Mobile App Tutorial
Animation with Custom View Class and Animated Object
Bouncing and Frame Based Animation

Description of View Based Animation and Control-Model-View Design process

In mobile device programming, the Control-Model-View (CMV) metaphor for software design allows the programmer to model the user input, logic, and animation/output for their game, animation, or simulation. The CMV metaphor as related to Java-Android is as follows:

Control: Represents the "user" input and timed events within the software game. By design, Android programming is event driven. The user touches the screen, shakes the phone, presses a button and the software responds with an action. In the structure of an animated game, usually a timer will trigger animation events 30 times a second in addition to user input. In the structure of an Android program, the Control is the "Main Activity" that first opens on execution of the program.

Model: Represents the data structure of the objects and the functions that govern the flow of logic and action in a game. I think of the model in two parts:

1. The actual game pieces as modeled in classes.
2. The actions of the game pieces in relation to movement and each other. (Functions)
3. The "Game Logic." (Rules of the game)

In Android software development, the Model can be a separate Class that "holds the objects and action" or the Model can be interwoven within the View Class.

View: The View's job is to retrieve positions, orientation, data, and graphics from the Model and draw them on the Screen. The Runnable and the Handler will call events 30 times a second that causes the view to retrieve this data and then draw the image to the screen. In Android Programming the View is usually represented as a "GameView.java" class that extends the Android class of View. The View holds the "Game Loop" that calls on the Model to update the positions of the game.

This project will introduce the Model and View Phase of App design. We will design an extension of the View Class called AnimationView that will manipulate the model and draw shapes to the phone screen based on the Model. In subsequent lessons we will work on handling graphics, more complicated models, and User inputs via the Main Class for control.

Phase 1: Create the App Project

Process:

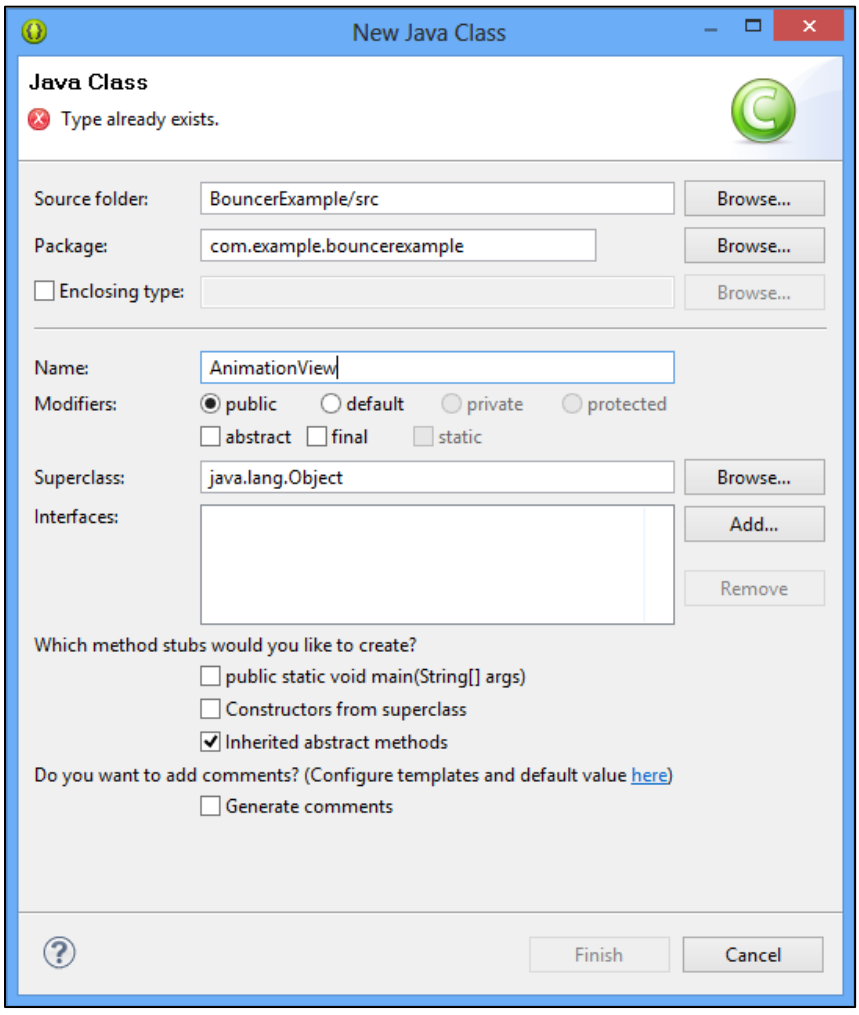
1. Start Eclipse and select New Project -> “Android Application Project”
2. Fill out the fields with the following:
    a. Application Name: BouncerExample
    b. Project Name: Bouncer
    c. Package Name: com.example.bouncerexample
3. Click Next
4. Click Next at the Configure Project Screen
5. Click Next at the Configure Launcher Screen
6. Click Next at the Create Activity Screen
7. Fill out the following fields in New Blank Activity Screen
    a. Activity Name: Main
    b. LayoutName: activity_main
    c. Navigation Type: None
8. Click “Finish”

Phase 2: Set up the Java Objects

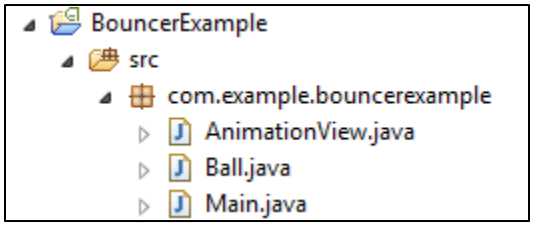
We will work through the App Structure from defining classes first because the XML depends on a custom AnimationView class.

Process:

1. Click and expand the Bouncer/src/com.example.bouncerexample and you will see the Main.java class
2. RightClick on the com.example.bouncerexample and select New -> Class
3. Name the class AnimationView and click Finish



4. We will also create a Ball.java class to represent an object we will animate. Right click on com.example.bouncer and create a Ball class. You should have three classes: AnimationView.java, Ball.java, and Main.java (Note the similarity to the Board Game Project)



5. We now need to modify the code in the AnimationView.java class to make it extend View. Open the AnimationView class and make the following modifications:

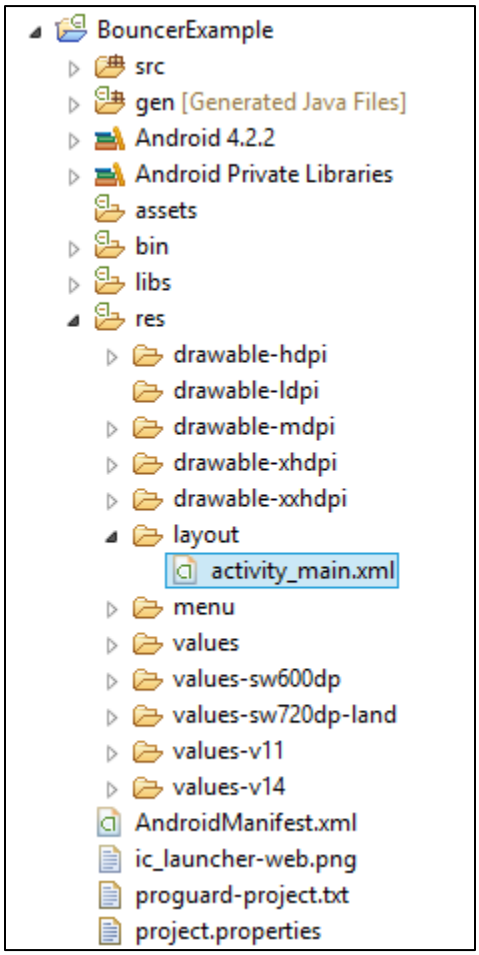
```
1  package com.example.bouncerexample;
2
3  import android.content.Context;
4  import android.util.AttributeSet;
5  import android.view.View;
6
7  public class AnimationView extends View{
8
9      public AnimationView(Context context, AttributeSet attrs) {
10         super(context, attrs);
11   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

Phase 3: Android XML User Interface Design

With the AnimationView Class setup with the essential elements of a View Class, we can write XML. This App only has one XML object – the custom View Class AnimationView. Code the XML Exactly as you see it below.

Process:

1. Open the activity_main.xml file:



2. Delete the XML and type the following to set up a Linear View and the custom AnimationView object:

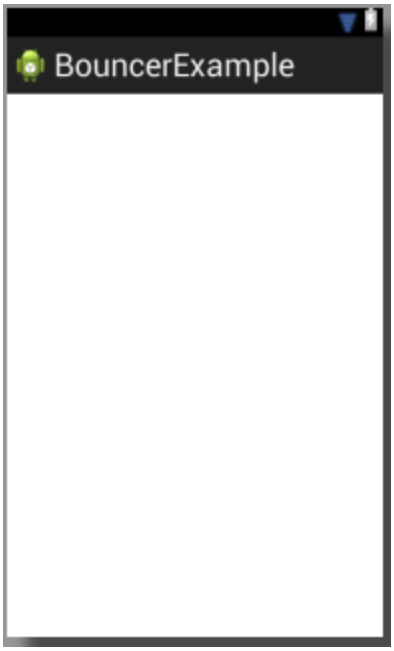
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Main" >

    <com.example.bouncerexample.AnimationView
        android:id = "@+id/animationView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
                                        />

</LinearLayout>

```

3. Note that the Graphic Layout will look blank. That is because the AnimationView Class represents a blank space on which we will create a canvas to do the animations.



Phase 4: Creating a Ball Class to represent the objects we wish to animate.

We will create a Ball class to will hold the following fields:

Point p:	Represents the x and y position of the Ball
Integer c:	Represents the color of the Ball.
Integer r:	Represents the Radius of the Ball.
Integer dx:	Represents the change in x position of the Ball. (Horizontal Speed)
Integer dy:	Represents the change in y position of the Ball. (Vertical Speed)
Paint paint:	Android Object holding the color for drawing on the canvas

The Ball class will have the following functions:

getX():	Returns the x position of the ball as an integer
getY():	Returns the y position of the ball as an integer
getRadius():	Returns the Radius of the ball as an integer
getPaint():	Returns the Paint of the Ball as a Paint Object
setColor(int col):	Sets the Color of the Ball
goTo(int x, int y):	Sets the x and y positions of the Ball
setDX(int speed):	Sets the Horizontal Speed of the Ball
setDY():	Sets the Vertical Speed of the Ball
move():	Changes the x and y position by the dx and dy values
bounce():	Bounces off the Sides of the Canvas

Process:

1. Define the fields for the Ball Class:

```
 1 package com.example.bouncerexample;
 2
 3 import android.graphics.Canvas;
 4 import android.graphics.Paint;
 5 import android.graphics.Point;
 6
 7 public class Ball {
 8
 9     private Point p; // Structure storing Point of Ball
10     private int c; // integer for color
11     private int r; // integer for radius
12     private int dx; // integer for change in x position
13     private int dy; // integer for change in y position
14     private Paint paint; // Holds color for drawing on canvas
15
```

2. The Constructor will define the values for the Fields and allow the program to create instances of the Ball:

```
15
16      public Ball (int x, int y, int col, int radius) {
17          p = new Point(x, y); // set x and y position
18          c = col; // set the color integer
19          r = radius; // sets the size
20          paint = new Paint(); // creates paint object
21          paint.setColor(c); // sets color for paint object
22          // For an experiment
23          dx = 0;
24          dy = 0;
25      } // End Constructor for Ball
26
```

3. Write the Getter Functions for the Ball:

```
26
27      // Getters
28      // X Position
29      public int getX() {
30          return p.x;
31      }
32
33      // Y Position
34      public int getY() {
35          return p.y;
36      }
37
38      // Radius
39      public int getRadius() {
40          return r;
41      }
42
43      // Paint
44      public Paint getPaint() {
45          return paint;
46      }
47
```

4. Write the Setters for the Ball:

```
47
48      // Setters
49      public void setColor(int col){
50          c = col;
51      }
52
53      public void goTo(int x, int y) {
54          p.x = x;
55          p.y = y;
56      }
57
58      public void setDX(int speed) {
59          dx = speed;
60      }
61
62      public void setDY(int speed) {
63          dy = speed;
64      }
65
```

5. Write the Functions for Moving and Bouncing off the Edges:

```
65
66      // Function for moving and bouncing
67
68      public void move() {
69          p.x = p.x + dx;
70          p.y = p.y + dy;
71      }
72
73      // Bounce off Edge
74
75      public void bounce(Canvas canvas) {
76          move();
77          if (p.x > canvas.getWidth() || p.x < 0) {
78              dx = dx * -1;
79          }
80          if (p.y > canvas.getHeight() || p.y < 0) {
81              dy = dy * -1;
82          }
83      } // end bounce()
84
```

6. Write the Functions for Bouncing off other Balls and finish the Class Ball:

```
84
85      // Collision Detection with other Ball Objects
86
87      public void bounceOff(Ball b) {
88          if ((Math.abs(b.getX()-p.x) < b.getRadius() + r) && (Math.abs(b.getY() - p.y)< b.getRadius() + r)) {
89              dx = dx * -1;
90              dy = dy * -1;
91          }
92      } // end bounceOff
93
94  } // end class Ball
95
96
```

7. We are finished with the Class Ball. We will now move to the AnimationView Object

Phase 5: Writing the Code for the AnimationView class

The AnimationView class performs two roles in this Application. First, the AnimationView class will use a Handler and Runnable to cycle and run the model and draw the graphics on the Device Screen. Second, the AnimationView will run the model to have the Balls objects calculate their positions, bounce off the edge of the screen, and do collision detection.

Process:

1. Go to the AnimationView class. We wrote some code for this class in Phase 2. Add the following fields: (Remember to import Android Classes as needed)

```
 8
 9 public class AnimationView extends View{
10
11     // Rate for Animation
12     private final int FRAME_RATE = 15;
13
14     // Paint Object - for Setting Graphic Colors
15     private Paint paint;
16
17     // Handler for animation timing
18     private Handler h;
19
20     // Objects in Animation
21     Ball myBall;
22
23
```

2. Write the Constructor for the AnimationView class:

```
24
25      public AnimationView(Context context, AttributeSet attrs) {
26          super(context, attrs);
27
28          // Set the Handler Object
29          h = new Handler();
30
31          // Paint Object
32          paint = new Paint();
33          paint.setColor(Color.BLUE); // Set Paint Color
34
35          // Create Ball Objects
36          myBall = new Ball(100, 100, Color.BLUE, 50);
37
38          // Set Speed of myBall
39          myBall.setDX(10);
40          myBall.setDY(10);
41
42      } // end Constructor AnimationView
43
```

3. We now need to code the onDraw() function. In Android View objects, the onDraw() function 'repaints' the screen. By adding drawing commands (via the Canvas object c) we can draw shapes and image files onto the View. The onDraw() will also calculate the model and then call the Runnable to set up the animation and Game Loop Cycle.

```
44
45      protected void onDraw(Canvas c) {
46          // Calculate the Model
47          myBall.bounce(c); // myBall Move and Bounce off Walls - model
48
49          // Draw the images taking data from Ball models
50          c.drawCircle(myBall.getX(), myBall.getY(), myBall.getRadius(), myBall.getPaint());
51
52          // Call the Runnable to re calculate the model and
53          // Draw the Animation
54          h.postDelayed(r, FRAME_RATE);
55      } // End onDraw
56
```

4. Note that on line 54 the h.postDelated() function is called. We now need to define the Runnable r. Notice that the Runnable does NOT call 'onDraw()' to complete the cycle. Instead, the Runnable r calls the function 'invalidate()'. The 'invalidate()' function within a View class will automatically clear the screen and then call the onDraw() function. This sets up the animation cycle. Because the FRAME_RATE is set to almost 60 times per second, the clear and onDraw() gives the illusion of continuous motion.

```
56
57      private Runnable r = new Runnable() {
58          public void run() {
59              invalidate(); // Calls the onDraw in View Objects
60          }
61      };  // end Runnable r
62
63  } // end Class AnimationView
64
```

Phase 6: Finishing the App with the Main.java class

The Main.java class extends Activity and binds the AnimationView object to the XML representation of AnimationView. If this App had user controlled events such as Buttons or the Accelerometer, the listeners would be coded into the Main.java class. We will build these in later lessons. For now, complete the code in the Main.java file:

Process

1. Open Main.java and complete the Code as follows: (Yes, this is all there is!)

```
1  package com.example.bouncerexample;
2
3  import android.os.Bundle;
4  import android.app.Activity;
5  import android.view.Menu;
6
7  public class Main extends Activity {
8
9      AnimationView animationView;
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15
16         animationView = (AnimationView)findViewById(R.id.animationView);
17     } // End onCreate
18
19     @Override
20     public boolean onCreateOptionsMenu(Menu menu) {
21         // Inflate the menu; this adds items to the action bar if it is present.
22         getMenuInflater().inflate(R.menu.main, menu);
23         return true;
24     }
25
26 } // end class Main
27
```

Phase 7: Adding more Ball Objects

We have done a lot of coding for what seems to be a simple App. However, this coding provides a foundation to add more Ball Objects and User Events. We will now add two additional Ball objects.

Process:

1. Go to AnimationView.java and look at the Fields. Add two more Ball objects to the Fields

```
10
11 public class AnimationView extends View{
12
13     // Rate for Animation
14     private final int FRAME_RATE = 15;
15
16     // Paint Object - for Setting Graphic Colors
17     private Paint paint;
18
19     // Handler for animation timing
20     private Handler h;
21
22     // Objects in Animation
23     Ball myBall;
24     Ball greenBall;
25     Ball redBall;
26
```

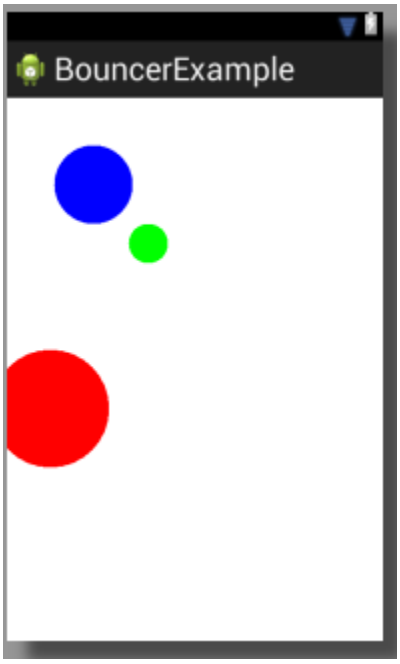
2. On the Constructor, initialize and set the values for greenBall and redBall.

```
37
38         // Create Ball Objects
39         myBall = new Ball(100, 100, Color.BLUE, 50);
40         greenBall = new Ball(200, 200, Color.GREEN, 25);
41         redBall = new Ball(50, 400, Color.RED, 75);
42
43         // Set Speed of myBall
44         myBall.setDX(10);
45         myBall.setDY(10);
46
47         // Set Speeds of greenBall and redBall
48         greenBall.setDX(-20);
49         greenBall.setDY(-15);
50         redBall.setDX(5);
51         redBall.setDY(-5);
52
```

3. Modify the onDraw to include code to model and draw the additional Ball objects:

```
55
56      protected void onDraw(Canvas c) {
57          // Calculate the Model
58          myBall.bounce(c); // myBall Move and Bounce off Walls - model
59          greenBall.bounce(c); // greenBall Bounce
60          redBall.bounce(c); // redBall Bounce
61
62          // Draw the images taking data from Ball models
63          c.drawCircle(myBall.getX(), myBall.getY(), myBall.getRadius(), myBall.getPaint());
64          c.drawCircle(greenBall.getX(), greenBall.getY(), greenBall.getRadius(), greenBall.getPaint());
65          c.drawCircle(redBall.getX(), redBall.getY(), redBall.getRadius(), redBall.getPaint());
66
67          // Call the Runnable to re calculate the model and
68          // Draw the Animation
69          h.postDelayed(r, FRAME_RATE);
70      } // End onDraw
71
```

4. Save, Test, and Run. You should see three Balls bouncing on the Canvas.



Assignment Grading: (Maximum 25 Points)

Create an Animation Sample that explores the Canvas draw functions, Android Color, velocity change, and Game Object class structure.

1. Complete the directions as presented: (17 Points)
2. Add additional Android Canvas Drawing elements via instances of Ball or another class your create: (1 Point per element up to 2 Points)
3. Implement different colors within the Additional Objects: (1 Point per color up to 2 Points)
4. Incorporate Code from the Pedometer App to have Accelerometer events control the velocity of the Animated Objects: (Up to 3 Points)
5. Make a JING demonstrating the Bouncing App Running on Emulator. (Up to 3 Points)

Extra Points will be added to previous projects for extra credit