - representing the initial position of the object int collision id	float rotation - rotation of the object String id	method applies the force to the object and the object shifts based on the type of force that is applied on it. getX() method - to get the x position of the object getY() method - to get the y position of the object createBody method - to create the body of the object in the physics world setPosition method - to set the position of the object's body in the physics world destroyBody method - to remove the object's body from the physics paintShape abstract method - this method will be implemented in its inherited classes to give it a shape

The main idea of creating a PhysicalObject class is to offer the option of extensibility to allow for one to create different kinds of objects in the screen, be it main masses or objects that connect the masses together (connectors and obstacles).

The following classes inherit from PhysicalObject - Connectors, Obstacles and CirclePhysicalObject (representing a mass that is circular in shape).

The CirclePhysicalObject represents an implemented PhysicalObject class and represents a physical circular object with a defined mass that will be connected to each other through connectors and will interact with each other in the physics world. Other classes of this type that extend from physical object could be SquarePhysicalObject or OvalPhysicalObject.

It will have the following constructors, instance variables and methods:

Constructor parameters	Instance Variables	Methods
Constructor 1: String id int collisionid JGColor color double radius Constructor 2: String id int collisionid JGColor color double radius mass Constructor 3: String id int collisionid JGColor color double radius String gfxname (name of Sprite from media table used) mass	double radius	init - this initializes the object based on the mass that is given to the object and places it in the physics world paintShape() - this paints the shape of the object onto the canvas of the main GUI exertForce() - this is an abstract method that each of the subclass of the Connectors are to implement

The Connector abstract class represents the objects that connect between masses and it will have subclass of Spring.

Constructor parameters	Instance Variables	Methods
Constructor 1: String id Body mass1 Body mass2 String mass1_id String mass2_id		paintShape() - abstract method init() - abstract method move() - this will provide the length and location of both ends of the connector and update the GUI with this information

The Spring class will implement the Connector abstract class and will have the following methods and constructors:

Constructor parameters	Instance Variables	Methods
Constructor 1: String id Body mass1 Body mass2 String mass1_id String mass2_id Constructor 2: String id Body mass1 Body mass2 String gfxstring String mass1_id String mass2_id rest_length Constructor 3: String id Body mass1 Body mass2 String gfxstring String mass1_id String mass2_id spring_constant Constructor 4: String id Body mass1 Body mass2 String mass1_id String mass2_id rest_length spring_constant	double spring_constant double rest_length double length	paintShape() - this will paint the shape of the spring on the GUI init() - this method will help create the body in the physics world and connect the 2 bodies in the physics world exertForce() - this method will calculate the force that is exerted by the spring on both of the objects that the spring is connected to set_length(double length) - this method sets the length of the connector

The Muscle class will extend from the Spring class as the muscle class only bears the difference in that it exerts a force influenced by the internal forces of the muscle.

Constructor parameters	Instance Variables	Methods
Constructor will have an additional parameter as	double internal_force	exertForce() - this method will overwrite the exertForce

compared to the Spring class of internal_force, signifying the force that the Spring exerts on the masses it is connected to.		method that is inherited from the Spring class calculate the force that is exerted by the muscle on both of the objects that the spring is muscle to

The Obstructor abstract class refers to any object in the physics world that the objects might run into. This includes walls or any other form of object that we may want to place in the physics world for the object to bounce off. Therefore, it would have the abstract method of exertForce(), that would allow the obstructor object to exert a force on the mass that bounces onto it.

The obstructor class would have the following constructor, instance variables and methods:

Constructor parameters	Instance Variables	Methods
Constructor 1: String id double force_magnitude double exponent Constructor 1: String id double force_magnitude double exponent double x double y	double force_magnitude double exponent	abstract init() - this method, when implemented initializes the obstructor object in the physics world, based on the information given abstract exertForce() - this method returns the force that is exerted on the object that collides with it abstract paintShape()

The Wall object is an implementation of the obstructor class and refers to the walls within the physics world. It will have the following constructor, instance variables and methods:

Constructor parameters	Instance Variables	Methods
Constructor 1: String id double force_magnitude double exponent String sideOfWall	double force_magnitude double exponent double start_position double end_position double length	init() - this method initializes the obstructor object in the physics world, based on the information given exertForce() - this method returns the force that is

Constructor 2: String id double force_magnitude double exponent double x double y Constructor 3: String id double force_magnitude double exponent String gfxname double x double y		exerted on the object that collides with it paintShape() - this method draws the wall based on where it is defined in the parameters exertForce() - returns the force exerted on the object based on the force_magnitude of the wall and the exponent

Force will be an abstract class with a method that makes overarches all possible forces. The reason that we have an abstract class is because all forces have direction and magnitude, and we will be able to extend more forces later on if we desire.

Force will be a subclass that extends from Vec2. The reason that we extend from the Vec2 class is because all forces can be thought of as a vector, and the Vec2 class simply has an x/y direction setup. This allows us to add other kinds of vectors in the future.

There are going to be five main forces that are subclasses of the forces superclass: gravitational force, spring force (which can be bidirectional), collisional force (which factors in collisions off of walls), viscous force, and center of mass force.

Gravitational force will have two components: 1) direction and 2) magnitude
Spring force: which grabs the spring constants from each spring and applies the force upon the two objects to which the spring is attached. The force is based on Hooke's Law: F= -k * (change in x)
Collisional force will be based on magnitude and some exponent (that will determine how far the colliding objects will travel.
Lastly, viscous force will simply have magnitude - this will determine how fast objects are able to travel through the environment.

Forces don't care about x/y position or even the physical location of the object. The forces will be enacted dependent on their location, but the forces themselves don't actually need to know where the object is located as they will be active at all times or when objects collide with walls. Instead, what will happen is the forces will be grabbed by the objects and then applied to themselves.

Lastly, there is the matter of center of mass force. The center of mass is where the center of all the conglomerate mass (the total sum of all masses) is located. This is what will determine the rate at which the objects will fall and will be useful for other calculations in the future.

Parsers

A final aspect of the design that will be critical for everything else to work will be to create parsers for the masses, springs, forces, and their respective subclasses. We will need to add some kind of data structure that will be able to take in all of the variables and then allow us to retrieve that information in order to generate constructors which will then allow us to place objects in our physical world while also allowing us to create the physical forces and connectors that we will need in order to simulate the given physics environment.

Constructor parameters	Instance Variables	Methods
Constructor 1: x_dir y_dir magnitude	double force_magnitude double x_dir double y_dir	init() - this method initializes the obstructor object in the physics world, based on the information given

Aside from these forces and objects, we need an environment/world to store all these objects. This environment would store all the objects, connectors and obstructors and also exert forces on the object. This environment will also communicate with the GUI what is happening on in the physics world - therefore, the environment will need to have a world variable that stores information of the physics environment

The environment will have the following instance variables and methods:

Instance Variables	Methods
- All the 4 walls surrounding the world - World variable, referring to the physics world - Force gravity - boolean gravityOn	- changeGravity - createObject - joinObjects - createObstructor

- List<PhysicalObjects>, consisting of the list of PhysicalObjects in the screen - double viscosity	- setViscosity