CSM3505

Native Mobile Programming

Lab 5

Alaa Aldeen Abouzeid
s56608

Task 1: Advanced Android in Kotlin 01.1: Using Android Notifications

MainActivity.kt

```kotlin
/ Copyright (C) 2019 Google Inc. .../

package com.example.android.eggtimernotifications

import ...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)
        if (savedInstanceState == null) {
            supportFragmentManager.beginTransaction()
                .replace(R.id.container, EggTimerFragment.newInstance())
                .commitNow()
        }
    }
}
```

AlarmReceiver.kt

```kotlin
/ Copyright (C) 2019 Google Inc. .../

package com.example.android.eggtimernotifications.receiver

import ...

class AlarmReceiver: BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        // TODO: Step 1.10 [Optional] remove toast
//        Toast.makeText(context, context.getText(R.string.eggs_ready), Toast.LENGTH_SHORT).show()

        // TODO: Step 1.9 add call to sendNotification
        val notificationManager = ContextCompat.getSystemService(
            context,
            NotificationManager::class.java
        ) as NotificationManager

        notificationManager.sendNotification(
            context.getText(R.string.eggs_ready).toString(),
            context
        )

    }

}
```

NotificationUtils.kt

```kotlin

// TODO: Step 1.1 extension function to send messages (GIVEN)
/**
 * Builds and delivers the notification.
 *
 * @param context, activity context.
 */
@SuppressLint( ...value: "WrongConstant")
fun NotificationManager.sendNotification(messageBody: String, applicationContext: Context) {
    // Create the content intent for the notification, which launches
    // this activity
    // TODO: Step 1.11 create intent
    val contentIntent = Intent(applicationContext, MainActivity::class.java)
    // TODO: Step 1.12 create PendingIntent
    val contentPendingIntent = PendingIntent.getActivity(
        applicationContext,
        NOTIFICATION_ID,
        contentIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
    )

    // TODO: Step 2.0 add style
    val eggImage = BitmapFactory.decodeResource(
        applicationContext.resources,
        R.drawable.cooked_egg
    )
    val bigPicStyle = NotificationCompat.BigPictureStyle()
        .bigPicture(eggImage)
        .bigLargeIcon( b: null)
```

```kotlin

    // TODO: Step 2.2 add snooze action
    val snoozeIntent = Intent(applicationContext, SnoozeReceiver::class.java)
    val snoozePendingIntent: PendingIntent = PendingIntent.getBroadcast(
        applicationContext,
        REQUEST_CODE,
        snoozeIntent,
        FLAGS
    )

    // TODO: Step 1.2 get an instance of NotificationCompat.Builder
    // Build the notification
    val builder = NotificationCompat.Builder(
        applicationContext,
        "egg_channel"
    )

        // TODO: Step 1.8 use the new 'breakfast' notification channel



        // TODO: Step 1.3 set title, text and icon to builder
        .setSmallIcon(R.drawable.cooked_egg)
        .setContentTitle(
            applicationContext
                .getString(R.string.notification_title)
        )
        .setContentText(messageBody)

```

```
80
81          // TODO: Step 1.13 set content intent
82          .setContentIntent(contentPendingIntent)
83          .setAutoCancel(true)
84
85          // TODO: Step 2.1 add style to builder
86          .setStyle(bigPicStyle)
87          .setLargeIcon(eggImage)
88
89          // TODO: Step 2.3 add snooze action
90          .addAction(
91              R.drawable.egg_icon,
92              "Snooze",
93              snoozePendingIntent
94          )
95
96          // TODO: Step 2.5 set priority
97          .setPriority(NotificationCompat.PRIORITY_HIGH)
98      // TODO: Step 1.4 call notify
99      notify(NOTIFICATION_ID, builder.build())
100 }
101
102 // TODO: Step 1.14 Cancel all notifications
103 /**
104  * Cancels all notifications.
105  *
106  */
107 fun NotificationManager.cancelNotifications() {
108     cancelAll()
```

EggTimerFragment.kt

```
35  class EggTimerFragment : Fragment() {
36
37      private val TOPIC = "breakfast"
38
39      override fun onCreateView(
40          inflater: LayoutInflater, container: ViewGroup?,
41          savedInstanceState: Bundle?
42      ): View {
43
44          val binding: FragmentEggTimerBinding = DataBindingUtil.inflate(
45              inflater, R.layout.fragment_egg_timer, container,  attachToParent: false
46          )
47
48          val viewModel = ViewModelProviders.of( fragment: this).get(EggTimerViewModel::class.java)
49
50          binding.eggTimerViewModel = viewModel
51          binding.lifecycleOwner = this.viewLifecycleOwner
52
53          // TODO: Step 1.7 call create channel
54          createChannel(
55              getString(R.string.egg_notification_channel_id),
56              getString(R.string.egg_notification_channel_name)
57          )
```

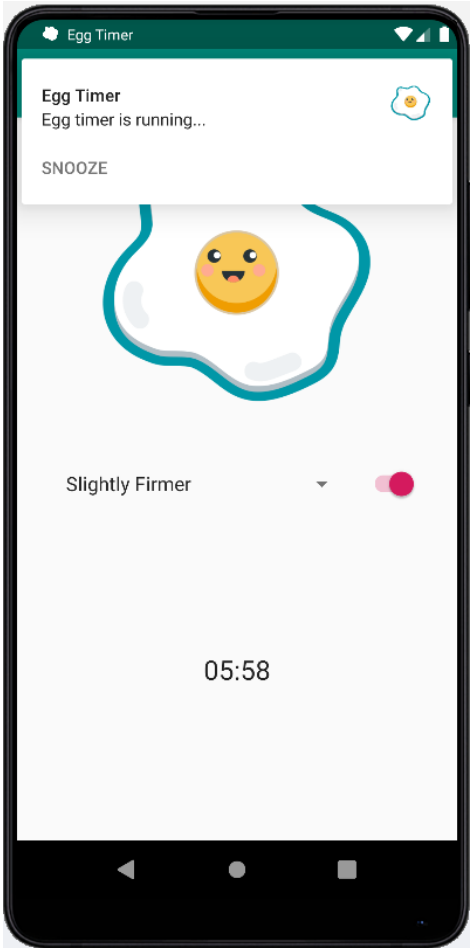
```
        return binding.root
    }
    private fun createChannel(channelId: String, channelName: String) {
        // TODO: Step 1.6 START create a channel
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val notificationChannel = NotificationChannel(
                channelId,
                channelName,
                // TODO: Step 2.4 change importance
                NotificationManager.IMPORTANCE_LOW
            )
            // TODO: Step 2.6 disable badges for this channel

            notificationChannel.enableLights( lights: true)
            notificationChannel.lightColor = Color.RED
            notificationChannel.enableVibration( vibration: true)
            notificationChannel.description = "Time for breakfast"

            val notificationManager = requireActivity().getSystemService(
                NotificationManager::class.java
            )
            notificationManager.createNotificationChannel(notificationChannel)
        }
        // TODO: Step 1.6 END create channel
    }



    companion object {
        fun newInstance() = EggTimerFragment()
    }
```

```
    companion object {
        fun newInstance() = EggTimerFragment()
    }
}
```

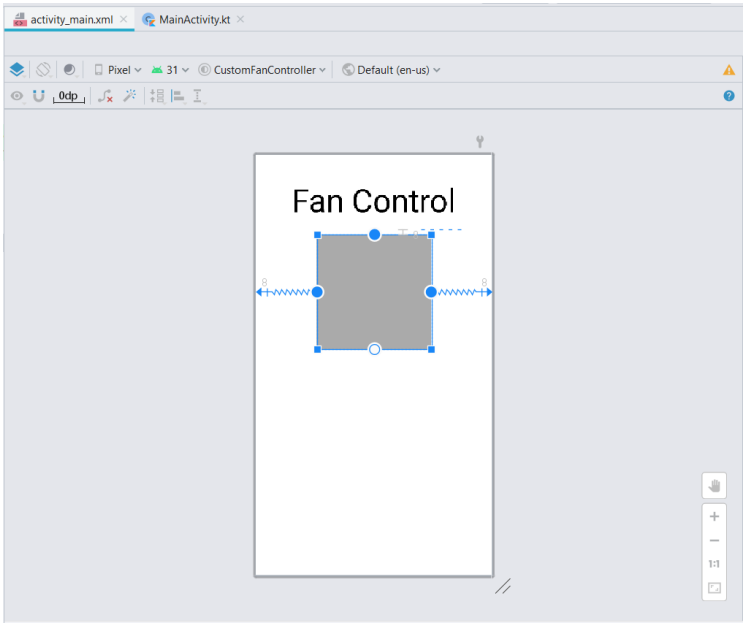
EggTimerViewModel.kt

```
class EggTimerViewModel(private val app: Application) : AndroidViewModel(app) {

    private val REQUEST_CODE = 0
    private val TRIGGER_TIME = "TRIGGER_AT"

    private val minute: Long = 60_000L
    private val second: Long = 1_000L

    private val timerLengthOptions: IntArray
    private val notifyPendingIntent: PendingIntent

    private val alarmManager = app.getSystemService(Context.ALARM_SERVICE) as AlarmManager
    private var prefs =
        app.getSharedPreferences( name: "com.example.android.eggtimernotifications", Context.MODE_PRIVATE)
    private val notifyIntent = Intent(app, AlarmReceiver::class.java)

    private val _timeSelection = MutableLiveData<Int>()
    val timeSelection: LiveData<Int>
        get() = _timeSelection

    private val _elapsedTime = MutableLiveData<Long>()
    val elapsedTime: LiveData<Long>
        get() = _elapsedTime

```

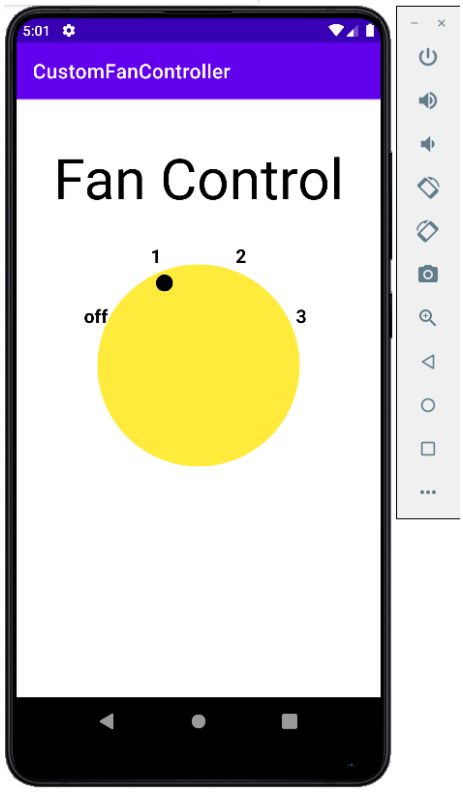
```
57      private var _alarmOn = MutableLiveData<Boolean>()
58      val isAlarmOn: LiveData<Boolean>
59          get() = _alarmOn
60
61
62      private lateinit var timer: CountDownTimer
63
64      init {
65          _alarmOn.value = PendingIntent.getBroadcast(
66              getApplication(),
67              REQUEST_CODE,
68              notifyIntent,
69              PendingIntent.FLAG_NO_CREATE
70          ) != null
71
72          notifyPendingIntent = PendingIntent.getBroadcast(
73              getApplication(),
74              REQUEST_CODE,
75              notifyIntent,
76              PendingIntent.FLAG_UPDATE_CURRENT
77          )
78
79          timerLengthOptions = app.resources.getIntArray(R.array.minutes_array)
80
81          //If alarm is not null, resume the timer back for this alarm
82          if (_alarmOn.value!!) {
83              createTimer()
84          }
85      }
86      fun setAlarm(isChecked: Boolean) {
87          when (isChecked) {
88              true -> timeSelection.value?.let { startTimer(it) }
89              false -> cancelNotification()
90          }
91      }
92
93      /**
94       * Sets the desired interval for the alarm
95       *
96       * @param timerLengthSelection, interval timerLengthSelection value.
97       */
98      fun setTimeSelected(timerLengthSelection: Int) {
99          _timeSelection.value = timerLengthSelection
100     }
```

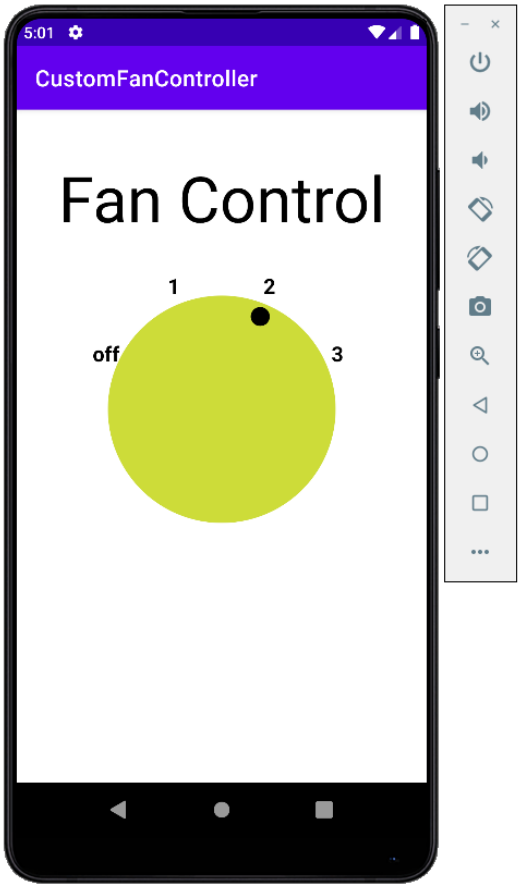


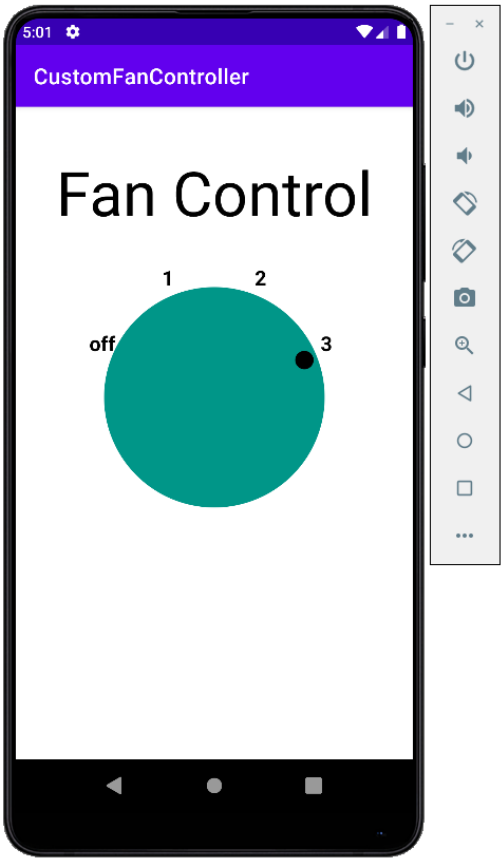
Task 2: Advanced Android in Kotlin 02.1:Creating Custom Views




5:00
CustomFanController
Fan Control
1
2
off
3


5:01
CustomFanController
Fan Control
1
2
off
3


5:01
CustomFanController
Fan Control
1
2
off
3


5:01
CustomFanController
Fan Control
1
2
off
3

DialView.kt

```kotlin
package com.android.example.customfancontroller

import ...


private enum class FanSpeed(val label: Int) {
    OFF("off"),
    LOW("1"),
    MEDIUM("2"),
    HIGH("3");

    fun next() = when (this) {
        OFF -> LOW
        LOW -> MEDIUM
        MEDIUM -> HIGH
        HIGH -> OFF
    }

}

private const val RADIUS_OFFSET_LABEL = 30
private const val RADIUS_OFFSET_INDICATOR = -35


class DialView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleAttr: Int = 0
) : View(context, attrs, defStyleAttr){

    private var fanSpeedLowColor = 0
    private var fanSpeedMediumColor = 0
    private var fanSpeedMaxColor = 0

    init {
        isClickable = true

        context.withStyledAttributes(attrs, R.styleable.DialView) {  this: TypedArray
            fanSpeedLowColor = getColor(R.styleable.DialView_fanColor1,  defValue: 0)
            fanSpeedMediumColor = getColor(R.styleable.DialView_fanColor2,  defValue: 0)
            fanSpeedMaxColor = getColor(R.styleable.DialView_fanColor3,  defValue: 0)
        }


    }

    override fun performClick(): Boolean {
        if (super.performClick()) return true

        fanSpeed = fanSpeed.next()
        contentDescription = resources.getString(fanSpeed.label)

        invalidate()
        return true
    }
```

```
65  private val paint = Paint(Paint.ANTI_ALIAS_FLAG).apply {  this: Paint
66      style = Paint.Style.FILL
67      textAlign = Paint.Align.CENTER
68      textSize = 55.0f
69      typeface = Typeface.create( familyName: "", Typeface.BOLD)
70  }
71
72  private var radius = 0.0f                  // Radius of the circle.
73  private var fanSpeed = FanSpeed.OFF        // The active selection.
74
75  // position variable which will be used to draw label and indicator circle position
76  private val pointPosition: PointF = PointF( x: 0.0f,  y: 0.0f)
77
78  override fun onSizeChanged(width: Int, height: Int, oldWidth: Int, oldHeight: Int) {
79      radius = (min(width, height) / 2.0 * 0.8).toFloat()
80  }
81
82  override fun onDraw(canvas: Canvas) {
83
84      // Set dial background color to green if selection not off.
85      paint.color = if (fanSpeed == FanSpeed.OFF) Color.GRAY else Color.GREEN
86
87
88      paint.color = when (fanSpeed) {
89          FanSpeed.OFF -> Color.GRAY
90          FanSpeed.LOW -> fanSpeedLowColor
91          FanSpeed.MEDIUM -> fanSpeedMediumColor
92          FanSpeed.HIGH -> fanSpeedMaxColor
93      } as Int
94
```

```
97      // Draw the dial.
98      canvas.drawCircle((width / 2).toFloat(), ( height / 2).toFloat(), radius, paint)
99      // Draw the indicator circle.
100     val markerRadius = radius + RADIUS_OFFSET_INDICATOR
101     pointPosition.computeXYForSpeed(fanSpeed, markerRadius)
102     paint.color = Color.BLACK
103     canvas.drawCircle(pointPosition.x, pointPosition.y,  radius: radius / 12, paint)
104     // Draw the text labels.
105     val labelRadius = radius + RADIUS_OFFSET_LABEL
106     for (i in FanSpeed.values()) {
107         pointPosition.computeXYForSpeed(i, labelRadius)
108         val label = resources.getString(i.label)
109         canvas.drawText(label, pointPosition.x, pointPosition.y, paint)
110     }
111 }
112
113
114 private fun PointF.computeXYForSpeed(pos: FanSpeed, radius: Float) {
115     // Angles are in radians.
116     val startAngle = Math.PI * (9 / 8.0)
117     val angle = startAngle + pos.ordinal * (Math.PI / 4)
118     x = (radius * cos(angle)).toFloat() + width / 2
119     y = (radius * sin(angle)).toFloat() + height / 2
120 }
121
122 }
123
```

dimens.xml

```xml
<resources>
    <dimen name="text_view_padding">16dp</dimen>
    <dimen name="default_margin">8dp</dimen>
    <dimen name="margin_top">24dp</dimen>
    <dimen name="fan_dimen">250dp</dimen>
</resources>
```

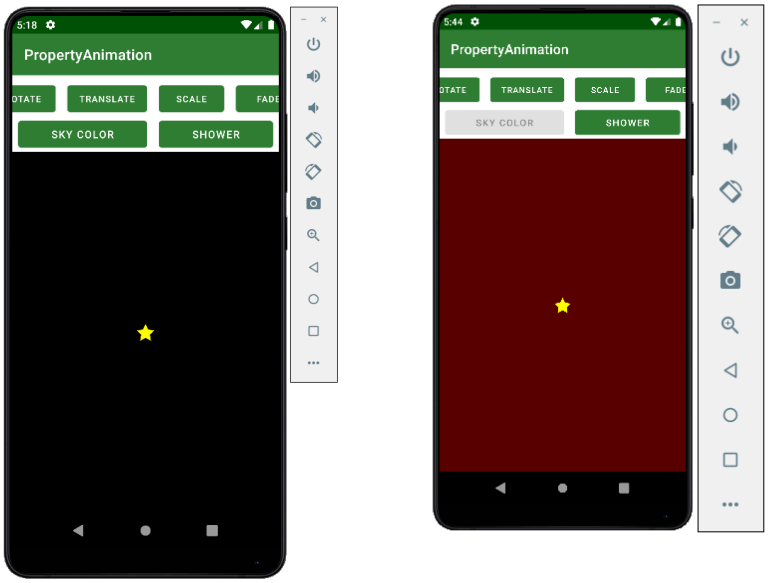
colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

attrs.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <declare-styleable name="DialView">
        <attr name="fanColor1" format="color" />
        <attr name="fanColor2" format="color" />
        <attr name="fanColor3" format="color" />
    </declare-styleable>

</resources>
```

Task 3: Advanced Android in Kotlin 03.1:Property Animation



MainActivity.kt

```kotlin
package com.google.samples.propertyanimation

import android.animation.*
import android.graphics.Color
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.view.ViewGroup
import android.view.animation.AccelerateInterpolator
import android.view.animation.LinearInterpolator
import android.widget.Button
import android.widget.FrameLayout
import android.widget.ImageView
import androidx.appcompat.widget.AppCompatImageView


class MainActivity : AppCompatActivity() {

    lateinit var star: ImageView
    lateinit var rotateButton: Button
    lateinit var translateButton: Button
    lateinit var scaleButton: Button
    lateinit var fadeButton: Button
    lateinit var colorizeButton: Button
    lateinit var showerButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
```

```
        star = findViewById(R.id.star)
        rotateButton = findViewById<Button>(R.id.rotateButton)
        translateButton = findViewById<Button>(R.id.translateButton)
        scaleButton = findViewById<Button>(R.id.scaleButton)
        fadeButton = findViewById<Button>(R.id.fadeButton)
        colorizeButton = findViewById<Button>(R.id.colorizeButton)
        showerButton = findViewById<Button>(R.id.showerButton)

        rotateButton.setOnClickListener {  it: View!
            rotater()
        }

        translateButton.setOnClickListener {  it: View!
            translater()
        }

        scaleButton.setOnClickListener {  it: View!
            scaler()
        }

        fadeButton.setOnClickListener {  it: View!
            fader()
        }

        colorizeButton.setOnClickListener {  it: View!
            colorizer()
        }

        showerButton.setOnClickListener {  it: View!
            shower()
        }
```

```
    private fun rotater() {
        val animator = ObjectAnimator.ofFloat(star, View.ROTATION,  ...values: -360f, 0f)
        animator.duration = 1000
        animator.disableViewDuringAnimation(rotateButton)
        animator.start()
    }

    private fun translater() {

        // Translate the view 200 pixels to the right and back

        val animator = ObjectAnimator.ofFloat(star, View.TRANSLATION_X,  ...values: 200f)
        animator.repeatCount = 1
        animator.repeatMode = ObjectAnimator.REVERSE
        animator.disableViewDuringAnimation(translateButton)
        animator.start()
    }

    private fun scaler() {

        // Scale the view up to 4x its default size and back

        val scaleX = PropertyValuesHolder.ofFloat(View.SCALE_X,  ...values: 4f)
        val scaleY = PropertyValuesHolder.ofFloat(View.SCALE_Y,  ...values: 4f)
        val animator = ObjectAnimator.ofPropertyValuesHolder(star, scaleX, scaleY)
        animator.repeatCount = 1
        animator.repeatMode = ObjectAnimator.REVERSE
        animator.disableViewDuringAnimation(scaleButton)
        animator.start()
    }
```

```
96      private fun fader() {
97          val animator = ObjectAnimator.ofFloat(star, View.ALPHA,  ...values: 0f)
98          animator.repeatCount = 1
99          animator.repeatMode = ObjectAnimator.REVERSE
100         animator.disableViewDuringAnimation(fadeButton)
101         animator.start()
102     }
103     @SuppressLint( ...value: "ObjectAnimatorBinding")
104     private fun colorizer() {
105
106         var animator = ObjectAnimator.ofArgb(star.parent,
107                 propertyName: "backgroundColor", Color.BLACK, Color.RED)
108         animator.setDuration(500)
109         animator.repeatCount = 1
110         animator.repeatMode = ObjectAnimator.REVERSE
111         animator.disableViewDuringAnimation(colorizeButton)
112         animator.start()
113     }
114
115     private fun shower() {
116
117         val container = star.parent as ViewGroup
118         val containerW = container.width
119         val containerH = container.height
120         var starW: Float = star.width.toFloat()
121         var starH: Float = star.height.toFloat()
122
```

```
123         // Create the new star (an ImageView holding our drawable) and add it to the container
124         val newStar = AppCompatImageView( context: this)
125         newStar.setImageResource(R.drawable.ic_star)
126         newStar.layoutParams = FrameLayout.LayoutParams(FrameLayout.LayoutParams.WRAP_CONTENT,
127             FrameLayout.LayoutParams.WRAP_CONTENT)
128         container.addView(newStar)
129
130         // Scale the view randomly between 10-160% of its default size
131         newStar.scaleX = Math.random().toFloat() * 1.5f + .1f
132         newStar.scaleY = newStar.scaleX
133         starW *= newStar.scaleX
134         starH *= newStar.scaleY
135
136         // Position the view at a random place between the left and right edges of the container
137         newStar.translationX = Math.random().toFloat() * containerW - starW / 2
138
139         val mover = ObjectAnimator.ofFloat(newStar, View.TRANSLATION_Y, -starH, containerH + starH)
140         mover.interpolator = AccelerateInterpolator( factor: 1f)
141
142         // Create an animator to rotateButton the view around its center up to three times
143         val rotator = ObjectAnimator.ofFloat(newStar, View.ROTATION,
144             (Math.random() * 1080).toFloat())
145         rotator.interpolator = LinearInterpolator()
146
147         val set = AnimatorSet()
148         set.playTogether(mover, rotator)
149         set.duration = (Math.random() * 1500 + 500).toLong()
```

```
150
151         // When the animation is done, remove the created view from the container
152         set.addListener(object : AnimatorListenerAdapter() {
153             override fun onAnimationEnd(animation: Animator?) {
154                 container.removeView(newStar)
155             }
156         })
157
158         set.start()
159     }
160
161     private fun ObjectAnimator.disableViewDuringAnimation(view: View) {
162
163
164         addListener(object : AnimatorListenerAdapter() {
165             override fun onAnimationStart(animation: Animator?) {
166                 view.isEnabled = false
167             }
168
169             override fun onAnimationEnd(animation: Animator?) {
170                 view.isEnabled = true
171             }
172         })
173     }
174
175 }
```