(README file and UML should give the graders quick insight into the purposes are for every class, interface, etc. that you include in your model.

interfaces, classes, methods and variables you need.)

Our project is to build an application that helps to create simple but effective 2D animations from shapes. The current assignment is to design a model that represent an animation.

In our design, we have three interfaces, and a serials of abstract classes and concrete implementations were created. A short explanation of the methods in the interface and the implementing class follows:

1. Interface of shapes -- IShape.

This interface allow user to create and manage various kinds of 2D shapes, although currently we have only included rectangles and ovals.

Both shapes have a name, a reference position(a class point2D was created to represent the x and y coordinates of the reference position), color(represented by ColorRGB class), shapeproperty(refer to width and height in rectangles and x, y radius in ovals, see class JavaDoc for details), and a TimePeriod (a representation of the appearing time and disappearing time of the shape).

1. getName();
2. getPosition();
3. setPosition(Point2D point2D);
4. getColor();
5. setColor(ColorRGB color);
6. getShapeProperty();
7. setShapeProperty(ShapeProperty shapeProperty);
8. setPeriod(TimePeriod t);
9. getPeriod();

From (i) to (ix) are a series of getters and setters for user to get shape information and modify shape.

1. getShapeType(); an enum class was created to represent the type of shapes.

1. copyShape(); a method that return a copy of a shape. The copy is passed into transformation classes to be mutated according to user input.

An abstract class AbstractShape was created to prevent code duplication. All the member variables are initialized in the constructor.

The two implementing class is Rectangle and Oval. They both override the copyShape() and getShapeType() methods in the class.

1. Interface of transformations -- ITransformation.

This interface allows user to create and modify a transformation of a shape.

The methods included in the interface as followed:

1. getTransShape(); return the shape after it is transformed.
2. getTimePeriod(); return the time interval of the transformation.
3. getTransType(); return the type of the transformation.

A abstract class AbstractTransformation was created for reducing code duplication.

For each transformation, a copy of a shape is passed in to be mutated, the same copy is reused for all the transformations of that shape. This will allow the shape to demonstrate different transformations that are happening at the same time at a certain tick.

The transformations also have a member variable timePeriod representing the time interval of the transformation. This time interval must be within the time interval of the shape’s existence.

For this assignment, we included three transformations as classes implementing the interface:

1. Move -- a shape can move from a reference point to a new reference point on the canvas within a certain time interval while the shape is present. The whole shape has to stay inside the canvas after the move. The new reference point can not be the same as the original reference point.

This class has two additional field properties “fromPosition” and “toPosition” indicating the current position of the shape and the next position after the move.

1. Scale -- a shape can change one or two of its dimensions within a certain time interval while the shape is present. The whole shape has to stay inside the canvas after the transformation.

This class has two additional field properties “fromShapeProperty” and “toShapeProperty” indicating the current dimensions of the shape and the new dimensions after the change.

The whole shape has to stay inside the canvas after the change. We have to make sure the new dimensions won’t have the shape go out of boundary.

1. ChangeColor -- a shape can change its color within a certain time interval while the shape is present. This class has two additional field properties “fromColor” and “toColor” representing the current color of the shape and the new color after the change.

The member variables of the transformations are all initialized in the constructor. A toString method is included to generate the text description of the transformations.

1. Interface of animations -- IAnimator.

This is the interface where the animation is realized in the model.

The methods are as followed:

1. addShape(IShape s); add a shape to the list of shapes.
2. removeShape(IShape shape); remove a shape from the list.
3. getShapeList(); get a list of shapes in the animation.
4. addTransformations(ITransformation t); add a transformation to the list.
5. removeTransformations(ITransformation t); remove a transformation from the list.

1. getTransList(); get a list of transformations.
2. getTransInfo(); return a text description of all the transformations in the animation.
3. getShapesInfo(); return a text description of all the shapes in the animation.
4. getShapeAtTick(); return a list of shapes at a certain tick. The list can be used for actually playing the animation.

There is one implementation class of this interface, AnimationModel.

It has two member variables. The shapeList contains all the shapes in the animation. The transList is the container for all the transformations of the animation. The user can add shapes and transformations into the list if there is no duplication in the list. Further, the animation must adhere to certain constraints. For example, one cannot have move the same rectangle to the left and right during overlapping time intervals.

User can also delete unwanted shapes or transformations from the corresponding list.

We choose the List as the container for our data because it is easy to filter and sort with the stream API.

In our design, we use one copy of a certain shape for all the transformations that belongs to that shape. This will ensure the shape can demonstrate all the transformations that are happening at the same time at a certain tick.

A method getShapeAtTick() is placed in the interface as a placeholder for now. For the next assignment, when the controller calls model and pass in a certain tick, we can filter through the list of transformations and get a list of transformations that should happen at that tick, and calculate the corresponding values of the member variables of a shape with the fomula provided. Lastly we can come up with a list of shape at that tick.