In this assignment we work on three views for the Easy Animator application.

First of all , there is a **IView interface**. The interface has all functionalities for the three different views and then individual views suppress or provide defaults for functionalities they do not implement. The methods in the interface as followed:

1. SwingPanle getPanel()
2. Void refresh()
3. String getViewType() : return the type of the view. The types are “text”, “svg”, or “visual”.
4. Void getOutPut()
5. String play()
6. And (2) are implemented in the visual view. (1) is for getting the swing panel to set list of “shapes at tick”, (2) is for repainting the canvas to form the animation. (4) and (5) are implemented in the text view and SVG view. (4) is for generating the text description of the animation and (5) return the output.

Secondly, we implement the **AnimationBuilder interface**. This interface function as an adaptor between the model we created in the last assignment and the file input for populating the model. In order to accomplish the adaptation, **we make the following changes to our model:**

1. Added a new constructor in the AbstractShape class with “name” as the only parameter for the construction of a IShape.
2. Removed “timePeriod” from Rectangle and Oval class constructors. We set the time period (the time period from showing to disappearing) for a shape after the given file is completely parsed.
3. Added a new transformation type “NoChange”. This is the transformation when the shape doesn’t change position, color or size.
4. Added “from attributes” in all transformation classes to store the properties before every transformation(in the previous version,we only have “to attributes” in every transformation class), and provided getters for them in the ITransformation interface.
5. Added “x”, “y”, “width” and “height” as new member variables in AnimationModel. These data determine the size of the canvas. Getters for these new member variables are also added.
6. The “get shape at tick” method was completed in the AnimationModel. We first abstained the list of all transformations and filtered out a list of transformations happening at a given tick. When the list of “shape at tick” is null, the animation stops.

Then the tweening formula were applied to calculate the change to an attribute of a shape based on the type of transformation. A copy of the shape is set with appropriate attribute information and added to the list of “Shape at tick” if it is not in the list. If the shape already exist in the list of “Shape at tick”, which means at a certain tick, this shape is going through multiple transformations, the new change of the attribute is added to the same shape that is already exist in the list. At last, a list of “shape at tick” is returned.

After making necessary changes to the model, we built the **AnimiationBuilderImpl** which implement the AnimationBuilder interface. The main method in the class is “addMotion”. This populate the transformation list of the model. We first go through the shape list to find the shape with matching name of interest. If the shape doesn’t have any kind of transformation, the initial state is then applied to the shape and the first transformation of the shape is added to the transformation list. If the shape has already gone through some kind of transformation, the same shape is passed into the transformation constructor for building a new transformation. Because the name is unique to every shape, once we find the shape with matching name and apply attribute information and transformation to the shape, we can break out of the loop.

Second to last step, we **create the three views** implementing the IView interface.

1. TextView:
2. SVGView:
3. VisualView:

We first set up a frame with the canvas size information from the model. Then we added a scrollPane and a SwingPanel in the frame. The SwingPanel is the same size as the frame. The **paintComponent** method in the SwingPanel draws the shapes with appropriate dimensions and colors base on the information from the list of “ shape at tick”. This list is generated by the model and passed in the SwingPanel through the **setShapeAtTick** method in the SwingPanel. The VisualView will **refresh(repaint)** itself at a certain time interval base on the speed of the animation.

This is realized by a timer in the preliminary controller and the controller serves as an action listener for the event.

In the assignment, we use the **Easy Animator** as the entry point of the program. A preliminary controller is created for better segregation of responsibilities. After parsing information from the command line arguments, the **AnimationBuilderImpl** populates the model, the **viewFactory** generates the corresponding view. **Model, view and the tempo(speed)** are then passed into the controller. Inside the controller, one of the three views is called base on the type of view.