ADSV 2420, Advanced Programming I
JavaFX III, Animation

Problem 1 Step-by-Step (15 points)

1. To your existing FX1 project add a new package named anim (always in Project mode, not File mode).

2. The abstract class ***Animation*** is the root class for JavaFX animations as shown in Figure 1.

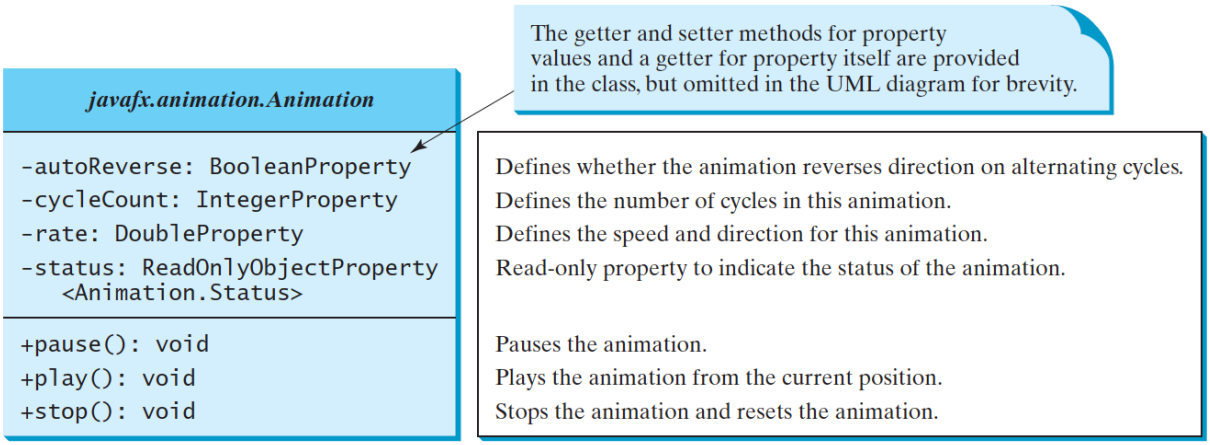


FIGURE 1

a. The **autoReverse** is a Boolean property that indicates whether an animation will reverse its direction on the next cycle.

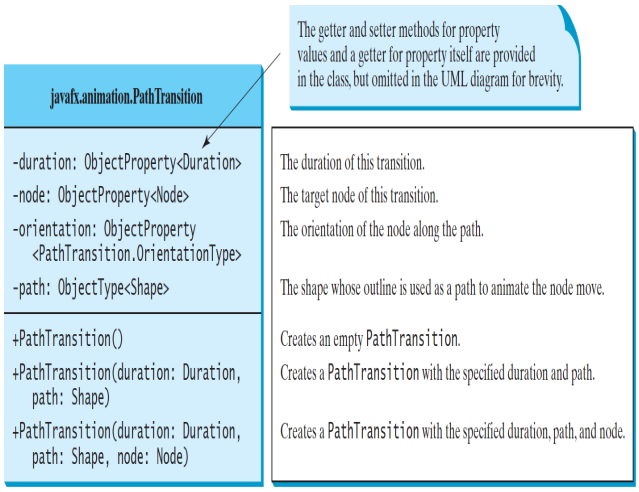
b. The **cycleCount** indicates the number of the cycles for the animation. You can use the constant Timeline.INDEFINTE to indicate an indefinite number of cycles.

c. The **rate** defines the speed of the animation. A negative rate value indicates the opposite direction for the animation.

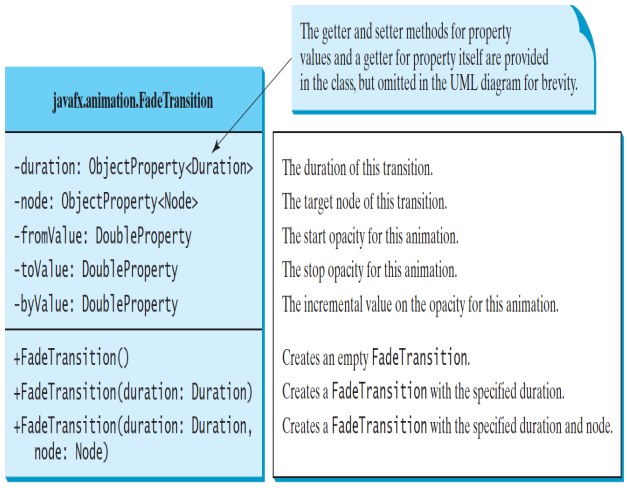
d. The **status** is a read-only property that indicates the status of the animation (Animation.Status.PAUSED, Animation.Status.RUNNING, and Animation.Status.STOPPED).

e. The methods **pause**(), **play**(), and **stop**() pauses, plays, and stops an animation.

3. The **PathTransition** and the **FadeTransition** classes are derived from class Animation.



The PathTransition class animates the the moves of a node along a path from one end to the other over a given time.



The FadeTransition class animates the change of the opacity in a node over a given time.

4. The class **Timeline** is derived from Animation can be used to program any animation using one or more KeyFrames.

Constructor Summary

Constructors

Constructor and Description
Timeline() The constructor of Timeline.
Timeline(double targetFramerate) The constructor of Timeline.
Timeline(double targetFramerate, **KeyFrame**... keyFrames) The constructor of Timeline.
Timeline(**KeyFrame**... keyFrames) The constructor of Timeline.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
ObservableList<**KeyFrame**>	**getKeyFrames**() Returns the **KeyFrames** of this Timeline.
void	**stop**() Stops the animation and resets the play head to its initial position.

5. Now we will animate a ball to bounce left and right, up and down. Create a new package, lab18 and under the package create the class BouncingBall which extends Pane as shown below. The ball will be bouncing inside the pane.

```
1   package animation.lab18;
2
3   import javafx.scene.layout.Pane;
4
5   /**
6    *
7    * @author ASDV2
8    */
9   public class BouncingBall extends Pane
10  {
11
12  }
```

6. Add properties to your class by adding lines 13 to 17 as shown below. The comments are explanatory of the properties. We create a ball which is a Circle of radius 20 pixels, with coordinates, x, y, with direction dx, dy and then we animate the ball using TimeLine.

```
1   package animation.lab18;
2
3   import javafx.animation.Timeline;
4   import javafx.scene.layout.Pane;
5   import javafx.scene.shape.Circle;
6
7   /**
8    *
9    * @author ASDV2
10   */
11  public class BouncingBall extends Pane
12  {
13      public final double radius = 20;                    //ball size 40
        private double x = radius, y = radius;              //coordinates of the ball
        private double dx = 1, dy = 1;                      //direction of the ball
        private Circle circle = new Circle(x, y, radius); // her Magesty, the ball
17      private Timeline animation;                         //animate the ball
18  }
```

7. Add the constructor for the bouncing ball. Click the bulb at line 30 to import and to implement all abstract methods.

```
23      public BouncingBall()
24      {
25          circle.setFill(Color.BROWN); // Set ball color
            getChildren().add(circle); // Place a ball into this pane
27
28          // Create an animation for moving the ball
29          animation = new Timeline(
                    new KeyFrame(Duration.millis(50), new EventHandler<ActionEvent>() );
31
32      }
```

8. The resulting code is shown below after clicking the bulb. The Timeline constructor takes as an argument a KeyFrame and the Keyfarme constructor takes as arguments the duration of the keyframe and and an EVENT HANDLER to handle the keyframe that occurs every 50 milliseconds.

```
25      public BouncingBall()
26      {
27          circle.setFill(Color.BROWN); // Set ball color
            getChildren().add(circle); // Place a ball into this pane
29
30          // Create an animation for moving the ball
31          animation = new Timeline(
                    new KeyFrame(Duration.millis(50), new EventHandler<ActionEvent>()
33                  {
34                      @Override
                        public void handle(ActionEvent event)
36                      {
37                          throw new UnsupportedOperationException("Not supported yet.");
38                      }
39                  })
40          );
41
42      }
```

9. Create a method **moveBall** which moves the ball as shown in lines 45 to 62. Then, call the method in line 37 to handle each keyframe every 50 milliseconds. Lastly, set the animation to play forever in line 41 and start the animation in line 42.

```
                    public void handle(ActionEvent event)
36                  {
37                      moveBall() ;
38                  }
39              })
40          );
41          animation.setCycleCount(Timeline.INDEFINITE);
42          animation.play(); // Start animation
43      }
44
45      private void moveBall()
46      {
47          // Check boundaries
48          if (x < radius || x > getWidth() - radius)
49          {
50              dx *= -1; // Change ball move direction
51          }
52          if (y < radius || y > getHeight() - radius)
53          {
54              dy *= -1; // Change ball move direction
55          }
56
57          // Adjust ball position by 1 or -1
58          x += dx;
59          y += dy;
60          circle.setCenterX(x);
61          circle.setCenterY(y);
62      }
```

10. Add to the class BouncingBall the methods below. The code is self explanatory. We increase or decrease the speed of the ball and we play or stop the ball. The rateProperty is the speed of the ball.

```
65      public void play()
66      {
67          animation.play();
68      }
69
70      public void pause()
71      {
72          animation.pause();
73      }
74
75      public void increaseSpeed()
76      {
77          animation.setRate(animation.getRate() * 1.5);
78      }
79
80      public void decreaseSpeed()
81      {
82          animation.setRate(
83                  animation.getRate() * 1.5 > 0 ? animation.getRate() / 1.5 : 0);
84      }
85
86      public DoubleProperty rateProperty()
87      {
88          return animation.rateProperty();
89      }
90
```

11. The complete code of class BouncingBall is shown below for your reference. You may enlarge the pdf to see it clearly.

```
1   kage animation.lab18;
2
3   rt javafx.animation.KeyFrame;
4   rt javafx.animation.Timeline;
5   rt javafx.beans.property.DoubleProperty;
6   rt javafx.event.ActionEvent;
7   rt javafx.event.EventHandler;
8   rt javafx.scene.layout.Pane;
9   rt javafx.scene.paint.Color;
10  rt javafx.scene.shape.Circle;
11  rt javafx.util.Duration;
12
13
14
15  @author ASDV2
16
17  lic class BouncingBall extends Pane
18
19
20      public final double radius = 20;                    //ball size 40
21      private double x = radius, y = radius;              //coordinates of the ball
22      private double dx = 1, dy = 1;                      //direction of the ball
        private Circle circle = new Circle(x, y, radius); // her Magesty, the ball
        private Timeline animation;                         //animate the ball
26
27      public BouncingBall()
28      {
            circle.setFill(Color.BROWN); // Set ball color
            getChildren().add(circle); // Place a ball into this pane
30
31          // Create an animation for moving the ball
32          animation = new Timeline(
                    new KeyFrame(Duration.millis(50), new EventHandler<ActionEvent>()
                    {
35                      @Override
                        public void handle(ActionEvent event)
                        {
38                          moveBall();
39                      }
40                  })
41          );
42          animation.setCycleCount(Timeline.INDEFINITE);
43          animation.play(); // Start animation
44      }
45
46      private void moveBall()
47      {
48          // Check boundaries
49          if (x < radius || x > getWidth() - radius)
50          {
51              dx *= -1; // Change ball move direction
52          }
53          if (y < radius || y > getHeight() - radius)
54          {
55              dy *= -1; // Change ball move direction
56          }
57
58          // Adjust ball position by 1 or -1
59          x += dx;
60          y += dy;
61          circle.setCenterX(x);
62          circle.setCenterY(y);
63      }
64
65      public void play()
66      {
67          animation.play();
68      }
69
70      public void pause()
71      {
72          animation.pause();
73      }
74
75      public void increaseSpeed()
76      {
77          animation.setRate(animation.getRate() * 1.5);
78      }
79
80      public void decreaseSpeed()
81      {
82          animation.setRate(
                    animation.getRate() * 1.5 > 0 ? animation.getRate() / 1.5 : 0);
84      }
85
86      public DoubleProperty rateProperty()
87      {
88          return animation.rateProperty();
89      }
90
91
```

12. Now we will build our standard JavaFX application to control the ball with the mouse and the keyboard. Create a class BallControl that extends Application. Click the bulb and add imports and all abstract methods.

```
1   package animation.lab18;
2
3   import javafx.application.Application;
4   import javafx.stage.Stage;
5
6   public class BallControl extends Application
7   {
8
9       @Override
        public void start(Stage primaryStage) throws Exception
11      {
12          throw new UnsupportedOperationException("Not supported yet.");
13      }
14
15  }
```

13. Follow our standard procedure of adding Node(s) to the Scene and the Scene to the Stage. That is, create the BouncingBall and add it as shown below:

```
1   package animation.lab18;
2
3   import javafx.application.Application;
4   import javafx.scene.Scene;
5   import javafx.stage.Stage;
6
7   public class BallControl extends Application
8   {
9
10      @Override
        public void start(Stage primaryStage) throws Exception
12      {
13          BouncingBall bouncingBall = new BouncingBall(); // Create a ball pane
14          // Create a scene and place it in the stage
15          Scene scene = new Scene(bouncingBall, 800, 600);
16          primaryStage.setTitle("Bouncing Ball Control"); // Set the stage title
17          primaryStage.setScene(scene); // Place the scene in the stage
18          primaryStage.show(); // Display the stage
19          // Must request focus after the primary stage is displayed
20          bouncingBall.requestFocus();
21      }
22
23  }
```

14. We want to *stop the ball* when the *mouse is pressed* and move the ball when the *mouse is released.* Below are the specifications for method setOnMousePressed of class Node. BouncingBall IS-A Node.

setOnMousePressed

```
public final void setOnMousePressed(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMousePressed.

Property description:

```
Defines a function to be called when a mouse button has been pressed on this Node.
```

15. Add line 24, shown below which will add a listener too the BouncingBall when the mouse is pressed. Click the bulb on the left to import and to implement all abstract methods. When the mouse is pressed we stop the ball from moving.

```
22          bouncingBall.requestFocus();
23
24          bouncingBall.setOnMousePressed( new EventHandler<MouseEvent>() );
25      }
```

16. After clicking the bulb of line 24 the resulting code is shown below with the addition of the main() method. Clean and build and click the mouse to stop the ball from moving.

```
1   package animation.lab18;
2
3   import javafx.application.Application;
4   import javafx.event.EventHandler;
5   import javafx.scene.Scene;
6   import javafx.scene.input.MouseEvent;
7   import javafx.stage.Stage;
8
9   public class BallControl extends Application
10  {
11      @Override
12      public void start(Stage primaryStage) throws Exception
13      {
14          BouncingBall bouncingBall = new BouncingBall(); // Create a ball pane
15          // Create a scene and place it in the stage
16          Scene scene = new Scene(bouncingBall, 800, 600);
17          primaryStage.setTitle("Bouncing Ball Control"); // Set the stage title
18          primaryStage.setScene(scene); // Place the scene in the stage
19          primaryStage.show(); // Display the stage
20          // Must request focus after the primary stage is displayed
21          bouncingBall.requestFocus();
22
23          bouncingBall.setOnMousePressed(new EventHandler<MouseEvent>()
24          {
25              @Override
26              public void handle(MouseEvent event)
27              {
28                  bouncingBall.pause();
29              }
30          });
31      }
32
33      public static void main(String[] args)
34      {
35          launch(args);
36      }
37  }
38
```

17. Similarly add an event handler for mouse released to move the ball. This time with a lamda expression for setOnMouseReleased.

```
32              bouncingBall.setOnMouseReleased(e ->
33              {
34              bouncingBall.play();
35              });
```

18. Similarly add an event handler for the keyboard to increase and decrease the speed of the ball. DO NOT USE the lambda expression shown below but an ANONYMOUS class. Clean and build and test your animation.

```
38          // Increase and decrease animation
39          bouncingBall.setOnKeyPressed(e ->
40                  {
41                      if (e.getCode() == KeyCode.UP)
42                      {
43                          bouncingBall.increaseSpeed();
44                      }
45                      else if (e.getCode() == KeyCode.DOWN)
46                      {
                            bouncingBall.decreaseSpeed();
48                      }
49          });
```

19. The complete code for class BallControl is shown below:

```
1   package animation.lab18;
2   import javafx.application.Application;
3   import javafx.event.EventHandler;
4   import javafx.scene.Scene;
5   import javafx.scene.input.KeyCode;
6   import javafx.scene.input.MouseEvent;
7   import javafx.stage.Stage;
8
9   public class BallControl extends Application
10  {
11      @Override
12      public void start(Stage primaryStage) throws Exception
13      {
14          BouncingBall bouncingBall = new BouncingBall(); // Create a ball pane
15          // Create a scene and place it in the stage
16          Scene scene = new Scene(bouncingBall, 800, 600);
17          primaryStage.setTitle("Bouncing Ball Control"); // Set the stage title
18          primaryStage.setScene(scene); // Place the scene in the stage
19          primaryStage.show(); // Display the stage
20          // Must request focus after the primary stage is displayed
21          bouncingBall.requestFocus();
22
23          bouncingBall.setOnMousePressed(new EventHandler<MouseEvent>()
24          {
25              @Override
26              public void handle(MouseEvent event)
27              {
28                  bouncingBall.pause();
29              }
30          });
31
32          bouncingBall.setOnMouseReleased(e ->
33          {
34          bouncingBall.play();
35          });
36          // Increase and decrease animation
37          bouncingBall.setOnKeyPressed(e ->
38                  {
39                      if (e.getCode() == KeyCode.UP)
40                      {
41                          bouncingBall.increaseSpeed();
42                      }
43                      else if (e.getCode() == KeyCode.DOWN)
44                      {
45                          bouncingBall.decreaseSpeed();
46                      }
47          });
48      }
49      public static void main(String[] args){ launch(args); }
50  }
51
```

Problem 2

Add to the BouncingBall class a Rectangle of width 70 and height 20. Make the ball of radius 10. The rectangle is a racket which appears at the center-bottom of your BouncingBall pane and can hit the ball back. The rectangle moves left and right upon events KeyCode.LEFT, and KeyCode.RIGHT by adding code to your existing keyboard handler of BallControl. To detect WHEN the ball hits the racket use the Rectangle method intersects and simply reverse the direction of the ball appropriately. Add to your BouncingBall Text nodes to keep track of the score of Human against Computer. The program's jar is posted under tennis.jar. Extra credit 1 point if you add one more ball. That is, play tennis with 2 balls at the same time.