UNIT-4

Working with Graphics and Animation

4.1 Working with graphics

The Android framework offers a variety of graphics rendering APIs for 2D and 3D that interact with manufacturer implementations of graphics drivers, so it is important to have a good understanding of how those APIs work at a higher level.

- Android provides a huge set of 2D-drawing APIs that allow you to create graphics.
- Android has visually appealing graphics and mind-blowing animations.
- The Android framework provides a rich set of powerful APIS for applying animation to UI elements and graphics as well as drawing custom 2D and 3D graphics.

Canvas: Android graphics provides low-level graphics tools such as canvases, color, filters, points, and rectangles which handle drawing to the screen directly. The Android framework provides a set of 2D-DRAWING APIs which allows user to provide their custom graphics onto a canvas or to modify existing views to customize their look and feel.

There are two ways to draw 2D graphics,

1. Draw your animation into a View object from your layout.
2. Draw your animation directly on a Canvas.

Some of the important methods of Canvas Class are as follows

1. drawText()
2. drawRoundRect()
3. drawCircle()
4. drawRect()
5. drawBitmap()
6. drawARGB()

You can use these methods in the onDraw() method to create your custom user interface.

Drawing an animation with a View is the best option for drawing simple graphics that do not need to change dynamically and are not part of a performance-intensive game. It is used when the user wants to display a static graphic or predefined animation.

Drawing an animation with Canvas is a better option when your application needs to re-draw itself regularly. For example, video games should be drawn to the Canvas on their own.

Example:

Step1: Create a New Project in Android Studio

Step2: Working with the activity_main.xml file

- Navigate to the app > res > layout > activity_main.xml and add the below code to that file. Below is the code for the activity_main.xml file. Add an image as shown below.

```
<?xmlthe  version="1.0" encoding="utf-8"?>
<RelativeLayout
   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">

   <ImageView
      android:id="@+id/image_view_1"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:ignore="ContentDescription"
      android:background="@color/black"/>

</RelativeLayout>
```

Step3: Working with the MainActivity.kt file

```
package org.main_act.drawlines


import android.annotation.SuppressLint
import android.graphics.Bitmap
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.widget.ImageView
import androidx.annotation.RequiresApi

class MainActivity : AppCompatActivity(), View.OnTouchListener {

    // Declaring ImageView, Bitmap, Canvas, Paint,
    // Down Coordinates and Up Coordinates
    private lateinit var mImageView: ImageView
    private lateinit var bitmap: Bitmap
    private lateinit var canvas: Canvas
    private lateinit var paint: Paint
    private var downX = 0f
    private var downY = 0f
    private var upX = 0f
    private var upY = 0f

    @RequiresApi(Build.VERSION_CODES.R)
    @SuppressLint("ClickableViewAccessibility")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initializing the ImageView
        mImageView = findViewById(R.id.image_view_1)

        // Getting the current window dimensions
        val currentDisplay = windowManager.currentWindowMetrics
        val dw = currentDisplay.bounds.width()
        val dh = currentDisplay.bounds.height()

        // Creating a bitmap with fetched dimensions
        bitmap = Bitmap.createBitmap(dw, dh, Bitmap.Config.ARGB_8888)

        // Storing the canvas on the bitmap
        canvas = Canvas(bitmap)

        // Initializing Paint to determine
        // stoke attributes like color and size
        paint = Paint()
        paint.color = Color.RED
        paint.strokeWidth = 10F

        // Setting the bitmap on ImageView
        mImageView.setImageBitmap(bitmap)

        // Setting onTouchListener on the ImageView
        mImageView.setOnTouchListener(this)
    }

    // When Touch is detected on the ImageView,
    // Initial and final coordinates are recorded
    // and a line is drawn between them.
    // ImagView is updated
    @SuppressLint("ClickableViewAccessibility")
    override fun onTouch(v: View?, event: MotionEvent?): Boolean {
        when (event!!.action) {
            MotionEvent.ACTION_DOWN -> {
                downX = event.x
                downY = event.y
            }

            MotionEvent.ACTION_UP -> {
                upX = event.x
```

```
                upY = event.y
                canvas.drawLine(downX, downY, upX, upY, paint)
                mImageView.invalidate()
            }
        }
        return true
    }
}
```

4.2 Animation

Animation is the process of adding a motion effect to any view, image, or text. With the help of an animation, you can add motion or change the shape of a specific view. Animation in Android is generally used to give your UI a rich look and feel.

Following are the three animation systems used in Android applications:

- Property Animation
- View Animation
- Drawable Animation

Property Animation: Property animation is the preferred method of animation in Android. This animation is a robust framework that lets you animate any properties of any objects, view or non-view objects. The android.animation provides classes that handle property animation.

Property Animation is one of the robust frameworks that allows animating for almost everything. This is one of the powerful and flexible animations which was introduced in Android 3.0. Property animation can be used to add any animation in the CheckBox, RadioButtons, and widgets other than any view.

View Animation: View Animation is also called Tween Animation. The android.view.animation provides classes that handle view animation.. This animation can be used to animate the content of a view. .It is limited to simple transformations such as moving, re-sizing and rotation, but not its background color.

View Animation can be used to add animation to a specific view to perform tweened animation on views. Tweened animation calculates animation information such as size, rotation, start point, and endpoint. These animations are slower and less flexible. An example of View animation can be used if we want to expand a specific layout in that place we can use View Animation. An example of View Animation can be seen in Expandable RecyclerView.

Drawable Animation: Drawable animation is implemented using the AnimationDrawable class.. This animation works by displaying a running sequence of 'Drawable' resources that is images, frame by frame inside a view object.

Drawable Animation is used if you want to animate one image over another. The simple way to understand is to animate drawables is to load the series of drawables one after another to create an animation. A simple example of drawable animation can be seen in many apps Splash screen on apps logo animation.

Example:

Step1: Create New Project

1. Open Android Studio
2. Go to File => New => New Project.
3. Then, select Empty Activity and click on next
   - Write the application name as DynamicEditTextKotlin
   - Select the minimum SDK you need, here we have selected 21 as the minimum SDK
   - Choose the language as Kotlin and click on the finish button.

Step2: Modify activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@+id/linearLayout"
        android:gravity="center"
        android:text="It’s an animation time"
        android:textSize="32sp"
        android:textColor="@color/colorPrimaryDark"
        android:textStyle="bold" />
    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:orientation="vertical">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:weightSum="2">
            <Button
                android:id="@+id/fade_in"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Fade In"
                android:textAllCaps="false" />
            <Button
                android:id="@+id/fade_out"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Fade Out"
                android:textAllCaps="false" />
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:weightSum="2">
            <Button
                android:id="@+id/zoom_in"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Zoom In"
                android:textAllCaps="false" />
            <Button
                android:id="@+id/zoom_out"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Zoom Out"
                android:textAllCaps="false" />
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:weightSum="2">
            <Button
                android:id="@+id/slide_down"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Slide Down"
                android:textAllCaps="false" />
            <Button
                android:id="@+id/slide_up"
```

```
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Slide Up"
            android:textAllCaps="false" />
        </LinearLayout>
        <LinearLayout
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:orientation="horizontal"
          android:weightSum="2">
          <Button
            android:id="@+id/bounce"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Bounce"
            android:textAllCaps="false" />
          <Button
            android:id="@+id/rotate"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Rotate"
            android:textAllCaps="false" />
        </LinearLayout>
    </LinearLayout>
</RelativeLayout>
```

bounce.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:fillAfter="true">
    <translate
        android:fromYDelta="100%"
        android:toYDelta="-20%"
        android:duration="300" />
    <translate
        android:startOffset="500"
        android:fromYDelta="-20%"
        android:toYDelta="10%"
        android:duration="150" />
    <translate
        android:startOffset="1000"
        android:fromYDelta="10%"
        android:toYDelta="0"
        android:duration="100" />
</set>
```

rotate.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/linear_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="0"
    android:toDegrees="360" />
```

Step3: Create MainActivity.kt file

```
package net.animate.animationsinkotlin


import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.os.Handler
import android.view.View
import android.view.animation.AnimationUtils
import kotlinx.android.synthetic.main.activity_main.*

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

        bounce.setOnClickListener {
            val animationBounce = AnimationUtils.loadAnimation(this, R.anim.bounce)
            textView.startAnimation(animationBounce)
        }
        rotate.setOnClickListener {
            val animationRotate = AnimationUtils.loadAnimation(this, R.anim.rotate)
            textView.startAnimation(animationRotate)
        }
    }
}
```

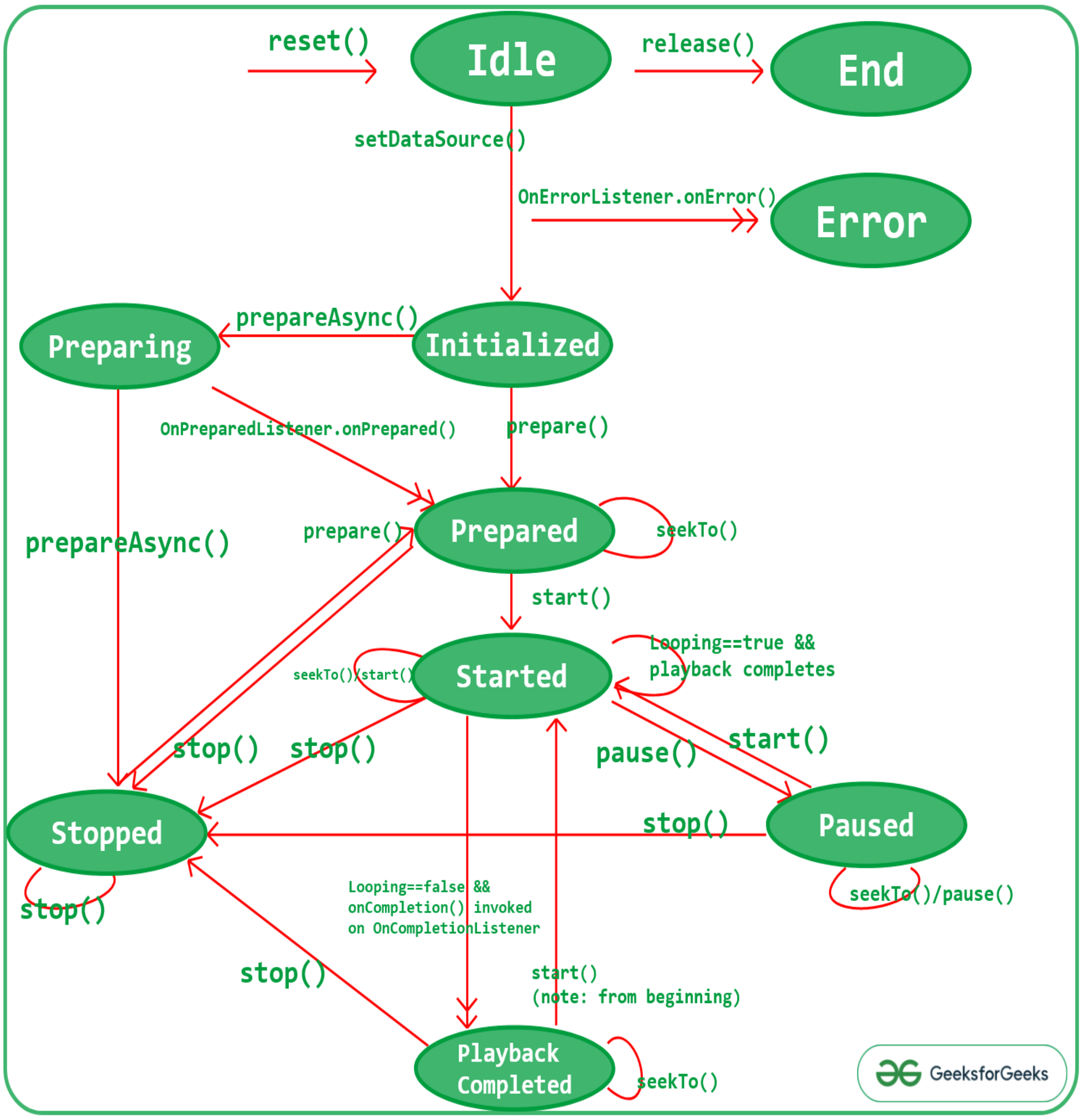
4.3 Media Player

MediaPlayer Class in Android is used to play media files. Those are Audio and Video files. It can also be used to play audio or video streams over the network. So in this article, the things discussed are:

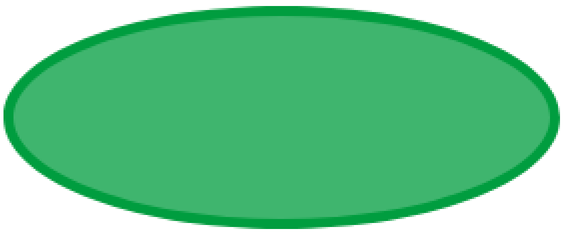
- MediaPlayer State diagram
- Creating a simple audio player using MediaPlayer API. Have a look at the following image. Note that we are going to implement this project using the Kotlin language.

State Diagram of the MediaPlayer Class

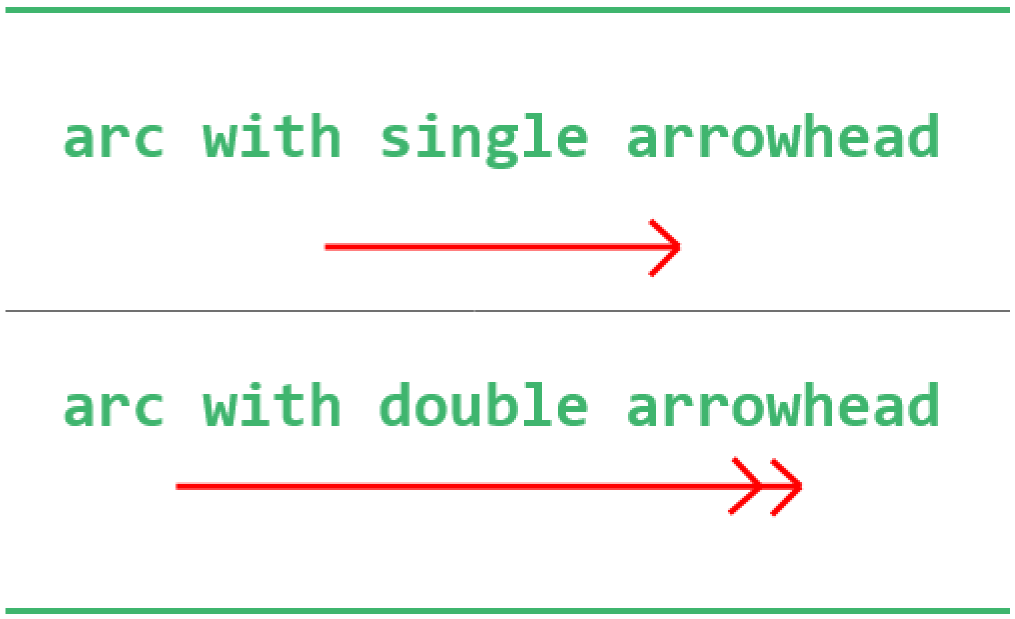
- The playing of the audio or video file using MediaPlayer is done using a state machine.
- The following image is the MediaPlayer state diagram.



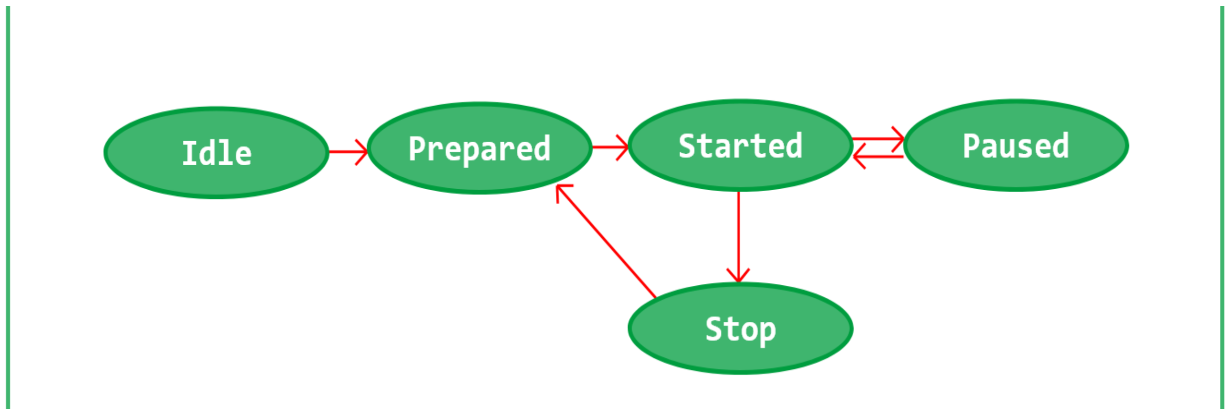
- In the above MediaPlayer state diagram, the oval shape represents the state of the MediaPlayer instance resides in.



- There are two types of arcs showing in the state diagram. One with the single arrowhead represents the synchronous method calls of the MediaPlayer instance and one with the double arrowhead represents the asynchronous calls.



- The release method which is one of the important element in the MediaPlayer API. This helps in releasing the Memory resources allocated for the Mediaplayer instance when it is not needed anymore. Refer to How to Clear or Release Audio Resources in Android? to know how the memory allocated by the Mediaplayer can be released. So that the memory management is done accordingly.
- If the stop() method is called using the Mediaplayer instance, then it needs to be prepared for the next playback.
- The MediaPlayer can be moved to the specific time position using the **seekTo()** method so that the MediaPlayer instance can continue playing the Audio or Video playback from that specified position.
- The focus of the audio playback should be managed accordingly using the AudioManager service which is discussed in the article How to Manage Audio Focus in Android?.
- The following image is the summarised version of the MediaPlayer state diagram.



Steps to create a simple MediaPlayer in Android

Step1: Create an empty activity project

- Create an empty activity Android Studio project. And select Kotlin as a programming language.

Step2: Create a raw resource folder

- Create a raw resource folder under the res folder and copy one of the .mp3 file extensions.

Step3: Working with the activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    tools:ignore="HardcodedText">

    <TextView
        android:id="@+id/headingText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:text="MEDIA PLAYER"
        android:textSize="18sp"
        android:textStyle="bold" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/headingText"
        android:layout_marginTop="16dp"
        android:gravity="center_horizontal">
        <Button
            android:id="@+id/stopButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:backgroundTint="@color/colorPrimary"
            android:text="STOP"
            android:textColor="@android:color/white" />

        <Button
            android:id="@+id/playButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:backgroundTint="@color/colorPrimary"
            android:text="PLAY"
            android:textColor="@android:color/white" />

        <Button
            android:id="@+id/pauseButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:backgroundTint="@color/colorPrimary"
            android:text="PAUSE"
            android:textColor="@android:color/white" />

    </LinearLayout>

</RelativeLayout>
```

Step4: Working with the MainActivity.kt file

- The MediaPlayer instance needs the attributes to be set before playing any audio or video file.
- Invoke the following inside the MainActivity.kt file. Comments are added for better understanding.

```
import android.media.MediaPlayer

import androidx.appcompat.app.AppCompatActivity

import android.os.Bundle

import android.widget.Button


class MainActivity : AppCompatActivity() {


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


        // create an instance of mediplayer for audio playback
        val mediaPlayer: MediaPlayer = MediaPlayer.create(applicationContext, R.raw.music)


        // register all the buttons using their appropriate IDs
        val bPlay: Button = findViewById(R.id.playButton)
        val bPause: Button = findViewById(R.id.pauseButton)
        val bStop: Button = findViewById(R.id.stopButton)


        //Handle the start button to
          //Start the audio playback
        bPlay.setOnClickListener {
            // start method is used to start
              // playing the audio file
            mediaPlayer.start()
        }


        //Handle the pause button to put the
        // MediaPlayer instance at the Pause state
        bPause.setOnClickListener {
            // pause() method can be used to
            //Pause the media player instance
            mediaPlayer.pause()
        }

        //Handle the stop button to stop playing
        // and prepare the MediaPlayer instance
        //For the next instance of play
        bStop.setOnClickListener {
            // stop() method is used to completely
            //Stop playing the media player instance
            MediaPlayer.stop()

            //After stopping the MediaPlayer instance
            //It again needs to be prepared
            //For the next instance of playback
            MediaPlayer.prepare()
        }
    }
}
```