Actor Class and Bouncing Exercise

Description:

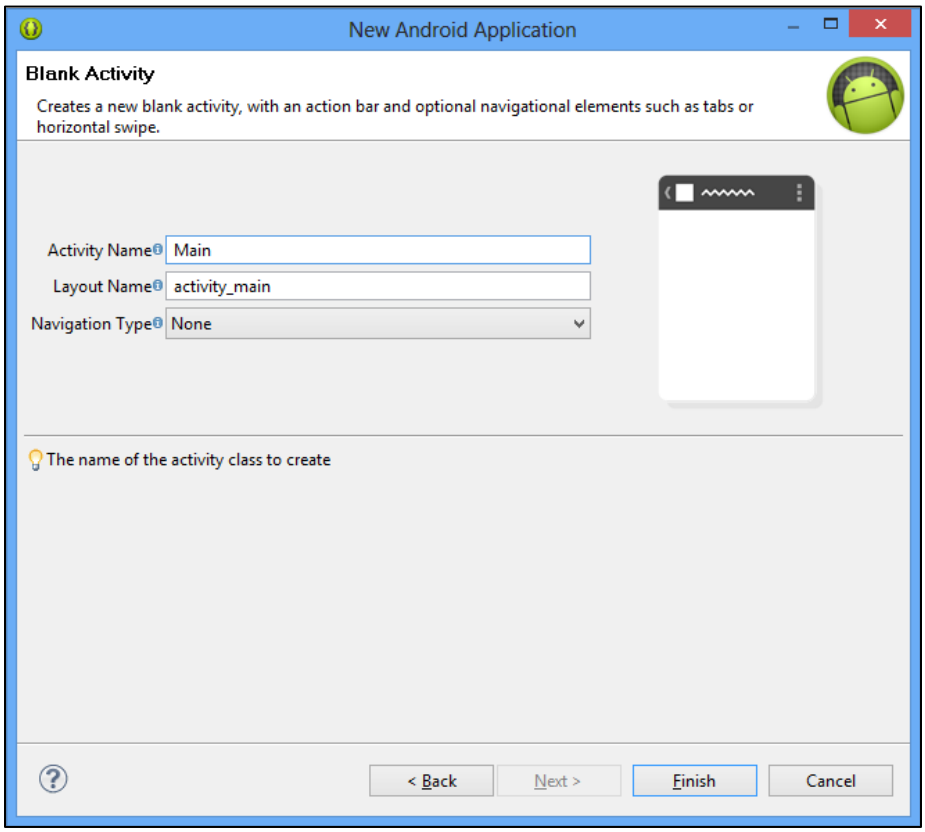
The goal of this activity is to have you write a Class 'Actor' to represent objects in a 2D animated environment. We will do the following:

1. Create a 3 Class Android App with a Main, AnimationView, and an Actor Class
2. Write the AnimationView as a custum View class
3. Write a Handler / Runnable Sequence to run the animation.
4. Create two Actor instances to demonstrate.

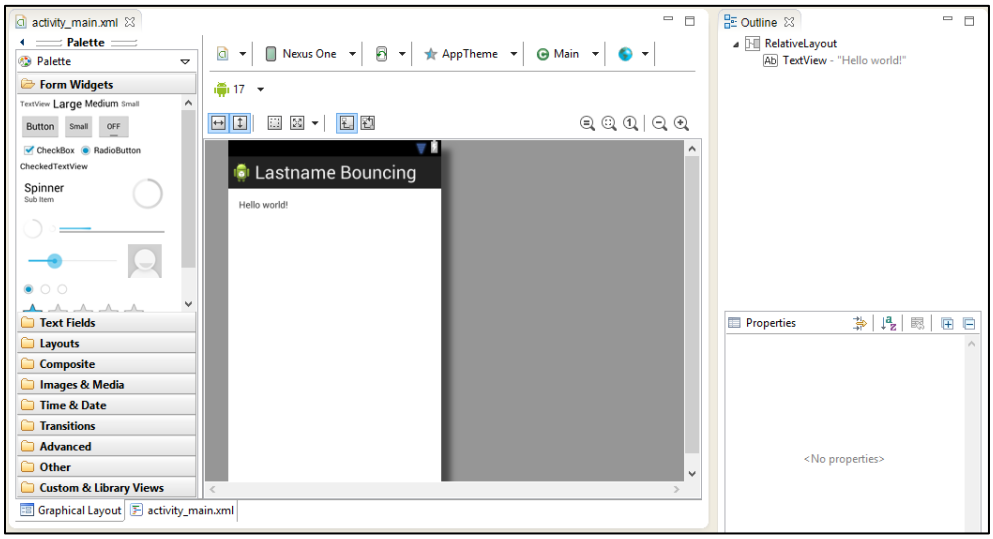
Process:

1. Create a New Android App and call it "LastnameBouncing".

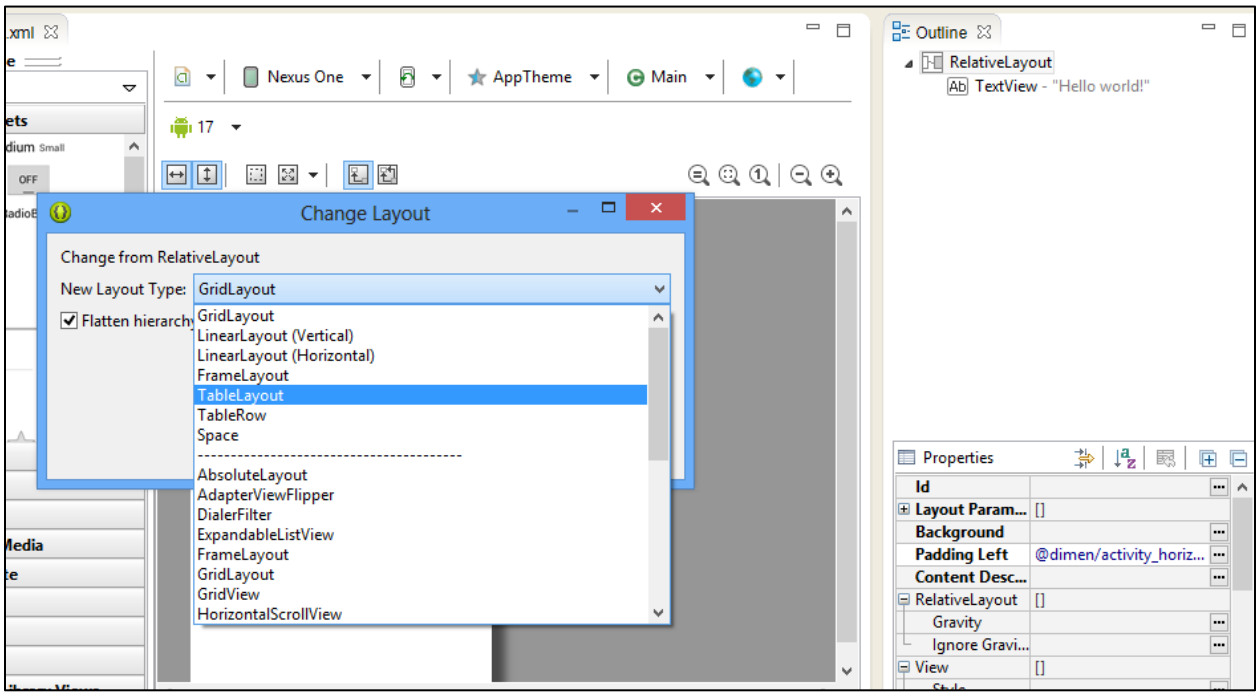
2. When prompted for the name of the Activity, call it "Main"



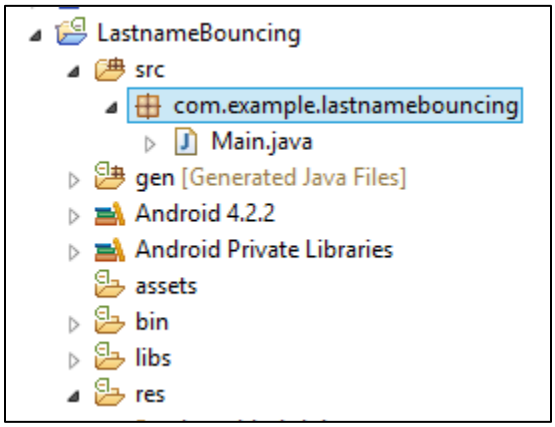
3. Go to the activity_main xml



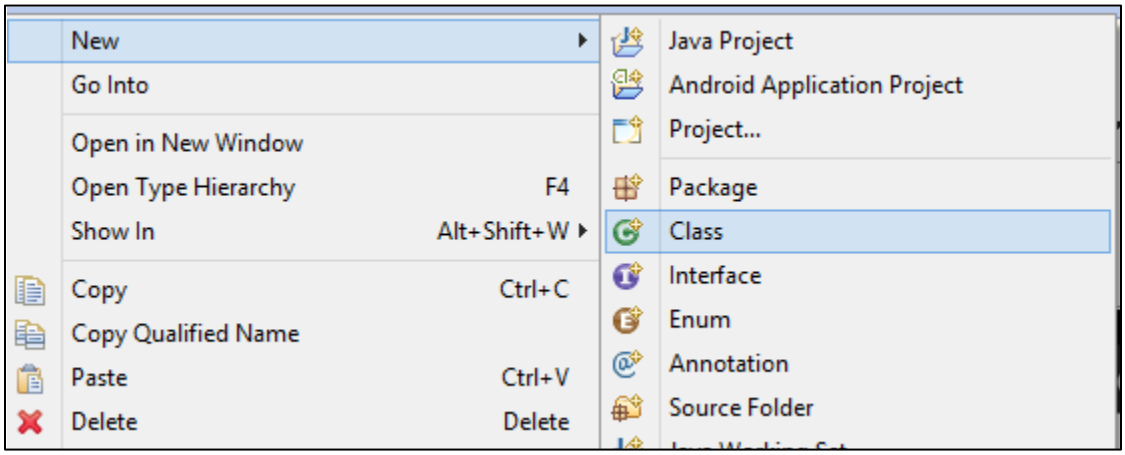
4. Change the Layout type to “TableLayout.” This is all we will do for now.



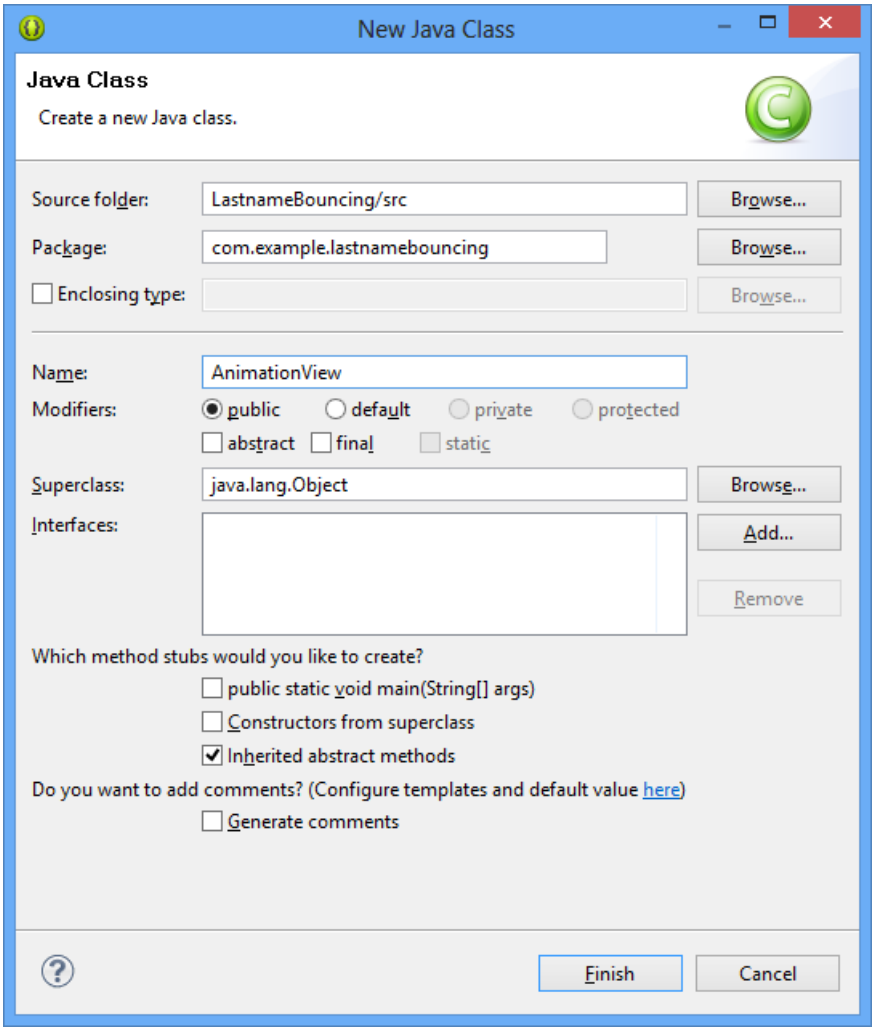
5. Expand the src folder so you see the Main.java class.



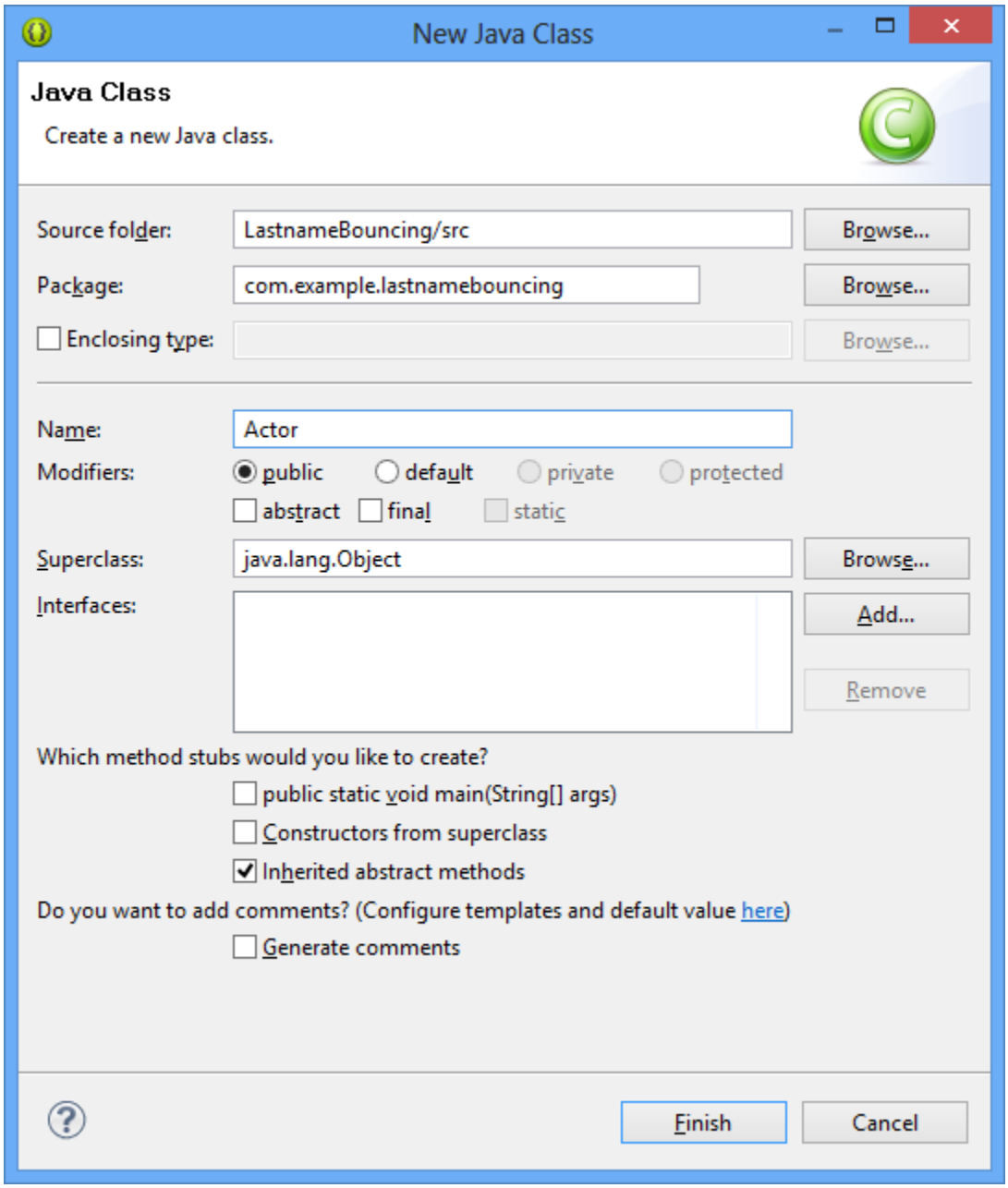
6. Right click on the 'com.example.lastnamebouncing' and select 'New-Class'



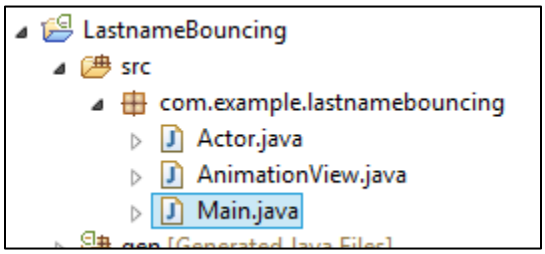
7. Name the class “AnimationView.”



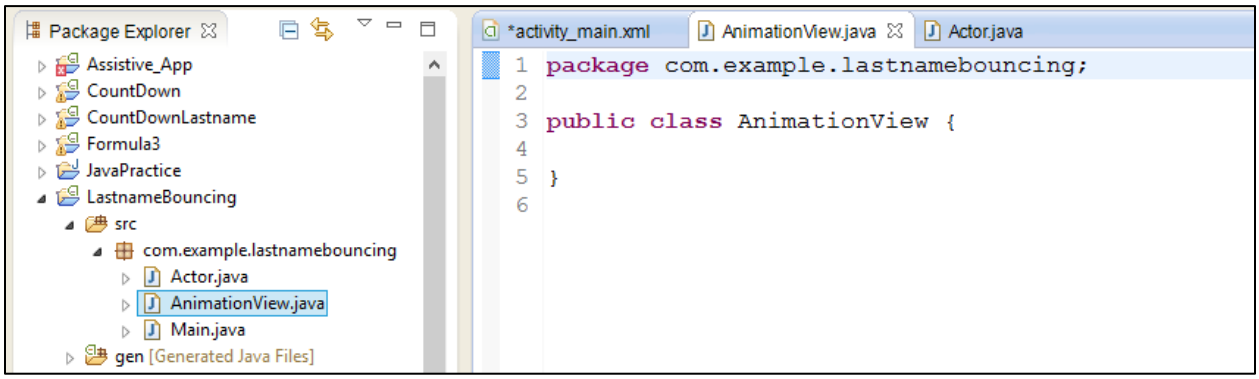
8. Repeat the process and make a new class called ‘Actor’



9. You should now see three classes; Main, AnimationView, and Actor.



10. Go to the AnimationView class.



11. Change the class declaration to the following:

```
6
7  public class AnimationView extends View {
8
```

12. Add the AttributeSet attrs to the parameters and the attrs to the super command.

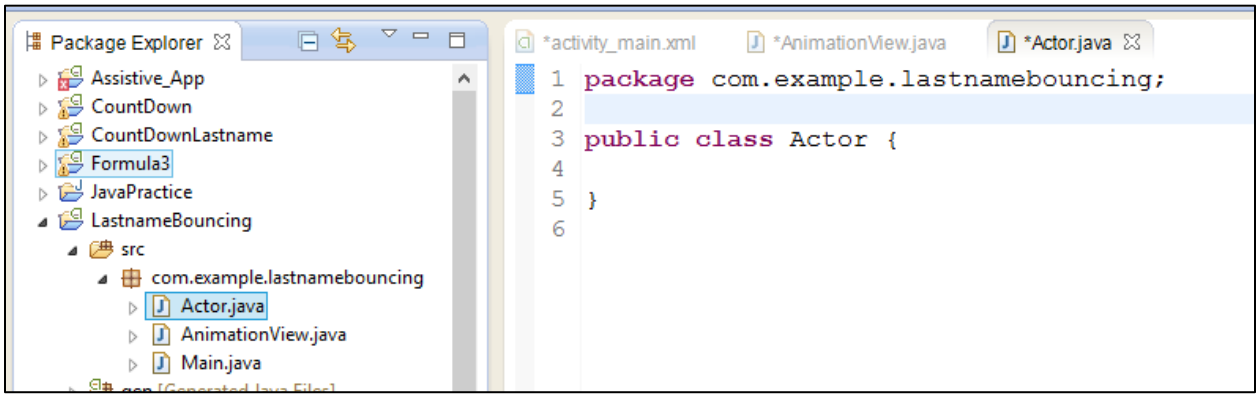
```
8
9      public AnimationView(Context context, AttributeSet attrs) {
10         super(context, attrs);
11         // TODO Auto-generated constructor stub
12     } // end onCreate
13
```

13. The entire AnimationView class should look like this:

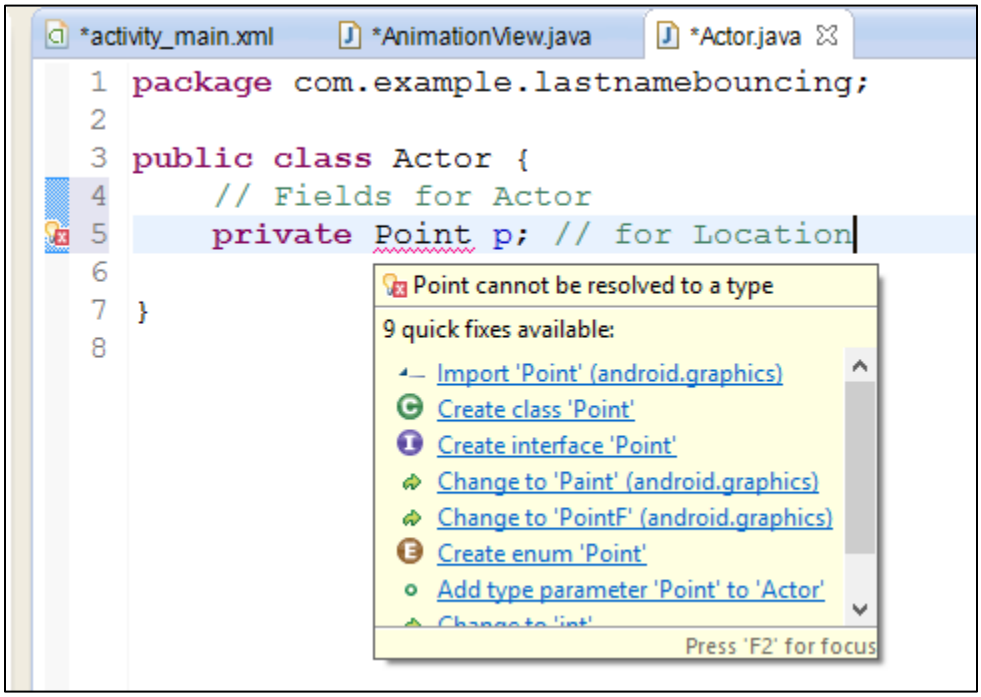
*activity_main.xml | *AnimationView.java | Actor.java

```
1  package com.example.lastnamebouncing;
2
3  import android.content.Context;
4  import android.util.AttributeSet;
5  import android.view.View;
6
7  public class AnimationView extends View {
8
9      public AnimationView(Context context, AttributeSet attrs) {
10         super(context, attrs);
11         // TODO Auto-generated constructor stub
12     } // end onCreate
13
14 } // end class AnimationView
15
16
```

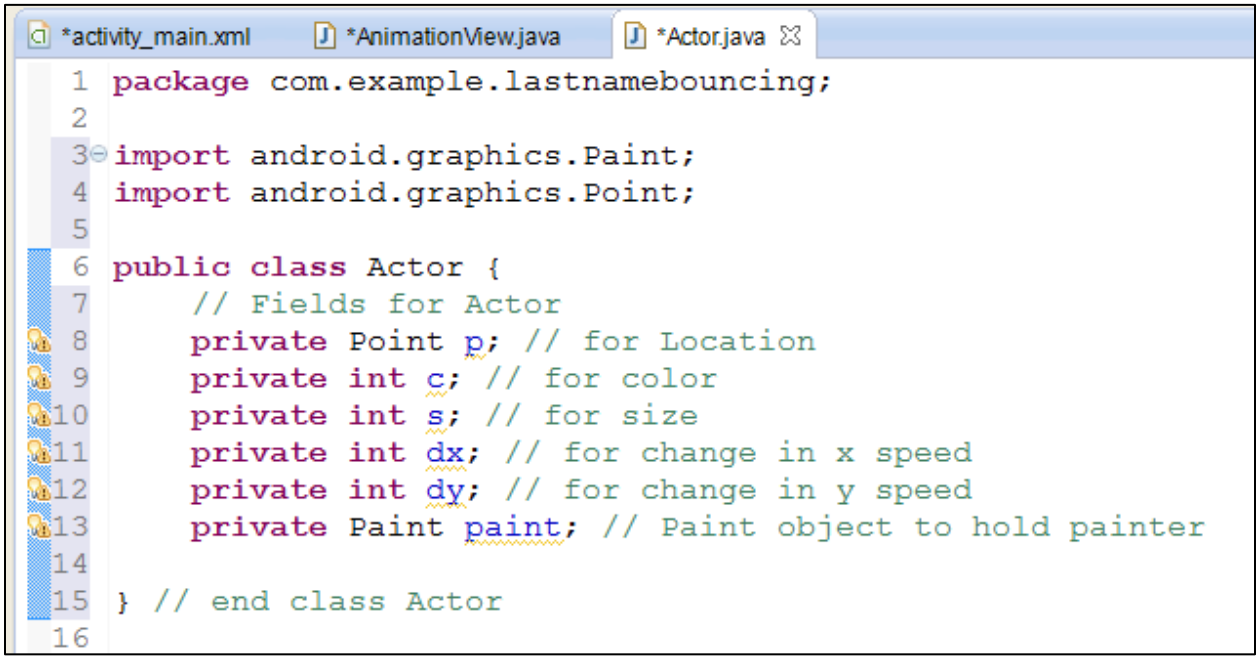
14. We will now build the Actor class. Go to the Actor.java class.



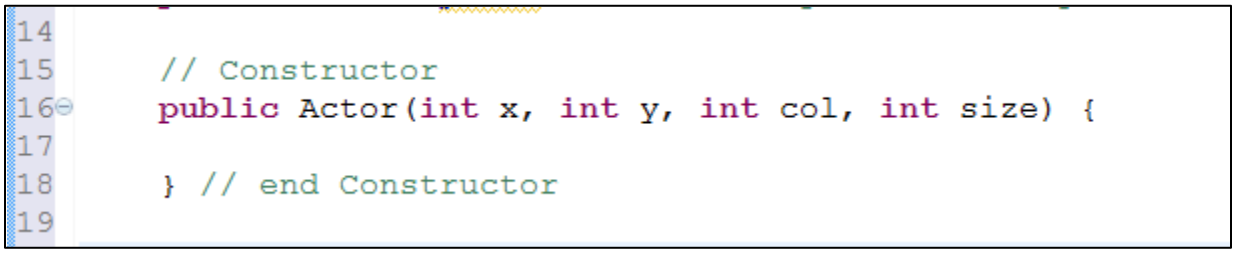
15. We will now build the fields for the Actor class. We will start with a Point object to store the X and Y coordinates.



16. Now fill in the rest of the Fields for the Actor class. Note the comments describing the purpose of the fields.



17. We will now build the constructor for the Actor class. Note that the constructor that sets the values for x and y position, color, and size of the Actor.



18. Inside the constructor we will initialize the values for all the fields.

```
14
15      // Constructor
16      public Actor(int x, int y, int col, int size) {
17          // Initialize Values
18          p = new Point(x, y); // set the x and y position
19          c = col; // set the color
20          s = size;  // sets the size
21          paint = new Paint(); // creates Paint object
22          paint.setColor(c); // sets Paint color
23          dx = 0; // sets x speed to 0
24          dy = 0; // sets y speed to 0
25
26      } // end Constructor
27
```

19. We now need to code “Access” functions to read the values from the data fields. Code these functions below the Constructor. (Not inside the constructor)

```
27
28      // Functions - What Actor can do
29
30      // Accessors or 'Getters' - get values
31
32      public int getX() {
33          return p.x;
34      } // end getX()
35
36      public int getY() {
37          return p.y;
38      } // end getY()
39
40      public Paint getPaint() {
41          return paint;
42      } // end getPaint()
43
44      public int getSize() {
45          return s;
46      } // end getSize()
47
48  } // end class Actor
```

20. We now need to write modifiers. This is where we allow other parts of the program to change the fields of an object. We will start with changing color:

```
48
49      // Modifiers - Change data inside Actor
50      public void setColor(int col) {
51          c = col;
52          paint.setColor(c);
53      }
54
```

21. Write the modifiers for Position and Speed.

```
54
55      public void goTo(int x, int y) {
56          p.x = x;
57          p.y = y;
58      }
59
60      public void setDX(int speed) {
61          dx = speed;
62      }
63
64      public void setDY(int speed) {
65          dy = speed;
66      }
67
```

22. Write the Modifiers to change the speed.

```
67
68      public void changeDX(float a) {
69          dx += a;
70      }
71
72      public void changeDY(float a) {
73          dy += a;
74      }
75
76      public void move() {
77          p.x += dx;
78          p.y += dy;
79      }
80
```

23. Build these functions and leave them blank. We will return to them later.

```
80
81      public void bounce(Canvas c) {
82          // Leave empty for now
83      }
84
85      public void bounceActor(Actor a) {
86          // Leave empty for now
87      }
88
```

24. Finally, we need the Actors to be able to 'draw themselves.' Code these functions to have the Actor be able to draw a circle and a square.

```
88
89      public void drawCircle(Canvas c) {
90          c.drawCircle(p.x, p.y, s, paint);
91      }
92
93      public void drawSquare(Canvas c) {
94          c.drawRect(p.x, p.y, p.x + s, p.y+s, paint);
95      }
96
97
98
99  } // end class Actor
```

25. We are finished with the Actor Class. Now go to the AnimationView class and create some Actor Fields. Also write the fields for the handler and the RATE.

```
 8
 9  public class AnimationView extends View {
10
11      // Create some Actors
12      private Actor joshua;
13      private Actor rebecca;
14
15      // Create the Handler for animation
16      private Handler h;
17      private int RATE = 30;  // about 30 Frames a Second
18
```

26. In the Constructor, initialize the Actors, giving them their starting position, color, and size. Also initialize the Handler h.

```
18
19      public AnimationView(Context context, AttributeSet attrs) {
20          super(context, attrs);
21          // TODO Auto-generated constructor stub
22
23          // Initialize the Actors
24          joshua = new Actor(100, 100, Color.RED, 10);
25          rebecca = new Actor(200, 200, Color.BLUE, 5);
26
27          // Initialize the Handler
28          h = new Handler();
29
30      } // end onCreate
31
```

27. Below the constructor, write the 'onDraw' method. This handlers the drawing of the App.

```
32
33      // Create the onDraw Method - in all View Classes
34      public void onDraw(Canvas c) {
35
36      }
37
38  } // end class AnimationView
```

28. In the onDraw() function, have joshua and rebecca call the drawSquare() and drawCircle() functions.

```
32
33      // Create the onDraw Method - in all View Classes
34      public void onDraw(Canvas c) {
35          // have the Actors draw themselves
36          joshua.drawCircle(c);
37          rebecca.drawSquare(c);
38
39      } // End onDraw
40
```

29. Now write the h.postDelated function to call the runnable. The error on the r is OK for now.

```
32
33      // Create the onDraw Method - in all View Classes
34      public void onDraw(Canvas c) {
35          // have the Actors draw themselves
36          joshua.drawCircle(c);
37          rebecca.drawSquare(c);
38
39          // Call the Runnable for Animation
40          h.postDelayed(r, RATE);
41
42      } // End onDraw
43
```

30. We will now define the Runnable r. Write the private inner class as shown.

```
65
66      // Create the Runnable
67      private Runnable r = new Runnable() {
68
69          @Override
70          public void run() {
71              // TODO Auto-generated method stub
72
73          }
74
75      };
76
```

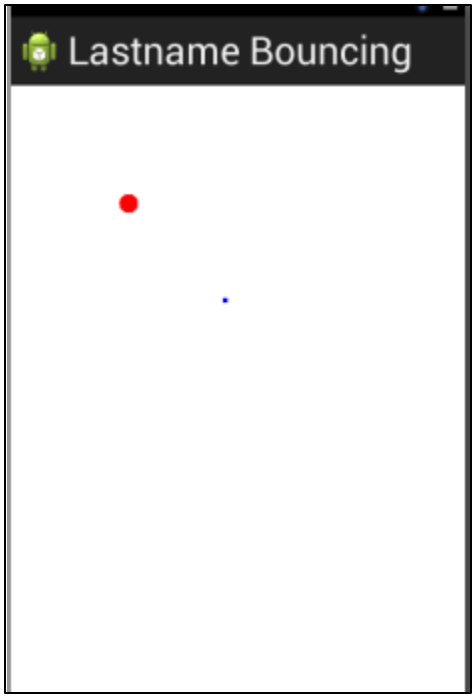
31. Write the invalidate() function inside the void run of the Runnable. In a View object, the invalidate() function will call onDraw() which will re-draw the screen.

```
65
66      // Create the Runnable
67      private Runnable r = new Runnable() {
68
69          @Override
70          public void run() {
71              // TODO Auto-generated method stub
72              // Call the Invalidate Method to reDraw
73              invalidate();
74          }
75
76      };
77
```

32. Go back to the XML for the activity_main and write the code below to link the AnimationView class to the XML

```
11
12      <com.example.lastnamebouncing.AnimationView
13          android:id = "@+id/animationView"
14          android:layout_width = "match_parent"
15          android:layout_height = "match_parent"
16      />
17
```

33. The View of the XML should look like this:


Lastname Bouncing

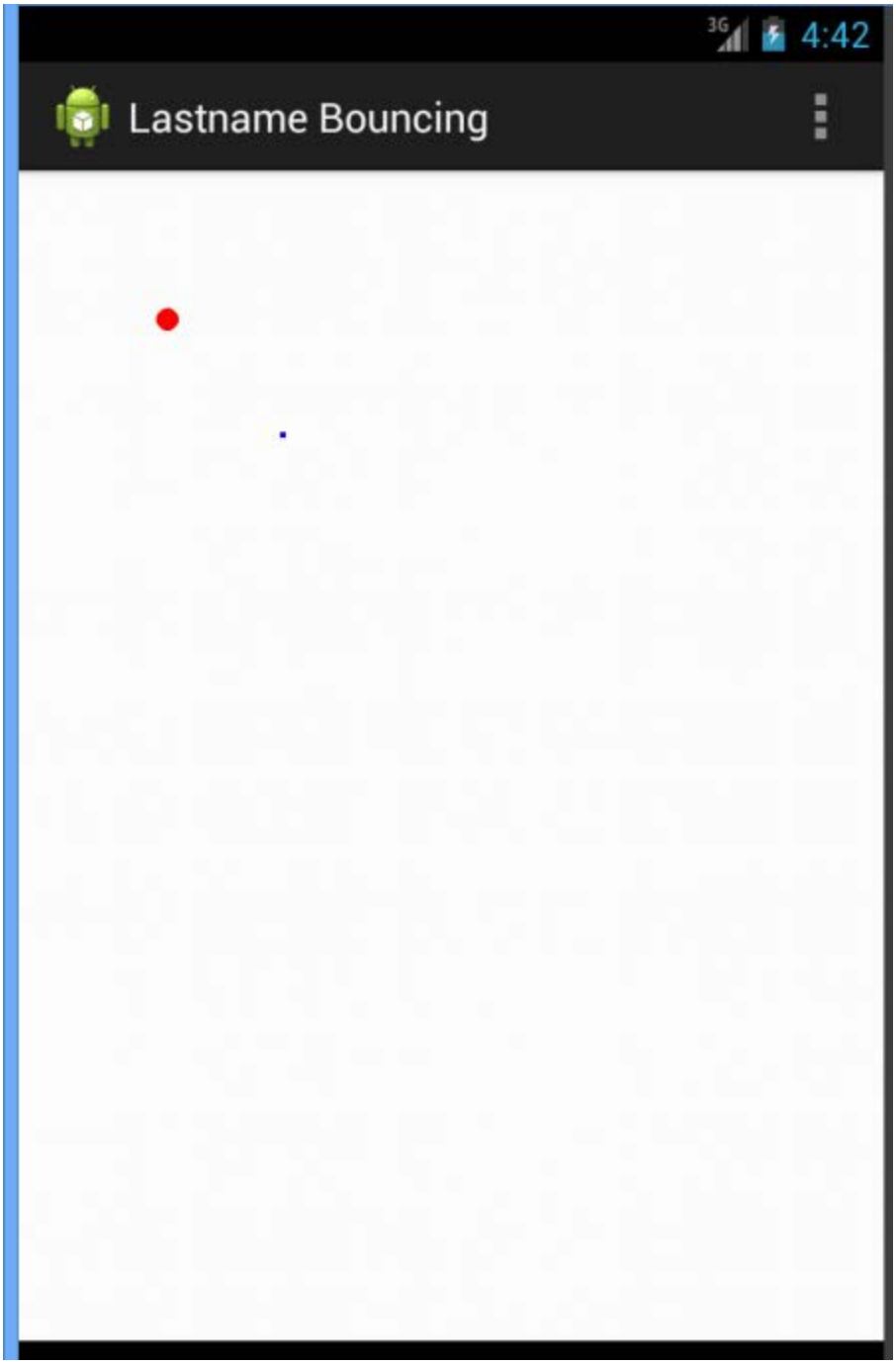
34. Almost done! Go to the Main class and write a field for the AnimationView

```
 6
 7 public class Main extends Activity {
 8
 9     // Field for Animation View
10     private AnimationView animationView;
11
```

35. Go to the onCreate() and bind the animationView object to the XML.

```
11
12    @Override
13    protected void onCreate(Bundle savedInstanceState) {
14        super.onCreate(savedInstanceState);
15        setContentView(R.layout.activity_main);
16
17        // Initialize the Animation View
18        animationView = (AnimationView)findViewById(R.id.animationView);
19
20    } // end onCreate
21
```

36. Save and run the App. You should see two small shapes.



37. We will now have these shapes move. Go to the Actor Class and write the code for bouncing:

```
80
81      public void bounce(Canvas c) {
82          if (p.x > c.getWidth() || p.x < 0) {
83              dx = dx * -1;
84          }
85          if (p.y > c.getHeight() || p.y < 0) {
86              dy = dy * -1;
87          }
88      }
89
```

38. Now the Actors will bounce off the side of the screen. Go to the AnimationView class on the onCreate() to give speed to the Actors rebecca and joshua.

```
19
20      public AnimationView(Context context, AttributeSet attrs) {
21          super(context, attrs);
22          // TODO Auto-generated constructor stub
23
24          // Initialize the Actors
25          joshua = new Actor(100, 100, Color.RED, 40);
26          rebecca = new Actor(200, 200, Color.BLUE, 50);
27
28          joshua.setDX(5);
29          joshua.setDY(5);
30
31          rebecca.setDX(10);
32          rebecca.setDY(-10);
33
34
35          // Initialize the Handler
36          h = new Handler();
37
38      } // end onCreate
39
```

39. In the onDraw() of the AnimationView, call the functions to move() and bounce() joshua and rebecca.

```
39
40      // Create the onDraw Method - in all View Classes
41      public void onDraw(Canvas c) {
42
43          // Actors Move
44          joshua.move();
45          rebecca.move();
46          joshua.bounce(c);
47          rebecca.bounce(c);
48
49          // have the Actors draw themselves
50          joshua.drawCircle(c);
51          rebecca.drawSquare(c);
52
53          // Call the Runnable for Animation
54          h.postDelayed(r, RATE);
55
56      } // End onDraw
57
```

40. Save and Run the App. The Actors should move (very slowly . . .). How can you add more Actors, change size, speed, color ??