## Hanoi University of Science and Technologies

IT3100E-123219 Object-oriented Programming

## Semester 20202

# 

## Mini-Project

# Interactive simulation of composition of forces

## Group 16

### Hoang Nguyen Minh Nhat - 20194445

### Pham Thanh Hung - 20194437

### Tran Quoc Lap - 20194443

# 

# Assignment of members

* + Detail for classes/methods

Minh Nhat: create class Force and other methods related to forces: frictionForce,..., class ArrowAnimation to demonstrate forces.

Quoc Lap: create class Object (inherited by Cube and Cylinder), GUI for Set-up Objects and Show/Hide Details and other methods related to Object

Thanh Hung: create class Monitor and surface classes, create Background GUI and actor GUI, help other members with their problems and optimize the application.

* + Claim clearly if you copy/copy with modify/use the idea of any source. Otherwise, you will receive 0 for the midterm score.

# Mini project description (Nhat)

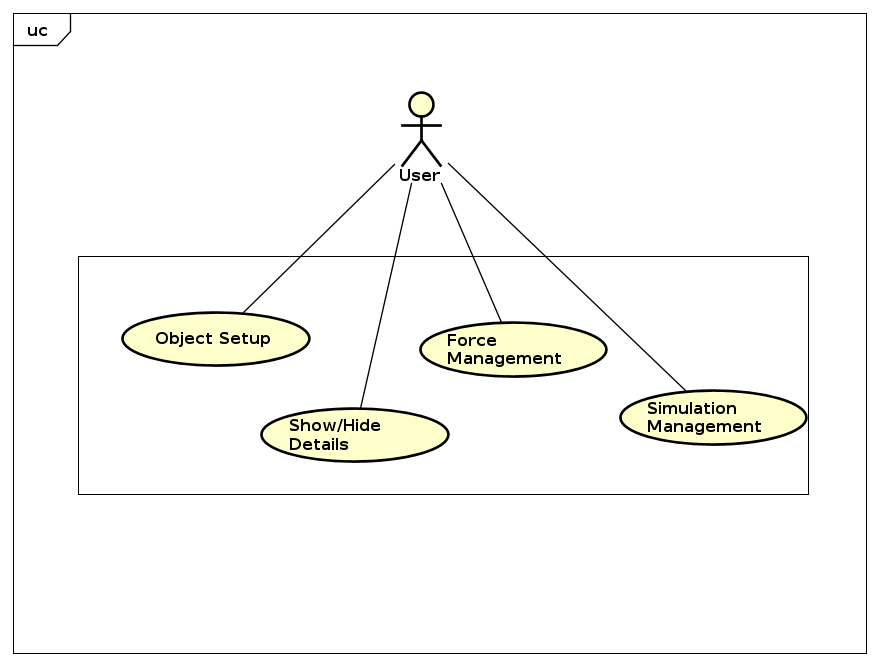
* + Describe in detail about your mini project requirement

The main requirement of our project is to create an application which simulates some basic Newton forces’ behavior and allow users to adjust some attributes for better observation.

In detail, we’ve created classes that facsimile the basic property of Force and its behaviors in real life (Force, Surface, Object - Cube and Cylinder). Along with them are some classes that combine the force and objects together (Monitor) and show animation on screen (the Animations).

* + Use case diagram and explanation: How the users interact to the software with use cases

There are 4 main use-cases that users can interact with the application:



1. Force management  
   Users can adjust the acting force’s power and direction through a slider. When the power value is positive, the acting force’s direction is left to right, vice versa. The application will display an arrow of which length is changed according to the value of force’s power.

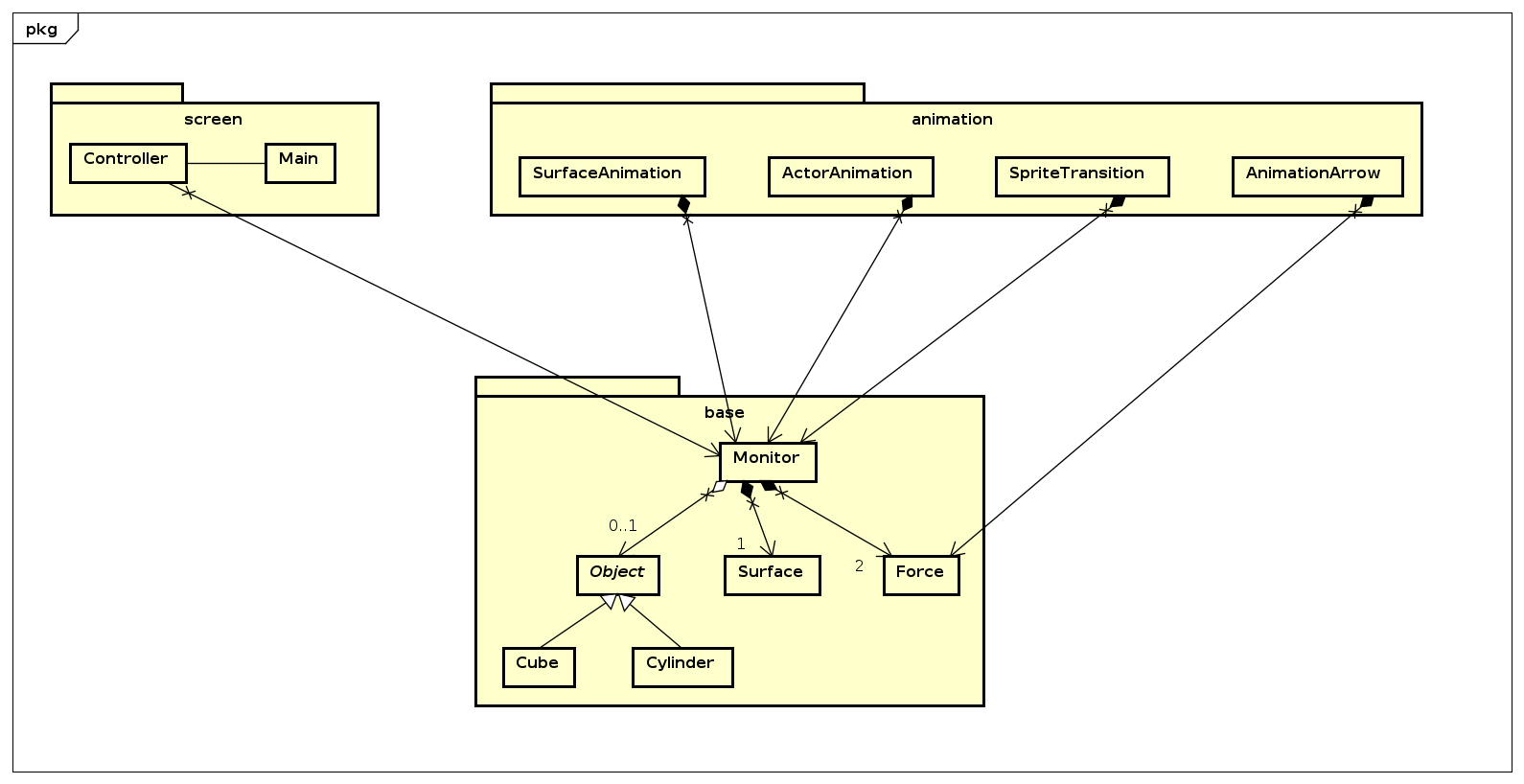
There is also a friction force caused by the surface to the object. This force has power depending on many conditions and is displayed similarly to the acting force.

Those two forces are computed to one total force to know the sum of forces that makes an object move or stay.

1. Simulation management  
   Users can choose to stop the simulator as well as to continue it.   
   Users can also reset the simulator to original default value
2. Show/Hide details  
   Users can tick boxes to choose to show the detailed information of the simulator or not. Details are the number value of object property and animation property (velocity, force power, acceleration, …)
3. Object set-up

This use-case includes: Putting an object onto the road, Lifting the object off the road, Changing the size of the object, Changing the mass of the object.

# Design (Hung & Lap)

1. A general class diagram: Class diagram may be with packages, including all classes without attributes/operations

What are they?

-> `base` (contains basic component that can be simulated via command-line interface)

-> `animation` (contains GUI component illustrating animation and transition of objects)

-> screen (visualize the simulation in a GUI using things created in `base` and `animation`)

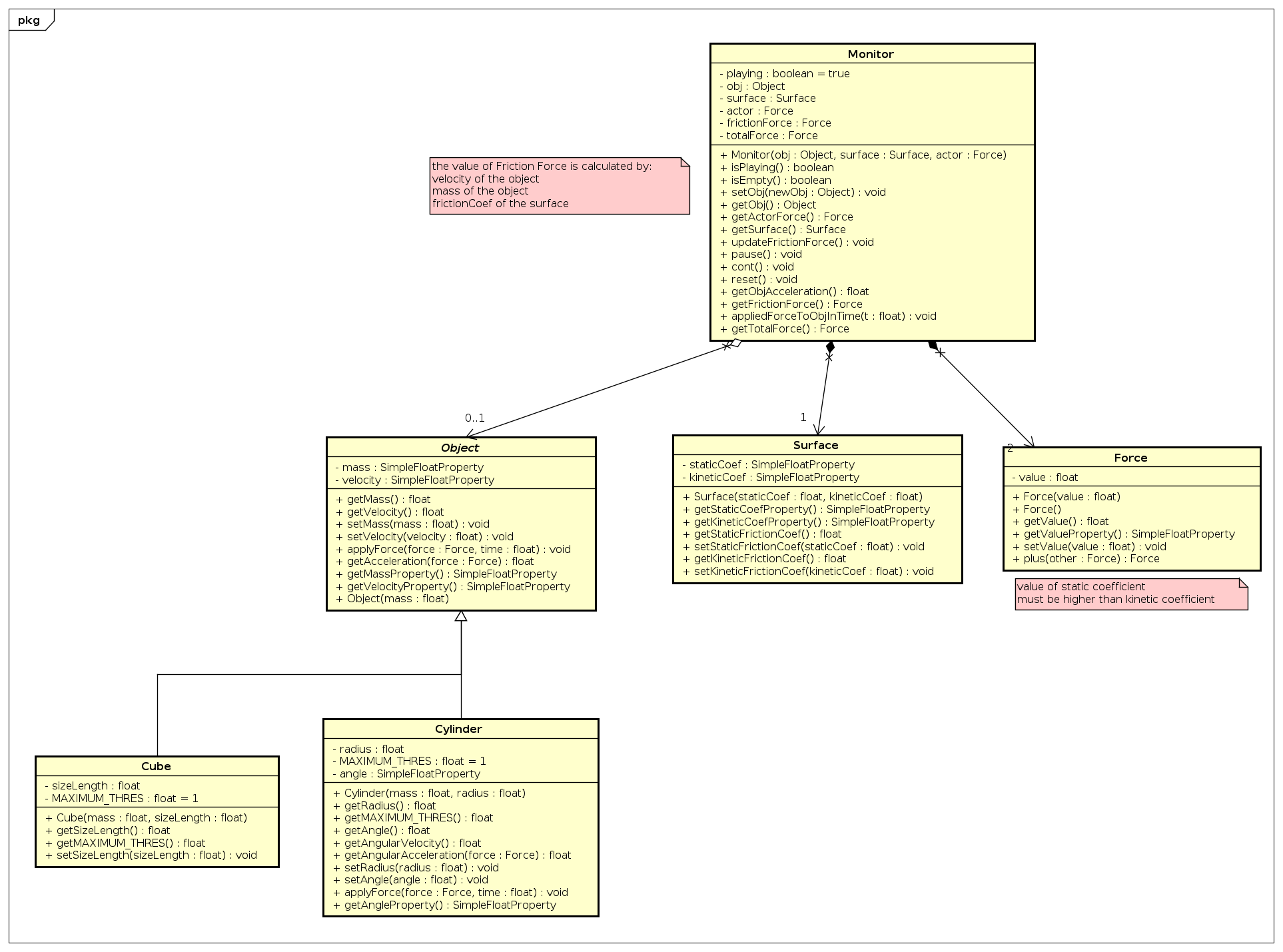
Why did we break them down that way?

-> reasonable with regard to the afore-created use-case diagram,

-> easy in management (:-)).

2. Several class diagrams for each package or several packages, with detail attributes/operations for each classackages, with detail attributes/operations for each class

What are they?



-> Under package `base`:

Force: attribute `value` (cardinality) of type SimpleFloatProperty so that we can capture any changes.

Object: abstract class contains attributes and methods that both Cylinder and Cube have in common: attributes mass and velocity (of type SimpleFloatProperty), methods: applyForce, getters and setters.

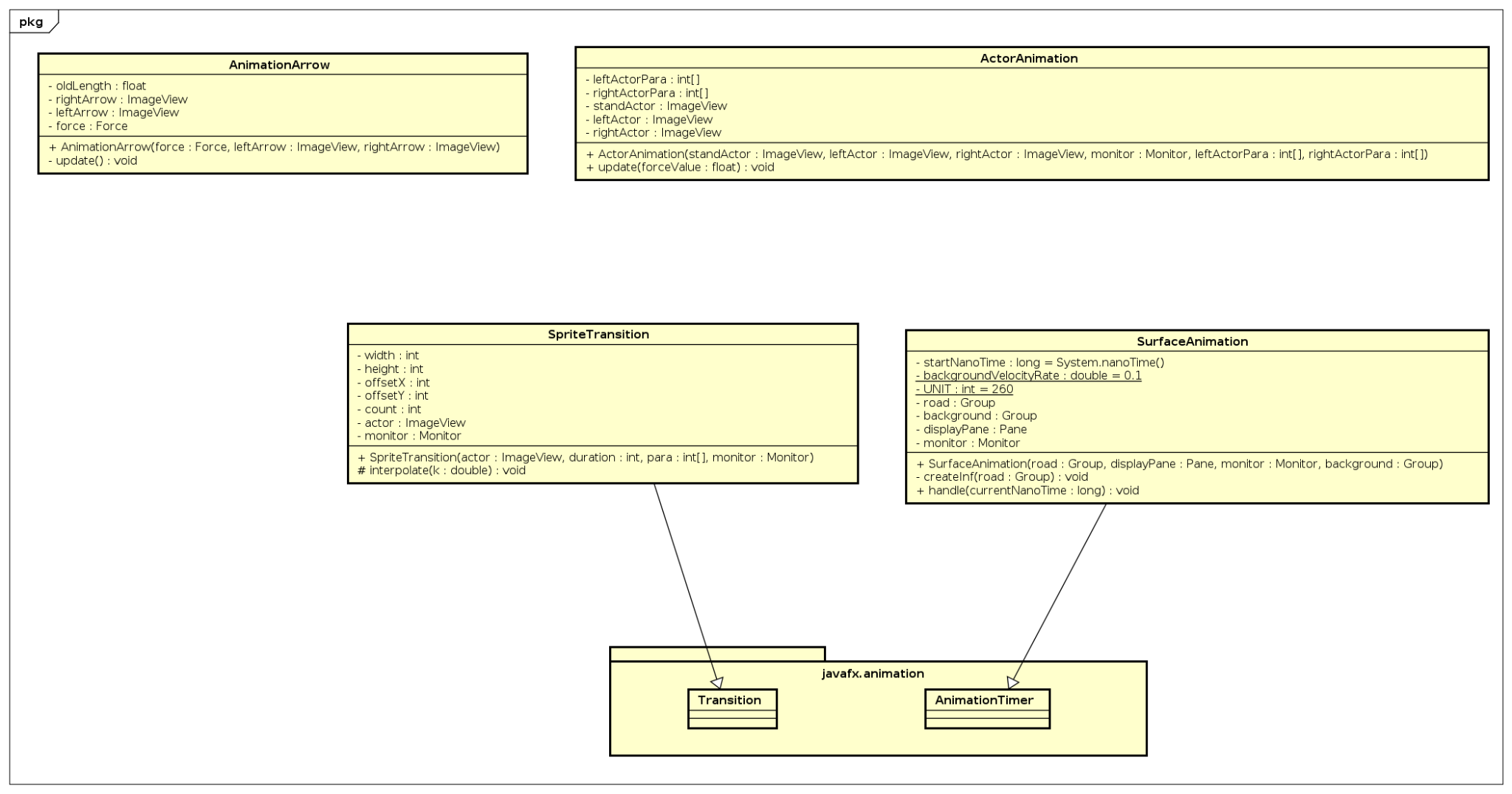
Cube: inherited from Object (Not java.lang.Object) with additional attributes: sizeLength (not exceed MAXIMUM\_THRES) and additional methods

Cylinder: inherited from Object (Not java.length.Object) with additional attributes: radius (not exceed MAXIMUM\_THRES), angle (of type SimpleFloatProperty angle rotated if the cylinder is rolling on the road) and additional methods.

When a force is applied on an object, the `applyForce` in Object will be called and make changes to the states of the object (velocity, acceleration, angle,...)

Surfaces: 2 attributes `kineticCoef` and `staticCoef` (of type SimpleFloatProperty), with getters and setters.

Monitor: a simulator of Newton’s law with fields: an Object instance, a Surface instance, 3 Force instances (actor , frictionForce, totalForce). Because users can pause the simulator whenever they want, we introduce a boolean attribute `playing`.

-> package `animation`:

AnimationArrow: We use an arrow as a GUI illustration of a Force on the screen so this class needs attributes: Force (of course :->) (composite relationship), rightArrow and leftArrow (ImageView for right and left direction accordingly), oldLength (float, in order to change the length of illustration according to Force.value). Thanks to the SimpleFloatProperty datatype of Force.value, anytime Force changes its value, the AnimationArrow will change the length of the force illustration.

ActorAnimation: in our screen, the dinosaur presents as the one who applies force on the object. We want the dinosaur to move left or right according to Force’s direction. ActorAnimation is responsible for this task. It works dependently on the state of the monitor so we set a composite relationship between ActorAnimation and Monitor.

SpriteTransition: We want the dinosaur to stamp as if it was running, and the speed of stamping is slow or fast according to the velocity of the object. SpriteTransition is responsible for this task.

SurfaceAnimation: We want the road to move slow or fast according to the velocity of the object. SurfaceAnimation is responsible for this task.

-> package `screen`

This is the place where our GUI application starts. It has 2 classes: `Main` and `Controller`

Controller: this is the controller for the FXML file of javaFX application.

This class will enable users to do the following tasks:

- drop & drag an object onto the road, out of the road.

- change the mass of the object via a JFXTextField text field component.

- change the force by sliding over the JFXSlider sliders component.

- show/hide details about forces, friction coefficients, velocity via checkboxes.

- play/pause/reset with 3 buttons

When the user drag and drop object, the `onDragDetected`, `onMouseDragged`, `onMouseReleased`, `onMouseDragReleased` will be called. If the user drop the object successfully onto the road then sliders, checkboxes will be editable, otherwise they won’t.

When the user changes the value of object’s mass, the value of force and friction coefficients, the changes will be captured thanks to `SimpleFloatProperty` data type of attributes in Objects, Surface, Force classes.

By designing the project in this way, we can work simultaneously on different parts of the project.

