```
package com.internshala.echo.fragments
import android.app.Activity
import android.content.Context
import android.media.AudioManager
import android.media.MediaPlayer
import android.net.Uri
import android.os.Bundle
import android.os.Handler
import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ImageButton
import android.widget.SeekBar
import android.widget.TextView
import com.cleveroad.audiovisualization.AudioVisualization
import com.cleveroad.audiovisualization.DbmHandler
import com.cleveroad.audiovisualization.GLAudioVisualizationView
import com.internshala.echo.CurrentSongHelper
import com.internshala.echo.R
import com.internshala.echo.Songs
import java.util.*
import java.util.concurrent.TimeUnit
/**
 * A simple [Fragment] subclass.
 */
class SongPlayingFragment : Fragment() {
    var myActivity: Activity? = null
    var mediaPlayer: MediaPlayer? = null
    var startTimeText: TextView? = null
    var endTimeText: TextView? = null
    var playPauseImageButton: ImageButton? = null
    var previousImageButton: ImageButton? = null
    var nextImageButton: ImageButton? = null
    var loopImageButton: ImageButton? = null
    var shuffleImageButton: ImageButton? = null
    var seekBar: SeekBar? = null
    var songArtistView: TextView? = null
    var songTitleView: TextView? = null
    var currentPosition: Int = 0
    var fetchSongs: ArrayList<Songs>? = null
    var currentSongHelper: CurrentSongHelper? = null
    var audioVisualization: AudioVisualization? = null
    var glView: GLAudioVisualizationView? = null

    /*Declaring the preferences for the shuffle and loop feature
    * the object is created as we will need them outside the scope of this class*/
    object Staticated {
        var MY_PREFS_SHUFFLE = "Shuffle feature"
        var MY_PREFS_LOOP = "Loop feature"
    }

    var updateSongTime = object : Runnable {
        override fun run() {
            val getCurrent = mediaPlayer?.currentPosition
            startTimeText?.setText(String.format("%d:%d",
                    TimeUnit.MILLISECONDS.toMinutes(getCurrent?.toLong() as Long),

TimeUnit.MILLISECONDS.toSeconds(TimeUnit.MILLISECONDS.toMinutes(getCurrent?.toLong() as
Long))))
            seekBar?.setProgress(getCurrent?.toInt() as Int)
            Handler().postDelayed(this, 1000)
        }
    }
    override fun onCreateView(inflater: LayoutInflater?, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        val view = inflater!!.inflate(R.layout.fragment_song_playing, container, false)
        seekBar = view?.findViewById(R.id.seekBar)
        startTimeText = view?.findViewById(R.id.startTime)
        endTimeText = view?.findViewById(R.id.endTime)
        playPauseImageButton = view?.findViewById(R.id.playPauseButton)
        nextImageButton = view?.findViewById(R.id.nextButton)
        previousImageButton = view?.findViewById(R.id.previousButton)
        loopImageButton = view?.findViewById(R.id.loopButton)
        shuffleImageButton = view?.findViewById(R.id.shuffleButton)
        songArtistView = view?.findViewById(R.id.songArtist)

        glView = view?.findViewById(R.id.visualizer_view)
        return view
    }
    override fun onViewCreated(view: View?, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        audioVisualization = glView as AudioVisualization
    }
    override fun onAttach(context: Context?) {
        super.onAttach(context)
        myActivity = context as Activity
    }
    override fun onAttach(activity: Activity?) {
        super.onAttach(activity)
        myActivity = activity
    }
    override fun onResume() {
        super.onResume()
        audioVisualization?.onResume()
    }
    override fun onPause() {
        audioVisualization?.onPause()
        super.onPause()
    }
    override fun onDestroyView() {
        audioVisualization?.release()
        super.onDestroyView()
    }
    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        currentSongHelper = CurrentSongHelper()
        currentSongHelper?.isPlaying = true
        currentSongHelper?.isLoop = false
        currentSongHelper?.isShuffle = false
        var path: String? = null
        var _songTitle: String? = null
        var _songArtist: String? = null
        var songId: Long = 0
        try {
            path = arguments.getString("path")
            _songTitle = arguments.getString("songTitle")
            _songArtist = arguments.getString("songArtist")
            songId = arguments.getInt("songId").toLong()
            currentPosition = arguments.getInt("position")
            fetchSongs = arguments.getParcelableArrayList("songData")
            currentSongHelper?.songPath = path
            currentSongHelper?.songTitle = _songTitle
            currentSongHelper?.songArtist = _songArtist
            currentSongHelper?.songId = songId
            currentSongHelper?.currentPosition = currentPosition
            updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
        } catch (e: Exception) {
            e.printStackTrace()
        }
        mediaPlayer = MediaPlayer()
        mediaPlayer?.setAudioStreamType(AudioManager.STREAM_MUSIC)
        try {
            mediaPlayer?.setDataSource(myActivity, Uri.parse(path))
            mediaPlayer?.prepare()
        } catch (e: Exception) {
            e.printStackTrace()
        }
        mediaPlayer?.start()
        processInformation(mediaPlayer as MediaPlayer)
        if (currentSongHelper?.isPlaying as Boolean) {
            playPauseImageButton?.setBackgroundResource(R.drawable.pause_icon)
        } else {
            playPauseImageButton?.setBackgroundResource(R.drawable.play_icon)
        }
        mediaPlayer?.setOnCompletionListener {
            onSongComplete()
        }
        clickHandler()

        var visualizationHandler = DbmHandler.Factory.newVisualizerHandler(myActivity as
Context, 0)
        audioVisualization?.linkTo(visualizationHandler)

        /*Now we want that when if user has turned shuffle or loop ON, then these settings
should persist even if the app is restarted after closing
        * This is done with the help of Shared Preferences
        * Shared preferences are capable of storing small amount of data in the form of
key-value pair*/

        /*Here we initialize the preferences for shuffle in a private mode
        * Private mode is chosen so that so other app us able to read the preferences apart
from our app*/
        var prefsForShuffle = myActivity?.getSharedPreferences(Staticated.MY_PREFS_SHUFFLE,
Context.MODE_PRIVATE)

        /*Here we extract the value of preferences and check if shuffle was ON or not*/
        var isShuffleAllowed = prefsForShuffle?.getBoolean("feaure", false)
        if (isShuffleAllowed as Boolean) {


            /*if shuffle was found activated, then we change the icon color and tun loop
OFF*/
            currentSongHelper?.isShuffle = true
            currentSongHelper?.isLoop = false
            shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_icon)
            loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
        } else {
            /*Else default is set*/
            currentSongHelper?.isShuffle = false
            shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_white_icon)
        }

        /*Similar to the shuffle we check the value for loop activation*/
        var prefsForLoop = myActivity?.getSharedPreferences(Staticated.MY_PREFS_LOOP,
Context.MODE_PRIVATE)

        /*Here we extract the value of preferences and check if loop was ON or not*/
        var isLoopAllowed = prefsForLoop?.getBoolean("feature", false)
        if (isLoopAllowed as Boolean) {

            /*If loop was activated we change the icon color and shuffle is turned OFF */
            currentSongHelper?.isShuffle = false
            currentSongHelper?.isLoop = true
            shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_white_icon)
            loopImageButton?.setBackgroundResource(R.drawable.loop_icon)
        } else {

            /*Else defaults are used*/
            loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
            currentSongHelper?.isLoop = false
        }
    }

    fun clickHandler() {
        shuffleImageButton?.setOnClickListener({

            /*Initializing the shared preferences in private mode
            * edit() used so that we can overwrite the preferences*/
            var editorShuffle =
myActivity?.getSharedPreferences(Staticated.MY_PREFS_SHUFFLE, Context.MODE_PRIVATE)?.edit()
            var editorLoop = myActivity?.getSharedPreferences(Staticated.MY_PREFS_LOOP,
Context.MODE_PRIVATE)?.edit()

            if (currentSongHelper?.isShuffle as Boolean) {
                shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_white_icon)
                currentSongHelper?.isShuffle = false

                /*If shuffle was activated previously, then we deactivate it*/
                /*The putBoolean() method is used for saving the boolean value against the
key which is feature here*/

                /*Now the preferences agains the block Shuffle feature will have a key:
feature and its value: false*/
                editorShuffle?.putBoolean("feature", false)
                editorShuffle?.apply()
            } else {

                currentSongHelper?.isShuffle = true
                currentSongHelper?.isLoop = false
                shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_icon)
                loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)

                /*Else shuffle is activated and if loop was activated then loop is
deactivated*/
                editorShuffle?.putBoolean("feature", true)
                editorShuffle?.apply()
                /*Similar to shuffle, the loop feature has a key:feature and its
value:false*/
                editorLoop?.putBoolean("feature", false)
                editorLoop?.apply()
            }
        })

        nextImageButton?.setOnClickListener({
            currentSongHelper?.isPlaying = true
            if (currentSongHelper?.isShuffle as Boolean) {
                playNext("PlayNextLikeNormalShuffle")
            } else {
                playNext("PlayNextNormal")
            }
        })
        previousImageButton?.setOnClickListener({
            currentSongHelper?.isPlaying = true
            if (currentSongHelper?.isLoop as Boolean) {
                loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
            }
            playPrevious()
        })
        loopImageButton?.setOnClickListener({
            /*The operation on preferences is completely analogous to shuffle, no addition
is there*/
            var editorShuffle =
myActivity?.getSharedPreferences(Staticated.MY_PREFS_SHUFFLE, Context.MODE_PRIVATE)?.edit()
            var editorLoop = myActivity?.getSharedPreferences(Staticated.MY_PREFS_LOOP,
Context.MODE_PRIVATE)?.edit()
            if (currentSongHelper?.isLoop as Boolean) {
                currentSongHelper?.isLoop = false
                loopImageButton?.setBackgroundResource(R.drawable.loop_white_icon)
                editorLoop?.putBoolean("feature", false)
                editorLoop?.apply()
            } else {
                currentSongHelper?.isLoop = true
                currentSongHelper?.isShuffle = false
                loopImageButton?.setBackgroundResource(R.drawable.loop_icon)
                shuffleImageButton?.setBackgroundResource(R.drawable.shuffle_white_icon)
                editorShuffle?.putBoolean("feature", false)
                editorShuffle?.apply()
                editorLoop?.putBoolean("feature", true)
                editorLoop?.apply()
            }
        })
        playPauseImageButton?.setOnClickListener({
            if (mediaPlayer?.isPlaying as Boolean) {
                mediaPlayer?.pause()
                currentSongHelper?.isPlaying = false
                playPauseImageButton?.setBackgroundResource(R.drawable.play_icon)
            } else {
                mediaPlayer?.start()
                currentSongHelper?.isPlaying = true
                playPauseImageButton?.setBackgroundResource(R.drawable.pause_icon)
            }
        })
    }
    fun playNext(check: String) {
        if (check.equals("PlayNextNormal", true)) {
            currentPosition = currentPosition + 1
        } else if (check.equals("PlayNextLikeNormalShuffle", true)) {
            var randomObject = Random()
            var randomPosition = randomObject.nextInt(fetchSongs?.size?.plus(1) as Int)
            currentPosition = randomPosition
        }
        if (currentPosition == fetchSongs?.size) {
            currentPosition = 0
        }
        currentSongHelper?.isLoop = false
        var nextSong = fetchSongs?.get(currentPosition)
        currentSongHelper?.songPath = nextSong?.songData
        currentSongHelper?.songTitle = nextSong?.songTitle
        currentSongHelper?.songArtist = nextSong?.artist
        currentSongHelper?.songId = nextSong?.songID as Long
        updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
        mediaPlayer?.reset()
        try {
            mediaPlayer?.prepare()
            mediaPlayer?.start()
            processInformation(mediaPlayer as MediaPlayer)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    fun playPrevious() {
        currentPosition = currentPosition - 1
        if (currentPosition == -1) {
            currentPosition = 0
        }
        if (currentSongHelper?.isPlaying as Boolean) {
            playPauseImageButton?.setBackgroundResource(R.drawable.pause_icon)
        } else {
            playPauseImageButton?.setBackgroundResource(R.drawable.play_icon)
        }
        currentSongHelper?.isLoop = false
        var nextSong = fetchSongs?.get(currentPosition)
        currentSongHelper?.songPath = nextSong?.songData
        currentSongHelper?.songTitle = nextSong?.songTitle
        currentSongHelper?.songArtist = nextSong?.artist
        currentSongHelper?.songId = nextSong?.songID as Long
        updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
        mediaPlayer?.reset()
        try {
            mediaPlayer?.setDataSource(myActivity, Uri.parse(currentSongHelper?.songPath))
            mediaPlayer?.prepare()
            mediaPlayer?.start()
            processInformation(mediaPlayer as MediaPlayer)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }
    fun onSongComplete() {
        if (currentSongHelper?.isShuffle as Boolean) {
            playNext("PlayNextLikeNormalShuffle")
            currentSongHelper?.isPlaying = true
        } else {
            if (currentSongHelper?.isLoop as Boolean) {
                currentSongHelper?.isPlaying = true
                var nextSong = fetchSongs?.get(currentPosition)
                currentSongHelper?.currentPosition = currentPosition
                currentSongHelper?.songPath = nextSong?.songData
                currentSongHelper?.songTitle = nextSong?.songTitle
                currentSongHelper?.songArtist = nextSong?.artist
                currentSongHelper?.songId = nextSong?.songID as Long
                updateTextViews(currentSongHelper?.songTitle as String,
currentSongHelper?.songArtist as String)
                mediaPlayer?.reset()
                try {
                    mediaPlayer?.setDataSource(myActivity,
Uri.parse(currentSongHelper?.songPath))
                    mediaPlayer?.prepare()
                    mediaPlayer?.start()
                } catch (e: Exception) {
                    e.printStackTrace()
                }
            } else {
                playNext("PlayNextNormal")
                currentSongHelper?.isPlaying = true
            }
        }
    }
    fun updateTextViews(songTitle: String, songArtist: String) {
        songTitleView?.setText(songTitle)
        songArtistView?.setText(songArtist)
    }
    fun processInformation(mediaPlayer: MediaPlayer) {
        val finalTime = mediaPlayer.duration
        val startTime = mediaPlayer.currentPosition
        seekBar?.max = finalTime
        startTimeText?.setText(String.format("%d: %d",
                TimeUnit.MILLISECONDS.toMinutes(startTime.toLong()),
                TimeUnit.MILLISECONDS.toSeconds(startTime.toLong()) -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(startTime.toLong())))
        )
        endTimeText?.setText(String.format("%d: %d",
                TimeUnit.MILLISECONDS.toMinutes(finalTime.toLong()),
                TimeUnit.MILLISECONDS.toSeconds(finalTime.toLong()) -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes(finalTime.toLong())))
        )
        seekBar?.setProgress(startTime)
        Handler().postDelayed(updateSongTime, 1000)
    }
}
```