CS4131
Mobile Application Development

Name: ________________________________ (   ) Date: _______________

Chapter 8: PiP, Transitions and Animations

8.1 Introduction to Chapter 8

This chapter will introduce features and implementations that will enhance the aesthetics of your app, which may make your app more appealing to the user. As this chapter features primarily "moving" components in animations and transitions, an optional chapter on Kotlin coroutines, which is Kotlin's answer to unappealing multi-layered nested blocks in multithreading codes to make for a more responsive app in the user-end.

8.2 Implementing Video Playback and Picture in Picture (PiP) mode

The VideoView is a visual component which, when added to the layout of an activity, provides a surface onto which a video may be played. Android currently supports the following video formats extensions: .3gp, .mp4, .mkv, .webm. You can read more about the supported video formats here: https://developer.android.com/guide/topics/media/media-formats

To support controls for the VideoView, the MediaController will be needed which is a View containing controls for a MediaPlayer, typicaly containing buttons like "Play/Pause", "Rewind", "Fast Forward" and a progress slider. It takes care of synchronizing the controls with the state of the MediaPlayer. The way to use this class is to instantiate it programmatically. The MediaController will create a default set of controls and put them in a window floating above your application.

Starting in Android 8.0 (API level 26), Android allows activities to launch in picture-in-picture (PiP) mode. PiP is a special type of multi-window mode mostly used for video playback. It lets the user watch a video in a small window pinned to a corner of the screen while navigating between apps or browsing content on the main screen.

While PiP mode may seem like a useful and attractive feature to use in apps which support video, there are some good practices pertaining to PiP mode:

- PiP might be disabled on devices that have low RAM, hence, you may want to do a check by calling: hasSystemFeature(PackageManager.FEATURE_PICTURE_IN_PICTURE)
- PiP is intended for activities that play full-screen video. When switching your activity into PiP mode, avoid showing anything except video content, when your activity enters PiP mode and hide UI elements
- When an activity is in PiP mode, by default it doesn't get input focus. To receive input events while in PiP mode, use MediaSession.setCallback()
- When your app is in PiP mode, video playback in the PiP window can cause audio interference with another app, such as a music player app or voice search app. To avoid this, request audio focus when you start playing the video, and handle audio focus change notifications

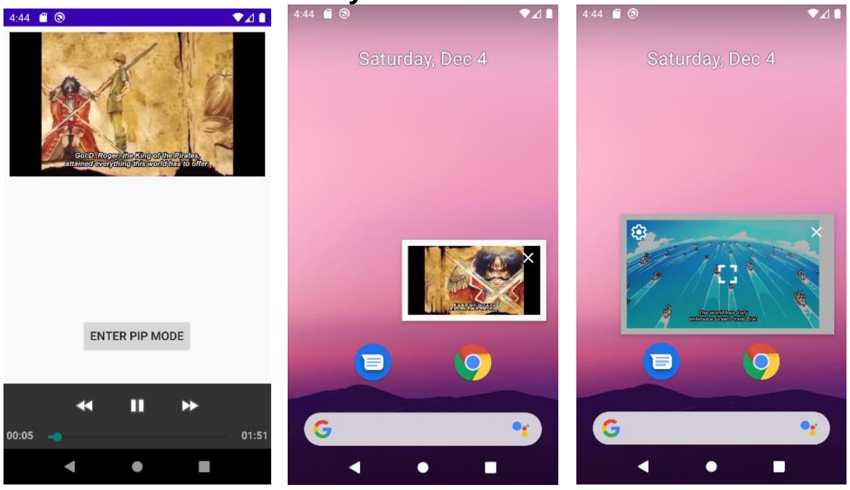
- When your app is about to enter PiP, note only the top activity will enter it. In some situations, such as on multi-window devices, it is possible the activity below will now be shown and become visible again alongside the PiP activity. You should handle this case accordingly.

You may read more about PiP mode here: https://developer.android.com/guide/topics/ui/picture-in-picture

Example 1: VideoPlaybackPiP
This app allows the user to play a video which is stored internally in the app and allows the user to go into PiP mode to display the video as a reduced screen on the side. When the user selects the video on PiP mode, the user is able to either close the video, set some settings, or transition the video back to in-app.

Note that the video continues playing during the transition from in-app to PiP mode and back to in-app.

Screenshots of VideoPlaybackPiP

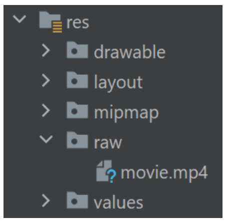


AndroidManifest.xml

```
...
        <activity
            android:name=".MainActivity"
            android:supportsPictureInPicture="true"
            android:configChanges="screenSize|smallestScreenSize|screenLayout|orientation"
            android:exported="true">
            <intent-filter>
...
```

strings.xml

```
<resources>
    <string name="app_name">Chp 8: VideoPlaybackPiP</string>
    <string name="pipBtn">Enter Pip Mode</string>
</resources>
```



MainActivity.kt

```
import android.app.PictureInPictureParams
import android.content.ContentValues.TAG
import android.content.res.Configuration
import android.graphics.Rect
import android.net.Uri
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.util.Rational
import android.view.View
import android.widget.Button
import android.widget.MediaController
import android.widget.VideoView

class MainActivity : AppCompatActivity() {
    private lateinit var videoView: VideoView
    private lateinit var mediaController: MediaController

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

        videoView = findViewById(R.id.videoView)
        val videoRes = R.raw.movie
        videoView.setVideoURI(Uri.parse("android.resource://$packageName/$videoRes"))
        videoView.setOnPreparedListener{ mediaPlayer ->
            Log.i(TAG, "Duration=${videoView.duration}")
        }

        mediaController = MediaController(this)
        mediaController.setAnchorView(videoView)
        videoView.setMediaController(mediaController)

        // Listener is called immediately after the user exits PiP but before animating.
        videoView.addOnLayoutChangeListener { _, left, top, right, bottom,
                                            oldLeft, oldTop, oldRight, oldBottom ->
            if (left != oldLeft || right != oldRight || top != oldTop || bottom != oldBottom){
                // The playerView's bounds changed, update to reflect its new bounds
                val sourceRectHint = Rect()
                videoView.getGlobalVisibleRect(sourceRectHint)
                setPictureInPictureParams(
                  PictureInPictureParams.Builder().setSourceRectHint(sourceRectHint).build()
                )
            }
        }
    }
    fun enterPipMode(view : View){
        val pipBtn = findViewById<Button>(R.id.pipBtn)
        val rational = Rational(videoView.width, videoView.height)
        val params = PictureInPictureParams.Builder()
            .setAspectRatio(rational).setSourceRectHint(Rect()).build()
        pipBtn.visibility = View.INVISIBLE
        videoView.setMediaController(null)
        enterPictureInPictureMode(params)
    }
    override fun onPictureInPictureModeChanged(isInPictureInPictureMode: Boolean,
        newConfig: Configuration?) {
        super.onPictureInPictureModeChanged(isInPictureInPictureMode, newConfig)
        val pipBtn = findViewById<Button>(R.id.pipBtn)
        if (!isInPictureInPictureMode){
            pipBtn.visibility = View.VISIBLE
            videoView.setMediaController(mediaController)
        }
    }
}
```

Explanation:

Line 23 to 25 Line 30 to 32	The code will set up the VideoView and MediaController objects, by linking the VideoView to the internal video resource, then linking the MediaController and VideoView objects together. Note that the setAnchorView() method is to anchor the controls onto the video controls.
Line 47 to 64	The functions support the entering of PiP mode. The button is attached to the enterPipMode function, and when called, it will set the dimensions of the PiP window to be of the same aspect ratio as the original video, in order not to unnecessarily stretch it horizontally or vertically. There is no MediaController set onto the video on PiP mode in this code. The overridden onPictureInPictureModeChanged handles hiding of UI elements when the app goes into PiP mode and conversely, the unhiding of UI elements when it goes out of PiP mode.
Line 35 to 45	The VideoView listener is setup to listen to want happens when its size changes (when it changes between normal to PiP and vice versa). This listener will perform an update so that the app can keep track of what are the current dimensions throughout the runtime of the app.

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <VideoView
        android:id="@+id/videoView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />
    <Button
        android:id="@+id/pipBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="128dp"
        android:onClick="enterPipMode"
        android:text="@string/pipBtn"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/videoView"
        app:layout_constraintVertical_bias="1.0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

8.3 Android Transitions with Code

Transitions allow the changes made to the layout and appearance of the views in a UI to be animated during application runtime. There are several different ways to implement Transitions from within application code. The first way is applying transitions to just the Views themselves which will be covered in Example 2 and 3, where example 2, just the set of transitions is applied onto the View while in example 3, the start and end layouts are provided to allow the transition framework to figure out how to animate from the start to end layout.

Before moving to the examples, explanation is needed to how transitions work programmatically. Android's transition framework allows you to animate all kinds of motion in your UI. You can select what type of animation you want (such to fade the views in/out or change the view sizes) and the transition framework figures out how to animate from the start to the end layout.

To put many different transitions together, you will require a TransitionSet. A TransitionSet is a parent of child transitions (including other TransitionSets). Using TransitionSets enables more complex choreography of transitions, where some sets play ORDERING_TOGETHER and others play ORDERING_SEQUENTIAL. You can add transitions into a transition set by using the addTransition() method.

The other concept which needs to be talked about is the use of **interpolators** in the transitions. An interpolator is an animation modifier that affects the **rate of change in an animation**. This allows your existing animation effects to be accelerated, decelerated, repeated, bounced, etc.

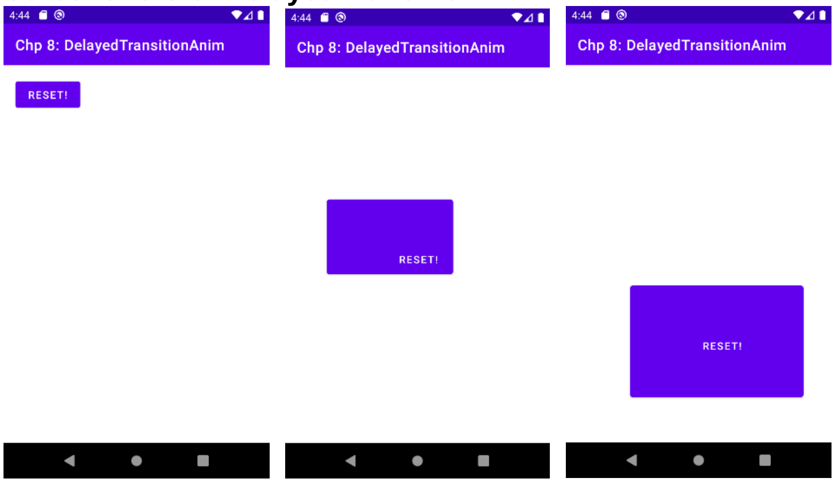
Interpolators	**Description**
AccelerateDecelerateInterpolator	The rate of change starts and ends slowly but accelerates through the middle.
AccelerateInterpolator	The rate of change starts out slowly then accelerates.
AnticipateInterpolator	The change starts backward then flings forward.
AnticipateOvershootInterpolator	The change starts backward then flings forward and overshoots the target value and finally goes back to the final value.
BaseInterpolator	An abstract class which is extended by default interpolators.
BounceInterpolator	The change bounces at the end.
CycleInterpolator	Repeats the animation for a specified number of cycles.
DecelerateInterpolator	The rate of change starts out quickly and and then decelerates.
LinearInterpolator	The rate of change is constant
OvershootInterpolator	The change flings forward and overshoots the last value then comes back.
PathInterpolator	Traverse a Path that extends from Point (0, 0) to (1, 1).

This article provides a very good visualization on the rate of change of animations of the interpolators applied on animating graphs:
https://medium.com/mobile-app-development-publication/illustrating-android-interpolator-31ea09051d78

Example 2: DelayedTransitionAnim

Upon touching anywhere on the screen (except the button), the button will fade then animate its way down to the bottom right of the screen. While moving to the bottom right of the screen, its size will increase and the button will perform a bounce animation before stopping. The button resets its position upon clicking the "RESET!" button.

Screenshots of DelayedTransitionAnim


Chp 8: DelayedTransitionAnim
RESET!
Chp 8: DelayedTransitionAnim
RESET!
Chp 8: DelayedTransitionAnim
RESET!

strings.xml

```
<resources>
    <string name="app_name">Chp 8: DelayedTransitionAnim</string>
    <string name="clickMe">Reset!</string>
</resources>
```

MainActivity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.transition.ChangeBounds
import android.transition.Fade
import android.transition.TransitionManager
import android.transition.TransitionSet
import android.view.MotionEvent
import android.view.View
import android.view.ViewGroup.LayoutParams.WRAP_CONTENT
import android.view.animation.AccelerateInterpolator
import android.view.animation.BounceInterpolator
import android.widget.Button
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.constraintlayout.widget.ConstraintSet

class MainActivity : AppCompatActivity() {
    private lateinit var myLayout : ConstraintLayout
    private lateinit var myTransition : TransitionSet

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        myLayout = findViewById(R.id.myLayout)

        val fade = Fade()
        val changeBounds = ChangeBounds()
        changeBounds.duration = 5000 // 2 seconds
        changeBounds.interpolator = BounceInterpolator()
        myTransition = TransitionSet()
        myTransition.addTransition(fade)
31            myTransition.addTransition(changeBounds)
32        }
33        override fun onTouchEvent(event: MotionEvent?): Boolean {
34            val button = findViewById<Button>(R.id.button)
35            button.minWidth = 500
36            button.minHeight = 350
37            TransitionManager.beginDelayedTransition(myLayout, myTransition)
38            val set = ConstraintSet()
39            set.connect(R.id.button, ConstraintSet.BOTTOM, ConstraintSet.PARENT_ID,
40                ConstraintSet.BOTTOM, 0)
41            set.connect(R.id.button, ConstraintSet.RIGHT, ConstraintSet.PARENT_ID,
42                ConstraintSet.RIGHT, 0)
43            set.constrainWidth(R.id.button, ConstraintSet.WRAP_CONTENT)
44            set.applyTo(myLayout)
45            return true
46        }
47        fun btnClick(view : View){
48            val button = findViewById<Button>(R.id.button)
49            button.width = WRAP_CONTENT
50            button.height = WRAP_CONTENT
51            val set = ConstraintSet()
52            set.connect(R.id.button, ConstraintSet.TOP, ConstraintSet.PARENT_ID,
53                ConstraintSet.TOP, 0)
54            set.connect(R.id.button, ConstraintSet.LEFT, ConstraintSet.PARENT_ID,
55                ConstraintSet.LEFT, 0)
56            set.constrainWidth(R.id.button, ConstraintSet.WRAP_CONTENT)
57            set.applyTo(myLayout)
58        }
59    }
```

Explanation:

Line 25 to 31	This is where the code for the animation is defined such that the transition set will include a fade before a bounce. The ChangeBounds() transition is used for the Button to move to a different part of the screen. Note that when adding the transitions into the TransitionSet, **the order matters**.
Line 33 to 46	This code is such that whenever the user touches anywhere on the screen except the Button, it will beginDelayedTransition() to apply the transition framework onto the Button with myTransition (the transition set defined). The latter few lines of code is to apply the constraints of the ConstraintLayout onto the Button.
Line 47 to 58	When the Button is clicked on, its width and height are reset and it is moved back to its original position in the ConstraintLayour.

activity_main.xml

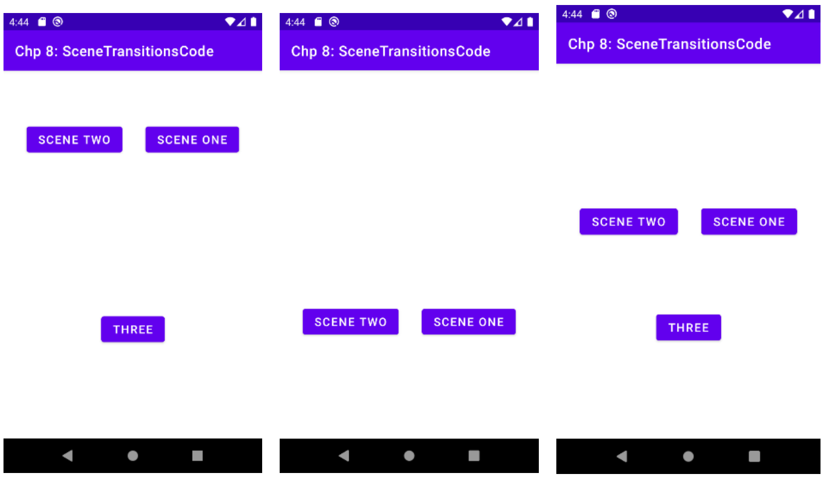
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/myLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="btnClick"
        android:text="@string/clickMe"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

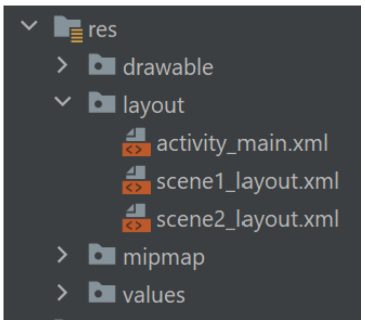
The following example will feature the use of start and end scene XML files to define the start and end points, and allow the transition framework to follow through the coded transitions to go from the starting layout to the ending layout.

<u>Example 3.1: SceneTransitionsCode</u>
When the "Scene Two" button is clicked on, the animation will start such that the button "Three" will fade away and the "Scene One" and "Scene Two" buttons will both drop and bounce at where the faded button "Three" is. When "Scene One" button is clicked after, "Three" will fade in and the "Scene One" and "Scene Two" buttons will both bounce up to its original position.

Note that in the project layout, it is such that the first screen is exactly the layout of scene1_layout.xml and the second screen is exactly the layout of scene2_layout.xml. Hence the transition framework animates its way to go from the first screen to the second screen and vice-versa.

Screenshots of SceneTransitionsCode


4:44
Chp 8: SceneTransitionsCode
SCENE TWO
SCENE ONE
THREE
4:44
Chp 8: SceneTransitionsCode
SCENE TWO
SCENE ONE
4:44
Chp 8: SceneTransitionsCode
SCENE TWO
SCENE ONE
THREE



strings.xml

```
<resources>
    <string name="app_name">Chp 8: SceneTransitionsCode</string>
    <string name="oneBtn">Scene Two</string>
    <string name="twoBtn">Scene One</string>
    <string name="threeBtn">Three</string>
</resources>
```

MainActivity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.transition.*
import android.view.View
import android.view.ViewGroup
import android.view.animation.*
```

```
7
8    class MainActivity : AppCompatActivity() {
9        private lateinit var rootContainer : ViewGroup
10       private lateinit var scene1: Scene
11       private lateinit var scene2: Scene
12       private lateinit var myTransition : TransitionSet
13
14       override fun onCreate(savedInstanceState: Bundle?) {
15           super.onCreate(savedInstanceState)
16           setContentView(R.layout.activity_main)
17
18           myTransition = TransitionSet()
19           myTransition.ordering = TransitionSet.ORDERING_SEQUENTIAL
20           val fade = Fade()
21           fade.addTarget(R.id.threeBtn)
22           myTransition.addTransition(fade)
23           val changeBounds = ChangeBounds()
24           changeBounds.duration = 2000 // 2 seconds
25           changeBounds.interpolator = BounceInterpolator()
26   //        changeBounds.interpolator = AccelerateInterpolator(1.5f)
27   //        changeBounds.interpolator = AccelerateDecelerateInterpolator()
28   //        changeBounds.interpolator = AnticipateInterpolator(1.5f)
29   //        changeBounds.interpolator = AnticipateOvershootInterpolator()
30   //        changeBounds.interpolator = CycleInterpolator(1.5f)
31   //        changeBounds.interpolator = DecelerateInterpolator(1.5f)
32   //        changeBounds.interpolator = LinearInterpolator()
33   //        changeBounds.interpolator = OvershootInterpolator(1.5f)
34           myTransition.addTransition(changeBounds)
35
36           rootContainer = findViewById<ViewGroup>(R.id.rootContainer)
37           scene1 = Scene.getSceneForLayout(rootContainer, R.layout.scene1_layout, this)
38           scene2 = Scene.getSceneForLayout(rootContainer, R.layout.scene2_layout, this)
39           scene1.enter()
40       }
41       fun goToScene2(view : View){TransitionManager.go(scene2, myTransition) }
42       fun goToScene1(view : View){TransitionManager.go(scene1, myTransition) }
43   }
```

Explanation:

Line 36 to 42	This code works in a similar fashion to what you saw in Example 2, in terms of how to define the transition set. The difference here is that the scenes are obtained from the respective layouts and the activity is set to display scene1 first. The respective functions are them called at appropriate times (they are linked to the respective buttons) to transition to the different scenes. **Note that you should observe the XML code for scene1_layout and scene2_layout and notice that the ids for the respective three buttons are the same for both scenes.** Note that the code also features commented out calls to other interpolators which you may use.

scene1_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/twoBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginTop="72dp"
        android:onClick="goToScene2"
        android:text="@string/oneBtn"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/oneBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="80dp"
        android:layout_marginTop="72dp"
        android:layout_marginEnd="32dp"
        android:onClick="goToScene1"
        android:text="@string/twoBtn"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toEndOf="@+id/twoBtn"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/threeBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="128dp"
        android:text="@string/threeBtn"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

scene2_layout.xml

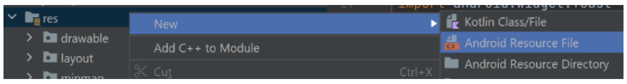
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/twoBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="32dp"
        android:layout_marginBottom="128dp"
        android:onClick="goToScene2"
        android:text="@string/oneBtn"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <Button
        android:id="@+id/oneBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="80dp"
        android:layout_marginEnd="32dp"
        android:layout_marginBottom="128dp"
        android:onClick="goToScene1"
        android:text="@string/twoBtn"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toEndOf="@+id/twoBtn"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

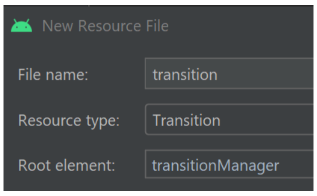
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/rootContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" />
```

8.4 Android Transitions with XML

XML files can also be used in place of coding the transitions out which may lead to your project structure being more organized and clearly defined (transitions are transitions, and not mixed together with the controller code). For transitions to be added in the form of XML files, you will need to right-click the res folder and create a new Android Resource File. From there, ensure that the file name is transition and the resource type is “Transition” and leave all other fields as their defaults. You will end up with a transition folder created with a transition.xml file.




New Resource File

File name: transition

Resource type: Transition

Root element: transitionManager

Example 3.2: SceneTransitionsXML

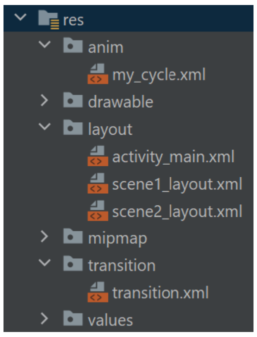
This app works in the exact same way as Example 3.1, except that this time, XML files are used to define the transitions.

MainActivity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.transition.Scene
import android.transition.Transition
import android.transition.TransitionInflater
import android.transition.TransitionManager
import android.view.View
import android.view.ViewGroup

class MainActivity : AppCompatActivity() {
    private lateinit var rootContainer : ViewGroup
    private lateinit var scene1: Scene
    private lateinit var scene2: Scene
    private lateinit var myTransition : Transition

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

        myTransition = TransitionInflater.from(this)
            .inflateTransition(R.transition.transition)

        rootContainer = findViewById(R.id.rootContainer)
        scene1 = Scene.getSceneForLayout(rootContainer, R.layout.scene1_layout, this)
        scene2 = Scene.getSceneForLayout(rootContainer, R.layout.scene2_layout, this)
        scene1.enter()
    }

    fun goToScene2(view : View){TransitionManager.go(scene2, myTransition) }
    fun goToScene1(view : View){TransitionManager.go(scene1, myTransition) }
}
```

transition.xml

```
<?xml version="1.0" encoding="utf-8"?>
<transitionSet xmlns:android="http://schemas.android.com/apk/res/android"
    android:transitionOrdering="sequential">
    <fade
        android:fadingMode="fade_in">
        <targets>
            <target android:targetId="@+id/threeBtn" />
        </targets>
    </fade>
    <changeBounds android:duration="2000"
        android:interpolator="@android:interpolator/bounce"/>
    <!--
    <changeBounds android:duration="2000"
        android:interpolator="@anim/my_cycle"/> -->
</transitionSet>
```



my_cycle.xml

```
<?xml version="1.0" encoding="utf-8"?>
<cycleInterpolator xmlns:android="http://schemas.android.com/apk/res/android"
    android:factor="1.2" />
```

Explanation:
If XML files are utlised the define the transition, the MainActivity code will only require you to inflate the transition.xml file, where inside you transition.xml file, you have your various transitions defined. Similar to Example 3.1, first a fade is defined, for threeBtn then after that, a BounceInterpolator is used to animate from scene1 to scene2 within 2 seconds. You may have noticed a commented out my_cycle animation. This my_cycle animation is within the anim folder of the resources and is only used if the interpolator has additional attributes (in this case, the cycle factor of 1.2) you need or want to define which cannot be readily defined in the transitions.xml file.

From here, there is much more you can do in terms of making transitions work for your app to make elements of your app visually stunning. You may want to read through the following to add transitions in a way which makes your app visually dynamic:

- https://developer.android.com/training/transitions/start-activity
- https://developer.android.com/training/transitions

8.5 Animating Objects

The property animation system is a robust framework that allows you to animate almost anything. You can define an animation to change any object property over time, regardless of whether it draws to the screen or not. A property animation changes a property's (a field in an object) value over a specified length of time. To animate something, you specify the object property that you want to animate, such as an object's position on the screen, how long you want to animate it for, and what values you want to animate between.

The view animation system provides the capability to only animate View objects, so if you wanted to animate non-View objects, you must implement your own code to do so. The view animation system is also constrained in the fact that it only exposes a few aspects of a View object to animate, such as the scaling and rotation of a View but not the background color, for instance.

Another disadvantage of the view animation system is that it only modified where the View was drawn, and not the actual View itself. For instance, if you animated a button to move across the screen, the button draws correctly, but the actual location where you can click the button does not change, so you must implement your own logic to handle this.

With the property animation system, these constraints are completely removed, and you can animate any property of any object (Views and non-Views) and the object itself is modified. The property animation system is also more robust in the way it carries out animation. At a high level, you assign animators to the properties that you want to animate, such as color, position, or size and can define aspects of the animation such as interpolation and synchronization of multiple animators.

Property animation utilizes the following classes to implement the animations on objects:

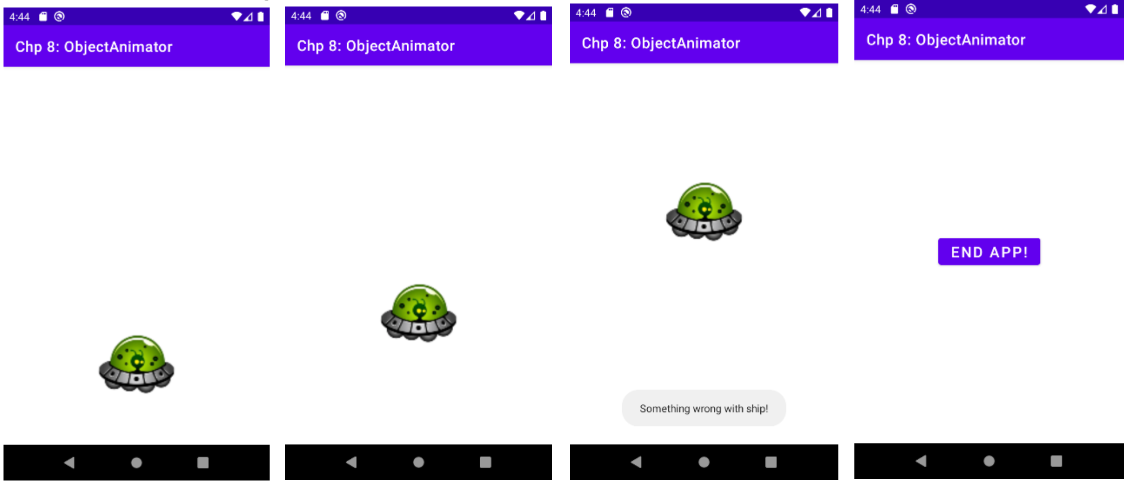
Animators	**Description**
ValueAnimator	The main timing engine for property animation that also computes the values for the property to be animated. There are **two pieces** to animating properties: calculating the

	animated values and setting those values on the object and property that is being animated. ValueAnimator does not carry out the second piece, so you must listen for updates to values calculated by the ValueAnimator and modify the objects that you want to animate with your own logic.
ObjectAnimator	Allows you to set a target object and object property to animate. This class updates the property accordingly when it computes a new value for the animation. ObjectAnimator makes the process of animating values on target objects easier, but ValueAnimator is much less restrictive
AnimatorSet	Provides a mechanism to group animations together so that they run in relation to one another. You can set animations to play together, sequentially, or after a specified delay.

<u>Example 4: ObjectAnimator</u>

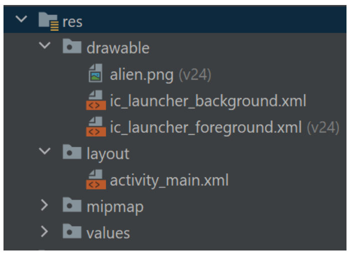
Upon a tap on anywhere on the screen, the alien ship, which is an ImageView, will start to animate its way up the screen such that it appears as it it is trying to take off upwards, but is having some issues doing so. After the alien has completely flew off the screen, a button will appear which will exit the app upon clicking on it.

Screenshots of ObjectAnimator



strings.xml

```
<resources>
    <string name="app_name">Chp 8: ObjectAnimator</string>
    <string name="takeOff">Alien took off!</string>
    <string name="repeat">Something wrong with ship!</string>
    <string name="dest">Alien is gone!</string>
    <string name="end">End App!</string>
</resources>
```



```
res
  drawable
    alien.png (v24)
    ic_launcher_background.xml
    ic_launcher_foreground.xml (v24)
  layout
    activity_main.xml
  mipmap
  values
```

MainActivity.kt

```
import android.animation.Animator
import android.animation.AnimatorSet
import android.animation.ObjectAnimator
import android.animation.ValueAnimator
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.view.animation.AccelerateInterpolator
import android.widget.Button
import android.widget.ImageView
import android.widget.Toast
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.constraintlayout.widget.ConstraintSet

class MainActivity : AppCompatActivity() {
    private val DEFAULT_ANIMATION_DURATION = 5000L
    private lateinit var alien : ImageView
    private lateinit var button : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        alien = findViewById(R.id.alien)
        button = findViewById(R.id.button)
        button.visibility = View.INVISIBLE
    }
    override fun onTouchEvent(event: MotionEvent?): Boolean {
        onStartAnimation()
        return true
    }
    fun onClickBtn(view : View){
        finish()
    }
    fun onStartAnimation(){
        val height : Float = (applicationContext.resources.displayMetrics.heightPixels)
                              .toFloat()
        val positionAnimator = ValueAnimator.ofFloat(0f, -height)
        positionAnimator.addUpdateListener {
            alien.translationY = it.animatedValue as Float
        }
        positionAnimator.interpolator= AccelerateInterpolator(1.5f)
        positionAnimator.duration = DEFAULT_ANIMATION_DURATION

        positionAnimator.addListener(object : Animator.AnimatorListener {
            override fun onAnimationStart(p0: Animator?) {
                val str = getString(R.string.takeOff)
                Toast.makeText(applicationContext, str, Toast.LENGTH_SHORT).show()
            }
            override fun onAnimationEnd(p0: Animator?) {
                val str = getString(R.string.dest)
                Toast.makeText(applicationContext, str, Toast.LENGTH_SHORT).show()
                button.visibility = View.VISIBLE
            }
            override fun onAnimationCancel(p0: Animator?) {
                TODO("Not yet implemented")
            }
            override fun onAnimationRepeat(p0: Animator?) {
                val str = getString(R.string.repeat)
                Toast.makeText(applicationContext, str, Toast.LENGTH_SHORT).show()
            }
        })
        positionAnimator.repeatMode = ValueAnimator.REVERSE
        positionAnimator.repeatCount = 2
        val rotationAnimator = ObjectAnimator.ofFloat(alien, "rotation", 0f, 360f)
        rotationAnimator.duration = 2000L
        val animatorSet = AnimatorSet()
        animatorSet.play(positionAnimator).after(rotationAnimator)
        animatorSet.start()
    }
}
```

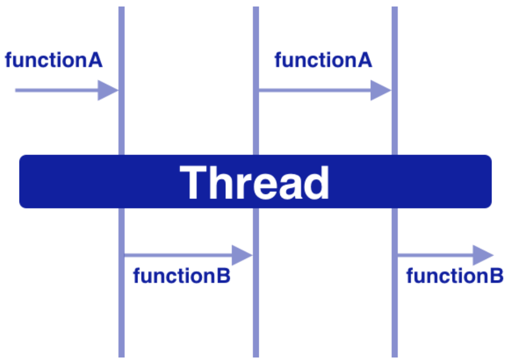
Explanation:

Line 17 Line 36 to 43	A ValueAnimator is set such that a translations about the y-axis of the phone screen will be implemented relative to the phone's screen size. Note that the y-axis is defined such that the negative y-axis is at the top of the screen. An AccelerateInterpolator is used so that the ship appears to be accelerating upward and the animation is given 5 seconds to fly up.
Line 45 to 62	The positionAnimator has an implemented listener which will listen for different parts of the animation such that appropriate Toast texts will be output like commentary on the ship's flight. The animation is made to repeat itself as well in the next few lines which is why the onAnimationRepeat method is also overridden.
Line 63 to 64	The positionAnimator is set to repeat itself two times such that it will reverse its steps in the positionAnimator which will make the ship appear that it is going up and down the screen, having problems with the flight.
Line 65 to 68	An ObjectAnimator is set to make the ship rotate a full circle in a duration of 2 seconds. Note that an AnimatorSet is also now defined where the positionAnimator is played AFTER the rotationAnimator (hence, rotate on the spot first, then translate), and do keep in mind, **the order matters**. Ensure to remember to start the mission.

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/myLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/alien"
        android:layout_width="200dp"
        android:layout_height="200dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0"
        app:srcCompat="@drawable/alien" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickBtn"
        android:text="@string/end"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

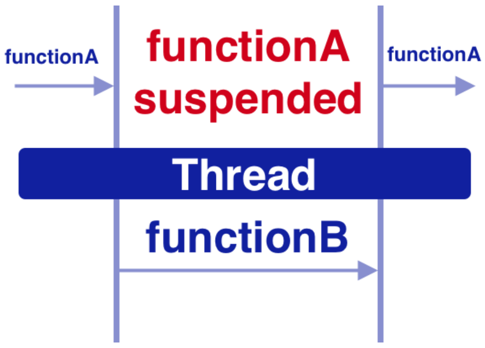
8.6 [Optional] Kotlin Coroutines for Multithreading



Kotlin coroutines are the answer to the usual multithreading framework which leads to callback hells and blocking states because there is no other simple way to guarantee thread-safe execution. The word coroutine is essentially the short form of cooperating routines, which means that we get getting functions to cooperate with each other and run tasks **asynchronously**.

On Android, coroutines help to manage long-running tasks that might otherwise block the main thread and cause your app to become unresponsive. Over 50% of professional developers who use coroutines have reported seeing increased productivity.

Android used to have the AsyncTask class which was intended to enable proper and easy use of the UI thread but it has since been deprecated due to inconsistent behavior on different versions of the platform and does not provide much utility. Coroutines have been introduced for asynchronous programming on Android, and some benefits include,



- **Lightweight:** It is possible to run many coroutines on a single thread due to support for **suspension** with the suspend keyword. Suspending saves memory over blocking while supporting many concurrent operations.
- **Fewer memory leaks:** Coroutines follow the principle of structured concurrency which means new coroutines can only be lauched within a specific CoroutineScope, which defines the coroutine's lifetime. This is done through the runBlocking keyword
- **Built-in cancellation support:** Cancellation is propagated automatically through the running coroutine hierarchy. Such support is given by the cancel() and join() methods for a running job.
- **Android Jetpack integration:** Many Jetpack libraries include extensions that provide full coroutines support.

To implement Kotlin coroutines, the following implementations need to be added into the dependencies block of build.gradle (app). Ensure to click on "Sync Now" to add the implementations.

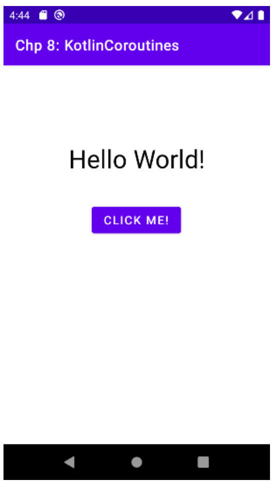
```
// Kotlin Coroutines
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.7'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.7'
```

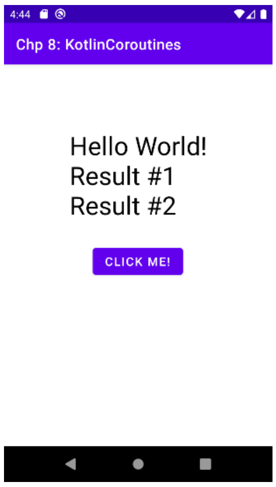
The following examples will showcase how coroutines will work in the context of Android development.

Example 5: KotlinCoroutines

Upon the click of the button, the texts Result#1 and Result#2, which are supposedly running on different functions, will appear to be output simultaneously in the TextViews in the app, due to apparent concurrency which is in actual fact, due to function suspension in the thread.

Screenshots of KotlinCoroutines





strings.xml

```xml
<resources>
    <string name="app_name">Chp 8: KotlinCoroutines</string>
    <string name="helloWorld">Hello World!</string>
    <string name="clickMe">Click Me!</string>
    <string name="result1">Result #1</string>
    <string name="result2">Result #2</string>
</resources>
```

MainActivity.kt

```kotlin
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import kotlinx.coroutines.*
import kotlinx.coroutines.Dispatchers.IO
import kotlinx.coroutines.Dispatchers.Main

class MainActivity : AppCompatActivity() {
    private lateinit var RESULT1 : String
    private lateinit var RESULT2 : String
    private lateinit var textView : TextView
    private lateinit var button : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        RESULT1 = getString(R.string.result1)
        RESULT2 = getString(R.string.result2)
        textView = findViewById(R.id.textView)
        button = findViewById(R.id.button)
        button.setOnClickListener{
            CoroutineScope(IO).launch{
                fakeAPIRequest()
            }
        }
    }
    private fun setText(input : String){
        val newText = textView.text.toString() + "\n$input"
        textView.text = newText
    }
    private suspend fun setTextOnMainThread(input : String){
        withContext(Main){
            setText(input)
        }
36      }
37      private suspend fun fakeAPIRequest(){
38          val result1 = getResult1FromAPI()
39          println("debug: $result1")
40          setTextOnMainThread(result1)
41          val result2 = getResult2FromAPI()
42          setTextOnMainThread(result2)
43      }
44      private suspend fun getResult1FromAPI(): String {
45          logThread("getResult1FromApi")
46          delay(100)
47          return RESULT1
48      }
49      private suspend fun getResult2FromAPI(): String {
50          logThread("getResult2FromApi")
51          delay(50)
52          return RESULT2
53      }
54      private fun logThread(methodName: String) {
55          println("debug: ${methodName}: ${Thread.currentThread().name}")
56      }
57  }
```

Explanation:

Line 5 to 7	Note the imports which are required for the use of the coroutines.
Line 22 to 26	A listener is set onto the button such that when it is clicked on, an IO coroutine scope is implemented which will launch the fakeAPIRequest()
Line 37 to 52	This fakeAPIRequest will run in concurrency with both getResult1FromAPI() and getResult2FromAPI() such that the running of the functions on the same thread will be managed such that they appear to be running at the same time, hence, improving the responsiveness of the app.

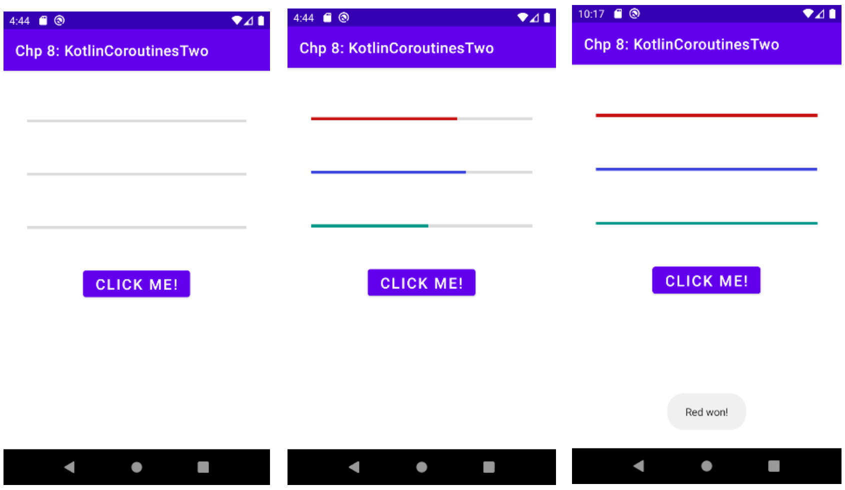
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textColor="@color/black"
        android:textSize="34sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.221" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/clickMe"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.115" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

<u>Example 6: KotlinCoroutinesTwo</u>

This app features a more visual representation of how concurrency will work with the help of Kotlin coroutines, through using progress bars of different colours. The bars will appear to look like they are running at the same time but in actual fact, it is a series of suspended functions which are managed on the thread to make them appear run simultaneously.

Screenshots of KotlinCoroutinesTwo


4:44
Chp 8: KotlinCoroutinesTwo
CLICK ME!
4:44
Chp 8: KotlinCoroutinesTwo
CLICK ME!
10:17
Chp 8: KotlinCoroutinesTwo
CLICK ME!
Red won!

strings.xml

```
<resources>
    <string name="app_name">Chp 8: KotlinCoroutinesTwo</string>
    <string name="clickMe">Click Me!</string>
</resources>
```

MainActivity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.ProgressBar
import android.widget.Toast
import kotlinx.coroutines.*
import kotlinx.coroutines.Dispatchers.Main
import java.util.*

class MainActivity : AppCompatActivity() {
    private var raceEnd = false
    private lateinit var progressBarRed : ProgressBar
    private lateinit var progressBarGreen : ProgressBar
    private lateinit var progressBarBlue : ProgressBar
    private var greenJob: Job? = null
    private var redJob: Job? = null
    private var blueJob: Job? = null

    override fun onCreate(savedInstanceState: Bundle?) {
```

```
20          super.onCreate(savedInstanceState)
21          setContentView(R.layout.activity_main)
22          val button = findViewById<Button>(R.id.button)
23          progressBarGreen = findViewById(R.id.progressBarGreen)
24          progressBarBlue = findViewById(R.id.progressBarBlue)
25          progressBarRed = findViewById(R.id.progressBarRed)
26          button.setOnClickListener{
27              startUpdate()
28          }
29      }
30      override fun onDestroy() {
31          super.onDestroy()
32          resetRun()
33      }
34      private fun startUpdate() {
35          resetRun()
36          greenJob = CoroutineScope(Main).launch{ startRunning(progressBarGreen) }
37          redJob = CoroutineScope(Main).launch { startRunning(progressBarRed) }
38          blueJob =CoroutineScope(Main).launch { startRunning(progressBarBlue) }
39      }
40      private suspend fun startRunning(progressBar: ProgressBar) {
41          progressBar.progress = 0
42          while (progressBar.progress < 1000 && !raceEnd) {
43              delay(10)
44              progressBar.progress += (1..10).random()
45          }
46          if (!raceEnd) {
47              raceEnd = true
48              Toast.makeText(this, "${progressBar.tooltipText} won!",
49                         Toast.LENGTH_SHORT).show()
50          }
51      }
52      private fun ClosedRange<Int>.random() = Random().nextInt(endInclusive - start) +  start
53      private fun resetRun() {
54          raceEnd = false
55          greenJob?.cancel()
56          blueJob?.cancel()
57          redJob?.cancel()
58      }
59  }
```

Explanation:

Line 36 to 38	For each progress bar, the coroutine scope is defined to be in the same scope (the main scope) and will start running the individual startRunning() functions on each progress bar concurrently.
Line 40 to 51	The bars will run such that every 0.01 second, the bars will increase progress by a randomized amount. The moment a progress bar "wins", a Toast is output and the race stops.
Line 30 to 33 Line 53 to 58	Upon reset of the bars or end of the app, ensure that no more jobs are running and clear the thread.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="32dp"
    tools:context=".MainActivity">

    <ProgressBar
        android:id="@+id/progressBarRed"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="40dp"
        android:layout_marginTop="16dp"
        android:max="1000"
        android:progressTint="#C81111"
        android:tooltipText="Red"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <ProgressBar
        android:id="@+id/progressBarBlue"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="40dp"
        android:layout_marginTop="32dp"
        android:max="1000"
        android:progressTint="#3B43DD"
        android:tooltipText="Blue"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/progressBarRed"
        app:layout_constraintVertical_bias="0.0" />

    <ProgressBar
        android:id="@+id/progressBarGreen"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="40dp"
        android:layout_marginTop="32dp"
        android:max="1000"
        android:progressTint="#009688"
        android:tooltipText="Green"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/progressBarBlue"
        app:layout_constraintVertical_bias="0.0" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="@string/clickMe"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/progressBarGreen"
        app:layout_constraintVertical_bias="0.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

[Reference]

[1] Android VideoView class: https://developer.android.com/reference/android/widget/VideoView
[2] Android MediaController class: https://developer.android.com/reference/android/widget/MediaController
[3] Android Transitions https://developer.android.com/training/transitions
[4] Interpolators: https://developer.android.com/reference/android/view/animation/Interpolator
[5] Android Animation Resource: https://developer.android.com/guide/topics/resources/animation-resource
[6] Android Property Animation: https://developer.android.com/guide/topics/graphics/prop-animation
[7] Android Animation Tutorial: https://www.raywenderlich.com/2785491-android-animation-tutorial-with-kotlin
[8] Kotlin Coroutines on Android: https://developer.android.com/kotlin/coroutines
[9] Kotlin Coroutines Documentation: https://kotlinlang.org/docs/coroutines-overview.html
[10] Mastering Kotlin Coroutines in Android: https://blog.mindorks.com/mastering-kotlin-coroutines-in-android-step-by-step-guide