In this assignment you will write definitions of three classes: PathAnimation, LinePathAnimation, and ArcPathAnimation. The PathAnimation class will be abstract. The LinePathAnimation and ArcPathAnimation classes will extend the PathAnimation class. They will not be abstract. Once you have written these class definitions, you will be able to test them using a program called AnimationApplication. The following statements appear in the main method of this program:

```java
GObject ball = new GOval(0, 0, 100, 100);
GPoint offset = new GPoint(-50,-50);

GLine line = new GLine(100, 300, 600, 300);
add(line);

PathAnimation animation1 = new LinePathAnimation(ball, offset, 200, 25, line);
animation1.show(this);
animation1.animate();
animation1.hide(this);
```

These statements cause Java to open a window and animate a moving ball, as indicated by the diagrams above. The program draws and erases a series of images. Each image is called a “frame”. Each frame shows the ball in a different position along a path. The two animations are quite similar. They differ only in the path along which the ball moves. This observation leads to a strategy for designing and implementing the three classes. The PathAnimation class should be responsible moving an object along a path, repeatedly drawing and erasing the object along the way. The LinePathAnimation and ArcPathAnimation classes will each implement a different kind of path.
1. Implement the **PathAnimation** class:
   a. Define **PathAnimation** to be a public abstract class. Provide the **PathAnimation** class with the following protected instance variables:
      i. A **GObject** instance variable (**figure**) representing the object to be animated;
      ii. Two **int** instance variables: the **duration** (number of frames) of the animation, and the time **delay** (milliseconds) between successive frames.
      iii. One **GPoint** instance variable (**offset**) representing a displacement of the figure from the path.
   
b. Define a constructor for the **PathAnimation** class with the following signature:
      ```java
      PathAnimation(GObject figure, GPoint offset, int duration, int delay)
      ```
      The **figure** parameter is an object that can be moved and drawn using operations in the `acm.graphics` package. The **offset** parameter indicates the point on the object that should follow the path. The **duration** parameter is the number of frames in the animation. The **delay** parameter is the time (in milliseconds) that should pass between drawing successive frames. The constructor should use these parameters to initialize the instance variables of the **PathAnimation** class.
   c. Define a public method **animate** for the **PathAnimation** class. This method should take no parameters. It should animate the motion of the **figure** instance variable. It should return nothing. This method should implement the following algorithm:
      For each value of **frame**, from 0 to **duration-1**, do the following:
      i. Invoke the **figure** object’s **setLocation** method so that the object is located at **position(frame)**.
      ii. Invoke the **GObject** method **pause(delay)** on the figure being animated, to cause Java to wait for **delay** milliseconds.
   d. Define a public method with signature: `void show(GraphicsProgram gProgram)` that includes the statement: `gProgram.add(figure);` that adds the figure to the scene.
   e. Define a public method with signature: `void hide(GraphicsProgram gProgram)` that includes the statement: `gProgram.remove(figure);` that removes the figure to the scene.
   f. Declare but do not implement a helper method that has the following signature:
      ```java
      protected abstract GPoint position(int frame)
      ```
   g. You cannot test your definition at this point, since the **PathAnimation** class is abstract, and cannot be instantiated.

2. Implement the **LinePathAnimation** class:
   a. Define **LinePathAnimation** to be a public (non-abstract) class that extends **PathAnimation**. Eventually you will need to define some instance variables for this class, in order to support the **position** method.
   b. Define a constructor for the **LinePathAnimation** class with the following signature:
      ```java
      LinePathAnimation(GObject figure, GPoint offset, int duration, int delay, GLine path)
      ```
      The **figure**, **offset**, **duration** and **delay** have the same meaning as in the **PathAnimation** class constructor described above. The **path** parameter represents a line along which the object will move during the animation. The constructor should use these parameters to initialize instance variables of the **LinePathAnimation**.
   c. Declare and implement the **position** method that was declared to be abstract in the **PathAnimation** class. In the **LinePathAnimation** class, this method has the following
signature: protected GPoint position(int frame). The input parameter frame represents an integer in the range 0 … duration-1. This method returns a GPoint object representing a position along the path of the animation. The GPoint should be chosen so that position(0) is path.getStartPoint() (one endpoint of the line); position(duration-1) is path.getEndPoint() (other endpoint of the line); and positions at intermediate frames are uniformly spaced along the line from position(0) to position(duration-1).

d. Test your LinePathAnimation and PathAnimation class definitions: First comment out the lines in AnimationApplication.java that construct and animate the ArcPathAnimation. Then compile and execute the program.

3. Implement the ArcPathAnimation class:
   a. Define a constructor for the ArcPathAnimation class with the following signature: ArcPathAnimation(GObject figure, GPoint offset, int duration, int delay, GArc path). The figure, duration and delay have the same meaning as in the PathAnimation class constructor described above. The path parameter is an elliptical arc along which the object will move during the animation. The constructor should use these parameters to initialize instance variables of the ArcPathAnimation.
   b. Declare and implement the position method that was declared to be abstract in the PathAnimation class. In the ArcPathAnimation class, this method has the following signature: protected GPoint position(int frame). The input parameter frame represents an integer in the range 0 … duration-1. This method returns a GPoint object representing a position along the path of the animation. The GPoint should be chosen so that position(0) is the point on the path at angle1 = path.getStartAngle(); position(duration-1) is the point on the circular path at angle2 = path.getStartAngle()+path.getSweepAngle(); and positions at intermediate frames are at uniformly spaced angles along the elliptical path from angle1 to angle2. In order to implement this procedure, you will need to use some Trigonometry. If a point with coordinates (x,y) lies on an ellipse of width w and height h, at an angle of θ with respect to the positive X-axis, then x = xc + (w/2) • cos(θ) and y = yc - (h/2) • sin(θ), where xc and yc are the coordinates of the center of the ellipse. (The negative sign in the formula for the y-coordinate is needed because in our application, y increases in the downward direction.)
   c. Test your ArcPathAnimation and PathAnimation class definitions: Uncomment the lines in AnimationApplication.java that you previously commented out. Then compile and execute the program.