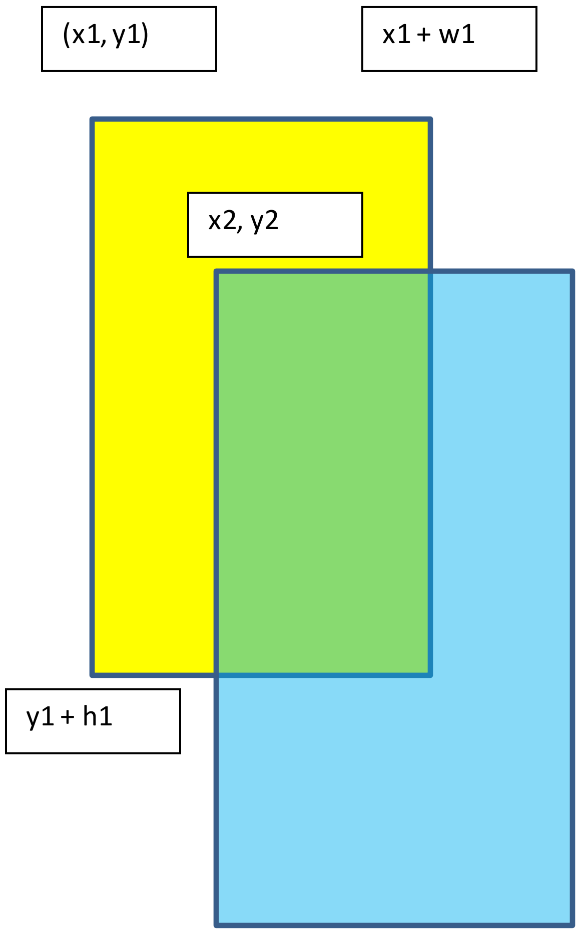
Collision Detection and Array Lists
Mobile Application Development
Marist School

In this series of lessons we will example how to implement collision detection between two Actors in our 2D Graphic/Animation Environment. We will also implement an Array List structure to animate and control a group of Actor instances.

Collision detection is the act of measuring if two Actors are overlapping or touching. We will simulate this by adding fields that store the width and height values of the Actor. We will then write a boolean function to return a true or false reporting on whether two actors are overlapping. The diagram below illustrates the function:



Question for collision:

x2 + w2 > x1 AND x2 < x1+w1

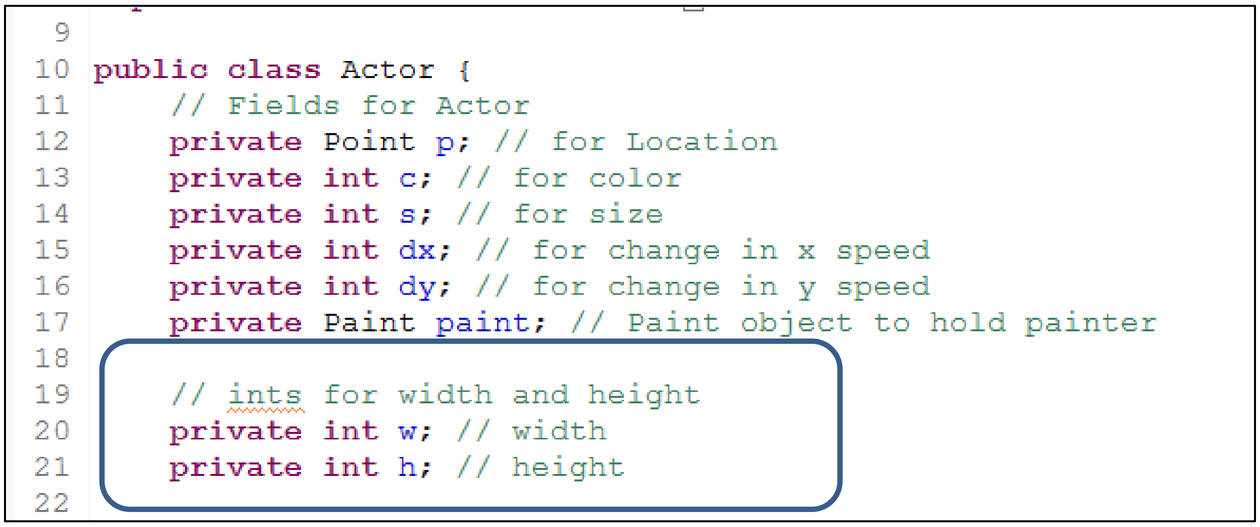
AND

y2 + h2 > y1 AND y2 < y1 + h1

We will now modify the Actor class to have a collision detection function:

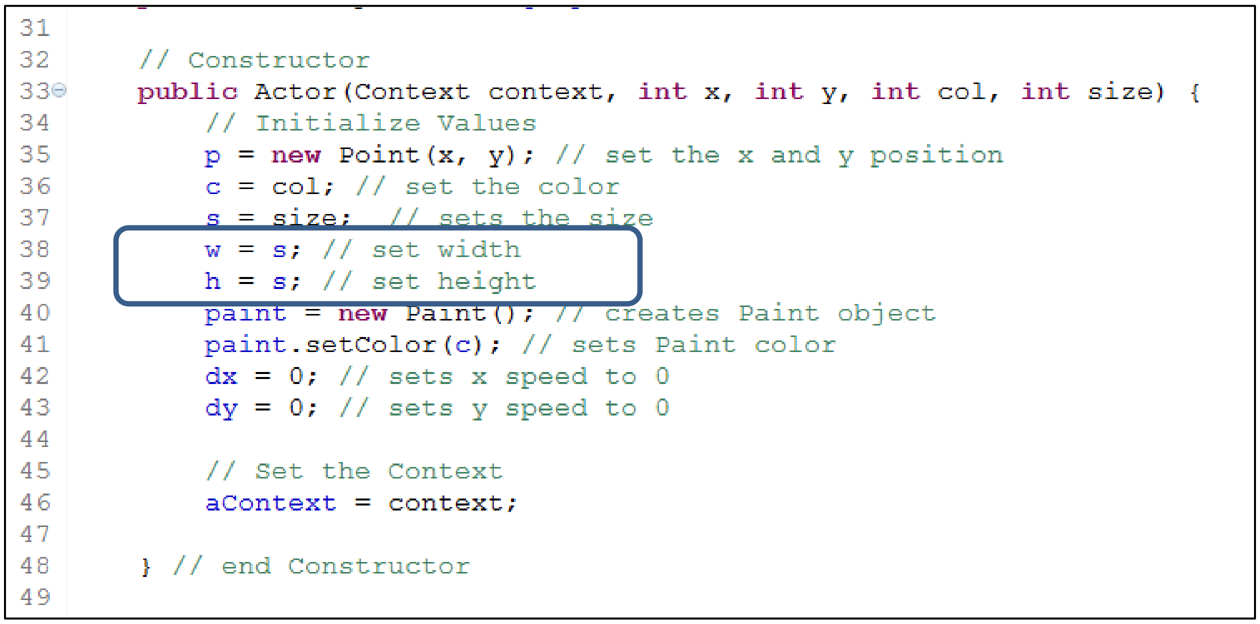
Process:

1. Go to the Actor Class and create two new fields: w for width and h for height:



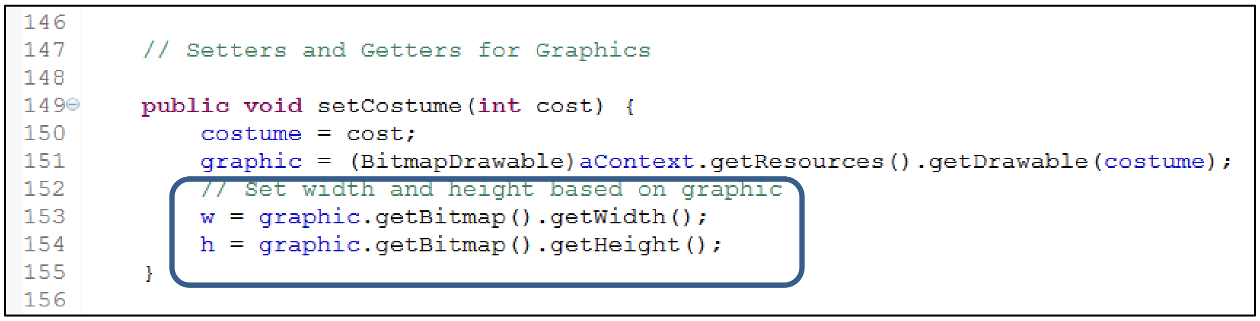
```java
public class Actor {
    // Fields for Actor
    private Point p; // for Location
    private int c; // for color
    private int s; // for size
    private int dx; // for change in x speed
    private int dy; // for change in y speed
    private Paint paint; // Paint object to hold painter

    // ints for width and height
    private int w; // width
    private int h; // height

```

2. We now need to initialize the values for these fields. We will grab these values from the size field or from the Bitmap if the Actor instance draws a picture. Go to the Actor constructor and add the two lines of code to set values for w and h.



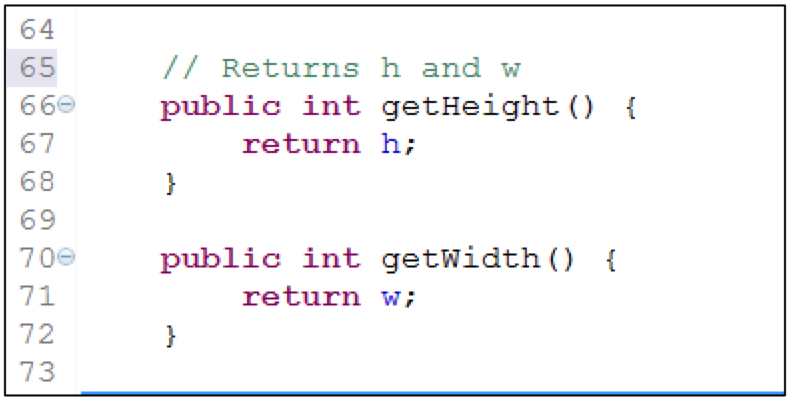
```java

    // Constructor
    public Actor(Context context, int x, int y, int col, int size) {
        // Initialize Values
        p = new Point(x, y); // set the x and y position
        c = col; // set the color
        s = size;  // sets the size
        w = s; // set width
        h = s; // set height
        paint = new Paint(); // creates Paint object
        paint.setColor(c); // sets Paint color
        dx = 0; // sets x speed to 0
        dy = 0; // sets y speed to 0

        // Set the Context
        aContext = context;

    } // end Constructor

```

3. Go to the setCostume() function (we wrote this in the last lesson) and add the lines of code to set the values for w and h from the dimensions of the graphic:



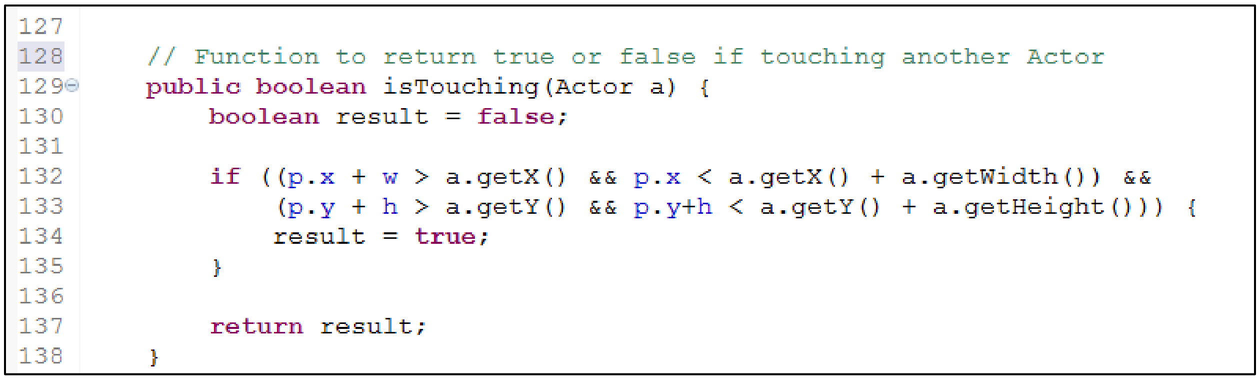
```

    // Setters and Getters for Graphics

    public void setCostume(int cost) {
        costume = cost;
        graphic = (BitmapDrawable)aContext.getResources().getDrawable(costume);
        // Set width and height based on graphic
        w = graphic.getBitmap().getWidth();
        h = graphic.getBitmap().getHeight();
    }

```

4. Write the accessors for the width and height (In the Actor Class):



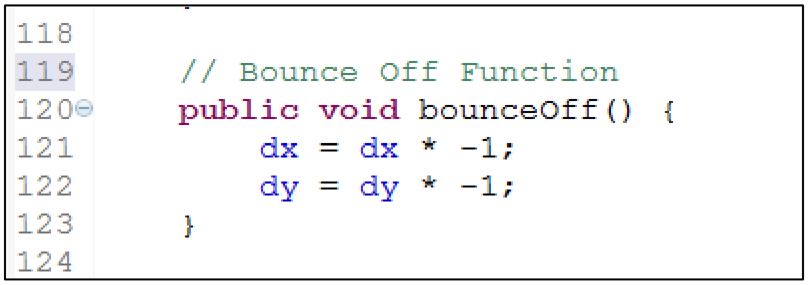
```

    // Returns h and w
    public int getHeight() {
        return h;
    }

    public int getWidth() {
        return w;
    }

```

5. We will now write a new function that returns true or false if the Actor "is touching" another Actor:



```

    // Function to return true or false if touching another Actor
    public boolean isTouching(Actor a) {
        boolean result = false;

        if ((p.x + w > a.getX() && p.x < a.getX() + a.getWidth()) &&
            (p.y + h > a.getY() && p.y+h < a.getY() + a.getHeight())) {
            result = true;
        }

        return result;
    }
```

6. We will now write a function to 'bounceOff' that will change the dx and dy values by -1:

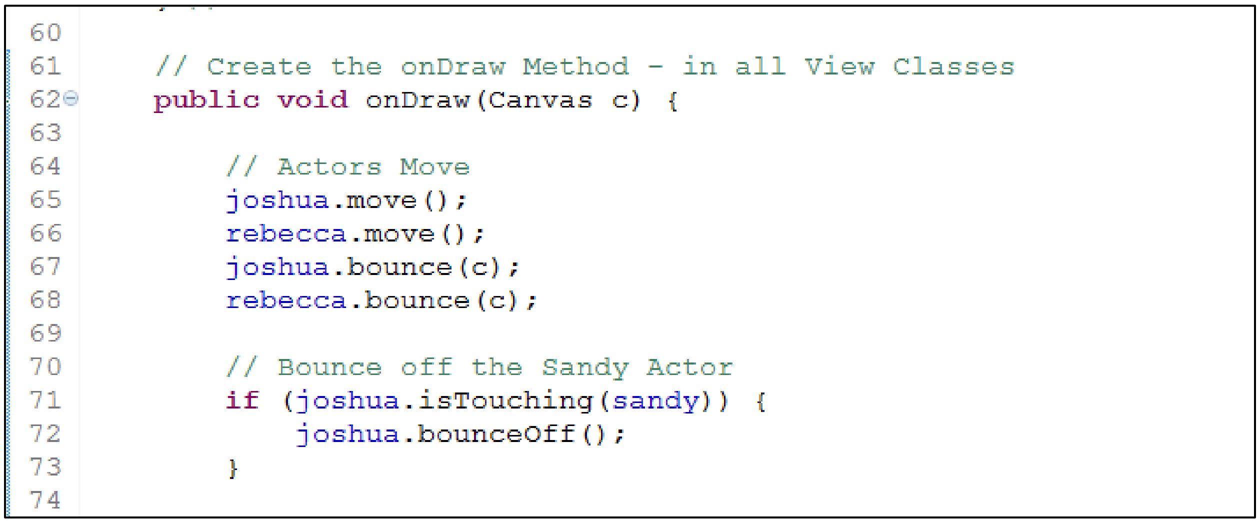


```
    // Bounce Off Function
    public void bounceOff() {
        dx = dx * -1;
        dy = dy * -1;
    }
```

7. We will now add some code in AnimationView to test this new function. Switch to AnimationView.

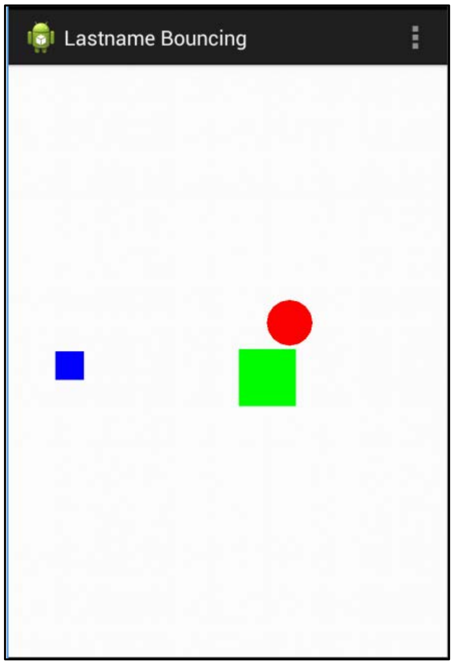
8. In my example, the Actor sandy is follows the touch of the user (the green square). We will write code to have the other objects bounce when touching Actor sandy.

9. Go to the onDraw() in the AnimationView class. Add the code for Actor joshua to bounce off sandy:

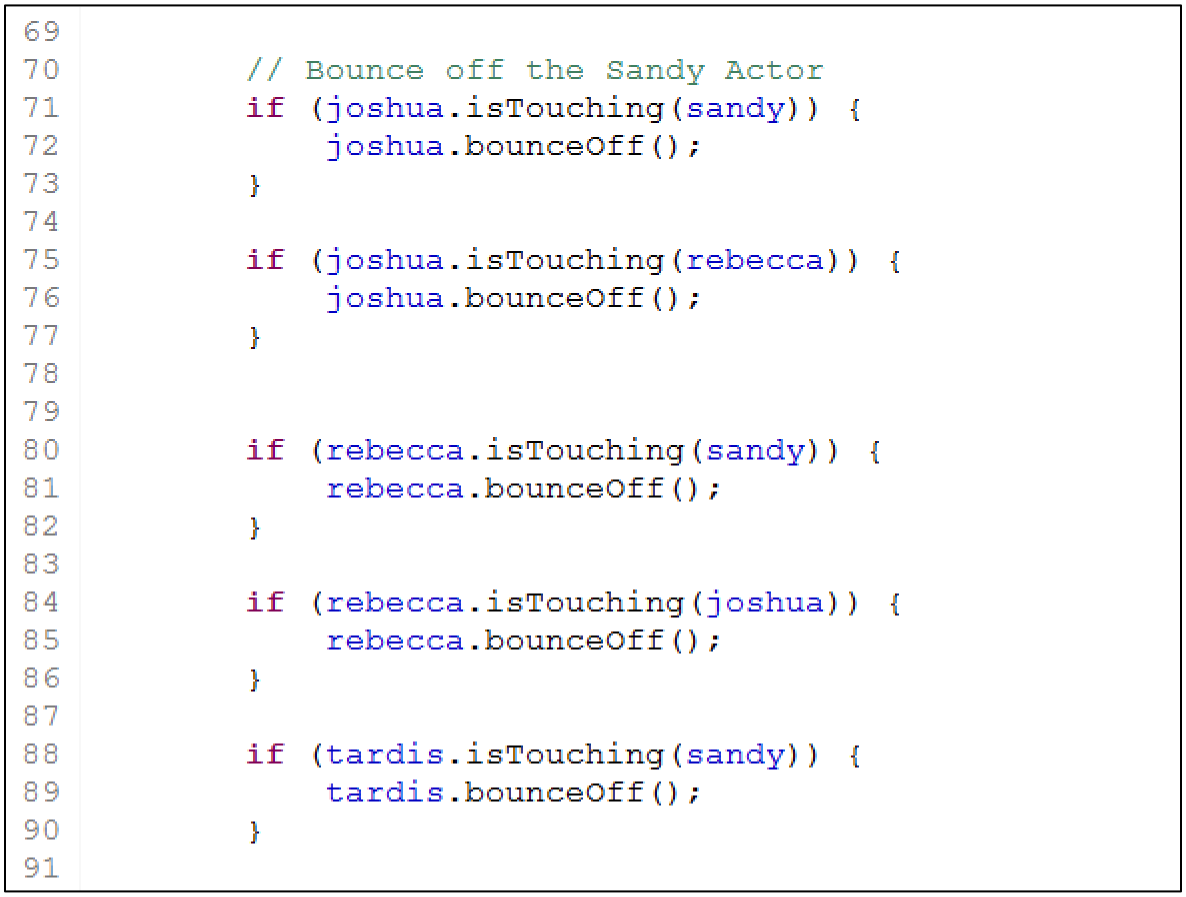


```

    // Create the onDraw Method - in all View Classes
    public void onDraw(Canvas c) {

        // Actors Move
        joshua.move();
        rebecca.move();
        joshua.bounce(c);
        rebecca.bounce(c);

        // Bounce off the Sandy Actor
        if (joshua.isTouching(sandy)) {
            joshua.bounceOff();
        }

```

10. Save and test the code. One of your Actors should be bouncing off another Actor (in my example it is the red circle bouncing off the green square.)



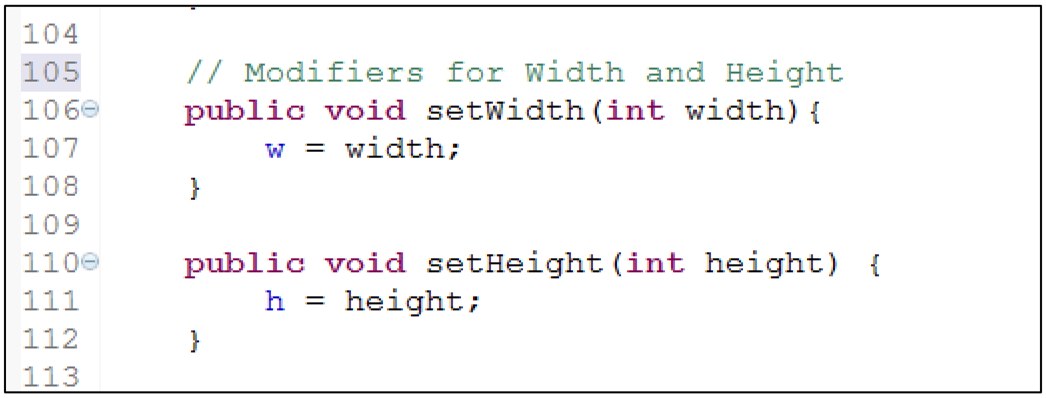
11. Add more code to have the Actors bounce. Here are some examples:



```java

        // Bounce off the Sandy Actor
        if (joshua.isTouching(sandy)) {
            joshua.bounceOff();
        }

        if (joshua.isTouching(rebecca)) {
            joshua.bounceOff();
        }


        if (rebecca.isTouching(sandy)) {
            rebecca.bounceOff();
        }

        if (rebecca.isTouching(joshua)) {
            rebecca.bounceOff();
        }

        if (tardis.isTouching(sandy)) {
            tardis.bounceOff();
        }

```

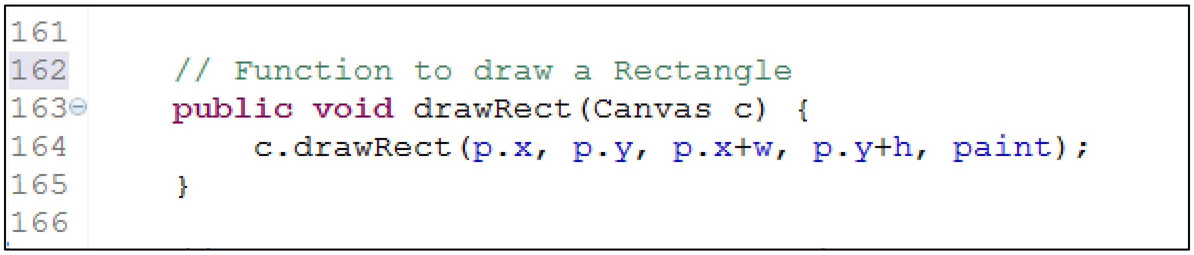
Creating a Breakout Game

12. We will now modify the program and Actors to create a Breakout like game. First, we need to add some functions to the Actor class. Go to the Actor class and add these modifiers for w and h:



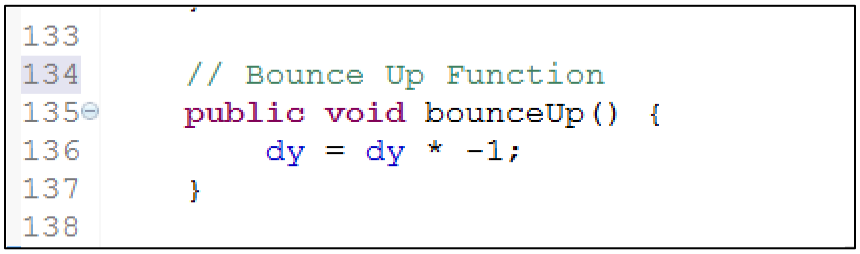
```
    // Modifiers for Width and Height
    public void setWidth(int width){
        w = width;
    }

    public void setHeight(int height) {
        h = height;
    }
```

13. Second, we need to add a function to draw a rectangle (so we can have a paddle). Add this function to the Actor class:



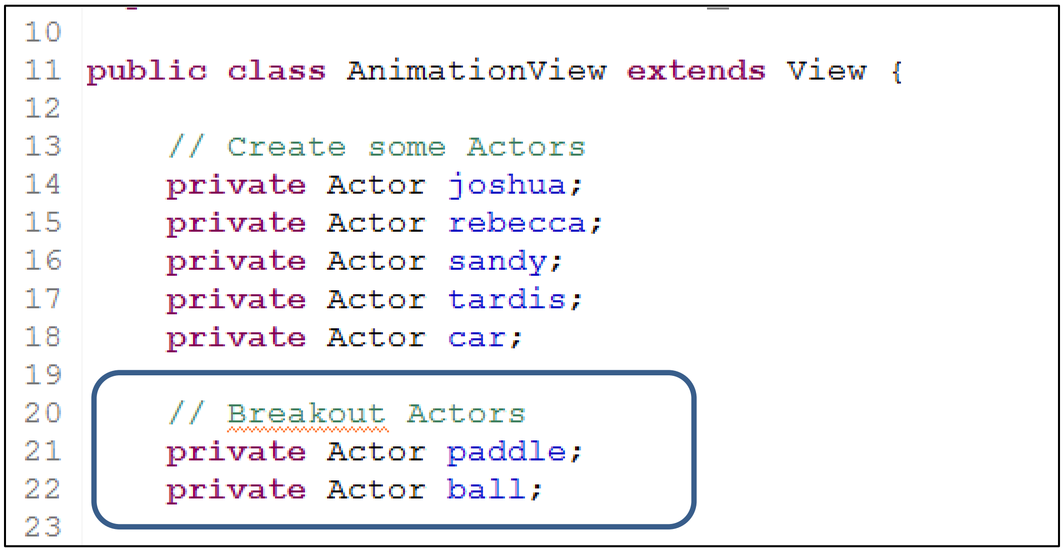
```
    // Function to draw a Rectangle
    public void drawRect(Canvas c) {
        c.drawRect(p.x, p.y, p.x+w, p.y+h, paint);
    }
```

14. We also want a Function to bounce up (change the bouncing behavior). Add this additional function to the Actor class.



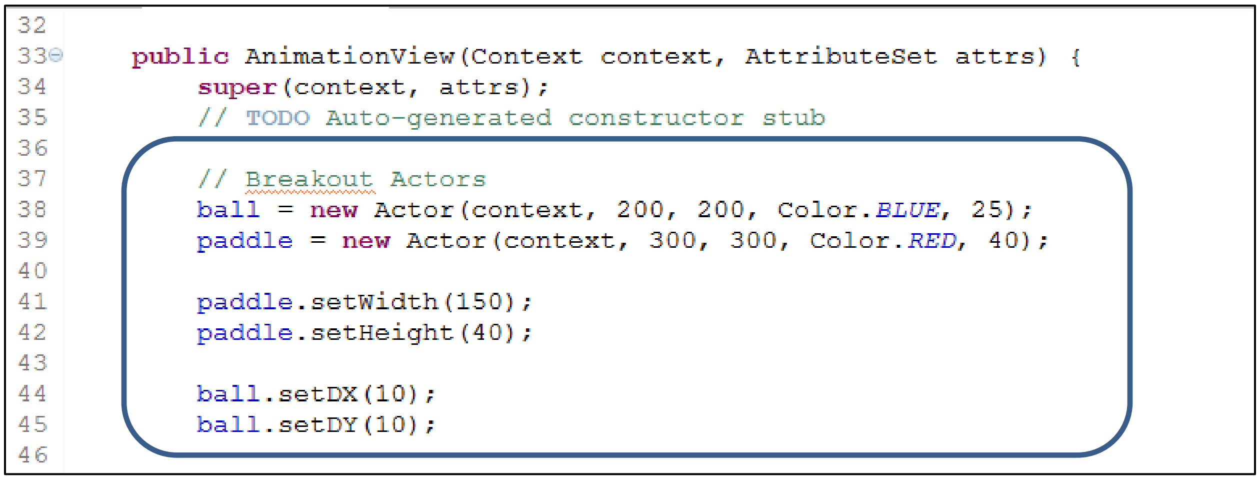
```
    // Bounce Up Function
    public void bounceUp() {
        dy = dy * -1;
    }
```

15. We will now create some new Actors in the AnimationView class. Go to the AnimationView class and add fields for Actors paddle and ball:



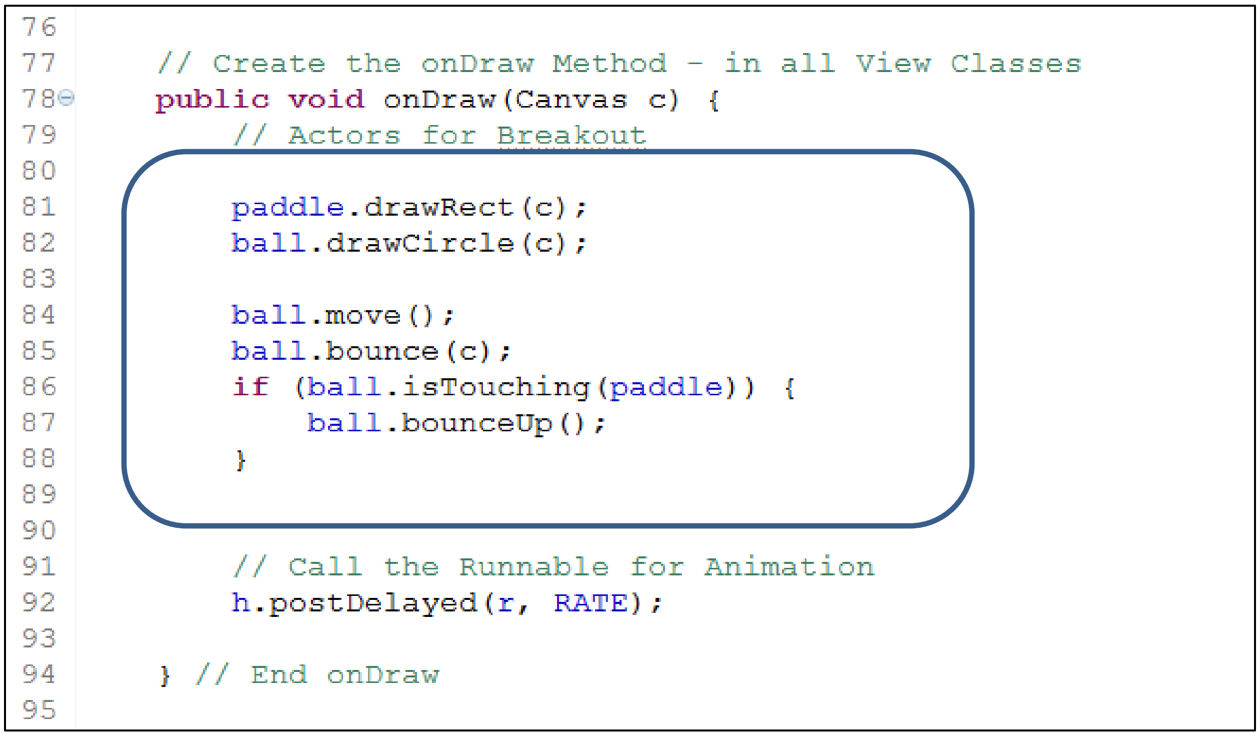
```

public class AnimationView extends View {

    // Create some Actors
    private Actor joshua;
    private Actor rebecca;
    private Actor sandy;
    private Actor tardis;
    private Actor car;

    // Breakout Actors
    private Actor paddle;
    private Actor ball;

```

16. Go to the constructor and write the initializers for the paddle and ball Actors:



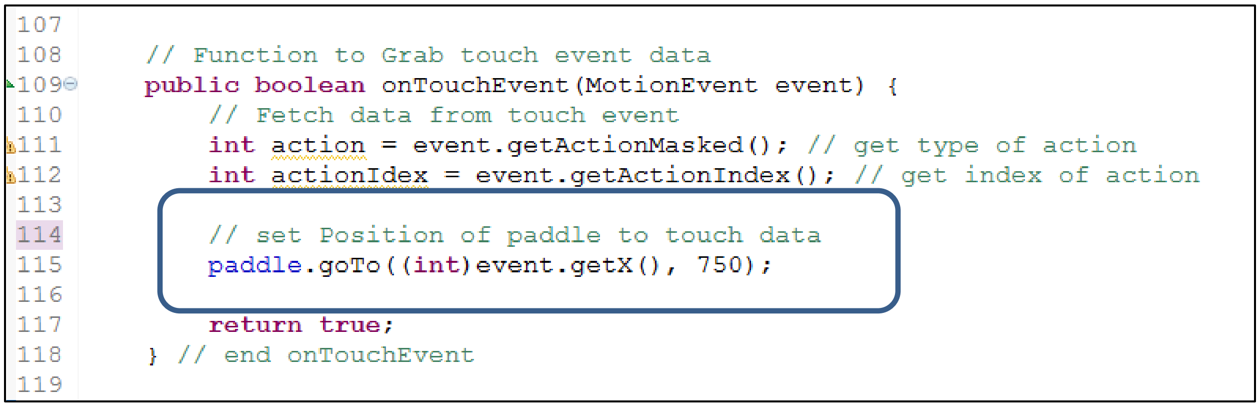
```

    public AnimationView(Context context, AttributeSet attrs) {
        super(context, attrs);
        // TODO Auto-generated constructor stub

        // Breakout Actors
        ball = new Actor(context, 200, 200, Color.BLUE, 25);
        paddle = new Actor(context, 300, 300, Color.RED, 40);

        paddle.setWidth(150);
        paddle.setHeight(40);

        ball.setDX(10);
        ball.setDY(10);

```

17. Now go to the onDraw() function and add the code to draw the paddle and ball and have the ball bounce off the paddle: (Note that I removed the older code for now)



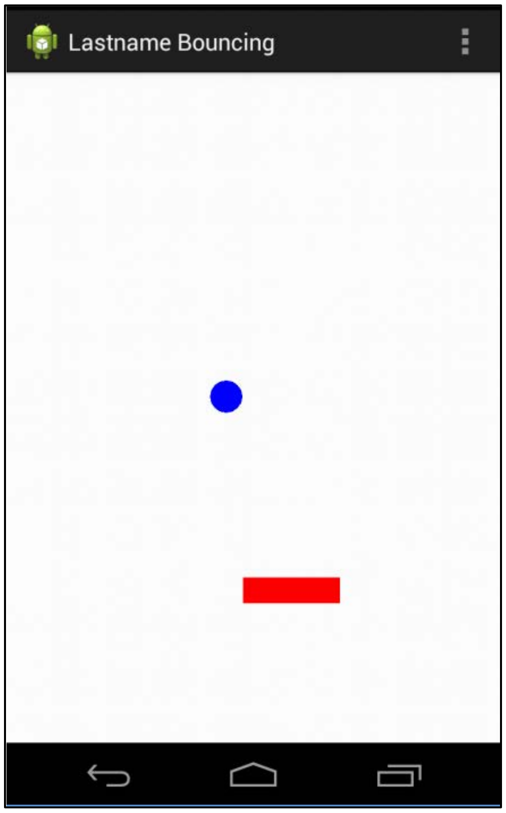
```

    // Create the onDraw Method - in all View Classes
    public void onDraw(Canvas c) {
        // Actors for Breakout

        paddle.drawRect(c);
        ball.drawCircle(c);

        ball.move();
        ball.bounce(c);
        if (ball.isTouching(paddle)) {
            ball.bounceUp();
        }


        // Call the Runnable for Animation
        h.postDelayed(r, RATE);

    } // End onDraw

```

18. We now need to have the paddle move with the touch. Go to the onTouchEvent() function and modify it so the paddle will stay with the x position of the touch.



```

    // Function to Grab touch event data
    public boolean onTouchEvent(MotionEvent event) {
        // Fetch data from touch event
        int action = event.getActionMasked(); // get type of action
        int actionIdex = event.getActionIndex(); // get index of action

        // set Position of paddle to touch data
        paddle.goTo((int)event.getX(), 750);

        return true;
    } // end onTouchEvent

```

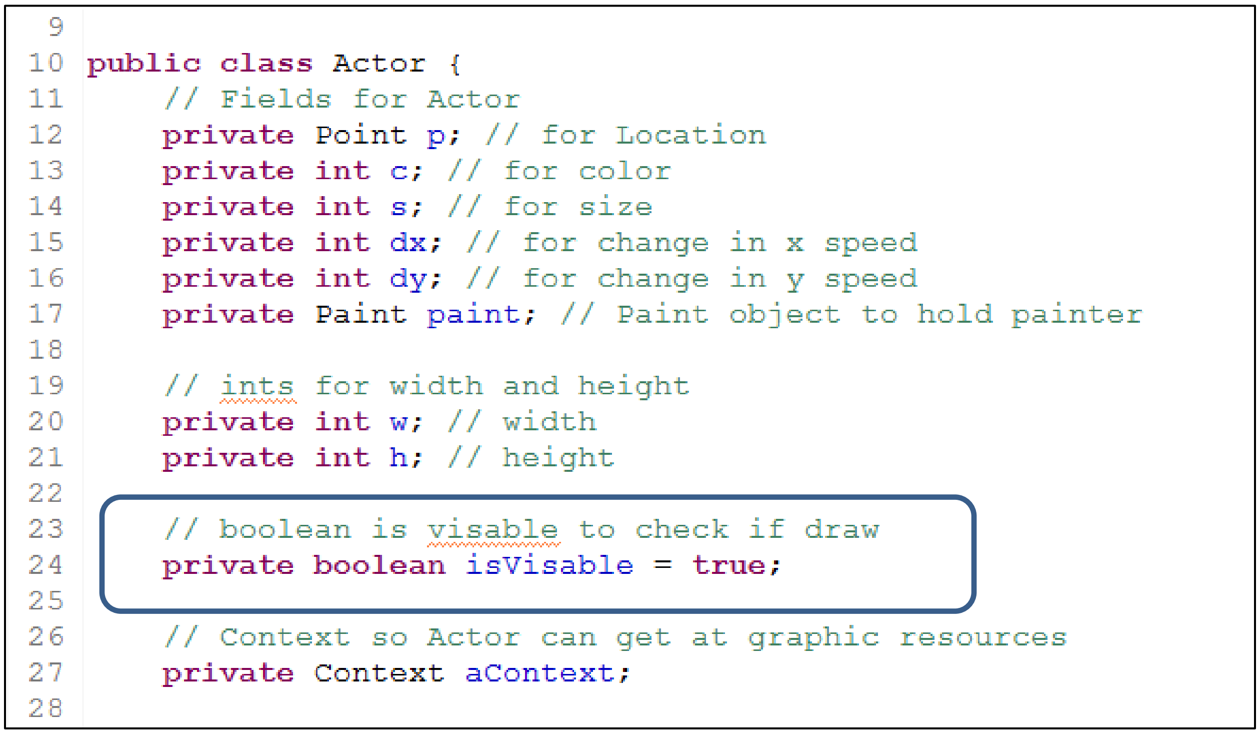
19. Save and run the code. The paddle should move with touch and the ball bounce off the paddle.



Using an ArrayList object to create, store, and manipulate multiple Actors.

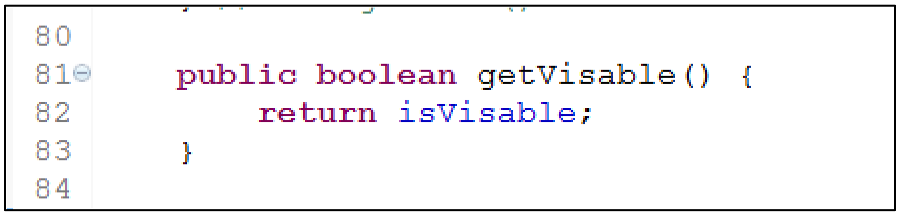
We now need to create the bricks using an ArrayList structure. We will also make some modifications for the Actor class for visability.

20. Add an isVisable field to the Actor class:



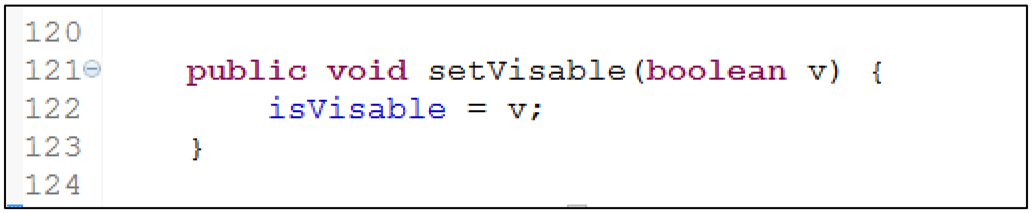
```java
public class Actor {
    // Fields for Actor
    private Point p; // for Location
    private int c; // for color
    private int s; // for size
    private int dx; // for change in x speed
    private int dy; // for change in y speed
    private Paint paint; // Paint object to hold painter

    // ints for width and height
    private int w; // width
    private int h; // height

    // boolean is visable to check if draw
    private boolean isVisable = true;

    // Context so Actor can get at graphic resources
    private Context aContext;
```

21. Write an accessor function so we can get isVisable values in other classes:



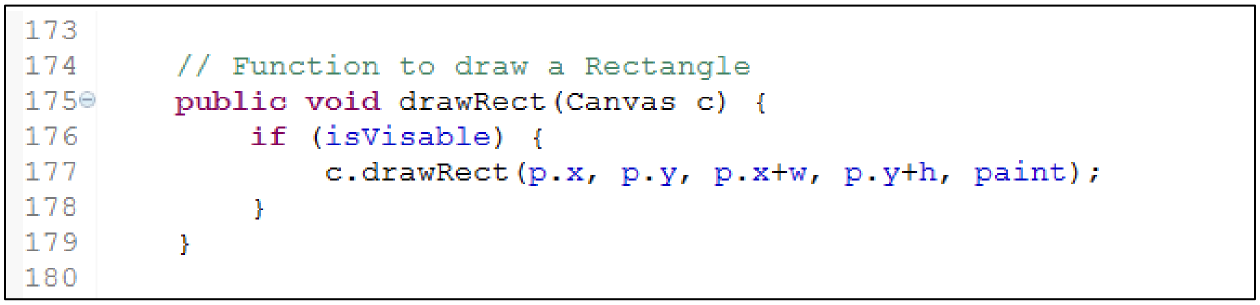
```java
    public boolean getVisable() {
        return isVisable;
    }
```

22. Write a modifier function so other classes can change the isVisable:



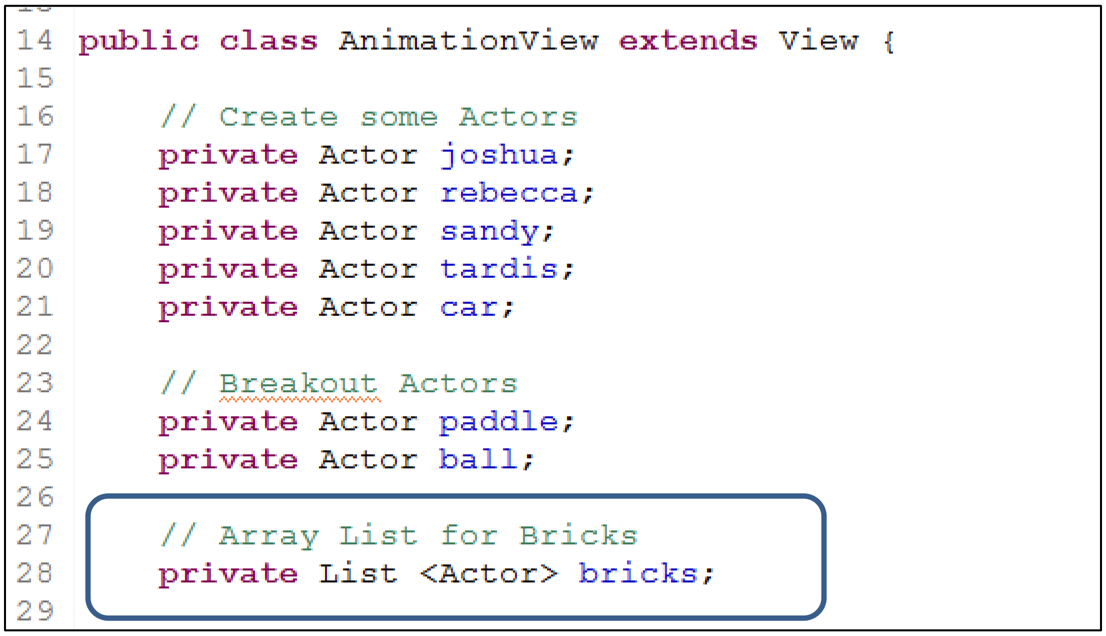
```java
    public void setVisable(boolean v) {
        isVisable = v;
    }
```

23. Modify the drawRect() function to only draw when isVisable is true:



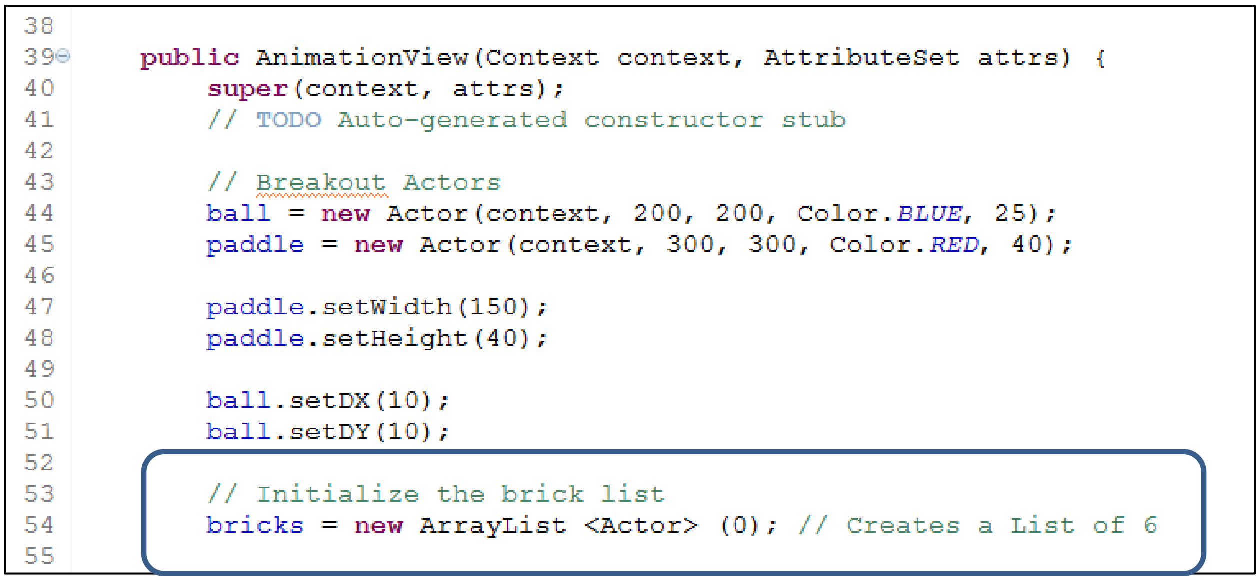
```java

    // Function to draw a Rectangle
    public void drawRect(Canvas c) {
        if (isVisable) {
            c.drawRect(p.x, p.y, p.x+w, p.y+h, paint);
        }
    }

```

24. Now go to the AnimationView class and find the fields. Add the following code to the fields to create a List of Actors named bricks



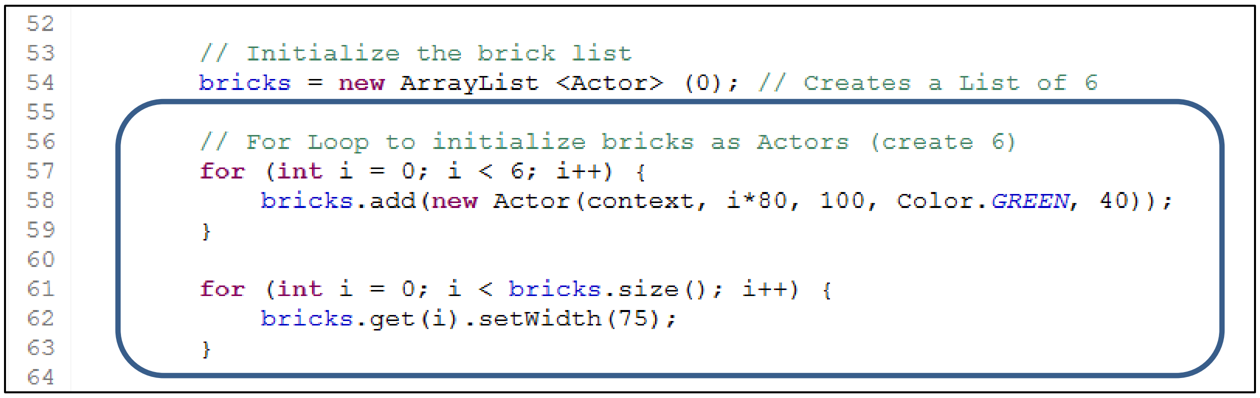
```java
public class AnimationView extends View {

    // Create some Actors
    private Actor joshua;
    private Actor rebecca;
    private Actor sandy;
    private Actor tardis;
    private Actor car;

    // Breakout Actors
    private Actor paddle;
    private Actor ball;

    // Array List for Bricks
    private List <Actor> bricks;

```

25. Now go to the constructor for the AnimationView class and write the code to initialize the ArrayList object:



```

    public AnimationView(Context context, AttributeSet attrs) {
        super(context, attrs);
        // TODO Auto-generated constructor stub

        // Breakout Actors
        ball = new Actor(context, 200, 200, Color.BLUE, 25);
        paddle = new Actor(context, 300, 300, Color.RED, 40);

        paddle.setWidth(150);
        paddle.setHeight(40);

        ball.setDX(10);
        ball.setDY(10);

        // Initialize the brick list
        bricks = new ArrayList <Actor> (0); // Creates a List of 6

```

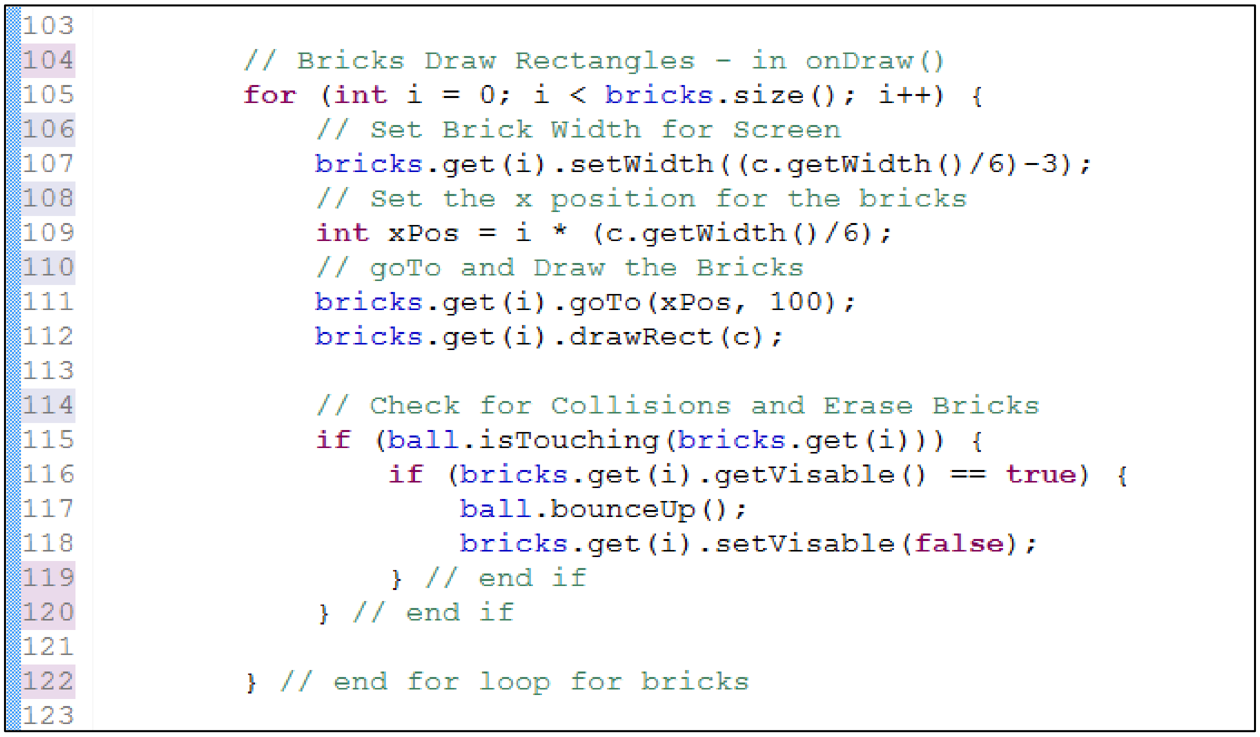
26. We will now use a for loop structure to add Actors to the bricks List. We will use a second for loop to set the Width of the Actors to 75.



```

        // Initialize the brick list
        bricks = new ArrayList <Actor> (0); // Creates a List of 6

        // For Loop to initialize bricks as Actors (create 6)
        for (int i = 0; i < 6; i++) {
            bricks.add(new Actor(context, i*80, 100, Color.GREEN, 40));
        }

        for (int i = 0; i < bricks.size(); i++) {
            bricks.get(i).setWidth(75);
        }

```

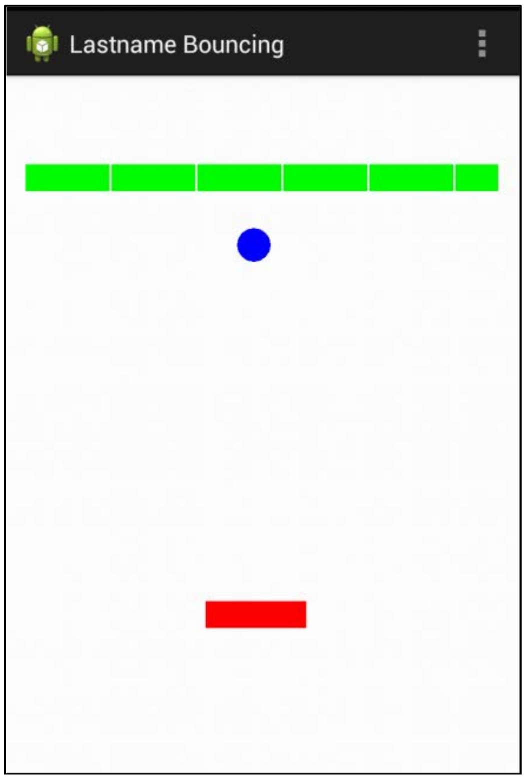
27. Now the magic begins! We want to use a for loop to iterate through the list of bricks and do the following:
    a. Calculate the Width of the brick based on the canvas size
    b. Calculate the placement of the bricks based on canvas size
    c. Draw each brick
    d. Check for collisions with the ball and then erase the bricks when needed

28. Go to the onDraw and add this section of code to handle the bricks:



```

        // Bricks Draw Rectangles - in onDraw()
        for (int i = 0; i < bricks.size(); i++) {
            // Set Brick Width for Screen
            bricks.get(i).setWidth((c.getWidth()/6)-3);
            // Set the x position for the bricks
            int xPos = i * (c.getWidth()/6);
            // goTo and Draw the Bricks
            bricks.get(i).goTo(xPos, 100);
            bricks.get(i).drawRect(c);

            // Check for Collisions and Erase Bricks
            if (ball.isTouching(bricks.get(i))) {
                if (bricks.get(i).getVisable() == true) {
                    ball.bounceUp();
                    bricks.get(i).setVisable(false);
                } // end if
            } // end if

        } // end for loop for bricks

```

29. Save and run the code. The result should look something like:



30. With some thought and creativity, you can add additional rows of bricks and even sound and scoring. Research the Canvas class to see what other functions you can add to the Actor. (Writing Text to screen, changing colors . . .).