```
package com.internshala.echo.fragments
import android.app.Activity
import android.content.Context
import android.os.Bundle
import android.provider.MediaStore
import android.support.v4.app.Fragment
import android.support.v7.widget.DefaultItemAnimator
import android.support.v7.widget.LinearLayoutManager
import android.support.v7.widget.RecyclerView
import android.view.*
import android.widget.ImageButton
import android.widget.LinearLayout
import android.widget.RelativeLayout
import android.widget.TextView
import com.internshala.echo.R
import com.internshala.echo.Songs
import com.internshala.echo.adapters.MainScreenAdapter
import java.util.*
/**
 * A simple [Fragment] subclass for displaying the songs on the main screen. Here we will
inflate our recycler view with the data of the songs on our phone.
 */
class MainScreenFragment : Fragment() {
    /*Let's see the usage of every variable declared here*/
    /*The arraylist is used for storing the songs along with the data associated with it*/
    var getSongsList: ArrayList<Songs>? = null
    /*The now playing bar at the bottom with basic play pause functionality*/
    var nowPlayingBottomBar: RelativeLayout? = null
    /*Play/pause button in bottom bar*/
    var playPauseButton: ImageButton? = null
    /*Variable which stores the song title*/
    var songTitle: TextView? = null
    /*The layout which is used to display the songs and the bottom bar*/
    var visibleLayout: RelativeLayout? = null
    /*The layout used to display the no songs present message to user when there are no
songs present*/
    var noSongs: RelativeLayout? = null
    /*Recycler view i.e. the list which is used for displaying the songs*/
    var recyclerView: RecyclerView? = null
    /*The variable used to store the context of the activity*/
    var myActivity: Activity? = null
    /*The variable used for main screen adapter in order to link the adapter with the
recycler view*/
    var _mainScreenAdapter: MainScreenAdapter? = null
    /*This method is called after the onCreateView() method when the fragment's activity
has been created and the view has been instantiated
    * It is used to do the final initialization once the other things are in place*/
    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        /*The variable getSongsList() is used to get store the arrayList returned by the
function getSongsFromPhone()*/
        getSongsList = getSongsFromPhone()
        /*Declaring the preferences to save the sorting order which we select*/
        val prefs = activity.getSharedPreferences("action_sort", Context.MODE_PRIVATE)
        val action_sort_ascending = prefs.getString("action_sort_ascending", "true")
        val action_sort_recent = prefs.getString("action_sort_recent", "false")
        /*If there are no songs we do not display the list instead we display no songs
message*/
```

```
        if (getSongsList == null) {
            visibleLayout?.visibility = View.INVISIBLE
            noSongs?.visibility = View.VISIBLE
        }
        /*If there are songs in the device, we display the list*/
        else {
            /*Here we initialize the main screen adapter and pass it the required
parameters i.e. the list of songs and the context*/
            _mainScreenAdapter = MainScreenAdapter(getSongsList as ArrayList<Songs>,
myActivity as Context)
            /*The layout manager defines the way a view will be set in the recycler view
            * There are different types of layout managers e.g. Linear, Grid, Staggered
grid
            * Here we are using the Linear layout manager which aligns the objects in a
linear way one under the other*/
            val mLayoutManager = LinearLayoutManager(myActivity)
            /*Here we put assign our layout manager to the recycler view's layout
manager*/
            recyclerView?.layoutManager = mLayoutManager
            /*It is similar to the item animator we used in the navigation drawer*/
            recyclerView?.itemAnimator = DefaultItemAnimator()
            /*Finally we set the adapter to the recycler view*/
            recyclerView?.adapter = _mainScreenAdapter
        }
        /*If the songs list is not empty, then we check whether applied any comparator
        * And we use that comparator and sort the list accordingly*/
        if (getSongsList != null) {
            if (action_sort_ascending!!.equals("true", ignoreCase = true)) {
                Collections.sort(getSongsList, Songs.Statified.nameComparator)
                _mainScreenAdapter?.notifyDataSetChanged()
            } else if (action_sort_recent!!.equals("true", ignoreCase = true)) {
                Collections.sort(getSongsList, Songs.Statified.dateComparator)
                _mainScreenAdapter?.notifyDataSetChanged()
            }
        }
    }
    /*The onCreateView() method is a mandatory method of the fragment life-cycle. A
fragment also has a life-cycle similar to an Activity.
    * A fragments life-cycle contains methods such as onAttach(), onCreate(),
onCreateView(), onActivityCreated(), onStart(), onResume(), and so-on similar to activity.
    * Please visit the link www.developer.android.com/guide/components/fragments.html  to
learn more about fragment life-cycle*/
    /*Now coming onto the onCreateView() method, it is called to have the fragment
instantiate its user interface*/
    override fun onCreateView(inflater: LayoutInflater?, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        // Inflating the layout for this fragment
        val view = inflater!!.inflate(R.layout.fragment_main_screen, container, false)
        /*This is used to tell the activity that the fragment has a menu*/
        setHasOptionsMenu(true)
        visibleLayout = view?.findViewById(R.id.visibleLayout)
        noSongs = view?.findViewById(R.id.noSongs)
        nowPlayingBottomBar = view?.findViewById(R.id.hiddenBarMainScreen)
        songTitle = view?.findViewById(R.id.songTitleMainScreen)
        playPauseButton = view?.findViewById(R.id.playPauseButton)
        recyclerView = view?.findViewById(R.id.contentMain)
        /*Here the view returned is inflated on the screen and is visible to the user*/
```

```
        return view
    }
    /*Called when the fragment is first attached to its context*/
    override fun onAttach(context: Context?) {
        super.onAttach(context)
        /*Here we assign our myActivity variable the value of the context of the activity
in which the fragment resides*/
        myActivity = context as Activity
    }
    /*Called when the fragment was first attached to its activity. This method was
deprecated which means that now it is no longer user.
    * The above method is also the same, but we still use it sometimes as to prevent some
crashes on some older devices*/
    override fun onAttach(activity: Activity?) {
        super.onAttach(activity)
        myActivity = activity
    }
    /*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))
            }
        }
        /*Returning the arraylist of songs*/
        return arrayList
    }
    override fun onCreateOptionsMenu(menu: Menu?, inflater: MenuInflater?) {
        menu?.clear()
```

```
        inflater?.inflate(R.menu.main, menu)
        return
    }
    /*Here we perform the actions of sorting according to the menu item clicked*/
    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        val switcher = item?.itemId
        if (switcher == R.id.action_sort_ascending) {
            /*Whichever action item is selected, we save the preferences and perform the
operation of comparison*/
            val editor = myActivity?.getSharedPreferences("action_sort",
Context.MODE_PRIVATE)?.edit()
            editor?.putString("action_sort_ascending", "true")
            editor?.putString("action_sort_recent", "false")
            editor?.apply()
            if (getSongsList != null) {
                Collections.sort(getSongsList, Songs.Statified.nameComparator)
            }
            _mainScreenAdapter?.notifyDataSetChanged()
            return false
        } else if (switcher == R.id.action_sort_recent) {
            val editortwo = myActivity?.getSharedPreferences("action_sort",
Context.MODE_PRIVATE)?.edit()
            editortwo?.putString("action_sort_recent", "true")
            editortwo?.putString("action_sort_ascending", "false")
            editortwo?.apply()
            if (getSongsList != null) {
                Collections.sort(getSongsList, Songs.Statified.dateComparator)
            }
            _mainScreenAdapter?.notifyDataSetChanged()
            return false
        }
        return super.onOptionsItemSelected(item)
    }
}
```