OOP
COURSE PROJECT:
FORCE REPRESENTATION

Team 7:

Phạm Việt Thành - 20194454 - thanh.pv194454@sis.hust.edu.vn

Nguyễn Tuấn Dũng - 20194427 - dung.nt194427@sis.hust.edu.vn

Phạm Văn Cường - 20194421 - cuong.pv194421@sis.hust.edu.vn

I. Assignment of members

1. Phạm Việt Thành - 20194454

a) Main class

- TestMotion class
- InvalidInputException class
- Integrating the custom exception into the main object classes.

b) GUI class

- Design the UI (Scene.fxml)
- Handle all animation class and functions
- Handle background scaling for main object corresponding to side-length and radius
- Handle all input text field, input dialog (with exception)
- Force slider handler
- Vector representation of forces: 30%

2. Phạm Văn Cường - 20194421

a) Main classes:

- force classes: 30%
- objects classes: 70%

b) GUI classes:

- Friction coef sliders
- Handled cases where the object's speed surpasses speed threshold
- Reset function

3. **Nguyễn Tuấn Dũng - 20194427**

a) Main classes:

- force classes: 70 %
- objects classes: 30 %

b) GUI classes

- Drag and drop feature
- Show and hide informations
- Vector representation of forces: 70 %
- Pause button

II. Mini-project Description

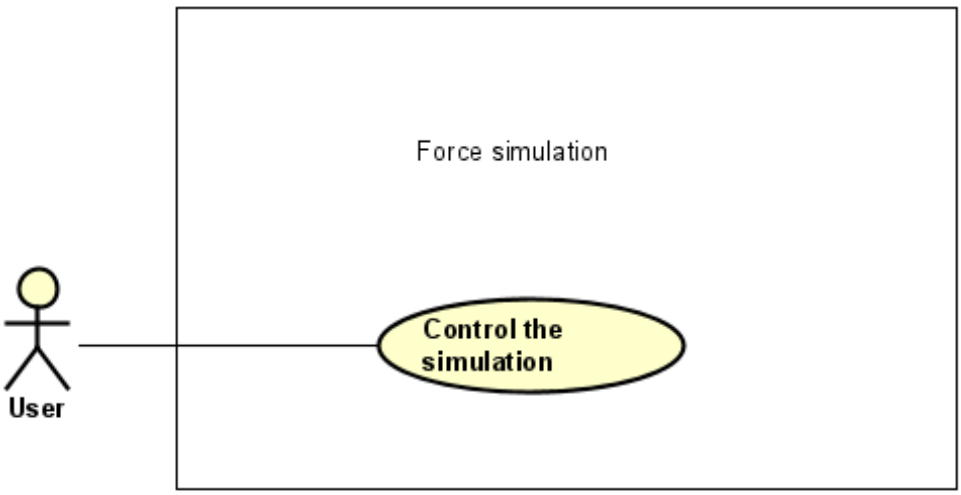
1. Mini-project requirements

- GUI: The design of this simulation is closely similar to this source: http://phet.colorado.edu/sims/html/forces-and-motion-basics/latest/forces-and-motion-basics_en.html
- In the simulation, the user controls a physical system. This includes three components, which are the main object, the surface (horizontal) and an actor force (horizontal). The user can control all the components and observe the motion of the main object.
- The main object: includes a cube-shaped object and a cylinder-shaped object. For each type of object, the user can specify the desired parameters as follows:

Object type	Desired parameters
Cube-shaped object	- Side-length (cannot exceed a maximum threshold) - Mass
Cylinder-shaped object	- Radius (cannot exceed a maximum threshold) - Mass

- To set up the main object, the user can drag one object (cube or cylinder) from the bottom left corner onto the surface, and an input dialog will pop up for the user to input the desired parameters.
- The actor force: the actor force is applied on the center of mass of the main object, it is represented by a horizontal arrow. The user can adjust the magnitude of the force by using the slider in the bottom center panel, or specify the value in the text field.
- The surface: The user can control the friction coefficients of the surface by using the sliders in the right corner. There are two friction coefficients: kinetic friction coefficient and static friction coefficient. The value of the static friction coefficient cannot be lower than that of the kinetic friction coefficient.
- Throughout the simulation process, the user can:
    - Change the applied force as well as the friction coefficients.
    - Pause, continue, reset the simulation.
    - Choose to show or hide detailed information such as directions and values of forces, mass, velocity and acceleration of the main object by using check boxes in the upper right corner.

2. Explanation of use case diagram



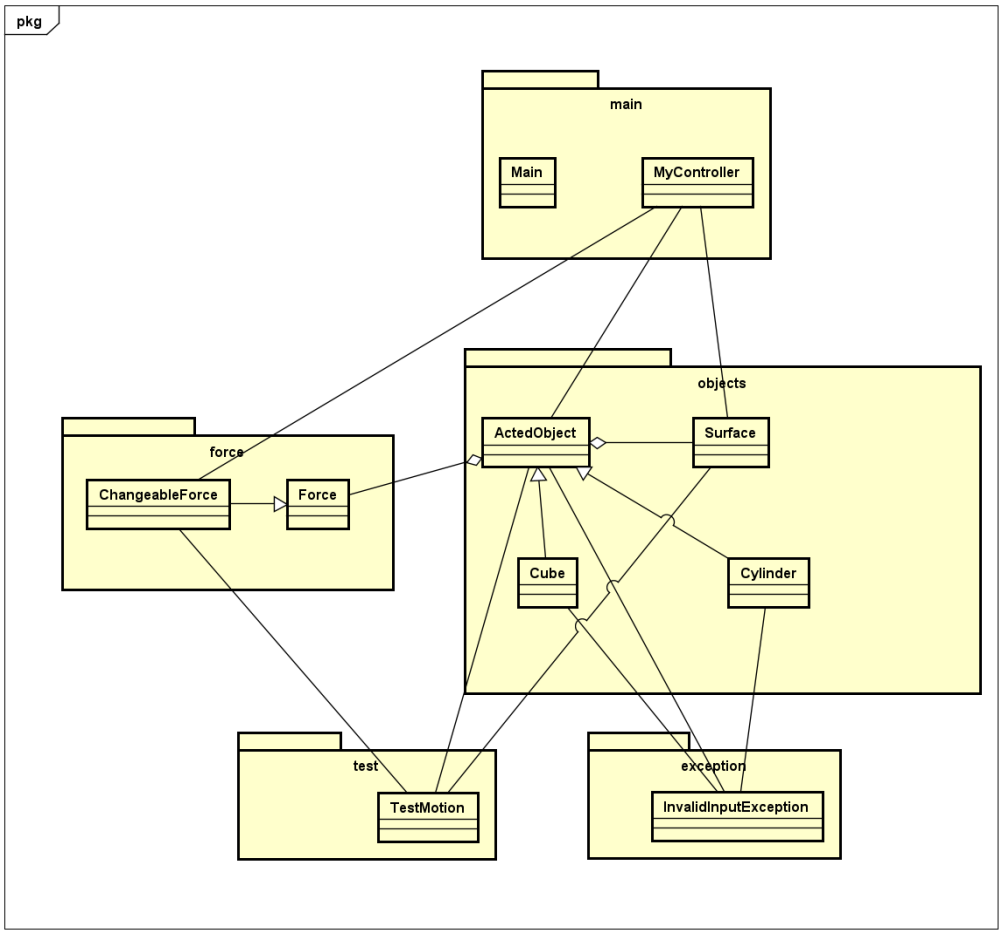
Our program has only one main use case: to set up and control the interactive simulation of composition of forces.

- After starting the program, the user can choose the type of object to simulate its forces and motions (either a cube or a cylinder) by dragging symbols from the bottom left of the screen. After choosing one of the objects, the user will be asked to enter the information of that object (mass and radius for the cylinder, mass and side-length for the cube).
- After completing these steps, the user can control the simulation by controlling the magnitude and direction of the acted force on the object (through a slider or an input field), or controlling the static and kinetic coefficients of the surface through sliders. While the program is simulating the motion of our object, the user can choose to show or hide information such as: directions and magnitudes of forces acting on the object and sum of forces, object's mass, velocity, acceleration, angular velocity and angular acceleration in the case of cylinder.

- The user can choose to pause and resume the current simulation.
- Finally, the user can press the Reset button to start a new simulation, or exit the program.

III. Design

1. General class diagram



We created five main packages for this problem:

- In the force package, we created a class Force to represent forces in general. ChangeableForce is a child class of Force. We will go into details about these two in the explanation for the force diagram.

- In the objects package, we created classes that represent two components of the simulation: the main object to apply force to, and the horizontal surface where the main object stays on. ActedObject has two children classes Cube and Cylinder. These two are slightly different in the way we simulate their motion as a cylinder-shaped object has both linear motion and rotational motion.
- The main package contains a javafx GUI main class that extends javafx.application.Application and a controller class.
- The exception package contains our custom Exception classes. We only have one, which will be used to handle the input cases (i.e. when the user input the main object's mass, radius or side length).
- The test package is only for testing the functionalities of the objects and forces before we use them in the GUI controller.

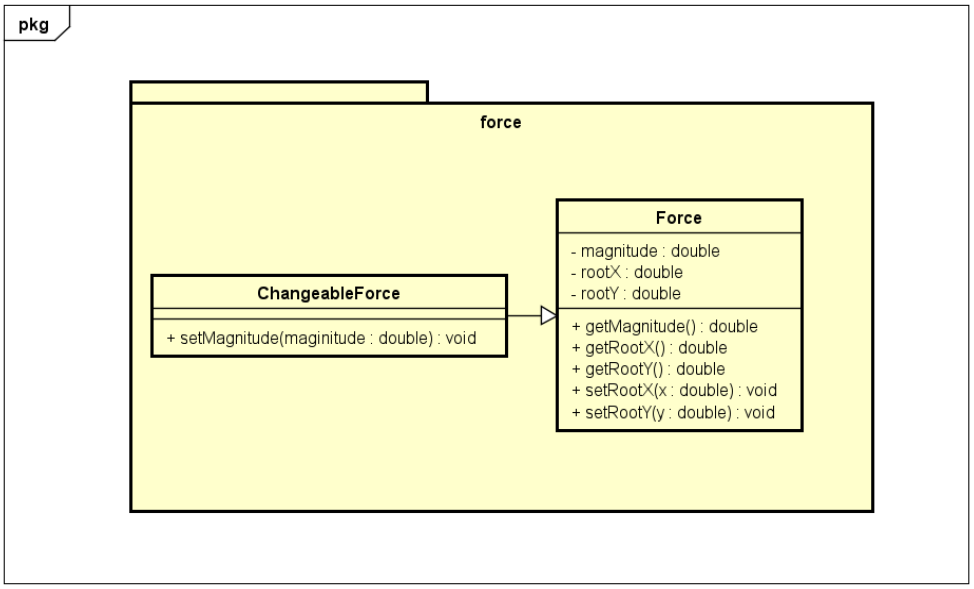
ActedObject aggregates Force and Surface as shown on the diagram. Because the actor that applies force on the main object doesn't need to be represented in this problem, we decided to make the horizontal force the actor itself. This may sound unintuitive at first but this is why we decided to design the system like that:

Every object is constantly affected by gravitational force. And since for this problem the main object constantly stays on the horizontal surface, it is constantly affected by a normal force and a frictional force if it ever moves. Thus, instead of creating other objects that constantly apply those forces on the main object, we decided to construct every force that applies on an object when the object itself is constructed. This kind of association between objects is certainly aggregation. This may not sound so convincing in the perspective of real world physics. But it makes the coding work tidier. We applied the same logic for the class Surface.

The system has three main components: a main object, a surface and an actor. We express this by using an ActedObject instance, a Surface instance and a ChangeableForce instance inside the MyController class. The visualization of the simulation is controlled through these 3 components.

2. Detailed class diagram

a. The force package

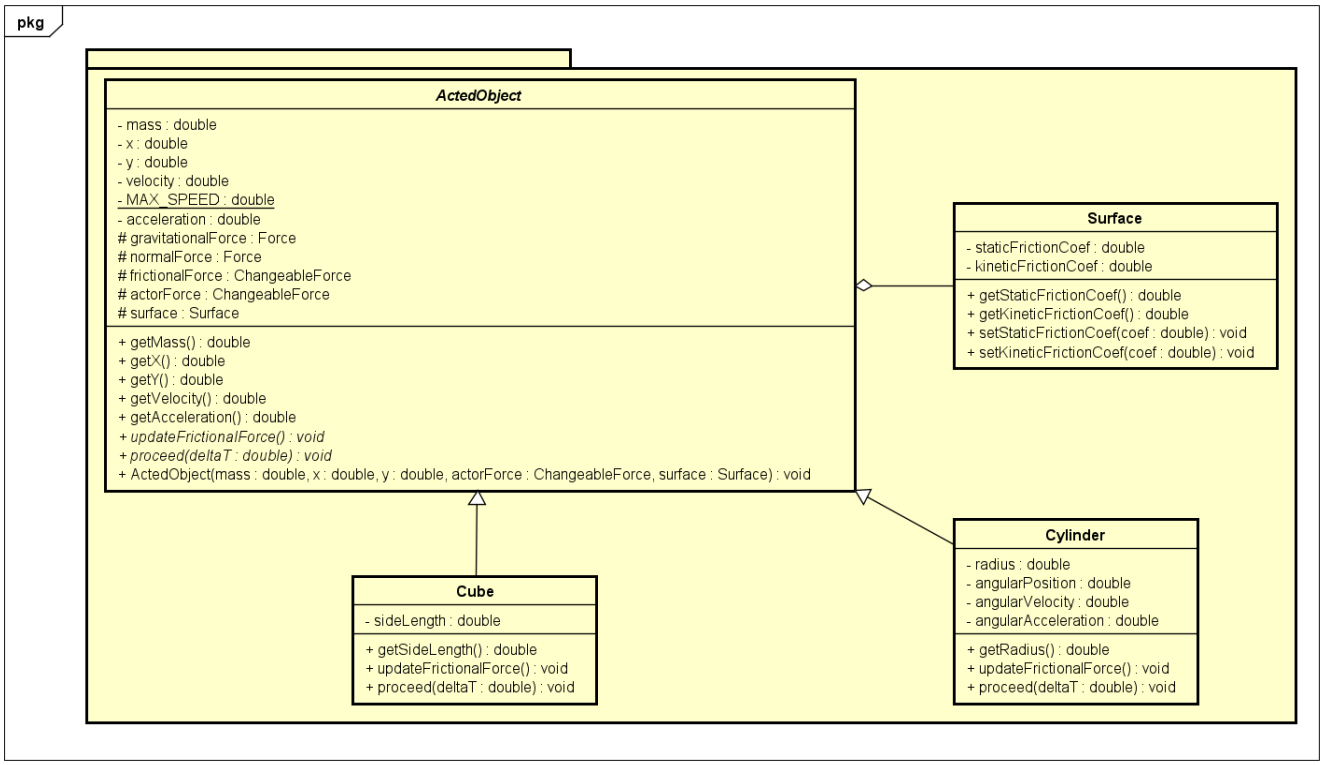


The Force package will contain classes that represent the forces acting on the object (gravitational force, normal force, frictional force and actor force).

- The Force class contains all the basic attributes and methods that every force acting on the objects share. This class is constructed by its magnitude (this attribute can be positive or negative depending on its current direction) and its root coordinates. We created the getters methods for these attributes so that we can use these values to calculate and simulate the object's simulation. We also created setters methods for the x and y coordinates, since the coordinates of the forces will change constantly with the object's motion, we would need to update their coordinates constantly.

- The ChangeableForce inherits from the Force class. This class represents the force that can change its magnitudes during the object's motion (frictional force, actor force). We added a magnitude setter for this class, so that we can update these forces' magnitudes accordingly.

b. The objects package



An Object instance has mass, x, y, velocity, acceleration. These attributes are private and their values can be obtained through getters. The value of these attributes are manipulated through various rules of physics. We implemented these rules in code inside the proceed(Double deltaT) method. There is a constant attribute called MAX_SPEED. This is used to limit the speed of the object. When the speed of the object surpasses this threshold, the actor must stop applying horizontal force on the object. There are 5 protected attributes: gravitationalForce, normalForce, frictionalForce, actorForce and surface. These are either Force,

ChangeableForce or Surface instances. All of these are constructed when an ActedObject instance is created. gravitationalForce, normalForce, and frictionalForce are constructed based on the attributes of the main object (i.e. x, y, mass). But actorForce and surface must be explicitly constructed and passed into the constructor of ActedObject as parameters. updateFrictionalForce() and proceed(Double deltaT) are abstract methods that Cube and Cylinder have to implement since they have different types of motion.

The Cube class and the Cylinder class are children classes of the ActedObject class to represent a cube or a cylinder. The Cube class contains the attribute sideLength, and the Cylinder class contains the attribute radius. Also these attributes have input conditions: the mass has to be positive, and the radius and side length has to be positive and above a specific threshold (60 m for the side length, and 30 m for the radius). If the user tries to pass in an invalid value for the constructor, the program will return our custom exception InvalidInputException, which prints out a message of the error for the user. We also created getter methods for these objects.

These two classes have different implementations of the abstract method updateFrictionalForce() of their parent class ActedObject. This method updates the value of the frictional force during our simulation. The frictional force will be calculated as provided in the Mini Project topics documents (Page 6). To keep things simple, we have to assume that the rotation motion of the cylinder is always rolling without slipping since including other cases will involve many other things like moment of inertia, torque,... which is out of the scope of this course.

The proceed methods from these classes are the methods that simulate the object's motion during the simulation. The method will update all of the object's properties that may change during the simulation: the velocity, acceleration, coordinates, forces' magnitudes and positions. This method has a passed in

parameter deltaT. The method updates the motion's properties every deltaT seconds (for the presentation, we set deltaT = 0.001s). By doing this, we can display the object's motion in real time.

The Surface class has two attributes: staticFrictionCoef and kineticFrictionCoef as well as their getters and setters. These are friction coefficients. An ActedObject instance will get the friction coefficients of the surface it's standing on through getters in order to calculate its frictional force.

c. exception package:

We didn't draw a class diagram for this package as there is only one class inside: the InvalidInputException class. This class extends java.lang.Exception and is thrown in the setter methods related to mass, radius and side length of classes in the objects package. It is used to check whether the input values provided by the users are valid or not. For the value of mass, it has to be positive. In the cases of radius and side length, they have to be positive and lower than the prefixed maximum values.

d. test package:

Again, we didn't draw a class diagram for this package as there is only one class with only one method (main). The use of this package is for testing the functionalities of Force, Objects and Surface, before fitting them into the GUI.

e. main package:

We didn't draw a class diagram for this package. The package contains a Main class which extends JavaFX Application class, MyController which controls the GUI, Scene.fxml which contains the design of the GUI and application.css which helps make a better style for the GUI.