```
package com.internshala.echo.fragments
import android.app.Activity
import android.content.Context
import android.media.MediaPlayer
import android.os.Bundle
import android.provider.MediaStore
import android.support.v4.app.Fragment
import android.support.v4.app.FragmentActivity
import android.support.v7.widget.DefaultItemAnimator
import android.support.v7.widget.LinearLayoutManager
import android.support.v7.widget.RecyclerView
import android.view.LayoutInflater
import android.view.Menu
import android.view.View
import android.view.ViewGroup
import android.widget.ImageButton
import android.widget.RelativeLayout
import android.widget.TextView
import com.internshala.echo.CurrentSongHelper
import com.internshala.echo.R
import com.internshala.echo.Songs
import com.internshala.echo.adapters.FavoriteAdapter
import com.internshala.echo.databases.EchoDatabase
/**
 * A simple [Fragment] subclass.
 */

class FavoriteFragment : Fragment() {

     var myActivity: Activity? = null
     var noFavorites: TextView? = null
     var nowPlayingBottomBar: RelativeLayout? = null
     var playPauseButton: ImageButton? = null
     var songTitle: TextView? = null
     var recyclerView: RecyclerView? = null
     var trackPosition: Int = 0

    /*This variable will be used for database instance*/
     var favoriteContent: EchoDatabase? = null

    /*Variable to store favorites*/
     var refreshList: ArrayList<Songs>? = null

    var getListfromDatabase: ArrayList<Songs>? = null

    object Statified {
          var mediaPlayer: MediaPlayer? = null
     }

    override fun onCreateView(inflater: LayoutInflater?, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
          val view = inflater!!.inflate(R.layout.fragment_favorite, container, false)
          favoriteContent = EchoDatabase(myActivity)
          noFavorites = view?.findViewById(R.id.noFavourites)
          nowPlayingBottomBar = view.findViewById(R.id.hiddenBarFavScreen)
          songTitle = view.findViewById(R.id.songTitle)
          playPauseButton = view.findViewById(R.id.playPauseButton)
          recyclerView = view.findViewById(R.id.favoriteRecycler)
          return view
     }
```

```
    override fun onAttach(context: Context?) {
        super.onAttach(context)
        myActivity = context as Activity
    }

    override fun onAttach(activity: Activity?) {
        super.onAttach(activity)
        myActivity = activity
    }

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

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        display_favorites_by_searching()
        bottomBarSetup()
    }

    override fun onResume() {
        super.onResume()
    }

    override fun onPrepareOptionsMenu(menu: Menu?) {
        super.onPrepareOptionsMenu(menu)
    }

    /*As the name suggests, this function is used to fetch the songs present in your phones
and returns the arraylist of the same*/
    fun getSongsFromPhone(): ArrayList<Songs>? {
        var arrayList = ArrayList<Songs>()

        /*A content resolver is used to access the data present in your phone
        * In this case it is used for obtaining the songs present your phone*/
        var contentResolver = myActivity?.contentResolver

        /*Here we are accessing the Media class of Audio class which in turn a class of
Media Store, which contains information about all the media files present
        * on our mobile device*/
        var songUri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI

        /*Here we make the request of songs to the content resolver to get the music files
from our device*/
        var songCursor = contentResolver?.query(songUri, null, null, null, null)

        /*In the if condition we check whether the number of music files are null or not.
The moveToFirst() function returns the first row of the results*/
        if (songCursor != null && songCursor.moveToFirst()) {
            val songId = songCursor.getColumnIndex(MediaStore.Audio.Media._ID)
            val songTitle = songCursor.getColumnIndex(MediaStore.Audio.Media.TITLE)
            val songArtist = songCursor.getColumnIndex(MediaStore.Audio.Media.ARTIST)
            val songData = songCursor.getColumnIndex(MediaStore.Audio.Media.DATA)
            val dateIndex = songCursor.getColumnIndex(MediaStore.Audio.Media.DATE_ADDED)

            /*moveToNext() returns the next row of the results. It returns null if there
is no row after the current row*/
            while (songCursor.moveToNext()) {

                var currentId = songCursor.getLong(songId)
```

```
                var currentTitle = songCursor.getString(songTitle)
                var currentArtist = songCursor.getString(songArtist)
                var currentData = songCursor.getString(songData)
                var currentDate = songCursor.getLong(dateIndex)

                /*Adding the fetched songs to the arraylist*/
                arrayList.add(Songs(currentId, currentTitle, currentArtist, currentData,
currentDate))
            }
        } else {
            return null
        }

        /*Returning the arraylist of songs*/
        return arrayList
    }

    /*The bottom bar setup function is used to place the bottom bar on the favorite screen
when we come back from the song playing screen to the favorite screen*/
    fun bottomBarSetup() {
        try {

            /*Calling the click handler function will help us handle the click events of
the bottom bar*/
            bottomBarClickHandler()

            /*We fetch the song title with the help of the current song helper and set it
to the song title in our bottom bar*/
            songTitle?.setText(SongPlayingFragment.Statified.currentSongHelper?.songTitle)

            /*If we are the on the favorite screen and not on the song playing screen when
the song finishes
            * we want the changes in the song to reflect on the favorite screen hence we
call the onSongComplete() function which help us in maintaining consistency*/
            SongPlayingFragment.Statified.mediaPlayer?.setOnCompletionListener({

songTitle?.setText(SongPlayingFragment.Statified.currentSongHelper?.songTitle)
                SongPlayingFragment.Staticated.onSongComplete()
            })

            /*While coming back from the song playing screen
            * if the song was playing then only the bottom bar is placed, else not
placed*/
            if (SongPlayingFragment.Statified.mediaPlayer?.isPlaying as Boolean) {
                nowPlayingBottomBar?.visibility = View.VISIBLE
            } else {
                nowPlayingBottomBar?.visibility = View.INVISIBLE
            }

            /*Since we are dealing with the media player object which can be null, hence
we handle all such exceptions using the try-catch block*/
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    /*The bottomBarClickHandler() function is used to handle the click events on the bottom
bar*/
    fun bottomBarClickHandler() {
```

```
        /*We place a click listener on the bottom bar*/
        nowPlayingBottomBar?.setOnClickListener({

            /*Using the same media player object*/
            Statified.mediaPlayer = SongPlayingFragment.Statified.mediaPlayer
            val songPlayingFragment = SongPlayingFragment()
            var args = Bundle()

            /*Here when we click on the bottom bar, we navigate to the song playing
fragment
            * Since we want the details of the same song which is playing to be displayed
in the song playing fragment
            * we pass the details of the current song being played to the song playing
fragment using Bundle*/
            args.putString("songArtist",
SongPlayingFragment.Statified.currentSongHelper?.songArtist)
            args.putString("songTitle",
SongPlayingFragment.Statified.currentSongHelper?.songTitle)
            args.putString("path",
SongPlayingFragment.Statified.currentSongHelper?.songPath)
            args.putInt("songId",
SongPlayingFragment.Statified.currentSongHelper?.songId?.toInt() as Int)
            args.putInt("songPosition",
SongPlayingFragment.Statified.currentSongHelper?.currentPosition?.toInt() as Int)
            args.putParcelableArrayList("songData",
SongPlayingFragment.Statified.fetchSongs)

            /*Here we put the additional string in the bundle
            * this tells us that the bottom bar was successfully setup*/
            args.putString("FavBottomBar", "success")

            /*Here we pass the bundle object to the song playing fragment*/
            songPlayingFragment.arguments = args

            /*The below lines are now familiar
            * These are used to open a fragment*/
            fragmentManager.beginTransaction()
                    .replace(R.id.details_fragment, songPlayingFragment)

                    /*The below piece of code is used to handle the back navigation
                    * This means that when you click the bottom bar and move on to the
next screen
                    * on pressing back button you navigate to the screen you came from*/
                    .addToBackStack("SongPlayingFragment")
                    .commit()
        })

        /*Apart from the click on the bottom bar we have a play/pause button in our bottom
bar
        * This button is used to play or pause the media player*/
        playPauseButton?.setOnClickListener({
            if (SongPlayingFragment.Statified.mediaPlayer?.isPlaying as Boolean) {

                /*If the song was already playing, we then pause it and save the it's
position
                * and then change the button to play button*/
                SongPlayingFragment.Statified.mediaPlayer?.pause()
                trackPosition = SongPlayingFragment.Statified.mediaPlayer?.currentPosition
as Int
                playPauseButton?.setBackgroundResource(R.drawable.play_icon)
            } else {
```

```
                    /*If the music was already paused and we then click on the button
                    * it plays the song from the same position where it was paused
                    * and change the button to pause button*/
                    SongPlayingFragment.Statified.mediaPlayer?.seekTo(trackPosition)
                    SongPlayingFragment.Statified.mediaPlayer?.start()
                    playPauseButton?.setBackgroundResource(R.drawable.pause_icon)
                }
            })
        }

        /*The below function is used to search the favorites and display*/
        fun display_favorites_by_searching() {

            /*Checking if database has any entry or not*/
            if (favoriteContent?.checkSize() as Int > 0) {

                /*New list for storing the favorites*/
                refreshList = ArrayList<Songs>()

                /*Getting the list of songs from database*/
                getListfromDatabase = favoriteContent?.queryDBList()

                /*Getting list of songs from phone storage*/
                val fetchListfromDevice = getSongsFromPhone()

                /*If there are no songs in phone then there cannot be any favorites*/
                if (fetchListfromDevice != null) {

                    /*Then we check all the songs in the phone*/
                    for (i in 0..fetchListfromDevice?.size - 1) {

                        /*We iterate through every song in database*/
                        for (j in 0..getListfromDatabase?.size as Int - 1) {

                            /*While iterating through all the songs we check for the songs
which are in both the lists
                            * i.e. the favorites songs*/
                            if (getListfromDatabase?.get(j)?.songID ===
fetchListfromDevice?.get(i)?.songID) {

                                /*on getting the favorite songs we add them to the refresh
list*/
                                refreshList?.add((getListfromDatabase as ArrayList<Songs>)
[j])
                            }
                        }
                    }
                } else {
                }

                /*If refresh list is null we display that there are no favorites*/
                if (refreshList == null) {
                    recyclerView?.visibility = View.INVISIBLE
                    noFavorites?.visibility = View.VISIBLE
                } else {

                    /*Else we setup our recycler view for displaying the favorite songs*/
                    val favoriteAdapter = FavoriteAdapter(refreshList as ArrayList<Songs>,
myActivity as Context)
```

```
                val mLayoutManager = LinearLayoutManager(activity)
                recyclerView?.layoutManager = mLayoutManager
                recyclerView?.itemAnimator = DefaultItemAnimator()
                recyclerView?.adapter = favoriteAdapter
                recyclerView?.setHasFixedSize(true)
            }
        } else {

            /*If initially the checkSize() function returned 0 then also we display the no
favorites present message*/
            recyclerView?.visibility = View.INVISIBLE
            noFavorites?.visibility = View.VISIBLE
        }
    }
}
```