**Overview:**

This project combines 2D shapes and animation to create something visually appealing. It allows for users to add shapes and edit the animation with respect to shape position, shape color and shape size. The project was completed in a group of three for the CS 5004 class at Northeastern University.

**Model Design Choices and Decisions:**

The project consists of:

* **Interfaces**
  + **IShape**
  + **AnimationChanges**
  + **EasyAnimatorModel**
* **Classes**
  + **AbstractShape (Abstract Class implements IShape)**
  + **Rectangle (extends Abstract Shape)**
  + **Oval (extends Abstract Shape)**
  + **MoveShape (implements AnimationChanges)**
  + **ChangeColor (implements AnimationChanges)**
  + **ScaleShape (implements AnimationChanges)**
  + **EasyAnimatorModel (implements EasyAnimatorModel)**

**IShape:**

This interface defines the methods that are going to be implemented on different shapes. It consists mainly of a constructor, getter and setter methods.

**AbstractShape (implements IShape):**

The AbstractShape class was created to reduce duplication of variables and functions that would be used by both the Rectangle and Oval classes. It implements most of the methods outlined in IShape. However, methods such as getPerimeter() and getArea() amongst others, are implemented in the classes that extended AbstractShape.

**Rectangle (extends AbstractShape):**

This class represents the Rectangle shape. It extends the class AbstractShape because a lot of  
its methods and fields are common to all shapes. Rectangle also contains a toString() method that displays its original state.

**Oval (extends AbstractShape):**

This class represents the Oval shape. It extends the class AbstractShape because a lot of  
its methods and fields are common to all shapes. Oval also contains a toString() method that displays its original state.

**AnimationChanges:**

This interface defines the methods that change the state of an IShape object. This interface  
will be implemented by class that represent the changes. We felt that this interface was required since classes that mutate the fields of IShape Objects share a lot of commonalities.

**MoveShape:**

This class represents the changes to the IShape Objects (x,y) position. It allows the user to move the  
IShape object from its current position to the new position on the 2D plane.

**ChangeColor:**

This class represents the changes to the IShape Object's color. It allows the user to change a shape’s color to any within the RGB spectrum.

**ScaleShape:**

This class represents the changes to the IShape Object's size. It allows the user to resize the shape to either a custom length and width or scale the Object using a scale factor. This class involves overloading of constructors.

**EasyAnimatorModel:**

This interface was created in order to allow easy testing of our controller. It also hides the implementation of the Model from the clients.

**EasyAnimatorModelImpl:**

This class serves as the **Model** class of our animation. The class is responsible for implementing the logic  
that updates the state of objects to eventually pass that information to the controller. The inputs from the controller will call methods in this class to update/mutate our shape objects.   
The class utilizes two types of data structures namely, an ArrayList<IShape> and a HashMap<String, AnimationChanges>. The ArrayList is responsible for storing original states of all shapes present in the animation, while the HashMap stores the mutations that have taken place to each shape. The ArrayList stores a clone of the original shape Objects.

The class contains a toString() method that when called returns a textual representation of the animation.

**View Design:**

As of right now, the only view possible with this Model class is the Text View.

**Text View:**

The text view allows the user to:

* View a full description of the original states of the shapes that are present in the animation.
* View all the changes made to every shape present in the animation.
* View changes made to a specific shape present in the animation.

**Run Design:**

In order to get a textual view of the animation, the user can call upon the toString() method of the EasyAnimatorModelImpl class which will print the entire animation to the console.

The user could also call upon the viewShapeJourney() method to view a particular shape’s cycle throughout the animation.