ANDROID PROGRAMMING EXERCISES

Kari Laitinen
http://www.naturalprogramming.com
2019-01-07 File created.
2019-01-30 Last modification.

EXERCISES WITH SquareBallRectangle

Among the Android example apps you can find an app named SquareBallRectangle, which shows a square, a ball, or a rectangle depending on what has been selected with radio buttons. The app consists of the following files

main/java/square/ball/rectangle/SquareBallRectangleActivity.java
main/java/square/ball/rectangle/SquareBallRectangleView.java
main/res/layout/activity_square_ball_rectangle.xml

You can make this app work in your Android Studio when you create a project with the name SquareBallRectangle. You must name the main activity class `SquareBallRectangleActivity`, and the layout file must be named **activity_square_ball_rectangle.xml**. It is also important that the package name is `square.ball.rectangle`

After you have created a project and followed the above naming rules, you can copy the mentioned files from the web to the corresponding folders in your Android project. When you copy the files, you can overwrite the files that Android Studio has created.

Exercise 1:

Modify the program so that in place of the rectangle a triangle will be drawn. You should modify the .xml file so that the text ’Triangle’ is selectable with the radio buttons.

The `Canvas` class does not provide a single method to draw a triangle, but the triangle can be drawn by using a class named `Path`. With the `Path` class it is possible to specify a path through selected graphical points. The `Path` class is used, for example, in the example app FlyingArrow. A `Path` object that specifies a triangle can be constructed, for example, with the following statements.

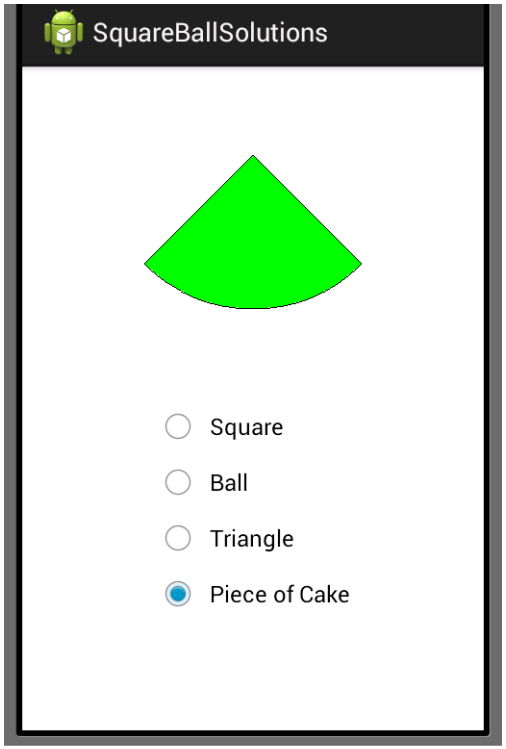
```
Path  triangle_path  =  new  Path() ;

triangle_path.moveTo( view_center_point_x, view_center_point_y - 80 ) ;
triangle_path.lineTo( view_center_point_x + 96, view_center_point_y + 80 ) ;
triangle_path.lineTo( view_center_point_x - 96, view_center_point_y + 80 ) ;
triangle_path.close() ;
```

When the `Path` object that represents the triangle is constructed with the above statements, the triangle will appear in the middle of the `View` object, and it is not necessary to adjust the zero point of the graphical coordinate system.

Exercise 2:

Improve the program so that a new shape named "Piece of Cake" can be selected. To make this happen you must specify a new RadioButton to the **.xml** file. Only the **.xml** file needs to be modified to make the new RadioButton visible. You can test your program after this modification, and do the actual drawing operation later.

When "Piece of Cake" is selected, the application should look like the following.



A "Piece of Cake" can be drawn by using a `Canvas` method named `drawArc()`. By studying the file **DrawingDemoActivity.java** of the DrawingDemo app, you can find out how to use the method.

For the `drawArc()` method angles are specified clockwise, not counter clockwise as in many other systems. Rectangles in Android are specified with the coordinates of upper left corner and lower right corner.

Exercise 3:

In this exercise you should improve the program so that this application will be a proper Android app. Android documents suggest that texts displayed by an app should not be put into layout files as is done in file **res/layout/activity_square_ball_rectangle.xml**. In well-made Android apps, texts should be stored in file **res/values/strings.xml**. This file probably already exists for your app.

Now your task is to put texts like "Square", "Ball", etc., to the **res/values/strings.xml** file and modify the file **activity_square_ball_rectangle.xml** so that it is able to refer to the strings in the **strings.xml** file. To do this, you should check out how strings are referred to in the layout file of the ButtonDemo application.

Doing this exercise does not modify the operation of the app. To see that your modifications work, you can modify some of the strings slightly. Please, keep the English strings in use. You do not have to modify the Java files in this exercise.

Exercise 4:

In this exercise you should add a new user interface language into your application. If you did the previous exercise correctly, it is easy to add a new language. I suggest that you add Finnish user interface texts to the app. You can do this by adding a new subfolder named `values-fi` to the `res` folder. You can do this with Windows File Explorer, or somehow with Android Studio. When the `values-fi` folder exists, you can make a copy of the **strings.xml** in the `values` folder, and put it into the `values-fi` folder. Now you just need to put Finnish words into the `values-fi/strings.xml` file.

Here are some English-Finnish translations: Square > Neliö, Ball > Pallo, Triangle > Kolmio, Piece of Cake >Kakkupala.

When you run your app in the emulator, it should automatically start using the Finnish texts when you change the language to Finnish in the emulator. (Finnish is Suomi in Finnish.) You may need to stop the application through the Settings before it changes the language.

Java files do not need to be modified in this exercise.

More user interface languages can be added to Android apps by creating new folders like `values-es`, `values-de`, etc., and each folder will then contain **.xml** files with suitable texts.

Exercise 5:

If you still have enthusiasm left, you can try out what is it like to do Kotlin programming. Kotlin is a new programming language that can be used to create Android applications.

There is a Kotlin version of this application named **SquareBallRectangleKt**. You can find it in **in_kotlin** subfolder. To take that application in use, you must create a new Android project in which you select Kotlin at the beginning. Package name and the name of the main activity class are the same as in the corresponding Java project.

After creating the project, you should copy the following files from the web:

main/java/square/ball/rectangle/SquareBallRectangleActivity.kt
main/java/square/ball/rectangle/SquareBallRectangleView.kt
main/res/layout/activity_square_ball_rectangle.xml

You could do the first exercise with the Kotlin version. After this you can say that you have some experience in using the Kotlin language.

I would like to warn you that Android Studio will warn about the Kotlin programs written by me. Most names in my programs are written with underscore characters, and for some reason Android Studio warns about those kinds of names.

Rest of these exercises will be based on Java.

EXERCISES WITH MovingBall

Among the Android examples you can find an app named MovingBall, whose main class is `MovingBallActivity`, which is stored in **MovingBallActivity.java**.

The MovingBall app consist of the following files in the file hierarchy of the project:

java/moving/ball/MovingBallActivity.java
java/moving/ball/MovingBallView.java
res/layout/activity_moving_ball.xml
res/menu/color_selection_menu.xml
res/values/strings.xml

All these files need to be copied to the local project in order to make the app operational in the Android Studio. The latest Android Studio does not automatically create the **menu** folder. You can create the **menu** folder as a subfolder of the **res** folder when you copy the file **color_selection_menu.xml.**

When you create the project, you must use `moving.ball` as package name, `MovingBallActivity` as the name of the main activity class, and `activity_moving_ball` as the name of the layout file. When a project is created, the latest Android Studio may not allow to set the names of the main activity class or the layout file. In this case you can edit the **AndroidManifest.xml** file and write `.MovingBallActivity` in place of `.MainActivity`

Do the following exercises after your MovingBall app works locally. Note that you have to do a long press on the COLOR button to get the menu visible.

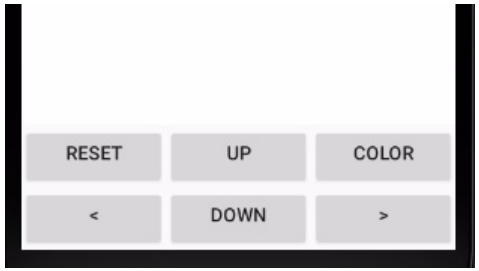
Exercise 1:

Add a new selectable color to the menu.

Exercise 2:

Add a Reset button to the app. With this button it must be possible to put the ball into initial center position, and set the ball color to initial value.

As you add a new button, you should to re-arrange the buttons in the following way

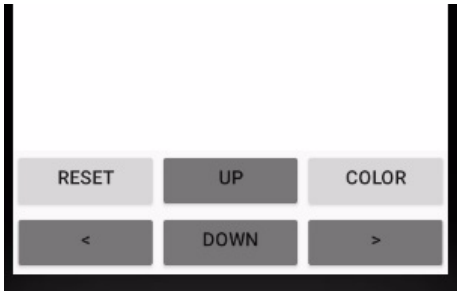


You can re-arrange the buttons by modifying the **activity_moving_ball.xml** file. With the attribute `android:layout_weight` it is possible to specify how much space is reserved for each item inside a `LinearLayout`. In the original file values "0.9" and "0.1" specify that 90% of space is reserved for `MovingBallView` and 10% is reserved for the `LinearLayout` that contains the buttons.

The values of `android:layout_weight` do not need to be numbers with decimal point. The attribute `android:layout_height` may need to be set to "0dp" when `android:layout_weight` is used.

Exercise 3:

Improve the user interface so that you give a different color for the buttons used to move the ball:



The ButtonDemo app shows one way to set the color of a button.

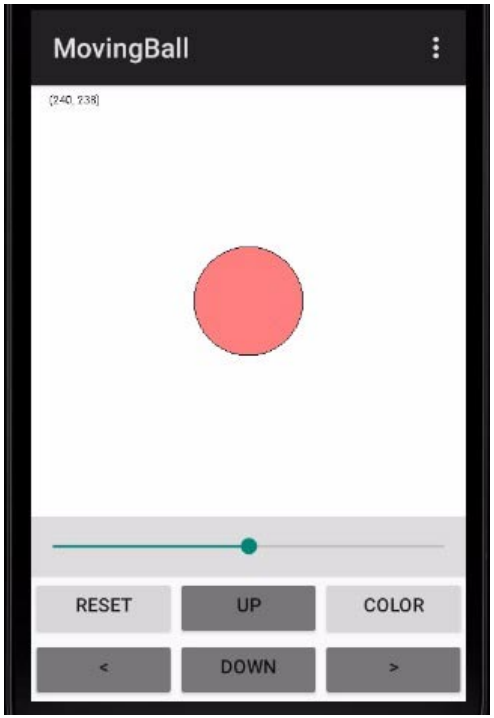
Exercise 4:

Improve the app so that the ball radius is initially 1/8 of the view width, and the ball 'jumps' to to initial middle point if it is moved so that it hits any of the four 'walls' of the view.

Exercise 5:

By studying example app TurningArrow you'll find out how the user interface component `SeekBar` can be used. With a `SeekBar` it is easy to adjust an integer value.

Now your task is to add a `SeekBar` to the MovingBall app, so that the user interface will look like the following



With the `SeekBar` it must be possible to adjust the transparency of the ball on the screen. The transparency (opacity) of a color is specified with the so-called alpha value. The Android `Paint` class has a method named `setAlpha()` with which the alpha value can be set.

The alpha value can be in the range 0 ... 255, ranging from completely transparent color to a fully opaque color. The application should modify the alpha value of the ball filling paint whenever the 'progress' of the `SeekBar` changes. The initial alpha value could be 128.

After you have added the `SeekBar` to the app, you must ensure that the Reset button works correctly. In a Reset operation, the ball alpha value must be set to its initial value, and the 'thumb' of the `SeekBar` must be put to its initial position.

EXERCISES WITH GesturesDemo

Android has a class named `GestureDetector` with which it is possible to find out what kinds of gestures the user does with the touch screen. Touch screen events can be forwarded to a `GestureDetector` object that generates gesture events which include the following.

- onDown: a finger is put on the screen
- onShowPress: the screen is touched for a short time
- onLongPress: a finger remains on the screen a little longer period
- onScroll: a finger moves on the screen
- onFling: a finger is moved quickly
- onSingleTapUp: a finger is lifted

GesturesDemo is an app with which you can explore these gesture events and see when they take place. You can find out, for example, that onSingleTapUp will not take place if a onLongPress has happened.

Use this app first and do then the following exercises. You can keep the original features of the program, as they can be useful when you add new features to the application. Read the comments at the beginning of **GesturesDemoActivity.java** to find out how to make the app work in your Android Studio.

Exercise 1:

Modify the program so that it will display a ball in that screen position which is long-pressed. This means that in the `onLongPress()` method a ball should be created. A ball can be most easily created with a special class. You can get a ready-to-use `Ball` class by copying it from file **MovingBallsWithPointerActivity.java**. You can place the `Ball` class into **GesturesDemoActivity.java** before all other classes.

To use a `Ball` object, you need to define a new data member (object reference) to the `GesturesDemoView` class. This new data field can be written like

```
Ball ball_on_screen ;
```

A `Ball` object can be created in the `onLongPress()` method in the following way.

```
ball_on_screen =
   new  Ball( (int) first_down_motion.getX(),
              (int) first_down_motion.getY(),
               Color.RED ) ;
```

When the `Ball` object is drawn in the `onDraw()` method, you must ensure that the object is really created before it is drawn. Drawing can thus be done in the following way.

```
if ( ball_on_screen != null )
{
   ball_on_screen.draw( canvas ) ;
}
```

Exercise 2:

Improve the feature of the previous exercise so that the ball that is created to the long-pressed position will get a random color.

In Android, colors are expressed as `int` values of the form `0xAARRGGBB`. A color value has components for Alpha, Red, Green, and Blue. The Alpha value must be `0xFF` to get a fully opaque color. (Prefix `0x` specifies a hexadecimal value in programming.)

A random color value can be made with the expression

```
(int) ( Math.random() * 0xFFFFFF ) + 0xFF000000
```

Exercise 3:

Improve the program so that many balls can exist on the screen. A new ball must always appear in the pressed position in a long press. You must now store many `Ball` objects. You can store the objects into a `ArrayList`-based array that can be created in the following way.

```
ArrayList<Ball> balls_on_screen = new ArrayList<Ball>();
```

An `ArrayList` is used in the original program, which should help in this exercise. `Ball` objects can be drawn with a ’foreach’ loop in the following way.

```
for ( Ball ball_to_draw : balls_on_screen )
{
   ball_to_draw.draw( canvas ) ;
}
```

Exercise 4:

Improve the program so that a `Ball` object can be deleted with an `onFling()` operation. This means that you must modify the `onFling()` method so that it deletes that `Ball` object on which the ’flinging’ begings. You must be able to remove a `Ball` object from the `ArrayList`.

In this exercise it is best to process the `ArrayList` starting from its end, because the newest and ’topmost’ `Ball` objects are at the end. In the `Ball` class there exists a method named `contains_point()` with which it is possible to discover if a point is inside the area of a `Ball`. You do not need to modify the `Ball` class. Here are some source code lines that might be helpful:

```
Point first_touched_point = new Point( /* write something here */ ) ;
int ball_index = balls_on_screen.size() ;
boolean ball_to_delete_is_found = false ;

while ( ball_index > 0 && ball_to_delete_is_found == false )
{
   ball_index -- ;

   if ( balls_on_screen.get( ball_index ).
                              contains_point( first_touched_point ) )
   {
      // here you should delete a ball.

      ball_to_delete_is_found = true ;
   }
}
```

Exercise 5:

Improve the program so that all existing balls get a new random color if the screen is tapped quickly, three times within 1.5 seconds.

You can make this feature into the `onSingleTapUp()` method which is called when a short tapping ends. The `MotionEvent` class provides a method named `getEventTime()` which returns milliseconds of the moment when the event took place. If you record the milliseconds of the last three `onSingleTapUp()` events it is possible to calculate when the events occur within the last 1.5 seconds.

This feature can probably be implemented in many ways. One possibility is to use an `ArrayList` where you store `long` values that describe event times:

```
ArrayList<Long> single_tap_up_event_times =
                                    new ArrayList<Long>() ;
```

The program must wait that the `ArrayList` contains three values before you start examining if the values are close to each other. On the other hand, you must delete old values from the beginning of the `ArrayList` so that it always contains only the last three time values. Then you just compare the first and last value in the `ArrayList`. (It might be helpful to remember that the original program keeps a certain number of text lines in an `ArrayList`.)

EXERCISES WITH BouncingBall

Note that these exercises are related to and app named BouncingBall. Avoid using another app named BouncingBallsAnimation.

BouncingBall is an application that displays a ball that rotates and bounces around the screen. The ball can be 'exploded' by touching it with a finger (or mouse). If the ball has exploded and the screen is touched again, the program creates a new ball that starts moving in a random direction.

You can make this app work locally on your computer by just downloading the file

BouncingBallActivity.java

The name of the main activity in your Android Studio project should be `BouncingBallActivity`, and the package name should be `bouncing.ball`

Exercise 1:

Modify the app so that the direction of the ball can be changed with an `onFling()` operation. The aim is that the direction of the ball will be determined by the direction of the finger movement. The finger does not need to touch the ball.

You should study the app named GesturesDemo to find out how to react to gestures on the touch screen. You must write something clever into the `onFling()` method. Here is a list of

things you have to do:

- Make the `BouncingBallView` class implement the `GestureDetector.OnGestureListener` interface. This means that all methods of that interface must be written to the class.
- Create a `GestureDetector` object
- Modify `onTouchEvent()` method so that it 'forwards' the `MotionEvent` objects to the `GestureDetector` object
- Modify the `onFling()` method so that it can calculate a new direction for the ball.
- Add a new method to the `Bouncer` class to set a new direction for the ball. The name of the method can be `set_direction()`

While doing this exercise, it is best first to ensure that you can actually react to 'flingings'. You could, for example, call the `enlarge()` method for the ball inside the `onFling()` method. When you can notice that 'flinging' makes the ball larger, you can continue to make a new direction for the ball.

The `onFling()` method gets `MotionEvent` objects as parameters. These parameters contain the coordinates of the start and end points of the 'flinging'. From this information it is possible to calculate how the finger moved in x and y directions.. By using the `Math.atan2()` method it is possible to calculate a new direction for the ball.

On the following page you can find the source code of the `onFling()` method which calculates a new direction for the ball.

```
public boolean onFling( MotionEvent first_down_motion,
                        MotionEvent last_move_motion,
                        float velocity_x, float velocity_y )
{
   float movement_x = last_move_motion.getX() - first_down_motion.getX() ;
   float movement_y = last_move_motion.getY() - first_down_motion.getY() ;

   /*  The following code seems to change the direction correctly.
       We must negate the value of the y coordinate so that the
       coordinates correspond to the mathematical system used by atan2().
   */

   movement_y = -movement_y ;

   float  new_direction = (float) Math.atan2( movement_y, movement_x ) ;

   ball_on_screen.set_direction( new_direction ) ;

   return true ;
}
```

The `Bouncer` class must contain the following method to make the above code work.

```
public void set_direction( float new_direction )
{
   bouncer_direction = new_direction ;
}
```

Exercise 2:

Improve the program so that it will count the collisions when the the ball collides with the 'walls' of the bouncing area. These collisions can be detected in the `move()` method of the `Bouncer` class, and the number of collisions should be displayed in the `onDraw()` method of the `BouncingBallView` class.

There must be some mechanism to handle the information of the collisions. One way is to add a data field like

```
int collision_counter = 0 ;
```

to the `Bouncer` class. This variable can be incremented always when a collision takes place. This way a `Bouncer` object knows how many times it has collided with the walls. Then there could be a method with which another object can read the number of collisions.

Exercise 3:

Improve the program so that it becomes a kind of game in which the player tries to control the movement of the ball so that it will not hit the walls. If the ball has hit the walls, say, 3 times, the ball will explode and the current ’game’ ends. After a ’game’ has ended it must be possible to start a new game.

Android has standard classes with which it is possible to build various kinds of dialogs. You should use a dialog to find out if the user of the app wants to start a new game. Among the teacher’s example apps there is an app named AlertDialogDemo, which shows one way to make a dialog to start a new game.

If making a dialog seems to be a too difficult task, you can try invent some easier way to start a new game. For example, long-pressing a certain part of the sceen could be a way to initiate a new game.