index page

```
D:\DWONLOADS\PROJECT\OPEN SOURCE PROJECTS\TIMBER-MASTER\APP\SRC\MAIN\JAVA
□□□□com
    □□□□naman14
        □□□□timber
            □□□□activities
            □□□□adapters
            □□□□cast
            □□□□dataloaders
            □□□□dialogs
            □□□□fragments
            □□□□helpers
            □□□□lastfmapi
            □   □□□□callbacks
            □   □□□□models
            □□□□listeners
            □□□□models
            □□□□nowplaying
            □□□□permissions
            □□□□provider
            □□□□slidinguppanel
            □□□□subfragments
            □□□□timely
            □   □□□□animation
            □   □□□□model
            □       □□□□core
            □       □□□□number
            □□□□transition
            □□□□utils
            □□□□widgets
                □□□□desktop
```

```java
/*
 * Copyright (C) 2012 Andrew Neal
 * Copyright (C) 2014 The CyanogenMod Project
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber;

import android.Manifest;
import android.app.Activity;
import android.content.ComponentName;
import android.content.ContentResolver;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.ContextWrapper;
import android.content.Intent;
import android.content.ServiceConnection;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.net.Uri;
import android.os.IBinder;
import android.os.RemoteException;
import android.provider.BaseColumns;
import android.provider.MediaStore;
import android.widget.Toast;

import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.helpers.MusicPlaybackTrack;
import com.naman14.timber.utils.TimberUtils.IdType;

import java.util.Arrays;
import java.util.WeakHashMap;

import static android.support.v4.content.PermissionChecker.checkSelfPermission;

public class MusicPlayer {

    private static final WeakHashMap<Context, ServiceBinder> mConnectionMap;
    private static final long[] sEmptyList;
    public static ITimberService mService = null;
    private static ContentValues[] mContentValuesCache = null;

    static {
        mConnectionMap = new WeakHashMap<Context, ServiceBinder>();
        sEmptyList = new long[0];
    }

    public static final ServiceToken bindToService(final Context context,
                                                   final ServiceConnection callback) {

        Activity realActivity = ((Activity) context).getParent();
        if (realActivity == null) {
            realActivity = (Activity) context;
        }
        final ContextWrapper contextWrapper = new ContextWrapper(realActivity);
        contextWrapper.startService(new Intent(contextWrapper, MusicService.class));
        final ServiceBinder binder = new ServiceBinder(callback,
                contextWrapper.getApplicationContext());
        if (contextWrapper.bindService(
                new Intent().setClass(contextWrapper, MusicService.class), binder, 0)) {
```

```java
            mConnectionMap.put(contextWrapper, binder);
            return new ServiceToken(contextWrapper);
        }
        return null;
    }

    public static void unbindFromService(final ServiceToken token) {
        if (token == null) {
            return;
        }
        final ContextWrapper mContextWrapper = token.mWrappedContext;
        final ServiceBinder mBinder = mConnectionMap.remove(mContextWrapper);
        if (mBinder == null) {
            return;
        }
        mContextWrapper.unbindService(mBinder);
        if (mConnectionMap.isEmpty()) {
            mService = null;
        }
    }

    public static final boolean isPlaybackServiceConnected() {
        return mService != null;
    }

    public static void next() {
        try {
            if (mService != null) {
                mService.next();
            }
        } catch (final RemoteException ignored) {
        }
    }

    public static void initPlaybackServiceWithSettings(final Context context) {

    }

    public static void asyncNext(final Context context) {
        final Intent previous = new Intent(context, MusicService.class);
        previous.setAction(MusicService.NEXT_ACTION);
        context.startService(previous);
    }

    public static void previous(final Context context, final boolean force) {
        final Intent previous = new Intent(context, MusicService.class);
        if (force) {
            previous.setAction(MusicService.PREVIOUS_FORCE_ACTION);
        } else {
            previous.setAction(MusicService.PREVIOUS_ACTION);
        }
        context.startService(previous);
    }

    public static void playOrPause() {
        try {
            if (mService != null) {
                if (mService.isPlaying()) {
                    mService.pause();
                } else {
                    mService.play();
                }
            }
        } catch (final Exception ignored) {
        }
    }

    public static void cycleRepeat() {
        try {
            if (mService != null) {
                switch (mService.getRepeatMode()) {
                    case MusicService.REPEAT_NONE:
                        mService.setRepeatMode(MusicService.REPEAT_ALL);
                        break;
                    case MusicService.REPEAT_ALL:
                        mService.setRepeatMode(MusicService.REPEAT_CURRENT);
                        if (mService.getShuffleMode() != MusicService.SHUFFLE_NONE) {
                            mService.setShuffleMode(MusicService.SHUFFLE_NONE);
                        }
                        break;
                    default:
                        mService.setRepeatMode(MusicService.REPEAT_NONE);
                        break;
                }
            }
        } catch (final RemoteException ignored) {
        }
    }

    public static void cycleShuffle() {
        try {
            if (mService != null) {
                switch (mService.getShuffleMode()) {
                    case MusicService.SHUFFLE_NONE:
                        mService.setShuffleMode(MusicService.SHUFFLE_NORMAL);
                        if (mService.getRepeatMode() == MusicService.REPEAT_CURRENT) {
                            mService.setRepeatMode(MusicService.REPEAT_ALL);
                        }
                        break;
                    case MusicService.SHUFFLE_NORMAL:
                        mService.setShuffleMode(MusicService.SHUFFLE_NONE);
                        break;
                    case MusicService.SHUFFLE_AUTO:
                        mService.setShuffleMode(MusicService.SHUFFLE_NONE);
                        break;
                    default:
                        break;
                }
            }
        } catch (final RemoteException ignored) {
        }
    }

    public static final boolean isPlaying() {
        if (mService != null) {
            try {
                return mService.isPlaying();
            } catch (final RemoteException ignored) {
            }
        }
        return false;
    }

    public static final int getShuffleMode() {
        if (mService != null) {
            try {
                return mService.getShuffleMode();
            } catch (final RemoteException ignored) {
            }
        }
        return 0;
    }

    public static void setShuffleMode(int mode) {
        try {
            if (mService != null) {
                mService.setShuffleMode(mode);
            }
        } catch (RemoteException ignored) {
        }
    }

    public static final int getRepeatMode() {
        if (mService != null) {
            try {
                return mService.getRepeatMode();
            } catch (final RemoteException ignored) {
            }
        }
        return 0;
    }

    public static final String getTrackName() {
        if (mService != null) {
            try {
                return mService.getTrackName();
            } catch (final RemoteException ignored) {
            }
        }
        return null;
    }

    public static final String getArtistName() {
        if (mService != null) {
            try {
                return mService.getArtistName();
            } catch (final RemoteException ignored) {
            }
        }
        return null;
    }

    public static final String getAlbumName() {
        if (mService != null) {
            try {
                return mService.getAlbumName();
            } catch (final RemoteException ignored) {
            }
        }
        return null;
    }

    public static final long getCurrentAlbumId() {
        if (mService != null) {
            try {
                return mService.getAlbumId();
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final long getCurrentAudioId() {
        if (mService != null) {
            try {
                return mService.getAudioId();
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final MusicPlaybackTrack getCurrentTrack() {
        if (mService != null) {
            try {
                return mService.getCurrentTrack();
            } catch (final RemoteException ignored) {
            }
        }
        return null;
    }

    public static final MusicPlaybackTrack getTrack(int index) {
        if (mService != null) {
            try {
                return mService.getTrack(index);
            } catch (final RemoteException ignored) {
            }
        }
        return null;
    }

    public static final long getNextAudioId() {
        if (mService != null) {
            try {
                return mService.getNextAudioId();
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final long getPreviousAudioId() {
        if (mService != null) {
            try {
                return mService.getPreviousAudioId();
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final long getCurrentArtistId() {
        if (mService != null) {
            try {
                return mService.getArtistId();
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final int getAudioSessionId() {
        if (mService != null) {
            try {
                return mService.getAudioSessionId();
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final long[] getQueue() {
        try {
            if (mService != null) {
                return mService.getQueue();
            } else {
            }
        } catch (final RemoteException ignored) {
        }
        return sEmptyList;
    }

    public static final long getQueueItemAtPosition(int position) {
        try {
            if (mService != null) {
                return mService.getQueueItemAtPosition(position);
            } else {
            }
        } catch (final RemoteException ignored) {
        }
        return -1;
    }

    public static final int getQueueSize() {
        try {
            if (mService != null) {
                return mService.getQueueSize();
            } else {
            }
        } catch (final RemoteException ignored) {
        }
        return 0;
    }

    public static final int getQueuePosition() {
        try {
            if (mService != null) {
                return mService.getQueuePosition();
            }
        } catch (final RemoteException ignored) {
        }
        return 0;
    }

    public static void setQueuePosition(final int position) {
        if (mService != null) {
            try {
                mService.setQueuePosition(position);
            } catch (final RemoteException ignored) {
            }
        }
    }

    public static void refresh() {
        try {
            if (mService != null) {
                mService.refresh();
            }
        } catch (final RemoteException ignored) {
        }
    }

    public static final int getQueueHistorySize() {
        if (mService != null) {
            try {
                return mService.getQueueHistorySize();
            } catch (final RemoteException ignored) {
            }
        }
        return 0;
    }

    public static final int getQueueHistoryPosition(int position) {
        if (mService != null) {
            try {
                return mService.getQueueHistoryPosition(position);
            } catch (final RemoteException ignored) {
            }
        }
        return -1;
    }

    public static final int[] getQueueHistoryList() {
        if (mService != null) {
            try {
                return mService.getQueueHistoryList();
            } catch (final RemoteException ignored) {
            }
```

```java
        }
        return null;
    }

    public static final int removeTrack(final long id) {
        try {
            if (mService != null) {
                return mService.removeTrack(id);
            }
        } catch (final RemoteException ingored) {
        }
        return 0;
    }

    public static final boolean removeTrackAtPosition(final long id, final int position) {
        try {
            if (mService != null) {
                return mService.removeTrackAtPosition(id, position);
            }
        } catch (final RemoteException ingored) {
        }
        return false;
    }

    public static void moveQueueItem(final int from, final int to) {
        try {
            if (mService != null) {
                mService.moveQueueItem(from, to);
            } else {
            }
        } catch (final RemoteException ignored) {
        }
    }

    public static void playArtist(final Context context, final long artistId, int position, boolean shuffle) {
        final long[] artistList = getSongListForArtist(context, artistId);
        if (artistList != null) {
            playAll(context, artistList, position, artistId, IdType.Artist, shuffle);
        }
    }

    public static void playAlbum(final Context context, final long albumId, int position, boolean shuffle) {
        final long[] albumList = getSongListForAlbum(context, albumId);
        if (albumList != null) {
            playAll(context, albumList, position, albumId, IdType.Album, shuffle);
        }
    }

    public static void playAll(final Context context, final long[] list, int position,
                               final long sourceId, final IdType sourceType,
                               final boolean forceShuffle) {
        if (list == null || list.length == 0 || mService == null) {
            return;
        }
        try {
            if (forceShuffle) {
                mService.setShuffleMode(MusicService.SHUFFLE_NORMAL);
            }
            final long currentId = mService.getAudioId();
            final int currentQueuePosition = getQueuePosition();
            if (position != -1 && currentQueuePosition == position && currentId == list[position]) {
                final long[] playlist = getQueue();
                if (Arrays.equals(list, playlist)) {
                    mService.play();
                    return;
                }
            }
            if (position < 0) {
                position = 0;
            }
            mService.open(list, forceShuffle ? -1 : position, sourceId, sourceType.mId);
            mService.play();
        } catch (final RemoteException ignored) {
        } catch (IllegalStateException e) {
            e.printStackTrace();
        }
    }

    public static void playNext(Context context, final long[] list, final long sourceId, final IdType sourceType) {
        if (mService == null) {
            return;
        }
        try {
            mService.enqueue(list, MusicService.NEXT, sourceId, sourceType.mId);
            final String message = makeLabel(context, R.plurals.NNNtrackstoqueue, list.length);
            Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
        } catch (final RemoteException ignored) {
        }
    }

    public static void shuffleAll(final Context context) {
        Cursor cursor = SongLoader.makeSongCursor(context, null, null);
        final long[] trackList = SongLoader.getSongListForCursor(cursor);
        if (trackList.length == 0 || mService == null) {
            return;
        }
        try {
            mService.setShuffleMode(MusicService.SHUFFLE_NORMAL);
            if (getQueuePosition() == 0 && mService.getAudioId() == trackList[0] && Arrays.equals(trackList, getQueue())) {
                    mService.play();
                    return;
            }
            mService.open(trackList, -1, -1, IdType.NA.mId);
            mService.play();
            cursor.close();
        } catch (final RemoteException ignored) {
        }
    }

    public static final long[] getSongListForArtist(final Context context, final long id) {
        final String[] projection = new String[]{
                BaseColumns._ID
        };
        final String selection = MediaStore.Audio.AudioColumns.ARTIST_ID + "=" + id + " AND "
                + MediaStore.Audio.AudioColumns.IS_MUSIC + "=1";
        Cursor cursor = context.getContentResolver().query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection, selection, null,
                MediaStore.Audio.AudioColumns.ALBUM_KEY + "," + MediaStore.Audio.AudioColumns.TRACK);
        if (cursor != null) {
            final long[] mList = SongLoader.getSongListForCursor(cursor);
            cursor.close();
            cursor = null;
            return mList;
        }
        return sEmptyList;
    }

    public static final long[] getSongListForAlbum(final Context context, final long id) {
        final String[] projection = new String[]{
                BaseColumns._ID
        };
        final String selection = MediaStore.Audio.AudioColumns.ALBUM_ID + "=" + id + " AND " + MediaStore.Audio.AudioColumns
                + "=1";
        Cursor cursor = context.getContentResolver().query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection, selection, null,
                MediaStore.Audio.AudioColumns.TRACK + ", " + MediaStore.Audio.Media.DEFAULT_SORT_ORDER);
        if (cursor != null) {
            final long[] mList = SongLoader.getSongListForCursor(cursor);
            cursor.close();
            cursor = null;
            return mList;
        }
        return sEmptyList;
    }

    public static final int getSongCountForAlbumInt(final Context context, final long id) {
        int songCount = 0;
        if (id == -1) {
            return songCount;
        }

        Uri uri = ContentUris.withAppendedId(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI, id);
        Cursor cursor = context.getContentResolver().query(uri,
                new String[]{MediaStore.Audio.AlbumColumns.NUMBER_OF_SONGS}, null, null, null);
        if (cursor != null) {
            cursor.moveToFirst();
            if (!cursor.isAfterLast()) {
                if (!cursor.isNull(0)) {
                    songCount = cursor.getInt(0);
                }
            }
            cursor.close();
            cursor = null;
        }

        return songCount;
    }

    public static final String getReleaseDateForAlbum(final Context context, final long id) {
        if (id == -1) {
            return null;
        }
        Uri uri = ContentUris.withAppendedId(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI, id);
        Cursor cursor = context.getContentResolver().query(uri, new String[]{
                MediaStore.Audio.AlbumColumns.FIRST_YEAR
        }, null, null, null);
        String releaseDate = null;
        if (cursor != null) {
            cursor.moveToFirst();
            if (!cursor.isAfterLast()) {
                releaseDate = cursor.getString(0);
            }
            cursor.close();
            cursor = null;
        }
        return releaseDate;
    }

    public static void seek(final long position) {
        if (mService != null) {
            try {
                mService.seek(position);
            } catch (final RemoteException ignored) {
            } catch (IllegalStateException ignored) {

            }
        }
    }

    public static void seekRelative(final long deltaInMs) {
        if (mService != null) {
            try {
                mService.seekRelative(deltaInMs);
            } catch (final RemoteException ignored) {
            } catch (final IllegalStateException ignored) {

            }
        }
    }
    public static final long position() {
        if (mService != null) {
            try {
                return mService.position();
            } catch (final RemoteException ignored) {
            } catch (final IllegalStateException ex) {

            }
        }
        return 0;
    }

    public static final long duration() {
        if (mService != null) {
            try {
                return mService.duration();
            } catch (final RemoteException ignored) {
            } catch (final IllegalStateException ignored) {

            }
        }
        return 0;
    }

    public static void clearQueue() {
        if (mService!=null) {
            try {
                mService.removeTracks(0, Integer.MAX_VALUE);
            } catch (final RemoteException ignored) {
            }
        }
    }

    public static void addToQueue(final Context context, final long[] list, long sourceId,
                                  IdType sourceType) {
        if (mService == null) {
            return;
        }
        try {
            mService.enqueue(list, MusicService.LAST, sourceId, sourceType.mId);
            final String message = makeLabel(context, R.plurals.NNNtrackstoqueue, list.length);
            Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
        } catch (final RemoteException ignored) {
        }
    }

    public static final String makeLabel(final Context context, final int pluralInt,
                                         final int number) {
        return context.getResources().getQuantityString(pluralInt, number, number);
    }

    public static void addToPlaylist(final Context context, final long[] ids, final long playlistid) {
        final int size = ids.length;
        final ContentResolver resolver = context.getContentResolver();
        final String[] projection = new String[]{
                "max(" + "play_order" + ")",
        };
        final Uri uri = MediaStore.Audio.Playlists.Members.getContentUri("external", playlistid);
        Cursor cursor = null;
        int base = 0;

        try {
            cursor = resolver.query(uri, projection, null, null, null);

            if (cursor != null && cursor.moveToFirst()) {
                base = cursor.getInt(0) + 1;
            }
        } finally {
            if (cursor != null) {
                cursor.close();
```

```java
                cursor = null;
            }
        }

        int numinserted = 0;
        for (int offSet = 0; offSet < size; offSet += 1000) {
            makeInsertItems(ids, offSet, 1000, base);
            numinserted += resolver.bulkInsert(uri, mContentValuesCache);
        }
        final String message = context.getResources().getQuantityString(
                R.plurals.NNNtrackstoplaylist, numinserted, numinserted);
        Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
    }

    public static void makeInsertItems(final long[] ids, final int offset, int len, final int base) {
        if (offset + len > ids.length) {
            len = ids.length - offset;
        }

        if (mContentValuesCache == null || mContentValuesCache.length != len) {
            mContentValuesCache = new ContentValues[len];
        }
        for (int i = 0; i < len; i++) {
            if (mContentValuesCache[i] == null) {
                mContentValuesCache[i] = new ContentValues();
            }
            mContentValuesCache[i].put(MediaStore.Audio.Playlists.Members.PLAY_ORDER, base + offset + i);
            mContentValuesCache[i].put(MediaStore.Audio.Playlists.Members.AUDIO_ID, ids[offset + i]);
        }
    }

    public static final long createPlaylist(final Context context, final String name) {
        if (name != null && name.length() > 0) {
            final ContentResolver resolver = context.getContentResolver();
            final String[] projection = new String[]{
                    MediaStore.Audio.PlaylistsColumns.NAME
            };
            final String selection = MediaStore.Audio.PlaylistsColumns.NAME + " = '" + name + "'";
            Cursor cursor = resolver.query(MediaStore.Audio.Playlists.EXTERNAL_CONTENT_URI,
                    projection, selection, null, null);
            if (cursor.getCount() <= 0) {
                final ContentValues values = new ContentValues(1);
                values.put(MediaStore.Audio.PlaylistsColumns.NAME, name);
                final Uri uri = resolver.insert(MediaStore.Audio.Playlists.EXTERNAL_CONTENT_URI,
                        values);
                return Long.parseLong(uri.getLastPathSegment());
            }
            if (cursor != null) {
                cursor.close();
                cursor = null;
            }
            return -1;
        }
        return -1;
    }

    public static final void openFile(final String path) {
        if (mService != null) {
            try {
                mService.openFile(path);
            } catch (final RemoteException ignored) {
            }
        }
    }

    public static final class ServiceBinder implements ServiceConnection {
        private final ServiceConnection mCallback;
        private final Context mContext;
        public ServiceBinder(final ServiceConnection callback, final Context context) {
            mCallback = callback;
            mContext = context;
        }

        @Override
        public void onServiceConnected(final ComponentName className, final IBinder service) {
            mService = ITimberService.Stub.asInterface(service);
            if (mCallback != null) {
                mCallback.onServiceConnected(className, service);
            }
            initPlaybackServiceWithSettings(mContext);
        }

        @Override
        public void onServiceDisconnected(final ComponentName className) {
            if (mCallback != null) {
                mCallback.onServiceDisconnected(className);
            }
            mService = null;
        }
    }

    public static final class ServiceToken {
        public ContextWrapper mWrappedContext;

        public ServiceToken(final ContextWrapper context) {
            mWrappedContext = context;
        }
    }
}
```

```java
/*
 * Copyright (C) 2012 Andrew Neal
 * Copyright (C) 2014 The CyanogenMod Project
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber;

import android.Manifest;
import android.annotation.SuppressLint;
import android.annotation.TargetApi;
import android.app.AlarmManager;
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.ContentResolver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.database.ContentObserver;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.media.AudioManager;
import android.media.AudioManager.OnAudioFocusChangeListener;
import android.media.MediaMetadataEditor;
import android.media.MediaMetadataRetriever;
import android.media.MediaPlayer;
import android.media.RemoteControlClient;
import android.media.audiofx.AudioEffect;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.os.Looper;
import android.os.Message;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.os.RemoteException;
import android.os.SystemClock;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.AlbumColumns;
import android.provider.MediaStore.Audio.AudioColumns;
import android.support.v4.app.NotificationManagerCompat;
import android.support.v4.media.MediaMetadataCompat;
import android.support.v4.media.session.MediaSessionCompat;
import android.support.v4.media.session.PlaybackStateCompat;
import android.support.v7.graphics.Palette;
import android.text.TextUtils;
import android.util.Log;

import com.naman14.timber.helpers.MediaButtonIntentReceiver;
import com.naman14.timber.helpers.MusicPlaybackTrack;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.models.LastfmUserSession;
import com.naman14.timber.lastfmapi.models.ScrobbleQuery;
import com.naman14.timber.permissions.Nammu;
import com.naman14.timber.provider.MusicPlaybackState;
import com.naman14.timber.provider.RecentStore;
import com.naman14.timber.provider.SongPlayCount;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.utils.TimberUtils.IdType;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.io.IOException;
import java.lang.ref.WeakReference;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Random;
import java.util.TreeSet;

import de.Maxr1998.trackselectorlib.ModNotInstalledException;
import de.Maxr1998.trackselectorlib.NotificationHelper;
import de.Maxr1998.trackselectorlib.TrackItem;

@SuppressLint("NewApi")
public class MusicService extends Service {
    public static final String PLAYSTATE_CHANGED = "com.naman14.timber.playstatechanged";
    public static final String POSITION_CHANGED = "com.naman14.timber.positionchanged";
    public static final String META_CHANGED = "com.naman14.timber.metachanged";
    public static final String QUEUE_CHANGED = "com.naman14.timber.queuechanged";
    public static final String PLAYLIST_CHANGED = "com.naman14.timber.playlistchanged";
    public static final String REPEATMODE_CHANGED = "com.naman14.timber.repeatmodechanged";
    public static final String SHUFFLEMODE_CHANGED = "com.naman14.timber.shufflemodechanged";
    public static final String TRACK_ERROR = "com.naman14.timber.trackerror";
    public static final String TIMBER_PACKAGE_NAME = "com.naman14.timber";
    public static final String MUSIC_PACKAGE_NAME = "com.android.music";
    public static final String SERVICECMD = "com.naman14.timber.musicservicecommand";
    public static final String TOGGLEPAUSE_ACTION = "com.naman14.timber.togglepause";
    public static final String PAUSE_ACTION = "com.naman14.timber.pause";
    public static final String STOP_ACTION = "com.naman14.timber.stop";
    public static final String PREVIOUS_ACTION = "com.naman14.timber.previous";
    public static final String PREVIOUS_FORCE_ACTION = "com.naman14.timber.previous.force";
    public static final String NEXT_ACTION = "fcom.naman14.timber.next";
    public static final String REPEAT_ACTION = "com.naman14.timber.repeat";
    public static final String SHUFFLE_ACTION = "com.naman14.timber.shuffle";
    public static final String FROM_MEDIA_BUTTON = "frommediabutton";
    public static final String REFRESH = "com.naman14.timber.refresh";
    public static final String UPDATE_LOCKSCREEN = "com.naman14.timber.updatelockscreen";
    public static final String CMDNAME = "command";
    public static final String CMDTOGGLEPAUSE = "togglepause";
    public static final String CMDSTOP = "stop";
    public static final String CMDPAUSE = "pause";
    public static final String CMDPLAY = "play";
    public static final String CMDPREVIOUS = "previous";
    public static final String CMDNEXT = "next";
    public static final String CMDNOTIF = "buttonId";
    public static final String UPDATE_PREFERENCES = "updatepreferences";
    public static final String CHANNEL_ID = "timber_channel_01";
    public static final int NEXT = 2;
    public static final int LAST = 3;
    public static final int SHUFFLE_NONE = 0;
    public static final int SHUFFLE_NORMAL = 1;
    public static final int SHUFFLE_AUTO = 2;
    public static final int REPEAT_NONE = 0;
    public static final int REPEAT_CURRENT = 1;
    public static final int REPEAT_ALL = 2;
    public static final int MAX_HISTORY_SIZE = 1000;
    private static final String TAG = "MusicPlaybackService";
    private static final boolean D = false;
    private static final String SHUTDOWN = "com.naman14.timber.shutdown";
    private static final int IDCOLIDX = 0;
    private static final int TRACK_ENDED = 1;
    private static final int TRACK_WENT_TO_NEXT = 2;
    private static final int RELEASE_WAKELOCK = 3;
    private static final int SERVER_DIED = 4;
    private static final int FOCUSCHANGE = 5;
    private static final int FADEDOWN = 6;
    private static final int FADEUP = 7;
    private static final int IDLE_DELAY = 5 * 60 * 1000;
    private static final long REWIND_INSTEAD_PREVIOUS_THRESHOLD = 3000;
    private static final String[] PROJECTION = new String[]{
            "audio._id AS _id", MediaStore.Audio.Media.ARTIST, MediaStore.Audio.Media.ALBUM,
            MediaStore.Audio.Media.TITLE, MediaStore.Audio.Media.DATA,
            MediaStore.Audio.Media.MIME_TYPE, MediaStore.Audio.Media.ALBUM_ID,
            MediaStore.Audio.Media.ARTIST_ID
    };
    private static final String[] ALBUM_PROJECTION = new String[]{
            MediaStore.Audio.Albums.ALBUM, MediaStore.Audio.Albums.ARTIST,
            MediaStore.Audio.Albums.LAST_YEAR
    };
    private static final String[] NOTIFICATION_PROJECTION = new String[]{
            "audio._id AS _id", AudioColumns.ALBUM_ID, AudioColumns.TITLE,
            AudioColumns.ARTIST, AudioColumns.DURATION
    };
    private static final Shuffler mShuffler = new Shuffler();
    private static final int NOTIFY_MODE_NONE = 0;
    private static final int NOTIFY_MODE_FOREGROUND = 1;
    private static final int NOTIFY_MODE_BACKGROUND = 2;
    private static final String[] PROJECTION_MATRIX = new String[]{
            "_id", MediaStore.Audio.Media.ARTIST, MediaStore.Audio.Media.ALBUM,
            MediaStore.Audio.Media.TITLE, MediaStore.Audio.Media.DATA,
            MediaStore.Audio.Media.MIME_TYPE, MediaStore.Audio.Media.ALBUM_ID,
            MediaStore.Audio.Media.ARTIST_ID
    };
    private static LinkedList<Integer> mHistory = new LinkedList<>();
    private final IBinder mBinder = new ServiceStub(this);
    private MultiPlayer mPlayer;
    private String mFileToPlay;
    private WakeLock mWakeLock;
    private AlarmManager mAlarmManager;
    private PendingIntent mShutdownIntent;
    private boolean mShutdownScheduled;
    private NotificationManagerCompat mNotificationManager;
    private Cursor mCursor;
    private Cursor mAlbumCursor;
    private AudioManager mAudioManager;
    private SharedPreferences mPreferences;
    private boolean mServiceInUse = false;
    private boolean mIsSupposedToBePlaying = false;
    private long mLastPlayedTime;
    private int mNotifyMode = NOTIFY_MODE_NONE;
    private long mNotificationPostTime = 0;
    private boolean mQueueIsSaveable = true;
    private boolean mPausedByTransientLossOfFocus = false;

    private MediaSessionCompat mSession;
    @SuppressWarnings("deprecation")
    private RemoteControlClient mRemoteControlClient;

    private ComponentName mMediaButtonReceiverComponent;

    private int mCardId;

    private int mPlayPos = -1;

    private int mNextPlayPos = -1;

    private int mOpenFailedCounter = 0;
    private int mMediaMountedCount = 0;

    private int mShuffleMode = SHUFFLE_NONE;

    private int mRepeatMode = REPEAT_NONE;

    private int mServiceStartId = -1;

    private ArrayList<MusicPlaybackTrack> mPlaylist = new ArrayList<MusicPlaybackTrack>(100);

    private long[] mAutoShuffleList = null;

    private MusicPlayerHandler mPlayerHandler;
    private final OnAudioFocusChangeListener mAudioFocusListener = new OnAudioFocusChangeListener() {

        @Override
        public void onAudioFocusChange(final int focusChange) {
            mPlayerHandler.obtainMessage(FOCUSCHANGE, focusChange, 0).sendToTarget();
        }
    };
    private HandlerThread mHandlerThread;
    private BroadcastReceiver mUnmountReceiver = null;
    private MusicPlaybackState mPlaybackStateStore;
    private boolean mShowAlbumArtOnLockscreen;
    private boolean mActivateXTrackSelector;
    private SongPlayCount mSongPlayCount;
    private RecentStore mRecentStore;
    private final BroadcastReceiver mIntentReceiver = new BroadcastReceiver() {

        @Override
        public void onReceive(final Context context, final Intent intent) {
            final String command = intent.getStringExtra(CMDNAME);


            handleCommandIntent(intent);

        }
    };
    private ContentObserver mMediaStoreObserver;

    @Override
    public IBinder onBind(final Intent intent) {
        if (D) Log.d(TAG, "Service bound, intent = " + intent);
        cancelShutdown();
        mServiceInUse = true;
        return mBinder;
    }

    @Override
    public boolean onUnbind(final Intent intent) {
        if (D) Log.d(TAG, "Service unbound");
        mServiceInUse = false;
        saveQueue(true);

        if (mIsSupposedToBePlaying || mPausedByTransientLossOfFocus) {

            return true;

        } else if (mPlaylist.size() > 0 || mPlayerHandler.hasMessages(TRACK_ENDED)) {
            scheduleDelayedShutdown();
            return true;
        }
        stopSelf(mServiceStartId);

        return true;
    }

    @Override
    public void onRebind(final Intent intent) {
        cancelShutdown();
```

```java
        mServiceInUse = true;
    }

    @Override
    public void onCreate() {
        if (D) Log.d(TAG, "Creating service");
        super.onCreate();

        mNotificationManager = NotificationManagerCompat.from(this);
        createNotificationChannel();

        // gets a pointer to the playback state store
        mPlaybackStateStore = MusicPlaybackState.getInstance(this);
        mSongPlayCount = SongPlayCount.getInstance(this);
        mRecentStore = RecentStore.getInstance(this);


        mHandlerThread = new HandlerThread("MusicPlayerHandler",
                android.os.Process.THREAD_PRIORITY_BACKGROUND);
        mHandlerThread.start();


        mPlayerHandler = new MusicPlayerHandler(this, mHandlerThread.getLooper());


        mAudioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        mMediaButtonReceiverComponent = new ComponentName(getPackageName(),
                MediaButtonIntentReceiver.class.getName());
        mAudioManager.registerMediaButtonEventReceiver(mMediaButtonReceiverComponent);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
            setUpMediaSession();
        else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH)
            setUpRemoteControlClient();

        mPreferences = getSharedPreferences("Service", 0);
        mCardId = getCardId();

        registerExternalStorageListener();

        mPlayer = new MultiPlayer(this);
        mPlayer.setHandler(mPlayerHandler);

        // Initialize the intent filter and each action
        final IntentFilter filter = new IntentFilter();
        filter.addAction(SERVICECMD);
        filter.addAction(TOGGLEPAUSE_ACTION);
        filter.addAction(PAUSE_ACTION);
        filter.addAction(STOP_ACTION);
        filter.addAction(NEXT_ACTION);
        filter.addAction(PREVIOUS_ACTION);
        filter.addAction(PREVIOUS_FORCE_ACTION);
        filter.addAction(REPEAT_ACTION);
        filter.addAction(SHUFFLE_ACTION);
        filter.addAction(AudioManager.ACTION_AUDIO_BECOMING_NOISY);
        filter.addAction(Intent.ACTION_SCREEN_ON);
        // Attach the broadcast listener
        registerReceiver(mIntentReceiver, filter);

        mMediaStoreObserver = new MediaStoreObserver(mPlayerHandler);
        getContentResolver().registerContentObserver(
                MediaStore.Audio.Media.INTERNAL_CONTENT_URI, true, mMediaStoreObserver);
        getContentResolver().registerContentObserver(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, true, mMediaStoreObserver);

        // Initialize the wake lock
        final PowerManager powerManager = (PowerManager) getSystemService(Context.POWER_SERVICE);
        mWakeLock = powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, getClass().getName());
        mWakeLock.setReferenceCounted(false);
        final Intent shutdownIntent = new Intent(this, MusicService.class);
        shutdownIntent.setAction(SHUTDOWN);

        mAlarmManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
        mShutdownIntent = PendingIntent.getService(this, 0, shutdownIntent, 0);

        scheduleDelayedShutdown();

        reloadQueueAfterPermissionCheck();
        notifyChange(QUEUE_CHANGED);
        notifyChange(META_CHANGED);
        //Try to push LastFMCache
        if (LastfmUserSession.getSession(this) != null) {
            LastFmClient.getInstance(this).Scrobble(null);
        }
        PreferencesUtility pref = PreferencesUtility.getInstance(this);
        mShowAlbumArtOnLockscreen = pref.getSetAlbumartLockscreen();
        mActivateXTrackSelector = pref.getXPosedTrackselectorEnabled();
    }

    @SuppressWarnings("deprecation")
    @TargetApi(Build.VERSION_CODES.ICE_CREAM_SANDWICH)
    private void setUpRemoteControlClient() {
        //Legacy for ICS
        if (mRemoteControlClient == null) {
            Intent mediaButtonIntent = new Intent(Intent.ACTION_MEDIA_BUTTON);
            mediaButtonIntent.setComponent(mMediaButtonReceiverComponent);
            PendingIntent mediaPendingIntent = PendingIntent.getBroadcast(this, 0, mediaButtonIntent, 0);

            // create and register the remote control client
            mRemoteControlClient = new RemoteControlClient(mediaPendingIntent);
            mAudioManager.registerRemoteControlClient(mRemoteControlClient);
        }

        mRemoteControlClient.setTransportControlFlags(
                RemoteControlClient.FLAG_KEY_MEDIA_PLAY |
                        RemoteControlClient.FLAG_KEY_MEDIA_PAUSE |
                        RemoteControlClient.FLAG_KEY_MEDIA_PREVIOUS |
                        RemoteControlClient.FLAG_KEY_MEDIA_NEXT |
                        RemoteControlClient.FLAG_KEY_MEDIA_STOP);
    }

    private void setUpMediaSession() {
        mSession = new MediaSessionCompat(this, "Timber");
        mSession.setCallback(new MediaSessionCompat.Callback() {
            @Override
            public void onPause() {
                pause();
                mPausedByTransientLossOfFocus = false;
            }

            @Override
            public void onPlay() {
                play();
            }

            @Override
            public void onSeekTo(long pos) {
                seek(pos);
            }

            @Override
            public void onSkipToNext() {
                gotoNext(true);
            }

            @Override
            public void onSkipToPrevious() {
                prev(false);
```

```java
            }

            @Override
            public void onStop() {
                pause();
                mPausedByTransientLossOfFocus = false;
                seek(0);
                releaseServiceUiAndStop();
            }
        });
        mSession.setFlags(MediaSessionCompat.FLAG_HANDLES_TRANSPORT_CONTROLS
                         | MediaSessionCompat.FLAG_HANDLES_MEDIA_BUTTONS);
    }

    @Override
    public void onDestroy() {
        if (D) Log.d(TAG, "Destroying service");
        super.onDestroy();
        //Try to push LastFMCache
        if (LastfmUserSession.getSession(this).isLogedin()) {
            LastFmClient.getInstance(this).Scrobble(null);
        }
        // Remove any sound effects
        final Intent audioEffectsIntent = new Intent(
                AudioEffect.ACTION_CLOSE_AUDIO_EFFECT_CONTROL_SESSION);
        audioEffectsIntent.putExtra(AudioEffect.EXTRA_AUDIO_SESSION, getAudioSessionId());
        audioEffectsIntent.putExtra(AudioEffect.EXTRA_PACKAGE_NAME, getPackageName());
        sendBroadcast(audioEffectsIntent);


        mAlarmManager.cancel(mShutdownIntent);

        mPlayerHandler.removeCallbacksAndMessages(null);

        if (TimberUtils.isJellyBeanMR2())
            mHandlerThread.quitSafely();
        else mHandlerThread.quit();

        mPlayer.release();
        mPlayer = null;

        mAudioManager.abandonAudioFocus(mAudioFocusListener);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
            mSession.release();

        getContentResolver().unregisterContentObserver(mMediaStoreObserver);

        closeCursor();

        unregisterReceiver(mIntentReceiver);
        if (mUnmountReceiver != null) {
            unregisterReceiver(mUnmountReceiver);
            mUnmountReceiver = null;
        }

        mWakeLock.release();
    }

    @Override
    public int onStartCommand(final Intent intent, final int flags, final int startId) {
        if (D) Log.d(TAG, "Got new intent " + intent + ", startId = " + startId);
        mServiceStartId = startId;

        if (intent != null) {
            final String action = intent.getAction();

            if (SHUTDOWN.equals(action)) {
                mShutdownScheduled = false;
                releaseServiceUiAndStop();
                return START_NOT_STICKY;
            }

            handleCommandIntent(intent);
        }

        scheduleDelayedShutdown();

        if (intent != null && intent.getBooleanExtra(FROM_MEDIA_BUTTON, false)) {
            MediaButtonIntentReceiver.completeWakefulIntent(intent);
        }

        return START_NOT_STICKY; //no sense to use START_STICKY with using startForeground
    }

    void scrobble() {
        if (LastfmUserSession.getSession(this).isLogedin()) {
            Log.d("Scrobble", "to LastFM");
            String trackname = getTrackName();
            if (trackname != null)
                LastFmClient.getInstance(this).Scrobble(new ScrobbleQuery(getArtistName(), trackname, System.currentTimeMill
        }
    }

    private void releaseServiceUiAndStop() {
        if (isPlaying()
                || mPausedByTransientLossOfFocus
                || mPlayerHandler.hasMessages(TRACK_ENDED)) {
            return;
        }

        if (D) Log.d(TAG, "Nothing is playing anymore, releasing notification");
        cancelNotification();
        mAudioManager.abandonAudioFocus(mAudioFocusListener);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
            mSession.setActive(false);

        if (!mServiceInUse) {
            saveQueue(true);
            stopSelf(mServiceStartId);
        }
    }

    private void handleCommandIntent(Intent intent) {
        final String action = intent.getAction();
        final String command = SERVICECMD.equals(action) ? intent.getStringExtra(CMDNAME) : null;

        if (D) Log.d(TAG, "handleCommandIntent: action = " + action + ", command = " + command);

        if (NotificationHelper.checkIntent(intent)) {
            goToPosition(mPlayPos + NotificationHelper.getPosition(intent));
            return;
        }

        if (CMDNEXT.equals(command) || NEXT_ACTION.equals(action)) {
            gotoNext(true);
        } else if (CMDPREVIOUS.equals(command) || PREVIOUS_ACTION.equals(action)
                || PREVIOUS_FORCE_ACTION.equals(action)) {
            prev(PREVIOUS_FORCE_ACTION.equals(action));
        } else if (CMDTOGGLEPAUSE.equals(command) || TOGGLEPAUSE_ACTION.equals(action)) {
            if (isPlaying()) {
                pause();
                mPausedByTransientLossOfFocus = false;
            } else {
                play();
            }
        } else if (CMDPAUSE.equals(command) || PAUSE_ACTION.equals(action)) {
            pause();
            mPausedByTransientLossOfFocus = false;
        } else if (CMDPLAY.equals(command)) {
            play();
        } else if (CMDSTOP.equals(command) || STOP_ACTION.equals(action)) {
            pause();
            mPausedByTransientLossOfFocus = false;
            seek(0);
            releaseServiceUiAndStop();
        } else if (REPEAT_ACTION.equals(action)) {
            cycleRepeat();
        } else if (SHUFFLE_ACTION.equals(action)) {
            cycleShuffle();
        } else if (UPDATE_PREFERENCES.equals(action)) {
            onPreferencesUpdate(intent.getExtras());
        }
        else if (AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(action)) {
            if (PreferencesUtility.getInstance(getApplicationContext()).pauseEnabledOnDetach()) {
                pause();
            }
        }
    }

    private void onPreferencesUpdate(Bundle extras) {
        mShowAlbumArtOnLockscreen = extras.getBoolean("lockscreen", mShowAlbumArtOnLockscreen);
        mActivateXTrackSelector = extras.getBoolean("xtrack",mActivateXTrackSelector);
        LastfmUserSession session = LastfmUserSession.getSession(this);
        session.mToken = extras.getString("lf_token", session.mToken);
        session.mUsername = extras.getString("lf_user", session.mUsername);
        if ("logout".equals(session.mToken)) {
            session.mToken = null;
            session.mUsername = null;
        }
        notifyChange(META_CHANGED);

    }

    private void updateNotification() {
        final int newNotifyMode;
        if (isPlaying()) {
            newNotifyMode = NOTIFY_MODE_FOREGROUND;
        } else if (recentlyPlayed()) {
            newNotifyMode = NOTIFY_MODE_BACKGROUND;
        } else {
            newNotifyMode = NOTIFY_MODE_NONE;
        }

        int notificationId = hashCode();
        if (mNotifyMode != newNotifyMode) {
            if (mNotifyMode == NOTIFY_MODE_FOREGROUND) {
                if (TimberUtils.isLollipop())
                    stopForeground(newNotifyMode == NOTIFY_MODE_NONE);
                else
                    stopForeground(newNotifyMode == NOTIFY_MODE_NONE || newNotifyMode == NOTIFY_MODE_BACKGROUND);
            } else if (newNotifyMode == NOTIFY_MODE_NONE) {
                mNotificationManager.cancel(notificationId);
                mNotificationPostTime = 0;
            }
        }

        if (newNotifyMode == NOTIFY_MODE_FOREGROUND) {
            startForeground(notificationId, buildNotification());
        } else if (newNotifyMode == NOTIFY_MODE_BACKGROUND) {
            mNotificationManager.notify(notificationId, buildNotification());
        }

        mNotifyMode = newNotifyMode;
    }

    private void cancelNotification() {
        stopForeground(true);
        mNotificationManager.cancel(hashCode());
        mNotificationPostTime = 0;
        mNotifyMode = NOTIFY_MODE_NONE;
    }

    private int getCardId() {
        if (TimberUtils.isMarshmallow()) {
            if (Nammu.checkPermission(Manifest.permission.READ_EXTERNAL_STORAGE)) {
                return getmCardId();
            } else return 0;
        } else {
            return getmCardId();
        }
    }

    private int getmCardId() {
        final ContentResolver resolver = getContentResolver();
        Cursor cursor = resolver.query(Uri.parse("content://media/external/fs_id"), null, null,
                null, null);
        int mCardId = -1;
        if (cursor != null && cursor.moveToFirst()) {
            mCardId = cursor.getInt(0);
            cursor.close();
            cursor = null;
        }
        return mCardId;
    }

    public void closeExternalStorageFiles(final String storagePath) {
        stop(true);
        notifyChange(QUEUE_CHANGED);
        notifyChange(META_CHANGED);
    }

    public void registerExternalStorageListener() {
        if (mUnmountReceiver == null) {
            mUnmountReceiver = new BroadcastReceiver() {


                @Override
                public void onReceive(final Context context, final Intent intent) {
                    final String action = intent.getAction();
                    if (action.equals(Intent.ACTION_MEDIA_EJECT)) {
                        saveQueue(true);
                        mQueueIsSaveable = false;
                        closeExternalStorageFiles(intent.getData().getPath());
                    } else if (action.equals(Intent.ACTION_MEDIA_MOUNTED)) {
                        mMediaMountedCount++;
                        mCardId = getCardId();
                        reloadQueueAfterPermissionCheck();
                        mQueueIsSaveable = true;
                        notifyChange(QUEUE_CHANGED);
                        notifyChange(META_CHANGED);
                    }
                }
            };
            final IntentFilter filter = new IntentFilter();
            filter.addAction(Intent.ACTION_MEDIA_EJECT);
            filter.addAction(Intent.ACTION_MEDIA_MOUNTED);
            filter.addDataScheme("file");
            registerReceiver(mUnmountReceiver, filter);
        }
    }

    private void scheduleDelayedShutdown() {
        if (D) Log.v(TAG, "Scheduling shutdown in " + IDLE_DELAY + " ms");
        mAlarmManager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                SystemClock.elapsedRealtime() + IDLE_DELAY, mShutdownIntent);
        mShutdownScheduled = true;
    }

    private void cancelShutdown() {
        if (D) Log.d(TAG, "Cancelling delayed shutdown, scheduled = " + mShutdownScheduled);
        if (mShutdownScheduled) {
            mAlarmManager.cancel(mShutdownIntent);
            mShutdownScheduled = false;
        }
    }

    private void stop(final boolean goToIdle) {
        if (D) Log.d(TAG, "Stopping playback, goToIdle = " + goToIdle);
        long duration = this.duration();
        long position = this.position();
        if (duration > 30000 && (position >= duration / 2) || position > 240000) {
            scrobble();
        }

        if (mPlayer.isInitialized()) {
            mPlayer.stop();
        }
        mFileToPlay = null;
        closeCursor();
        if (goToIdle) {
            setIsSupposedToBePlaying(false, false);
        } else {
            if (TimberUtils.isLollipop())
                stopForeground(false);
            else stopForeground(true);
        }
    }

    private int removeTracksInternal(int first, int last) {
        synchronized (this) {
            if (last < first) {
                return 0;
            } else if (first < 0) {
                first = 0;
            } else if (last >= mPlaylist.size()) {
                last = mPlaylist.size() - 1;
            }

            boolean gotonext = false;
            if (first <= mPlayPos && mPlayPos <= last) {
                mPlayPos = first;
                gotonext = true;
            } else if (mPlayPos > last) {
                mPlayPos -= last - first + 1;
            }
            final int numToRemove = last - first + 1;

            if (first == 0 && last == mPlaylist.size() - 1) {
                mPlayPos = -1;
                mNextPlayPos = -1;
                mPlaylist.clear();
                mHistory.clear();
            } else {
                for (int i = 0; i < numToRemove; i++) {
                    mPlaylist.remove(first);
                }

                ListIterator<Integer> positionIterator = mHistory.listIterator();
                while (positionIterator.hasNext()) {
                    int pos = positionIterator.next();
                    if (pos >= first && pos <= last) {
                        positionIterator.remove();
                    } else if (pos > last) {
                        positionIterator.set(pos - numToRemove);
                    }
                }
            }
            if (gotonext) {
                if (mPlaylist.size() == 0) {
                    stop(true);
                    mPlayPos = -1;
                    closeCursor();
                } else {
                    if (mShuffleMode != SHUFFLE_NONE) {
                        mPlayPos = getNextPosition(true);
                    } else if (mPlayPos >= mPlaylist.size()) {
                        mPlayPos = 0;
                    }
                    final boolean wasPlaying = isPlaying();
                    stop(false);
                    openCurrentAndNext();
                    if (wasPlaying) {
                        play();
                    }
                }
                notifyChange(META_CHANGED);
            }
            return last - first + 1;
        }
    }

    private void addToPlayList(final long[] list, int position, long sourceId, TimberUtils.IdType sourceType) {
        final int addlen = list.length;
        if (position < 0) {
            mPlaylist.clear();
            position = 0;
        }

        mPlaylist.ensureCapacity(mPlaylist.size() + addlen);
        if (position > mPlaylist.size()) {
            position = mPlaylist.size();
        }

        final ArrayList<MusicPlaybackTrack> arrayList = new ArrayList<MusicPlaybackTrack>(addlen);
        for (int i = 0; i < list.length; i++) {
            arrayList.add(new MusicPlaybackTrack(list[i], sourceId, sourceType, i));
        }

        mPlaylist.addAll(position, arrayList);

        if (mPlaylist.size() == 0) {
            closeCursor();
            notifyChange(META_CHANGED);
        }
    }

    private void updateCursor(final long trackId) {
        updateCursor("_id=" + trackId, null);
    }

    private void updateCursor(final String selection, final String[] selectionArgs) {
        synchronized (this) {
            closeCursor();
            mCursor = openCursorAndGoToFirst(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                    PROJECTION, selection, selectionArgs);
        }
        updateAlbumCursor();
    }

    private void updateCursor(final Uri uri) {
        synchronized (this) {
            closeCursor();
            mCursor = openCursorAndGoToFirst(uri, PROJECTION, null, null);
        }
        updateAlbumCursor();
    }

    private void updateAlbumCursor() {
        long albumId = getAlbumId();
        if (albumId >= 0) {
            mAlbumCursor = openCursorAndGoToFirst(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI,
                    ALBUM_PROJECTION, "_id=" + albumId, null);
        } else {
            mAlbumCursor = null;
        }
    }

    private Cursor openCursorAndGoToFirst(Uri uri, String[] projection,
                                          String selection, String[] selectionArgs) {
        Cursor c = getContentResolver().query(uri, projection,
                selection, selectionArgs, null);
        if (c == null) {
            return null;
        }
        if (!c.moveToFirst()) {
            c.close();
            return null;
        }
        return c;
    }

    private synchronized void closeCursor() {
        if (mCursor != null) {
            mCursor.close();
            mCursor = null;
        }
        if (mAlbumCursor != null) {
            mAlbumCursor.close();
            mAlbumCursor = null;
        }
    }

    private void openCurrentAndNext() {
        openCurrentAndMaybeNext(true);
    }

    private void openCurrentAndMaybeNext(final boolean openNext) {
        synchronized (this) {
            closeCursor();

            if (mPlaylist.size() == 0) {
                return;
            }
            stop(false);

            boolean shutdown = false;

            updateCursor(mPlaylist.get(mPlayPos).mId);
            while (true) {
                if (mCursor != null
                        && openFile(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI + "/"
                        + mCursor.getLong(IDCOLIDX))) {
                    break;
                }

                closeCursor();
                if (mOpenFailedCounter++ < 10 && mPlaylist.size() > 1) {
                    final int pos = getNextPosition(false);
                    if (pos < 0) {
                        shutdown = true;
                        break;
                    }
                    mPlayPos = pos;
                    stop(false);
                    mPlayPos = pos;
                    updateCursor(mPlaylist.get(mPlayPos).mId);
                } else {
                    mOpenFailedCounter = 0;
                    Log.w(TAG, "Failed to open file for playback");
                    shutdown = true;
                    break;
                }
            }

            if (shutdown) {
                scheduleDelayedShutdown();
                if (mIsSupposedToBePlaying) {
                    mIsSupposedToBePlaying = false;
                    notifyChange(PLAYSTATE_CHANGED);
                }
            } else if (openNext) {
                setNextTrack();
            }
        }
    }

    private void sendErrorMessage(final String trackName) {
        final Intent i = new Intent(TRACK_ERROR);
        i.putExtra(TrackErrorExtra.TRACK_NAME, trackName);
        sendBroadcast(i);
    }

    private int getNextPosition(final boolean force) {
        if (mPlaylist == null || mPlaylist.isEmpty()) {
            return -1;
        }
        if (!force && mRepeatMode == REPEAT_CURRENT) {
            if (mPlayPos < 0) {
                return 0;
            }
            return mPlayPos;
        } else if (mShuffleMode == SHUFFLE_NORMAL) {
            final int numTracks = mPlaylist.size();


            final int[] trackNumPlays = new int[numTracks];
            for (int i = 0; i < numTracks; i++) {
                trackNumPlays[i] = 0;
            }


            final int numHistory = mHistory.size();
            for (int i = 0; i < numHistory; i++) {
                final int idx = mHistory.get(i).intValue();
                if (idx >= 0 && idx < numTracks) {
                    trackNumPlays[idx]++;
                }
            }

            if (mPlayPos >= 0 && mPlayPos < numTracks) {
                trackNumPlays[mPlayPos]++;
            }

            int minNumPlays = Integer.MAX_VALUE;
            int numTracksWithMinNumPlays = 0;
            for (int i = 0; i < trackNumPlays.length; i++) {
                if (trackNumPlays[i] < minNumPlays) {
                    minNumPlays = trackNumPlays[i];
                    numTracksWithMinNumPlays = 1;
                } else if (trackNumPlays[i] == minNumPlays) {
                    numTracksWithMinNumPlays++;
                }
            }


            if (minNumPlays > 0 && numTracksWithMinNumPlays == numTracks
                    && mRepeatMode != REPEAT_ALL && !force) {
                return -1;
            }
            int skip = mShuffler.nextInt(numTracksWithMinNumPlays);
            for (int i = 0; i < trackNumPlays.length; i++) {
                if (trackNumPlays[i] == minNumPlays) {
                    if (skip == 0) {
                        return i;
                    } else {
                        skip--;
                    }
                }
            }

            if (D)
                Log.e(TAG, "Getting the next position resulted did not get a result when it should have");
            return -1;
        } else if (mShuffleMode == SHUFFLE_AUTO) {
            doAutoShuffleUpdate();
            return mPlayPos + 1;
        } else {
            if (mPlayPos >= mPlaylist.size() - 1) {
                if (mRepeatMode == REPEAT_NONE && !force) {
                    return -1;
                } else if (mRepeatMode == REPEAT_ALL || force) {
                    return 0;
                }
                return -1;
            } else {
                return mPlayPos + 1;
            }
        }
    }

    private void setNextTrack() {
        setNextTrack(getNextPosition(false));
    }

    private void setNextTrack(int position) {
        mNextPlayPos = position;
        if (D) Log.d(TAG, "setNextTrack: next play position = " + mNextPlayPos);
        if (mNextPlayPos >= 0 && mPlaylist != null && mNextPlayPos < mPlaylist.size()) {
            final long id = mPlaylist.get(mNextPlayPos).mId;
            mPlayer.setNextDataSource(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI + "/" + id);
        } else {
            mPlayer.setNextDataSource(null);
        }
    }

    private boolean makeAutoShuffleList() {
        Cursor cursor = null;
        try {
            cursor = getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                    new String[]{
                            MediaStore.Audio.Media._ID
                    }, MediaStore.Audio.Media.IS_MUSIC + "=1", null, null);
            if (cursor == null || cursor.getCount() == 0) {
                return false;
            }
            final int len = cursor.getCount();
            final long[] list = new long[len];
            for (int i = 0; i < len; i++) {
                cursor.moveToNext();
                list[i] = cursor.getLong(0);
            }
            mAutoShuffleList = list;
            return true;
        } catch (final RuntimeException e) {
        } finally {
            if (cursor != null) {
                cursor.close();
                cursor = null;
```

```java
            }
        }
        return false;
    }

    private void doAutoShuffleUpdate() {
        boolean notify = false;
        if (mPlayPos > 10) {
            removeTracks(0, mPlayPos - 9);
            notify = true;
        }
        final int toAdd = 7 - (mPlaylist.size() - (mPlayPos < 0 ? -1 : mPlayPos));
        for (int i = 0; i < toAdd; i++) {
            int lookback = mHistory.size();
            int idx = -1;
            while (true) {
                idx = mShuffler.nextInt(mAutoShuffleList.length);
                if (!wasRecentlyUsed(idx, lookback)) {
                    break;
                }
                lookback /= 2;
            }
            mHistory.add(idx);
            if (mHistory.size() > MAX_HISTORY_SIZE) {
                mHistory.remove(0);
            }
            mPlaylist.add(new MusicPlaybackTrack(mAutoShuffleList[idx], -1, TimberUtils.IdType.NA, -1));
            notify = true;
        }
        if (notify) {
            notifyChange(QUEUE_CHANGED);
        }
    }

    private boolean wasRecentlyUsed(final int idx, int lookbacksize) {
        if (lookbacksize == 0) {
            return false;
        }
        final int histsize = mHistory.size();
        if (histsize < lookbacksize) {
            lookbacksize = histsize;
        }
        final int maxidx = histsize - 1;
        for (int i = 0; i < lookbacksize; i++) {
            final long entry = mHistory.get(maxidx - i);
            if (entry == idx) {
                return true;
            }
        }
        return false;
    }

    private void notifyChange(final String what) {
        if (D) Log.d(TAG, "notifyChange: what = " + what);

        // Update the lockscreen controls
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
            updateMediaSession(what);
        else if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH)
            updateRemoteControlClient(what);

        if (what.equals(POSITION_CHANGED)) {
            return;
        }

        final Intent intent = new Intent(what);
        intent.putExtra("id", getAudioId());
        intent.putExtra("artist", getArtistName());
        intent.putExtra("album", getAlbumName());
        intent.putExtra("albumid", getAlbumId());
        intent.putExtra("track", getTrackName());
        intent.putExtra("playing", isPlaying());

        sendStickyBroadcast(intent);

        final Intent musicIntent = new Intent(intent);
        musicIntent.setAction(what.replace(TIMBER_PACKAGE_NAME, MUSIC_PACKAGE_NAME));
        sendStickyBroadcast(musicIntent);

        if (what.equals(META_CHANGED)) {

            mRecentStore.addSongId(getAudioId());
            mSongPlayCount.bumpSongCount(getAudioId());

        } else if (what.equals(QUEUE_CHANGED)) {
            saveQueue(true);
            if (isPlaying()) {

                if (mNextPlayPos >= 0 && mNextPlayPos < mPlaylist.size()
                        && getShuffleMode() != SHUFFLE_NONE) {
                    setNextTrack(mNextPlayPos);
                } else {
                    setNextTrack();
                }
            }
        } else {
            saveQueue(false);
        }

        if (what.equals(PLAYSTATE_CHANGED)) {
            updateNotification();
        }

    }

    @SuppressWarnings("deprecation")
    @TargetApi(Build.VERSION_CODES.ICE_CREAM_SANDWICH)
    private void updateRemoteControlClient(final String what) {
        //Legacy for ICS
        if (mRemoteControlClient != null) {
            int playState = mIsSupposedToBePlaying
                    ? RemoteControlClient.PLAYSTATE_PLAYING
                    : RemoteControlClient.PLAYSTATE_PAUSED;
            if (what.equals(META_CHANGED) || what.equals(QUEUE_CHANGED)) {
                Bitmap albumArt = null;
                if (mShowAlbumArtOnLockscreen) {
                    albumArt = ImageLoader.getInstance().loadImageSync(TimberUtils.getAlbumArtUri(getAlbumId()).toString());
                    if (albumArt != null) {

                        Bitmap.Config config = albumArt.getConfig();
                        if (config == null) {
                            config = Bitmap.Config.ARGB_8888;
                        }
                        albumArt = albumArt.copy(config, false);
                    }
                }

                RemoteControlClient.MetadataEditor editor = mRemoteControlClient.editMetadata(true);
                editor.putString(MediaMetadataRetriever.METADATA_KEY_ALBUM, getAlbumName());
                editor.putString(MediaMetadataRetriever.METADATA_KEY_ARTIST, getArtistName());
                editor.putString(MediaMetadataRetriever.METADATA_KEY_TITLE, getTrackName());
                editor.putLong(MediaMetadataRetriever.METADATA_KEY_DURATION, duration());
                editor.putBitmap(MediaMetadataEditor.BITMAP_KEY_ARTWORK, albumArt);
                editor.apply();

            }
            mRemoteControlClient.setPlaybackState(playState);
        }
    }
```

```java
    private void updateMediaSession(final String what) {
        int playState = mIsSupposedToBePlaying
                ? PlaybackStateCompat.STATE_PLAYING
                : PlaybackStateCompat.STATE_PAUSED;

        if (what.equals(PLAYSTATE_CHANGED) || what.equals(POSITION_CHANGED)) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                mSession.setPlaybackState(new PlaybackStateCompat.Builder()
                        .setState(playState, position(), 1.0f)
                        .setActions(PlaybackStateCompat.ACTION_PLAY | PlaybackStateCompat.ACTION_PAUSE | PlaybackStateCompat
                                PlaybackStateCompat.ACTION_SKIP_TO_NEXT | PlaybackStateCompat.ACTION_SKIP_TO_PREVIOUS)
                        .build());
            }
        } else if (what.equals(META_CHANGED) || what.equals(QUEUE_CHANGED)) {
            Bitmap albumArt = null;
            if (mShowAlbumArtOnLockscreen) {
                albumArt = ImageLoader.getInstance().loadImageSync(TimberUtils.getAlbumArtUri(getAlbumId()).toString());
                if (albumArt != null) {

                    Bitmap.Config config = albumArt.getConfig();
                    if (config == null) {
                        config = Bitmap.Config.ARGB_8888;
                    }
                    albumArt = albumArt.copy(config, false);
                }
            }
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                mSession.setMetadata(new MediaMetadataCompat.Builder()
                        .putString(MediaMetadataCompat.METADATA_KEY_ARTIST, getArtistName())
                        .putString(MediaMetadataCompat.METADATA_KEY_ALBUM_ARTIST, getAlbumArtistName())
                        .putString(MediaMetadataCompat.METADATA_KEY_ALBUM, getAlbumName())
                        .putString(MediaMetadataCompat.METADATA_KEY_TITLE, getTrackName())
                        .putLong(MediaMetadataCompat.METADATA_KEY_DURATION, duration())
                        .putLong(MediaMetadataCompat.METADATA_KEY_TRACK_NUMBER, getQueuePosition() + 1)
                        .putLong(MediaMetadataCompat.METADATA_KEY_NUM_TRACKS, getQueue().length)
                        .putString(MediaMetadataCompat.METADATA_KEY_GENRE, getGenreName())
                        .putBitmap(MediaMetadataCompat.METADATA_KEY_ALBUM_ART, albumArt)
                        .build());

                mSession.setPlaybackState(new PlaybackStateCompat.Builder()
                        .setState(playState, position(), 1.0f)
                        .setActions(PlaybackStateCompat.ACTION_PLAY | PlaybackStateCompat.ACTION_PAUSE | PlaybackStateCompat
                                PlaybackStateCompat.ACTION_SKIP_TO_NEXT | PlaybackStateCompat.ACTION_SKIP_TO_PREVIOUS)
                        .build());
            }
        }
    }

    private void createNotificationChannel() {
        if (TimberUtils.isOreo()) {
            CharSequence name = "Timber";
            int importance = NotificationManager.IMPORTANCE_LOW;
            NotificationManager manager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
            NotificationChannel mChannel = new NotificationChannel(CHANNEL_ID, name, importance);
            manager.createNotificationChannel(mChannel);
        }
    }

    private Notification buildNotification() {
        final String albumName = getAlbumName();
        final String artistName = getArtistName();
        final boolean isPlaying = isPlaying();
        String text = TextUtils.isEmpty(albumName)
                ? artistName : artistName + " - " + albumName;

        int playButtonResId = isPlaying
                ? R.drawable.ic_pause_white_36dp : R.drawable.ic_play_white_36dp;

        Intent nowPlayingIntent = NavigationUtils.getNowPlayingIntent(this);
        PendingIntent clickIntent = PendingIntent.getActivity(this, 0, nowPlayingIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        Bitmap artwork;
        artwork = ImageLoader.getInstance().loadImageSync(TimberUtils.getAlbumArtUri(getAlbumId()).toString());

        if (artwork == null) {
            artwork = ImageLoader.getInstance().loadImageSync("drawable://" + R.drawable.ic_empty_music2);
        }

        if (mNotificationPostTime == 0) {
            mNotificationPostTime = System.currentTimeMillis();
        }

        android.support.v4.app.NotificationCompat.Builder builder = new android.support.v4.app.NotificationCompat.Builder(th
                .setSmallIcon(R.drawable.ic_notification)
                .setLargeIcon(artwork)
                .setContentIntent(clickIntent)
                .setContentTitle(getTrackName())
                .setContentText(text)
                .setWhen(mNotificationPostTime)
                .addAction(R.drawable.ic_skip_previous_white_36dp,
                        "",
                        retrievePlaybackAction(PREVIOUS_ACTION))
                .addAction(playButtonResId, "",
                        retrievePlaybackAction(TOGGLEPAUSE_ACTION))
                .addAction(R.drawable.ic_skip_next_white_36dp,
                        "",
                        retrievePlaybackAction(NEXT_ACTION));

        if (TimberUtils.isJellyBeanMR1()) {
            builder.setShowWhen(false);
        }

        if (TimberUtils.isLollipop()) {
            builder.setVisibility(Notification.VISIBILITY_PUBLIC);
            android.support.v4.media.app.NotificationCompat.MediaStyle style = new android.support.v4.media.app.Notification
                    .setMediaSession(mSession.getSessionToken())
                    .setShowActionsInCompactView(0, 1, 2, 3);
            builder.setStyle(style);
        }
        if (artwork != null && TimberUtils.isLollipop()) {
            builder.setColor(Palette.from(artwork).generate().getVibrantColor(Color.parseColor("#403f4d")));
        }

        if (TimberUtils.isOreo()) {
            builder.setColorized(true);
        }

        Notification n = builder.build();

        if (mActivateXTrackSelector) {
            addXTrackSelector(n);
        }

        return n;
    }

    private void addXTrackSelector(Notification n) {
        if (NotificationHelper.isSupported(n)) {
            StringBuilder selection = new StringBuilder();
            StringBuilder order = new StringBuilder().append("CASE _id \n");
            for (int i = 0; i < mPlaylist.size(); i++) {
                selection.append("_id=").append(mPlaylist.get(i).mId).append(" OR ");
                order.append("WHEN ").append(mPlaylist.get(i).mId).append(" THEN ").append(i).append("\n");
            }
            order.append("END");
            Cursor c = getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, NOTIFICATION_PROJECTION, sele
            if (c != null && c.getCount() != 0) {
                c.moveToFirst();
                ArrayList<Bundle> list = new ArrayList<>();
                do {
```

```java
                    TrackItem t = new TrackItem()
                            .setArt(TimberUtils.getAlbumArtUri(c.getLong(c.getColumnIndexOrThrow(AudioColumns.ALBUM_ID))))
                            .setTitle(c.getString(c.getColumnIndexOrThrow(AudioColumns.TITLE)))
                            .setArtist(c.getString(c.getColumnIndexOrThrow(AudioColumns.ARTIST)))
                            .setDuration(TimberUtils.makeShortTimeString(this, c.getInt(c.getColumnIndexOrThrow(AudioColumns
                    list.add(t.get());
                } while (c.moveToNext());
                try {
                    NotificationHelper.insertToNotification(n, list, this, getQueuePosition());
                } catch (ModNotInstalledException e) {
                    e.printStackTrace();
                }
                c.close();
            }
        }
    }

    private final PendingIntent retrievePlaybackAction(final String action) {
        final ComponentName serviceName = new ComponentName(this, MusicService.class);
        Intent intent = new Intent(action);
        intent.setComponent(serviceName);

        return PendingIntent.getService(this, 0, intent, 0);
    }

    private void saveQueue(final boolean full) {
        if (!mQueueIsSaveable) {
            return;
        }

        final SharedPreferences.Editor editor = mPreferences.edit();
        if (full) {
            mPlaybackStateStore.saveState(mPlaylist,
                    mShuffleMode != SHUFFLE_NONE ? mHistory : null);
            editor.putInt("cardid", mCardId);
        }
        editor.putInt("curpos", mPlayPos);
        if (mPlayer.isInitialized()) {
            editor.putLong("seekpos", mPlayer.position());
        }
        editor.putInt("repeatmode", mRepeatMode);
        editor.putInt("shufflemode", mShuffleMode);
        editor.apply();
    }

    private void reloadQueueAfterPermissionCheck() {
        if (TimberUtils.isMarshmallow()) {
            if (Nammu.checkPermission(Manifest.permission.READ_EXTERNAL_STORAGE)) {
                reloadQueue();
            }
        } else {
            reloadQueue();
        }
    }

    private void reloadQueue() {
        int id = mCardId;
        if (mPreferences.contains("cardid")) {
            id = mPreferences.getInt("cardid", ~mCardId);
        }
        if (id == mCardId) {
            mPlaylist = mPlaybackStateStore.getQueue();
        }
        if (mPlaylist.size() > 0) {
            final int pos = mPreferences.getInt("curpos", 0);
            if (pos < 0 || pos >= mPlaylist.size()) {
                mPlaylist.clear();
                return;
            }
            mPlayPos = pos;
            updateCursor(mPlaylist.get(mPlayPos).mId);
            if (mCursor == null) {
                SystemClock.sleep(3000);
                updateCursor(mPlaylist.get(mPlayPos).mId);
            }
            synchronized (this) {
                closeCursor();
                mOpenFailedCounter = 20;
                openCurrentAndNext();
            }
            if (!mPlayer.isInitialized()) {
                mPlaylist.clear();
                return;
            }

            final long seekpos = mPreferences.getLong("seekpos", 0);
            seek(seekpos >= 0 && seekpos < duration() ? seekpos : 0);

            if (D) {
                Log.d(TAG, "restored queue, currently at position "
                        + position() + "/" + duration()
                        + " (requested " + seekpos + ")");
            }

            int repmode = mPreferences.getInt("repeatmode", REPEAT_NONE);
            if (repmode != REPEAT_ALL && repmode != REPEAT_CURRENT) {
                repmode = REPEAT_NONE;
            }
            mRepeatMode = repmode;

            int shufmode = mPreferences.getInt("shufflemode", SHUFFLE_NONE);
            if (shufmode != SHUFFLE_AUTO && shufmode != SHUFFLE_NORMAL) {
                shufmode = SHUFFLE_NONE;
            }
            if (shufmode != SHUFFLE_NONE) {
                mHistory = mPlaybackStateStore.getHistory(mPlaylist.size());
            }
            if (shufmode == SHUFFLE_AUTO) {
                if (!makeAutoShuffleList()) {
                    shufmode = SHUFFLE_NONE;
                }
            }
            mShuffleMode = shufmode;
        }
    }

    public boolean openFile(final String path) {
        if (D) Log.d(TAG, "openFile: path = " + path);
        synchronized (this) {
            if (path == null) {
                return false;
            }

            if (mCursor == null) {
                Uri uri = Uri.parse(path);
                boolean shouldAddToPlaylist = true;
                long id = -1;
                try {
                    id = Long.valueOf(uri.getLastPathSegment());
                } catch (NumberFormatException ex) {
                    // Ignore
                }

                if (id != -1 && path.startsWith(
                        MediaStore.Audio.Media.EXTERNAL_CONTENT_URI.toString())) {
                    updateCursor(uri);

                } else if (id != -1 && path.startsWith(
                        MediaStore.Files.getContentUri("external").toString())) {
                    updateCursor(id);
```

```java
            } else if (path.startsWith("content://downloads/")) {

                String mpUri = getValueForDownloadedFile(this, uri, "mediaprovider_uri");
                if (D) Log.i(TAG, "Downloaded file's MP uri : " + mpUri);
                if (!TextUtils.isEmpty(mpUri)) {
                    if (openFile(mpUri)) {
                        notifyChange(META_CHANGED);
                        return true;
                    } else {
                        return false;
                    }
                } else {
                    updateCursorForDownloadedFile(this, uri);
                    shouldAddToPlaylist = false;
                }

            } else {
                String where = MediaStore.Audio.Media.DATA + "=?";
                String[] selectionArgs = new String[]{path};
                updateCursor(where, selectionArgs);
            }
            try {
                if (mCursor != null && shouldAddToPlaylist) {
                    mPlaylist.clear();
                    mPlaylist.add(new MusicPlaybackTrack(
                            mCursor.getLong(IDCOLIDX), -1, TimberUtils.IdType.NA, -1));
                    notifyChange(QUEUE_CHANGED);
                    mPlayPos = 0;
                    mHistory.clear();
                }
            } catch (final UnsupportedOperationException ex) {
                // Ignore
            }
        }

        mFileToPlay = path;
        mPlayer.setDataSource(mFileToPlay);
        if (mPlayer.isInitialized()) {
            mOpenFailedCounter = 0;
            return true;
        }

        String trackName = getTrackName();
        if (TextUtils.isEmpty(trackName)) {
            trackName = path;
        }
        sendErrorMessage(trackName);

        stop(true);
        return false;
    }
}

private void updateCursorForDownloadedFile(Context context, Uri uri) {
    synchronized (this) {
        closeCursor();
        MatrixCursor cursor = new MatrixCursor(PROJECTION_MATRIX);
        String title = getValueForDownloadedFile(this, uri, "title");
        cursor.addRow(new Object[]{
                null,
                null,
                null,
                title,
                null,
                null,
                null,
                null
        });
        mCursor = cursor;
```

```java
            mCursor.moveToFirst();
        }
    }

    private String getValueForDownloadedFile(Context context, Uri uri, String column) {

        Cursor cursor = null;
        final String[] projection = {
                column
        };

        try {
            cursor = context.getContentResolver().query(uri, projection, null, null, null);
            if (cursor != null && cursor.moveToFirst()) {
                return cursor.getString(0);
            }
        } finally {
            if (cursor != null) {
                cursor.close();
            }
        }
        return null;
    }

    public int getAudioSessionId() {
        synchronized (this) {
            return mPlayer.getAudioSessionId();
        }
    }

    public int getMediaMountedCount() {
        return mMediaMountedCount;
    }

    public int getShuffleMode() {
        return mShuffleMode;
    }

    public void setShuffleMode(final int shufflemode) {
        synchronized (this) {
            if (mShuffleMode == shufflemode && mPlaylist.size() > 0) {
                return;
            }

            mShuffleMode = shufflemode;
            if (mShuffleMode == SHUFFLE_AUTO) {
                if (makeAutoShuffleList()) {
                    mPlaylist.clear();
                    doAutoShuffleUpdate();
                    mPlayPos = 0;
                    openCurrentAndNext();
                    play();
                    notifyChange(META_CHANGED);
                    return;
                } else {
                    mShuffleMode = SHUFFLE_NONE;
                }
            } else {
                setNextTrack();
            }
            saveQueue(false);
            notifyChange(SHUFFLEMODE_CHANGED);
        }
    }

    public int getRepeatMode() {
        return mRepeatMode;
    }

    public void setRepeatMode(final int repeatmode) {
        synchronized (this) {
            mRepeatMode = repeatmode;
            setNextTrack();
            saveQueue(false);
            notifyChange(REPEATMODE_CHANGED);
        }
    }

    public int removeTrack(final long id) {
        int numremoved = 0;
        synchronized (this) {
            for (int i = 0; i < mPlaylist.size(); i++) {
                if (mPlaylist.get(i).mId == id) {
                    numremoved += removeTracksInternal(i, i);
                    i--;
                }
            }
        }
        if (numremoved > 0) {
            notifyChange(QUEUE_CHANGED);
        }
        return numremoved;
    }

    public boolean removeTrackAtPosition(final long id, final int position) {
        synchronized (this) {
            if (position >= 0 &&
                    position < mPlaylist.size() &&
                    mPlaylist.get(position).mId == id) {

                return removeTracks(position, position) > 0;
            }
        }
        return false;
    }

    public int removeTracks(final int first, final int last) {
        final int numremoved = removeTracksInternal(first, last);
        if (numremoved > 0) {
            notifyChange(QUEUE_CHANGED);
        }
        return numremoved;
    }

    public int getQueuePosition() {
        synchronized (this) {
            return mPlayPos;
        }
    }

    public void setQueuePosition(final int index) {
        synchronized (this) {
            stop(false);
            mPlayPos = index;
            openCurrentAndNext();
            play();
            notifyChange(META_CHANGED);
            if (mShuffleMode == SHUFFLE_AUTO) {
                doAutoShuffleUpdate();
            }
        }
    }

    public int getQueueHistorySize() {
        synchronized (this) {
            return mHistory.size();
        }
    }

    public int getQueueHistoryPosition(int position) {
```

```java
        synchronized (this) {
            if (position >= 0 && position < mHistory.size()) {
                return mHistory.get(position);
            }
        }

        return -1;
    }

    public int[] getQueueHistoryList() {
        synchronized (this) {
            int[] history = new int[mHistory.size()];
            for (int i = 0; i < mHistory.size(); i++) {
                history[i] = mHistory.get(i);
            }

            return history;
        }
    }

    public String getPath() {
        synchronized (this) {
            if (mCursor == null) {
                return null;
            }
            return mCursor.getString(mCursor.getColumnIndexOrThrow(AudioColumns.DATA));
        }
    }

    public String getAlbumName() {
        synchronized (this) {
            if (mCursor == null) {
                return null;
            }
            return mCursor.getString(mCursor.getColumnIndexOrThrow(AudioColumns.ALBUM));
        }
    }

    public String getTrackName() {
        synchronized (this) {
            if (mCursor == null) {
                return null;
            }
            return mCursor.getString(mCursor.getColumnIndexOrThrow(AudioColumns.TITLE));
        }
    }

    public String getGenreName() {
        synchronized (this) {
            if (mCursor == null || mPlayPos < 0 || mPlayPos >= mPlaylist.size()) {
                return null;
            }
            String[] genreProjection = {MediaStore.Audio.Genres.NAME};
            Uri genreUri = MediaStore.Audio.Genres.getContentUriForAudioId("external",
                    (int) mPlaylist.get(mPlayPos).mId);
            Cursor genreCursor = getContentResolver().query(genreUri, genreProjection,
                    null, null, null);
            if (genreCursor != null) {
                try {
                    if (genreCursor.moveToFirst()) {
                        return genreCursor.getString(
                                genreCursor.getColumnIndexOrThrow(MediaStore.Audio.Genres.NAME));
                    }
                } finally {
                    genreCursor.close();
                }
            }
            return null;
        }
    }
    public String getArtistName() {
        synchronized (this) {
            if (mCursor == null) {
                return null;
            }
            return mCursor.getString(mCursor.getColumnIndexOrThrow(AudioColumns.ARTIST));
        }
    }

    public String getAlbumArtistName() {
        synchronized (this) {
            if (mAlbumCursor == null) {
                return null;
            }
            return mAlbumCursor.getString(mAlbumCursor.getColumnIndexOrThrow(AlbumColumns.ARTIST));
        }
    }

    public long getAlbumId() {
        synchronized (this) {
            if (mCursor == null) {
                return -1;
            }
            return mCursor.getLong(mCursor.getColumnIndexOrThrow(AudioColumns.ALBUM_ID));
        }
    }

    public long getArtistId() {
        synchronized (this) {
            if (mCursor == null) {
                return -1;
            }
            return mCursor.getLong(mCursor.getColumnIndexOrThrow(AudioColumns.ARTIST_ID));
        }
    }

    public long getAudioId() {
        MusicPlaybackTrack track = getCurrentTrack();
        if (track != null) {
            return track.mId;
        }

        return -1;
    }

    public MusicPlaybackTrack getCurrentTrack() {
        return getTrack(mPlayPos);
    }

    public synchronized MusicPlaybackTrack getTrack(int index) {
        if (index >= 0 && index < mPlaylist.size() && mPlayer.isInitialized()) {
            return mPlaylist.get(index);
        }

        return null;
    }

    public long getNextAudioId() {
        synchronized (this) {
            if (mNextPlayPos >= 0 && mNextPlayPos < mPlaylist.size() && mPlayer.isInitialized()) {
                return mPlaylist.get(mNextPlayPos).mId;
            }
        }
        return -1;
    }

    public long getPreviousAudioId() {
        synchronized (this) {
            if (mPlayer.isInitialized()) {
                int pos = getPreviousPlayPosition(false);
                if (pos >= 0 && pos < mPlaylist.size()) {
                    return mPlaylist.get(pos).mId;
                }
            }
        }
        return -1;
    }

    public long seek(long position) {
        if (mPlayer.isInitialized()) {
            if (position < 0) {
                position = 0;
            } else if (position > mPlayer.duration()) {
                position = mPlayer.duration();
            }
            long result = mPlayer.seek(position);
            notifyChange(POSITION_CHANGED);
            return result;
        }
        return -1;
    }

    public void seekRelative(long deltaInMs) {
        synchronized (this) {
            if (mPlayer.isInitialized()) {
                final long newPos = position() + deltaInMs;
                final long duration = duration();
                if (newPos < 0) {
                    prev(true);
                    // seek to the new duration + the leftover position
                    seek(duration() + newPos);
                } else if (newPos >= duration) {
                    gotoNext(true);
                    // seek to the leftover duration
                    seek(newPos - duration);
                } else {
                    seek(newPos);
                }
            }
        }
    }

    public long position() {
        if (mPlayer.isInitialized()) {
            return mPlayer.position();
        }
        return -1;
    }

    public long duration() {
        if (mPlayer.isInitialized()) {
            return mPlayer.duration();
        }
        return -1;
    }

    public long[] getQueue() {
        synchronized (this) {
            final int len = mPlaylist.size();
            final long[] list = new long[len];
            for (int i = 0; i < len; i++) {
                list[i] = mPlaylist.get(i).mId;
            }
            return list;
        }
    }

    public long getQueueItemAtPosition(int position) {
        synchronized (this) {
            if (position >= 0 && position < mPlaylist.size()) {
                return mPlaylist.get(position).mId;
            }
        }

        return -1;
    }

    public int getQueueSize() {
        synchronized (this) {
            return mPlaylist.size();
        }
    }

    public boolean isPlaying() {
        return mIsSupposedToBePlaying;
    }

    private void setIsSupposedToBePlaying(boolean value, boolean notify) {
        if (mIsSupposedToBePlaying != value) {
            mIsSupposedToBePlaying = value;


            if (!mIsSupposedToBePlaying) {
                scheduleDelayedShutdown();
                mLastPlayedTime = System.currentTimeMillis();
            }

            if (notify) {
                notifyChange(PLAYSTATE_CHANGED);
            }
        }
    }

    private boolean recentlyPlayed() {
        return isPlaying() || System.currentTimeMillis() - mLastPlayedTime < IDLE_DELAY;
    }

    public void open(final long[] list, final int position, long sourceId, TimberUtils.IdType sourceType) {
        synchronized (this) {
            if (mShuffleMode == SHUFFLE_AUTO) {
                mShuffleMode = SHUFFLE_NORMAL;
            }
            final long oldId = getAudioId();
            final int listlength = list.length;
            boolean newlist = true;
            if (mPlaylist.size() == listlength) {
                newlist = false;
                for (int i = 0; i < listlength; i++) {
                    if (list[i] != mPlaylist.get(i).mId) {
                        newlist = true;
                        break;
                    }
                }
            }
            if (newlist) {
                addToPlayList(list, -1, sourceId, sourceType);
                notifyChange(QUEUE_CHANGED);
            }
            if (position >= 0) {
                mPlayPos = position;
            } else {
                mPlayPos = mShuffler.nextInt(mPlaylist.size());
            }
            mHistory.clear();
            openCurrentAndNext();
            if (oldId != getAudioId()) {
                notifyChange(META_CHANGED);
            }
        }
```

```java
    }

    public void stop() {
        stop(true);
    }

    public void play() {
        play(true);
    }

    public void play(boolean createNewNextTrack) {
        int status = mAudioManager.requestAudioFocus(mAudioFocusListener,
                AudioManager.STREAM_MUSIC, AudioManager.AUDIOFOCUS_GAIN);

        if (D) Log.d(TAG, "Starting playback: audio focus request status = " + status);

        if (status != AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
            return;
        }

        final Intent intent = new Intent(AudioEffect.ACTION_OPEN_AUDIO_EFFECT_CONTROL_SESSION);
        intent.putExtra(AudioEffect.EXTRA_AUDIO_SESSION, getAudioSessionId());
        intent.putExtra(AudioEffect.EXTRA_PACKAGE_NAME, getPackageName());
        sendBroadcast(intent);

        mAudioManager.registerMediaButtonEventReceiver(new ComponentName(getPackageName(),
                MediaButtonIntentReceiver.class.getName()));
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
            mSession.setActive(true);

        if (createNewNextTrack) {
            setNextTrack();
        } else {
            setNextTrack(mNextPlayPos);
        }

        if (mPlayer.isInitialized()) {
            final long duration = mPlayer.duration();
            if (mRepeatMode != REPEAT_CURRENT && duration > 2000
                    && mPlayer.position() >= duration - 2000) {
                gotoNext(true);
            }

            mPlayer.start();
            mPlayerHandler.removeMessages(FADEDOWN);
            mPlayerHandler.sendEmptyMessage(FADEUP);

            setIsSupposedToBePlaying(true, true);

            cancelShutdown();
            updateNotification();
            notifyChange(META_CHANGED);
        } else if (mPlaylist.size() <= 0) {
            setShuffleMode(SHUFFLE_AUTO);
        }
    }

    public void pause() {
        if (D) Log.d(TAG, "Pausing playback");
        synchronized (this) {
            mPlayerHandler.removeMessages(FADEUP);
            if (mIsSupposedToBePlaying) {
                final Intent intent = new Intent(
                        AudioEffect.ACTION_CLOSE_AUDIO_EFFECT_CONTROL_SESSION);
                intent.putExtra(AudioEffect.EXTRA_AUDIO_SESSION, getAudioSessionId());
                intent.putExtra(AudioEffect.EXTRA_PACKAGE_NAME, getPackageName());
                sendBroadcast(intent);

                mPlayer.pause();
                notifyChange(META_CHANGED);
                setIsSupposedToBePlaying(false, true);
            }
        }
    }

    public void gotoNext(final boolean force) {
        if (D) Log.d(TAG, "Going to next track");
        synchronized (this) {
            if (mPlaylist.size() <= 0) {
                if (D) Log.d(TAG, "No play queue");
                scheduleDelayedShutdown();
                return;
            }
            int pos = mNextPlayPos;
            if (pos < 0) {
                pos = getNextPosition(force);
            }

            if (pos < 0) {
                setIsSupposedToBePlaying(false, true);
                return;
            }

            stop(false);
            setAndRecordPlayPos(pos);
            openCurrentAndNext();
            play();
            notifyChange(META_CHANGED);
        }
    }

    public void goToPosition(int pos) {
        synchronized (this) {
            if (mPlaylist.size() <= 0) {
                if (D) Log.d(TAG, "No play queue");
                scheduleDelayedShutdown();
                return;
            }
            if (pos < 0) {
                return;
            }
            if (pos == mPlayPos) {
                if (!isPlaying()) {
                    play();
                }
                return;
            }
            stop(false);
            setAndRecordPlayPos(pos);
            openCurrentAndNext();
            play();
            notifyChange(META_CHANGED);
        }
    }

    public void setAndRecordPlayPos(int nextPos) {
        synchronized (this) {

            if (mShuffleMode != SHUFFLE_NONE) {
                mHistory.add(mPlayPos);
                if (mHistory.size() > MAX_HISTORY_SIZE) {
                    mHistory.remove(0);
                }
            }

            mPlayPos = nextPos;
        }
    }

    public void prev(boolean forcePrevious) {
        synchronized (this) {
            boolean goPrevious = getRepeatMode() != REPEAT_CURRENT &&
                    (position() < REWIND_INSTEAD_PREVIOUS_THRESHOLD || forcePrevious);

            if (goPrevious) {
                if (D) Log.d(TAG, "Going to previous track");
                int pos = getPreviousPlayPosition(true);

                if (pos < 0) {
                    return;
                }
                mNextPlayPos = mPlayPos;
                mPlayPos = pos;
                stop(false);
                openCurrent();
                play(false);
                notifyChange(META_CHANGED);
            } else {
                if (D) Log.d(TAG, "Going to beginning of track");
                seek(0);
                play(false);
            }
        }
    }

    public int getPreviousPlayPosition(boolean removeFromHistory) {
        synchronized (this) {
            if (mShuffleMode == SHUFFLE_NORMAL) {

                final int histsize = mHistory.size();
                if (histsize == 0) {
                    return -1;
                }
                final Integer pos = mHistory.get(histsize - 1);
                if (removeFromHistory) {
                    mHistory.remove(histsize - 1);
                }
                return pos.intValue();
            } else {
                if (mPlayPos > 0) {
                    return mPlayPos - 1;
                } else {
                    return mPlaylist.size() - 1;
                }
            }
        }
    }

    private void openCurrent() {
        openCurrentAndMaybeNext(false);
    }

    public void moveQueueItem(int index1, int index2) {
        synchronized (this) {
            if (index1 >= mPlaylist.size()) {
                index1 = mPlaylist.size() - 1;
            }
            if (index2 >= mPlaylist.size()) {
                index2 = mPlaylist.size() - 1;
            }

            if (index1 == index2) {
                return;
            }

            final MusicPlaybackTrack track = mPlaylist.remove(index1);
            if (index1 < index2) {
                mPlaylist.add(index2, track);
                if (mPlayPos == index1) {
                    mPlayPos = index2;
```

```java
                } else if (mPlayPos >= index1 && mPlayPos <= index2) {
                    mPlayPos--;
                }
            } else if (index2 < index1) {
                mPlaylist.add(index2, track);
                if (mPlayPos == index1) {
                    mPlayPos = index2;
                } else if (mPlayPos >= index2 && mPlayPos <= index1) {
                    mPlayPos++;
                }
            }
            notifyChange(QUEUE_CHANGED);
        }
    }

    public void enqueue(final long[] list, final int action, long sourceId, IdType sourceType) {
        synchronized (this) {
            if (action == NEXT && mPlayPos + 1 < mPlaylist.size()) {
                addToPlayList(list, mPlayPos + 1, sourceId, sourceType);
                mNextPlayPos = mPlayPos + 1;
                notifyChange(QUEUE_CHANGED);
            } else {
                addToPlayList(list, Integer.MAX_VALUE, sourceId, sourceType);
                notifyChange(QUEUE_CHANGED);
            }

            if (mPlayPos < 0) {
                mPlayPos = 0;
                openCurrentAndNext();
                play();
                notifyChange(META_CHANGED);
            }
        }
    }

    private void cycleRepeat() {
        if (mRepeatMode == REPEAT_NONE) {
            setRepeatMode(REPEAT_CURRENT);
            if (mShuffleMode != SHUFFLE_NONE) {
                setShuffleMode(SHUFFLE_NONE);
            }
        } else {
            setRepeatMode(REPEAT_NONE);
        }
    }

    private void cycleShuffle() {
        if (mShuffleMode == SHUFFLE_NONE) {
            setShuffleMode(SHUFFLE_NORMAL);
//            if (mRepeatMode == REPEAT_CURRENT) {
//                setRepeatMode(REPEAT_ALL);
//            }
        } else if (mShuffleMode == SHUFFLE_NORMAL || mShuffleMode == SHUFFLE_AUTO) {
            setShuffleMode(SHUFFLE_NONE);
        }
    }

    public void refresh() {
        notifyChange(REFRESH);
    }

    public void playlistChanged() {
        notifyChange(PLAYLIST_CHANGED);
    }

    public interface TrackErrorExtra {

        String TRACK_NAME = "trackname";
    }
    private static final class MusicPlayerHandler extends Handler {
        private final WeakReference<MusicService> mService;
        private float mCurrentVolume = 1.0f;


        public MusicPlayerHandler(final MusicService service, final Looper looper) {
            super(looper);
            mService = new WeakReference<MusicService>(service);
        }


        @Override
        public void handleMessage(final Message msg) {
            final MusicService service = mService.get();
            if (service == null) {
                return;
            }

            synchronized (service) {
                switch (msg.what) {
                    case FADEDOWN:
                        mCurrentVolume -= .05f;
                        if (mCurrentVolume > .2f) {
                            sendEmptyMessageDelayed(FADEDOWN, 10);
                        } else {
                            mCurrentVolume = .2f;
                        }
                        service.mPlayer.setVolume(mCurrentVolume);
                        break;
                    case FADEUP:
                        mCurrentVolume += .01f;
                        if (mCurrentVolume < 1.0f) {
                            sendEmptyMessageDelayed(FADEUP, 10);
                        } else {
                            mCurrentVolume = 1.0f;
                        }
                        service.mPlayer.setVolume(mCurrentVolume);
                        break;
                    case SERVER_DIED:
                        if (service.isPlaying()) {
                            final TrackErrorInfo info = (TrackErrorInfo) msg.obj;
                            service.sendErrorMessage(info.mTrackName);


                            service.removeTrack(info.mId);
                        } else {
                            service.openCurrentAndNext();
                        }
                        break;
                    case TRACK_WENT_TO_NEXT:
                        mService.get().scrobble();
                        service.setAndRecordPlayPos(service.mNextPlayPos);
                        service.setNextTrack();
                        if (service.mCursor != null) {
                            service.mCursor.close();
                            service.mCursor = null;
                        }
                        service.updateCursor(service.mPlaylist.get(service.mPlayPos).mId);
                        service.notifyChange(META_CHANGED);
                        service.updateNotification();
                        break;
                    case TRACK_ENDED:
                        if (service.mRepeatMode == REPEAT_CURRENT) {
                            service.seek(0);
                            service.play();
                        } else {
                            service.gotoNext(false);
                        }
                        break;
                    case RELEASE_WAKELOCK:
```

```java
                    service.mWakeLock.release();
                    break;
                case FOCUSCHANGE:
                    if (D) Log.d(TAG, "Received audio focus change event " + msg.arg1);
                    switch (msg.arg1) {
                        case AudioManager.AUDIOFOCUS_LOSS:
                        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
                            if (service.isPlaying()) {
                                service.mPausedByTransientLossOfFocus =
                                        msg.arg1 == AudioManager.AUDIOFOCUS_LOSS_TRANSIENT;
                            }
                            service.pause();
                            break;
                        case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
                            removeMessages(FADEUP);
                            sendEmptyMessage(FADEDOWN);
                            break;
                        case AudioManager.AUDIOFOCUS_GAIN:
                            if (!service.isPlaying()
                                    && service.mPausedByTransientLossOfFocus) {
                                service.mPausedByTransientLossOfFocus = false;
                                mCurrentVolume = 0f;
                                service.mPlayer.setVolume(mCurrentVolume);
                                service.play();
                            } else {
                                removeMessages(FADEDOWN);
                                sendEmptyMessage(FADEUP);
                            }
                            break;
                        default:
                    }
                    break;
                default:
                    break;
            }
        }
    }
}

private static final class Shuffler {

    private final LinkedList<Integer> mHistoryOfNumbers = new LinkedList<Integer>();

    private final TreeSet<Integer> mPreviousNumbers = new TreeSet<Integer>();

    private final Random mRandom = new Random();

    private int mPrevious;


    public Shuffler() {
        super();
    }


    public int nextInt(final int interval) {
        int next;
        do {
            next = mRandom.nextInt(interval);
        } while (next == mPrevious && interval > 1
                && !mPreviousNumbers.contains(Integer.valueOf(next)));
        mPrevious = next;
        mHistoryOfNumbers.add(mPrevious);
        mPreviousNumbers.add(mPrevious);
        cleanUpHistory();
        return next;
    }


    private void cleanUpHistory() {
        if (!mHistoryOfNumbers.isEmpty() && mHistoryOfNumbers.size() >= MAX_HISTORY_SIZE) {
            for (int i = 0; i < Math.max(1, MAX_HISTORY_SIZE / 2); i++) {
                mPreviousNumbers.remove(mHistoryOfNumbers.removeFirst());
            }
        }
    }
}

private static final class TrackErrorInfo {
    public long mId;
    public String mTrackName;

    public TrackErrorInfo(long id, String trackName) {
        mId = id;
        mTrackName = trackName;
    }
}

private static final class MultiPlayer implements MediaPlayer.OnErrorListener,
        MediaPlayer.OnCompletionListener {

    private final WeakReference<MusicService> mService;

    private MediaPlayer mCurrentMediaPlayer = new MediaPlayer();

    private MediaPlayer mNextMediaPlayer;

    private Handler mHandler;

    private boolean mIsInitialized = false;

    private String mNextMediaPath;


    public MultiPlayer(final MusicService service) {
        mService = new WeakReference<MusicService>(service);
        mCurrentMediaPlayer.setWakeMode(mService.get(), PowerManager.PARTIAL_WAKE_LOCK);

    }


    public void setDataSource(final String path) {
        try {
            mIsInitialized = setDataSourceImpl(mCurrentMediaPlayer, path);
            if (mIsInitialized) {
                setNextDataSource(null);
            }
        } catch (IllegalStateException e) {
            e.printStackTrace();
        }
    }


    private boolean setDataSourceImpl(final MediaPlayer player, final String path) {
        try {
            player.reset();
            player.setOnPreparedListener(null);
            if (path.startsWith("content://")) {
                player.setDataSource(mService.get(), Uri.parse(path));
            } else {
                player.setDataSource(path);
            }
            player.setAudioStreamType(AudioManager.STREAM_MUSIC);

            player.prepare();
        } catch (final IOException todo) {

            return false;
        } catch (final IllegalArgumentException todo) {
            return false;
        }
        player.setOnCompletionListener(this);
        player.setOnErrorListener(this);
        return true;
    }


    public void setNextDataSource(final String path) {
        mNextMediaPath = null;
        try {
            mCurrentMediaPlayer.setNextMediaPlayer(null);
        } catch (IllegalArgumentException e) {
            Log.i(TAG, "Next media player is current one, continuing");
        } catch (IllegalStateException e) {
            Log.e(TAG, "Media player not initialized!");
            return;
        }
        if (mNextMediaPlayer != null) {
            mNextMediaPlayer.release();
            mNextMediaPlayer = null;
        }
        if (path == null) {
            return;
        }
        mNextMediaPlayer = new MediaPlayer();
        mNextMediaPlayer.setWakeMode(mService.get(), PowerManager.PARTIAL_WAKE_LOCK);
        mNextMediaPlayer.setAudioSessionId(getAudioSessionId());
        try {
            if (setDataSourceImpl(mNextMediaPlayer, path)) {
                mNextMediaPath = path;
                mCurrentMediaPlayer.setNextMediaPlayer(mNextMediaPlayer);
            } else {
                if (mNextMediaPlayer != null) {
                    mNextMediaPlayer.release();
                    mNextMediaPlayer = null;
                }
            }
        } catch (IllegalStateException e) {
            e.printStackTrace();
        }
    }


    public void setHandler(final Handler handler) {
        mHandler = handler;
    }


    public boolean isInitialized() {
        return mIsInitialized;
    }


    public void start() {
        mCurrentMediaPlayer.start();
    }


    public void stop() {
        mCurrentMediaPlayer.reset();
        mIsInitialized = false;
    }


    public void release() {
        mCurrentMediaPlayer.release();
    }
```

```java
        public void pause() {
            mCurrentMediaPlayer.pause();
        }


        public long duration() {
            return mCurrentMediaPlayer.getDuration();
        }


        public long position() {
            return mCurrentMediaPlayer.getCurrentPosition();
        }


        public long seek(final long whereto) {
            mCurrentMediaPlayer.seekTo((int) whereto);
            return whereto;
        }


        public void setVolume(final float vol) {
            try {
                mCurrentMediaPlayer.setVolume(vol, vol);
            } catch (IllegalStateException e) {
                e.printStackTrace();
            }
        }

        public int getAudioSessionId() {
            return mCurrentMediaPlayer.getAudioSessionId();
        }

        public void setAudioSessionId(final int sessionId) {
            mCurrentMediaPlayer.setAudioSessionId(sessionId);
        }

        @Override
        public boolean onError(final MediaPlayer mp, final int what, final int extra) {
            Log.w(TAG, "Music Server Error what: " + what + " extra: " + extra);
            switch (what) {
                case MediaPlayer.MEDIA_ERROR_SERVER_DIED:
                    final MusicService service = mService.get();
                    final TrackErrorInfo errorInfo = new TrackErrorInfo(service.getAudioId(),
                            service.getTrackName());

                    mIsInitialized = false;
                    mCurrentMediaPlayer.release();
                    mCurrentMediaPlayer = new MediaPlayer();
                    mCurrentMediaPlayer.setWakeMode(service, PowerManager.PARTIAL_WAKE_LOCK);
                    Message msg = mHandler.obtainMessage(SERVER_DIED, errorInfo);
                    mHandler.sendMessageDelayed(msg, 2000);
                    return true;
                default:
                    break;
            }
            return false;
        }


        @Override
        public void onCompletion(final MediaPlayer mp) {
            if (mp == mCurrentMediaPlayer && mNextMediaPlayer != null) {
                mCurrentMediaPlayer.release();
                mCurrentMediaPlayer = mNextMediaPlayer;
                mNextMediaPath = null;
                mNextMediaPlayer = null;
                mHandler.sendEmptyMessage(TRACK_WENT_TO_NEXT);
            } else {
                mService.get().mWakeLock.acquire(30000);
                mHandler.sendEmptyMessage(TRACK_ENDED);
                mHandler.sendEmptyMessage(RELEASE_WAKELOCK);
            }
        }
    }

    private static final class ServiceStub extends ITimberService.Stub {

        private final WeakReference<MusicService> mService;

        private ServiceStub(final MusicService service) {
            mService = new WeakReference<MusicService>(service);
        }


        @Override
        public void openFile(final String path) throws RemoteException {
            mService.get().openFile(path);
        }

        @Override
        public void open(final long[] list, final int position, long sourceId, int sourceType)
                throws RemoteException {
            mService.get().open(list, position, sourceId, IdType.getTypeById(sourceType));
        }

        @Override
        public void stop() throws RemoteException {
            mService.get().stop();
        }

        @Override
        public void pause() throws RemoteException {
            mService.get().pause();
        }


        @Override
        public void play() throws RemoteException {
            mService.get().play();
        }

        @Override
        public void prev(boolean forcePrevious) throws RemoteException {
            mService.get().prev(forcePrevious);
        }

        @Override
        public void next() throws RemoteException {
            mService.get().gotoNext(true);
        }

        @Override
        public void enqueue(final long[] list, final int action, long sourceId, int sourceType)
                throws RemoteException {
            mService.get().enqueue(list, action, sourceId, IdType.getTypeById(sourceType));
        }

        @Override
        public void moveQueueItem(final int from, final int to) throws RemoteException {
            mService.get().moveQueueItem(from, to);
        }

        @Override
        public void refresh() throws RemoteException {
            mService.get().refresh();
        }

        @Override
        public void playlistChanged() throws RemoteException {
            mService.get().playlistChanged();
        }

        @Override
        public boolean isPlaying() throws RemoteException {
            return mService.get().isPlaying();
        }

        @Override
        public long[] getQueue() throws RemoteException {
            return mService.get().getQueue();
        }

        @Override
        public long getQueueItemAtPosition(int position) throws RemoteException {
            return mService.get().getQueueItemAtPosition(position);
        }

        @Override
        public int getQueueSize() throws RemoteException {
            return mService.get().getQueueSize();
        }

        @Override
        public int getQueueHistoryPosition(int position) throws RemoteException {
            return mService.get().getQueueHistoryPosition(position);
        }

        @Override
        public int getQueueHistorySize() throws RemoteException {
            return mService.get().getQueueHistorySize();
        }

        @Override
        public int[] getQueueHistoryList() throws RemoteException {
            return mService.get().getQueueHistoryList();
        }

        @Override
        public long duration() throws RemoteException {
            return mService.get().duration();
        }

        @Override
        public long position() throws RemoteException {
            return mService.get().position();
        }

        @Override
        public long seek(final long position) throws RemoteException {
            return mService.get().seek(position);
        }

        @Override
        public void seekRelative(final long deltaInMs) throws RemoteException {
            mService.get().seekRelative(deltaInMs);
        }

        @Override
        public long getAudioId() throws RemoteException {
            return mService.get().getAudioId();
        }

        @Override
        public MusicPlaybackTrack getCurrentTrack() throws RemoteException {
            return mService.get().getCurrentTrack();
        }

        @Override
        public MusicPlaybackTrack getTrack(int index) throws RemoteException {
            return mService.get().getTrack(index);
        }

        @Override
        public long getNextAudioId() throws RemoteException {
            return mService.get().getNextAudioId();
        }

        @Override
        public long getPreviousAudioId() throws RemoteException {
            return mService.get().getPreviousAudioId();
        }

        @Override
        public long getArtistId() throws RemoteException {
            return mService.get().getArtistId();
        }

        @Override
        public long getAlbumId() throws RemoteException {
            return mService.get().getAlbumId();
        }

        @Override
        public String getArtistName() throws RemoteException {
            return mService.get().getArtistName();
        }

        @Override
        public String getTrackName() throws RemoteException {
            return mService.get().getTrackName();
        }

        @Override
        public String getAlbumName() throws RemoteException {
            return mService.get().getAlbumName();
        }

        @Override
        public String getPath() throws RemoteException {
            return mService.get().getPath();
        }

        @Override
        public int getQueuePosition() throws RemoteException {
            return mService.get().getQueuePosition();
        }

        @Override
        public void setQueuePosition(final int index) throws RemoteException {
            mService.get().setQueuePosition(index);
        }

        @Override
        public int getShuffleMode() throws RemoteException {
            return mService.get().getShuffleMode();
        }

        @Override
        public void setShuffleMode(final int shufflemode) throws RemoteException {
            mService.get().setShuffleMode(shufflemode);
        }

        @Override
        public int getRepeatMode() throws RemoteException {
            return mService.get().getRepeatMode();
        }

        @Override
        public void setRepeatMode(final int repeatmode) throws RemoteException {
            mService.get().setRepeatMode(repeatmode);
        }

        @Override
        public int removeTracks(final int first, final int last) throws RemoteException {
            return mService.get().removeTracks(first, last);
        }


        @Override
        public int removeTrack(final long id) throws RemoteException {
            return mService.get().removeTrack(id);
        }


        @Override
        public boolean removeTrackAtPosition(final long id, final int position)
                throws RemoteException {
            return mService.get().removeTrackAtPosition(id, position);
        }


        @Override
        public int getMediaMountedCount() throws RemoteException {
            return mService.get().getMediaMountedCount();
        }


        @Override
        public int getAudioSessionId() throws RemoteException {
            return mService.get().getAudioSessionId();
        }

    }

    private class MediaStoreObserver extends ContentObserver implements Runnable {

        private static final long REFRESH_DELAY = 500;
        private Handler mHandler;

        public MediaStoreObserver(Handler handler) {
            super(handler);
            mHandler = handler;
        }

        @Override
        public void onChange(boolean selfChange) {


            mHandler.removeCallbacks(this);
            mHandler.postDelayed(this, REFRESH_DELAY);
        }

        @Override
        public void run() {

            Log.e("ELEVEN", "calling refresh!");
            refresh();
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\TimberApp.java

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber;

import android.content.Context;
import android.support.multidex.MultiDex;
import android.support.multidex.MultiDexApplication;

import com.afollestad.appthemeengine.ATE;
import com.crashlytics.android.Crashlytics;
import com.crashlytics.android.core.CrashlyticsCore;
import com.naman14.timber.permissions.Nammu;
import com.naman14.timber.utils.PreferencesUtility;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.ImageLoaderConfiguration;
import com.nostra13.universalimageloader.core.download.BaseImageDownloader;
import com.nostra13.universalimageloader.utils.L;

import java.io.IOException;
import java.io.InputStream;

import io.fabric.sdk.android.Fabric;

public class TimberApp extends MultiDexApplication {


    private static TimberApp mInstance;

    public static synchronized TimberApp getInstance() {
        return mInstance;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        mInstance = this;

        //disable crashlytics for debug builds
        Crashlytics crashlyticsKit = new Crashlytics.Builder()
                .core(new CrashlyticsCore.Builder().disabled(BuildConfig.DEBUG).build())
                .build();
        Fabric.with(this, crashlyticsKit);

        ImageLoaderConfiguration localImageLoaderConfiguration = new ImageLoaderConfiguration.Builder(this).imageDownloader(
            PreferencesUtility prefs = PreferencesUtility.getInstance(TimberApp.this);

            @Override
            protected InputStream getStreamFromNetwork(String imageUri, Object extra) throws IOException {
                if (prefs.loadArtistAndAlbumImages())
                    return super.getStreamFromNetwork(imageUri, extra);
                throw new IOException();
            }
        }).build();

        ImageLoader.getInstance().init(localImageLoaderConfiguration);
        L.writeLogs(false);
        L.disableLogging();
        L.writeDebugLogs(false);
        Nammu.init(this);

        if (!ATE.config(this, "light_theme").isConfigured()) {
            ATE.config(this, "light_theme")
                    .activityTheme(R.style.AppThemeLight)
                    .primaryColorRes(R.color.colorPrimaryLightDefault)
                    .accentColorRes(R.color.colorAccentLightDefault)
                    .coloredNavigationBar(false)
                    .usingMaterialDialogs(true)
                    .commit();
        }
        if (!ATE.config(this, "dark_theme").isConfigured()) {
            ATE.config(this, "dark_theme")
                    .activityTheme(R.style.AppThemeDark)
                    .primaryColorRes(R.color.colorPrimaryDarkDefault)
                    .accentColorRes(R.color.colorAccentDarkDefault)
                    .coloredNavigationBar(false)
                    .usingMaterialDialogs(true)
                    .commit();
        }
        if (!ATE.config(this, "light_theme_notoolbar").isConfigured()) {
            ATE.config(this, "light_theme_notoolbar")
                    .activityTheme(R.style.AppThemeLight)
                    .coloredActionBar(false)
                    .primaryColorRes(R.color.colorPrimaryLightDefault)
                    .accentColorRes(R.color.colorAccentLightDefault)
                    .coloredNavigationBar(false)
                    .usingMaterialDialogs(true)
                    .commit();
        }
        if (!ATE.config(this, "dark_theme_notoolbar").isConfigured()) {
            ATE.config(this, "dark_theme_notoolbar")
                    .activityTheme(R.style.AppThemeDark)
                    .coloredActionBar(false)
                    .primaryColorRes(R.color.colorPrimaryDarkDefault)
                    .accentColorRes(R.color.colorAccentDarkDefault)
                    .coloredNavigationBar(true)
                    .usingMaterialDialogs(true)
                    .commit();
        }

    }


}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber;

import android.annotation.TargetApi;
import android.content.Context;
import android.content.Intent;
import android.media.MediaDescription;
import android.media.browse.MediaBrowser;
import android.media.session.MediaSession;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.service.media.MediaBrowserService;
import android.support.annotation.Nullable;

import com.naman14.timber.dataloaders.AlbumLoader;
import com.naman14.timber.dataloaders.AlbumSongLoader;
import com.naman14.timber.dataloaders.ArtistAlbumLoader;
import com.naman14.timber.dataloaders.ArtistLoader;
import com.naman14.timber.dataloaders.ArtistSongLoader;
import com.naman14.timber.dataloaders.PlaylistLoader;
import com.naman14.timber.dataloaders.PlaylistSongLoader;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.models.Album;
import com.naman14.timber.models.Artist;
import com.naman14.timber.models.Playlist;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.TimberUtils;

import java.util.ArrayList;
import java.util.List;

@TargetApi(21)
public class WearBrowserService extends MediaBrowserService {

    public static final String MEDIA_ID_ROOT = "__ROOT__";
    public static final int TYPE_ARTIST = 0;
    public static final int TYPE_ALBUM = 1;
    public static final int TYPE_SONG = 2;
    public static final int TYPE_PLAYLIST = 3;
    public static final int TYPE_ARTIST_SONG_ALBUMS = 4;
    public static final int TYPE_ALBUM_SONGS = 5;
    public static final int TYPE_ARTIST_ALL_SONGS = 6;
    public static final int TYPE_PLAYLIST_ALL_SONGS = 7;

    MediaSession mSession;
    public static WearBrowserService sInstance;

    private Context mContext;
    private boolean mServiceStarted;

    public static WearBrowserService getInstance() {
        return sInstance;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        sInstance = this;
        mContext = this;
        mSession = new MediaSession(this, "WearBrowserService");
        setSessionToken(mSession.getSessionToken());
        mSession.setCallback(new MediaSessionCallback());
        mSession.setFlags(MediaSession.FLAG_HANDLES_MEDIA_BUTTONS | MediaSession.FLAG_HANDLES_TRANSPORT_CONTROLS);

    }

    @Override
    public int onStartCommand(Intent startIntent, int flags, int startId) {
        return START_STICKY;
    }

    @Override
    public void onDestroy() {
        mServiceStarted = false;
        mSession.release();
    }

    @Override
    public void onLoadChildren(String parentId, Result<List<MediaBrowser.MediaItem>> result) {

        result.detach();
        loadChildren(parentId, result);

    }

    @Nullable
    @Override
    public BrowserRoot onGetRoot(String clientPackageName, int clientUid, Bundle rootHints) {
        return new BrowserRoot(MEDIA_ID_ROOT, null);
    }

    private final class MediaSessionCallback extends MediaSession.Callback {

        @Override
        public void onPlay() {
            setSessionActive();
        }

        @Override
        public void onSeekTo(long position) {

        }

        @Override
        public void onPlayFromMediaId(final String mediaId, Bundle extras) {
            long songId = Long.parseLong(mediaId);
            setSessionActive();
            MusicPlayer.playAll(mContext, new long[]{songId}, 0, -1, TimberUtils.IdType.NA, false);
        }

        @Override
        public void onPause() {

        }

        @Override
        public void onStop() {
            setSessionInactive();
        }

        @Override
        public void onSkipToNext() {

        }

        @Override
        public void onSkipToPrevious() {

        }

        @Override
        public void onFastForward() {

        }

        @Override
        public void onRewind() {

        }

        @Override
        public void onCustomAction(String action, Bundle extras) {

        }
    }

    private void setSessionActive() {
        if (!mServiceStarted) {
            startService(new Intent(getApplicationContext(), WearBrowserService.class));
            mServiceStarted = true;
        }

        if (!mSession.isActive()) {
            mSession.setActive(true);
        }
    }

    private void setSessionInactive() {
        if (mServiceStarted) {
            stopSelf();
            mServiceStarted = false;
        }

        if (mSession.isActive()) {
            mSession.setActive(false);
        }
    }

    private void addMediaRoots(List<MediaBrowser.MediaItem> mMediaRoot) {
        mMediaRoot.add(new MediaBrowser.MediaItem(
                new MediaDescription.Builder()
                        .setMediaId(Integer.toString(TYPE_ARTIST))
                        .setTitle(getString(R.string.artists))
                        .setIconUri(Uri.parse("android.resource://" +
                                "naman14.timber/drawable/ic_empty_music2"))
                        .setSubtitle(getString(R.string.artists))
                        .build(), MediaBrowser.MediaItem.FLAG_BROWSABLE
        ));

        mMediaRoot.add(new MediaBrowser.MediaItem(
                new MediaDescription.Builder()
                        .setMediaId(Integer.toString(TYPE_ALBUM))
                        .setTitle(getString(R.string.albums))
                        .setIconUri(Uri.parse("android.resource://" +
                                "naman14.timber/drawable/ic_empty_music2"))
                        .setSubtitle(getString(R.string.albums))
                        .build(), MediaBrowser.MediaItem.FLAG_BROWSABLE
        ));

        mMediaRoot.add(new MediaBrowser.MediaItem(
                new MediaDescription.Builder()
                        .setMediaId(Integer.toString(TYPE_SONG))
                        .setTitle(getString(R.string.songs))
                        .setIconUri(Uri.parse("android.resource://" +
                                "naman14.timber/drawable/ic_empty_music2"))
                        .setSubtitle(getString(R.string.songs))
                        .build(), MediaBrowser.MediaItem.FLAG_BROWSABLE
        ));


        mMediaRoot.add(new MediaBrowser.MediaItem(
                new MediaDescription.Builder()
                        .setMediaId(Integer.toString(TYPE_PLAYLIST))
                        .setTitle(getString(R.string.playlists))
                        .setIconUri(Uri.parse("android.resource://" +
                                "naman14.timber/drawable/ic_empty_music2"))
                        .setSubtitle(getString(R.string.playlists))
                        .build(), MediaBrowser.MediaItem.FLAG_BROWSABLE
        ));

    }


    private void loadChildren(final String parentId, final Result<List<MediaBrowser.MediaItem>> result) {

        final List<MediaBrowser.MediaItem> mediaItems = new ArrayList<>();

        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(final Void... unused) {

                if (parentId.equals(MEDIA_ID_ROOT)) {
                    addMediaRoots(mediaItems);
                } else {
                    switch (Integer.parseInt(Character.toString(parentId.charAt(0)))) {
                        case TYPE_ARTIST:
                            List<Artist> artistList = ArtistLoader.getAllArtists(mContext);
                            for (Artist artist : artistList) {
                                String albumNmber = TimberUtils.makeLabel(mContext, R.plurals.Nalbums, artist.albumCount);
                                String songCount = TimberUtils.makeLabel(mContext, R.plurals.Nsongs, artist.songCount);
                                fillMediaItems(mediaItems, Integer.toString(TYPE_ARTIST_SONG_ALBUMS) + Long.toString(artist.
                                        "naman14.timber/drawable/ic_empty_music2"), TimberUtils.makeCombinedString(mContext,
                            }
                            break;
                        case TYPE_ALBUM:
                            List<Album> albumList = AlbumLoader.getAllAlbums(mContext);
                            for (Album album : albumList) {
                                fillMediaItems(mediaItems, Integer.toString(TYPE_ALBUM_SONGS) + Long.toString(album.id), alb
                            }
                            break;
                        case TYPE_SONG:
                            List<Song> songList = SongLoader.getAllSongs(mContext);
                            for (Song song : songList) {
                                fillMediaItems(mediaItems, String.valueOf(song.id), song.title, TimberUtils.getAlbumArtUri(s
                            }
                            break;
                        case TYPE_ALBUM_SONGS:
                            List<Song> albumSongList = AlbumSongLoader.getSongsForAlbum(mContext, Long.parseLong(parentId.su
                            for (Song song : albumSongList) {
                                fillMediaItems(mediaItems, String.valueOf(song.id), song.title, TimberUtils.getAlbumArtUri(s
                            }
                            break;
                        case TYPE_ARTIST_SONG_ALBUMS:
                            fillMediaItems(mediaItems, Integer.toString(TYPE_ARTIST_ALL_SONGS) + Long.parseLong(parentId.sub
                                    "naman14.timber/drawable/ic_empty_music2"), "All songs by artist", MediaBrowser.MediaIte
                            List<Album> artistAlbums = ArtistAlbumLoader.getAlbumsForArtist(mContext, Long.parseLong(parentI
                            for (Album album : artistAlbums) {
                                String songCount = TimberUtils.makeLabel(mContext, R.plurals.Nsongs, album.songCount);
                                fillMediaItems(mediaItems, Integer.toString(TYPE_ALBUM_SONGS) + Long.toString(album.id), alb

                            }
                            break;
                        case TYPE_ARTIST_ALL_SONGS:
                            List<Song> artistSongs = ArtistSongLoader.getSongsForArtist(mContext, Long.parseLong(parentId.su
                            for (Song song : artistSongs) {
                                fillMediaItems(mediaItems, String.valueOf(song.id), song.title, TimberUtils.getAlbumArtUri(s
```

```java
                            }
                            break;
                        case TYPE_PLAYLIST:
                            List<Playlist> playlistList = PlaylistLoader.getPlaylists(mContext, false);
                            for (Playlist playlist : playlistList) {
                                String songCount = TimberUtils.makeLabel(mContext, R.plurals.Nsongs, playlist.songCount);
                                fillMediaItems(mediaItems, Integer.toString(TYPE_PLAYLIST_ALL_SONGS) + Long.toString(playlis
                                        Uri.parse("android.resource://" +
                                                "naman14.timber/drawable/ic_empty_music2"), songCount, MediaBrowser.MediaIte
                            }
                            break;
                        case TYPE_PLAYLIST_ALL_SONGS:
                            List<Song> playlistSongs = PlaylistSongLoader.getSongsInPlaylist(mContext, Long.parseLong(parent
                            for (Song song : playlistSongs) {
                                fillMediaItems(mediaItems, String.valueOf(song.id), song.title, TimberUtils.getAlbumArtUri(s
                            }
                            break;

                    }
                }
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                result.sendResult(mediaItems);
            }
        }.execute();

    }

    private void fillMediaItems(List<MediaBrowser.MediaItem> mediaItems, String mediaId, String title, Uri icon, String subT
        mediaItems.add(new MediaBrowser.MediaItem(
                new MediaDescription.Builder()
                        .setMediaId(mediaId)
                        .setTitle(title)
                        .setIconUri(icon)
                        .setSubtitle(subTitle)
                        .build(), playableOrBrowsable
        ));
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\activities\BaseActivity.java

```java
/*
 * Copyright (C) 2012 Andrew Neal
 * Copyright (C) 2014 The CyanogenMod Project
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber.activities;

import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.media.AudioManager;
import android.media.session.MediaSessionManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.support.v4.app.FragmentManager;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.ATEActivity;
import com.google.android.gms.cast.framework.CastButtonFactory;
import com.google.android.gms.cast.framework.CastContext;
import com.google.android.gms.cast.framework.CastSession;
import com.google.android.gms.cast.framework.Session;
import com.google.android.gms.cast.framework.SessionManager;
import com.google.android.gms.cast.framework.SessionManagerListener;
import com.google.android.gms.cast.framework.media.widget.ExpandedControllerActivity;
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GoogleApiAvailability;
import com.naman14.timber.ITimberService;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.cast.SimpleSessionManagerListener;
import com.naman14.timber.cast.WebServer;
import com.naman14.timber.listeners.MusicStateListener;
import com.naman14.timber.slidinguppanel.SlidingUpPanelLayout;
import com.naman14.timber.subfragments.QuickControlsFragment;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;

import java.io.IOException;
import java.lang.ref.WeakReference;
import java.util.ArrayList;

import static com.naman14.timber.MusicPlayer.mService;

public class BaseActivity extends ATEActivity implements ServiceConnection, MusicStateListener {

    private final ArrayList<MusicStateListener> mMusicStateListener = new ArrayList<>();
    private MusicPlayer.ServiceToken mToken;
    private PlaybackStatus mPlaybackStatus;

    private CastSession mCastSession;
    private SessionManager mSessionManager;
    private final SessionManagerListener mSessionManagerListener =
            new SessionManagerListenerImpl();
    private WebServer castServer;

    public boolean playServicesAvailable = false;

    private class SessionManagerListenerImpl extends SimpleSessionManagerListener {
        @Override
        public void onSessionStarting(Session session) {
            super.onSessionStarting(session);
            startCastServer();
        }

        @Override
        public void onSessionStarted(Session session, String sessionId) {
            invalidateOptionsMenu();
            mCastSession = mSessionManager.getCurrentCastSession();
            showCastMiniController();
        }
        @Override
        public void onSessionResumed(Session session, boolean wasSuspended) {
            invalidateOptionsMenu();
            mCastSession = mSessionManager.getCurrentCastSession();
        }
        @Override
        public void onSessionEnded(Session session, int error) {
            mCastSession = null;
            hideCastMiniController();
            stopCastServer();
        }

        @Override
        public void onSessionResuming(Session session, String s) {
            super.onSessionResuming(session, s);
            startCastServer();
        }

        @Override
        public void onSessionSuspended(Session session, int i) {
            super.onSessionSuspended(session, i);
            stopCastServer();
        }
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mToken = MusicPlayer.bindToService(this, this);

        mPlaybackStatus = new PlaybackStatus(this);
        //make volume keys change multimedia volume even if music is not playing now
        setVolumeControlStream(AudioManager.STREAM_MUSIC);

        try {
            playServicesAvailable = GoogleApiAvailability
                    .getInstance().isGooglePlayServicesAvailable(this) == ConnectionResult.SUCCESS;
        } catch (Exception ignored) {

        }

        if (playServicesAvailable)
            initCast();
    }

    @Override
    protected void onStart() {
        super.onStart();

        final IntentFilter filter = new IntentFilter();
        // Play and pause changes
        filter.addAction(MusicService.PLAYSTATE_CHANGED);
        // Track changes
        filter.addAction(MusicService.META_CHANGED);
        // Update a list, probably the playlist fragment's
        filter.addAction(MusicService.REFRESH);
        // If a playlist has changed, notify us
        filter.addAction(MusicService.PLAYLIST_CHANGED);
        // If there is an error playing a track
        filter.addAction(MusicService.TRACK_ERROR);

        registerReceiver(mPlaybackStatus, filter);

    }

    @Override
    protected void onStop() {
        super.onStop();
    }

    @Override
    public void onResume() {
        if (playServicesAvailable) {
            mCastSession = mSessionManager.getCurrentCastSession();
            mSessionManager.addSessionManagerListener(mSessionManagerListener);
        }
        //For Android 8.0+: service may get destroyed if in background too long
        if(mService == null){
            mToken = MusicPlayer.bindToService(this, this);
        }
        onMetaChanged();
        super.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (playServicesAvailable) {
            mSessionManager.removeSessionManagerListener(mSessionManagerListener);
            mCastSession = null;
        }
    }

    @Override
    public void onServiceConnected(final ComponentName name, final IBinder service) {
        mService = ITimberService.Stub.asInterface(service);
        onMetaChanged();
    }


    private void initCast() {
        CastContext castContext = CastContext.getSharedInstance(this);
        mSessionManager = castContext.getSessionManager();
    }

    @Override
    public void onServiceDisconnected(final ComponentName name) {
        mService = null;
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Unbind from the service
        if (mToken != null) {
            MusicPlayer.unbindFromService(mToken);
```

```java
            mToken = null;
        }

        try {
            unregisterReceiver(mPlaybackStatus);
        } catch (final Throwable e) {
        }
        mMusicStateListener.clear();
    }

    @Override
    public void onMetaChanged() {
        // Let the listener know to the meta chnaged
        for (final MusicStateListener listener : mMusicStateListener) {
            if (listener != null) {
                listener.onMetaChanged();
            }
        }
    }

    @Override
    public void restartLoader() {
        // Let the listener know to update a list
        for (final MusicStateListener listener : mMusicStateListener) {
            if (listener != null) {
                listener.restartLoader();
            }
        }
    }

    @Override
    public void onPlaylistChanged() {
        // Let the listener know to update a list
        for (final MusicStateListener listener : mMusicStateListener) {
            if (listener != null) {
                listener.onPlaylistChanged();
            }
        }
    }

    public void setMusicStateListenerListener(final MusicStateListener status) {
        if (status == this) {
            throw new UnsupportedOperationException("Override the method, don't add a listener");
        }

        if (status != null) {
            mMusicStateListener.add(status);
        }
    }

    public void removeMusicStateListenerListener(final MusicStateListener status) {
        if (status != null) {
            mMusicStateListener.remove(status);
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);

        getMenuInflater().inflate(R.menu.menu_cast, menu);

        if (playServicesAvailable) {
            CastButtonFactory.setUpMediaRouteButton(getApplicationContext(),
                    menu,
                    R.id.media_route_menu_item);
        }

        if (!TimberUtils.hasEffectsPanel(BaseActivity.this)) {
            menu.removeItem(R.id.action_equalizer);
        }
        ATE.applyMenu(this, getATEKey(), menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                super.onBackPressed();
                return true;
            case R.id.action_settings:
                NavigationUtils.navigateToSettings(this);
                return true;
            case R.id.action_shuffle:
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        MusicPlayer.shuffleAll(BaseActivity.this);
                    }
                }, 80);

                return true;
            case R.id.action_search:
                NavigationUtils.navigateToSearch(this);
                return true;
            case R.id.action_equalizer:
                NavigationUtils.navigateToEqualizer(this);
                return true;

        }
        return super.onOptionsItemSelected(item);
    }

    @Nullable
    @Override
    public String getATEKey() {
        return Helpers.getATEKey(this);
    }

    public void setPanelSlideListeners(SlidingUpPanelLayout panelLayout) {
        panelLayout.setPanelSlideListener(new SlidingUpPanelLayout.PanelSlideListener() {

            @Override
            public void onPanelSlide(View panel, float slideOffset) {
                View nowPlayingCard = QuickControlsFragment.topContainer;
                nowPlayingCard.setAlpha(1 - slideOffset);
            }

            @Override
            public void onPanelCollapsed(View panel) {
                View nowPlayingCard = QuickControlsFragment.topContainer;
                nowPlayingCard.setAlpha(1);
            }

            @Override
            public void onPanelExpanded(View panel) {
                View nowPlayingCard = QuickControlsFragment.topContainer;
                nowPlayingCard.setAlpha(0);
            }

            @Override
            public void onPanelAnchored(View panel) {

            }

            @Override
            public void onPanelHidden(View panel) {
            }
        });
    }

    private final static class PlaybackStatus extends BroadcastReceiver {

        private final WeakReference<BaseActivity> mReference;


        public PlaybackStatus(final BaseActivity activity) {
            mReference = new WeakReference<BaseActivity>(activity);
        }

        @Override
        public void onReceive(final Context context, final Intent intent) {
            final String action = intent.getAction();
            BaseActivity baseActivity = mReference.get();
            if (baseActivity != null) {
                if (action.equals(MusicService.META_CHANGED)) {
                    baseActivity.onMetaChanged();
                } else if (action.equals(MusicService.PLAYSTATE_CHANGED)) {
//                    baseActivity.mPlayPauseProgressButton.getPlayPauseButton().updateState();
                } else if (action.equals(MusicService.REFRESH)) {
                    baseActivity.restartLoader();
                } else if (action.equals(MusicService.PLAYLIST_CHANGED)) {
                    baseActivity.onPlaylistChanged();
                } else if (action.equals(MusicService.TRACK_ERROR)) {
                    final String errorMsg = context.getString(R.string.error_playing_track,
                            intent.getStringExtra(MusicService.TrackErrorExtra.TRACK_NAME));
                    Toast.makeText(baseActivity, errorMsg, Toast.LENGTH_SHORT).show();
                }
            }
        }
    }

    public class initQuickControls extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            QuickControlsFragment fragment1 = new QuickControlsFragment();
            FragmentManager fragmentManager1 = getSupportFragmentManager();
            fragmentManager1.beginTransaction()
                    .replace(R.id.quickcontrols_container, fragment1).commitAllowingStateLoss();
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
        }

        @Override
        protected void onPreExecute() {
        }
    }

    public void showCastMiniController() {
        //implement by overriding in activities
    }

    public void hideCastMiniController() {
        //implement by overriding in activities
    }

    public CastSession getCastSession() {
        return mCastSession;
    }

    private void startCastServer() {
        castServer = new WebServer(this);
        try {
            castServer.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void stopCastServer() {
        if (castServer != null) {
            castServer.stop();
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\activities\BaseThemedActivity.j

```
package com.naman14.timber.activities;

import android.media.AudioManager;
import android.os.Bundle;
import android.support.annotation.Nullable;

import com.afollestad.appthemeengine.ATEActivity;
import com.naman14.timber.utils.Helpers;

/**
 * Created by naman on 31/12/15.
 */
public class BaseThemedActivity extends ATEActivity {

    @Nullable
    @Override
    public String getATEKey() {
        return Helpers.getATEKey(this);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //make volume keys change multimedia volume even if music is not playing now
        setVolumeControlStream(AudioManager.STREAM_MUSIC);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\activities\DonateActivity.java

```java
package com.naman14.timber.activities;

import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import com.anjlab.android.iab.v3.BillingProcessor;
import com.anjlab.android.iab.v3.SkuDetails;
import com.anjlab.android.iab.v3.TransactionDetails;
import com.naman14.timber.R;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * Created by naman on 29/10/16.
 */
public class DonateActivity extends BaseThemedActivity implements BillingProcessor.IBillingHandler {

    private static final String DONATION_1 = "naman14.timber.donate_1";
    private static final String DONATION_2 = "naman14.timber.donate_2";
    private static final String DONATION_3 = "naman14.timber.donate_3";
    private static final String DONATION_5 = "naman14.timber.donate_5";
    private static final String DONATION_10 = "naman14.timber.donate_10";
    private static final String DONATION_20 = "naman14.timber.donate_20";


    private boolean readyToPurchase = false;
    BillingProcessor bp;

    private LinearLayout productListView;
    private ProgressBar progressBar;
    private TextView status;

    private String action = "support";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_donate);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setTitle("Support development");
        action = getIntent().getAction();

        productListView = (LinearLayout) findViewById(R.id.product_list);
        progressBar = (ProgressBar) findViewById(R.id.progressBar);
        status = (TextView) findViewById(R.id.donation_status);

        if (action != null && action.equals("restore")) {
            status.setText("Restoring purchases..");
        }

        bp = new BillingProcessor(this, getString(R.string.play_billing_license_key), this);

    }
    @Override
    public void onBillingInitialized() {
        readyToPurchase = true;
        checkStatus();
        if (!(action != null && action.equals("restore")))
            getProducts();
    }

    @Override
    public void onProductPurchased(String productId, TransactionDetails details) {
        checkStatus();
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(DonateActivity.this, "Thanks for your support!", Toast.LENGTH_SHORT).show();
            }
        });
    }

    @Override
    public void onBillingError(int errorCode, Throwable error) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(DonateActivity.this, "Unable to process purchase", Toast.LENGTH_SHORT).show();
            }
        });

    }

    @Override
    public void onPurchaseHistoryRestored() {

    }

    @Override
    public void onDestroy() {
        if (bp != null)
            bp.release();
        super.onDestroy();
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (!bp.handleActivityResult(requestCode, resultCode, data))
            super.onActivityResult(requestCode, resultCode, data);
    }

    private void checkStatus() {
        new AsyncTask<Void, Void, Boolean>() {
            @Override
            protected Boolean doInBackground(Void... voids) {
                List<String> owned = bp.listOwnedProducts();
                return owned != null && owned.size() != 0;
            }

            @Override
            protected void onPostExecute(Boolean b) {
                super.onPostExecute(b);
                if (b) {
                    PreferencesUtility.getInstance(DonateActivity.this).setFullUnlocked(true);
                    status.setText("Thanks for your support!");
                    if (action!=null && action.equals("restore")) {
                        status.setText("Your purchases has been restored. Thanks for your support");
                        progressBar.setVisibility(View.GONE);
                    }
                    if (getSupportActionBar() != null)
                        getSupportActionBar().setTitle("Support development");
                } else {
```

```
                if (action!=null && action.equals("restore")) {
                    status.setText("No previous purchase found");
                    getProducts();
                }
            }
        }
    }.execute();
}

private void getProducts() {

    new AsyncTask<Void, Void, List<SkuDetails>>() {
        @Override
        protected List<SkuDetails> doInBackground(Void... voids) {

            ArrayList<String> products = new ArrayList<>();

            products.add(DONATION_1);
            products.add(DONATION_2);
            products.add(DONATION_3);
            products.add(DONATION_5);
            products.add(DONATION_10);
            products.add(DONATION_20);

            return bp.getPurchaseListingDetails(products);
        }

        @Override
        protected void onPostExecute(List<SkuDetails> productList) {
            super.onPostExecute(productList);

            if (productList == null)
                return;

            Collections.sort(productList, new Comparator<SkuDetails>() {
                @Override
                public int compare(SkuDetails skuDetails, SkuDetails t1) {
                    if (skuDetails.priceValue >= t1.priceValue)
                        return 1;
                    else if (skuDetails.priceValue <= t1.priceValue)
                        return -1;
                    else return 0;
                }
            });
            for (int i = 0; i < productList.size(); i++) {
                final SkuDetails product = productList.get(i);
                View rootView = LayoutInflater.from(DonateActivity.this).inflate(R.layout.item_donate_product, productLi

                TextView detail = (TextView) rootView.findViewById(R.id.product_detail);
                detail.setText(product.priceText);

                rootView.findViewById(R.id.btn_donate).setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View view) {
                        if (readyToPurchase)
                            bp.purchase(DonateActivity.this, product.productId);
                        else
                            Toast.makeText(DonateActivity.this, "Unable to initiate purchase", Toast.LENGTH_SHORT).show(
                    }
                });

                productListView.addView(rootView);

            }
            progressBar.setVisibility(View.GONE);
        }
    }.execute();
}

@Override
```

```
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                super.onBackPressed();
                return true;
            default:
                break;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.activities;

import android.Manifest;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.support.design.widget.NavigationView;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TextView;

import com.afollestad.appthemeengine.customizers.ATEActivityThemeCustomizer;
import com.anjlab.android.iab.v3.BillingProcessor;
import com.google.android.gms.cast.framework.CastButtonFactory;
import com.google.android.gms.cast.framework.CastContext;
import com.google.android.gms.cast.framework.CastSession;
import com.google.android.gms.cast.framework.Session;
import com.google.android.gms.cast.framework.SessionManager;
import com.google.android.gms.cast.framework.SessionManagerListener;
import com.google.android.gms.cast.framework.media.widget.ExpandedControllerActivity;
import com.google.android.gms.cast.framework.media.widget.MiniControllerFragment;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.cast.ExpandedControlsActivity;
import com.naman14.timber.cast.SimpleSessionManagerListener;
import com.naman14.timber.cast.WebServer;
import com.naman14.timber.fragments.AlbumDetailFragment;
import com.naman14.timber.fragments.ArtistDetailFragment;
import com.naman14.timber.fragments.FoldersFragment;
import com.naman14.timber.fragments.MainFragment;
import com.naman14.timber.fragments.PlaylistFragment;
import com.naman14.timber.fragments.QueueFragment;
import com.naman14.timber.permissions.Nammu;
import com.naman14.timber.permissions.PermissionCallback;
import com.naman14.timber.slidinguppanel.SlidingUpPanelLayout;
import com.naman14.timber.subfragments.LyricsFragment;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class MainActivity extends BaseActivity implements ATEActivityThemeCustomizer {

    private SlidingUpPanelLayout panelLayout;
    private NavigationView navigationView;
    private TextView songtitle, songartist;
    private ImageView albumart;
    private String action;
    private Map<String, Runnable> navigationMap = new HashMap<String, Runnable>();
    private Handler navDrawerRunnable = new Handler();
    private Runnable runnable;
    private DrawerLayout mDrawerLayout;
    private boolean isDarkTheme;

    private Runnable navigateLibrary = new Runnable() {
        public void run() {
            navigationView.getMenu().findItem(R.id.nav_library).setChecked(true);
            Fragment fragment = new MainFragment();
            FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
            transaction.replace(R.id.fragment_container, fragment).commitAllowingStateLoss();

        }
    };

    private Runnable navigatePlaylist = new Runnable() {
        public void run() {
            navigationView.getMenu().findItem(R.id.nav_playlists).setChecked(true);
            Fragment fragment = new PlaylistFragment();
            FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
            transaction.hide(getSupportFragmentManager().findFragmentById(R.id.fragment_container));
            transaction.replace(R.id.fragment_container, fragment).commit();

        }
    };

    private Runnable navigateFolder = new Runnable() {
        public void run() {
            navigationView.getMenu().findItem(R.id.nav_folders).setChecked(true);
            Fragment fragment = new FoldersFragment();
            FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
            transaction.hide(getSupportFragmentManager().findFragmentById(R.id.fragment_container));
            transaction.replace(R.id.fragment_container, fragment).commit();

        }
    };

    private Runnable navigateQueue = new Runnable() {
        public void run() {
            navigationView.getMenu().findItem(R.id.nav_queue).setChecked(true);
            Fragment fragment = new QueueFragment();
            FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
            transaction.hide(getSupportFragmentManager().findFragmentById(R.id.fragment_container));
            transaction.replace(R.id.fragment_container, fragment).commit();

        }
    };

    private Runnable navigateAlbum = new Runnable() {
        public void run() {
            long albumID = getIntent().getExtras().getLong(Constants.ALBUM_ID);
            Fragment fragment = AlbumDetailFragment.newInstance(albumID, false, null);
            FragmentManager fragmentManager = getSupportFragmentManager();
            fragmentManager.beginTransaction()
                    .replace(R.id.fragment_container, fragment).commit();
        }
    };
    private Runnable navigateArtist = new Runnable() {
        public void run() {
            long artistID = getIntent().getExtras().getLong(Constants.ARTIST_ID);
            Fragment fragment = ArtistDetailFragment.newInstance(artistID, false, null);
            FragmentManager fragmentManager = getSupportFragmentManager();
            fragmentManager.beginTransaction()
                    .replace(R.id.fragment_container, fragment).commit();
        }
    };

    private Runnable navigateLyrics = new Runnable() {
        public void run() {
            Fragment fragment = new LyricsFragment();
            FragmentManager fragmentManager = getSupportFragmentManager();
            fragmentManager.beginTransaction()
                    .replace(R.id.fragment_container, fragment).commit();
        }
    };

    private Runnable navigateNowplaying = new Runnable() {
        public void run() {
            navigateLibrary.run();
            startActivity(new Intent(MainActivity.this, NowPlayingActivity.class));
        }
    };

    private final PermissionCallback permissionReadstorageCallback = new PermissionCallback() {
        @Override
        public void permissionGranted() {
            loadEverything();
        }

        @Override
        public void permissionRefused() {
            finish();
        }
    };


    @Override
    public void onCreate(Bundle savedInstanceState) {

        action = getIntent().getAction();

        isDarkTheme = PreferenceManager.getDefaultSharedPreferences(this).getBoolean("dark_theme", false);

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        navigationMap.put(Constants.NAVIGATE_LIBRARY, navigateLibrary);
        navigationMap.put(Constants.NAVIGATE_PLAYLIST, navigatePlaylist);
        navigationMap.put(Constants.NAVIGATE_QUEUE, navigateQueue);
        navigationMap.put(Constants.NAVIGATE_NOWPLAYING, navigateNowplaying);
        navigationMap.put(Constants.NAVIGATE_ALBUM, navigateAlbum);
        navigationMap.put(Constants.NAVIGATE_ARTIST, navigateArtist);
        navigationMap.put(Constants.NAVIGATE_LYRICS, navigateLyrics);

        mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
        panelLayout = (SlidingUpPanelLayout) findViewById(R.id.sliding_layout);

        navigationView = (NavigationView) findViewById(R.id.nav_view);
        View header = navigationView.inflateHeaderView(R.layout.nav_header);

        albumart = (ImageView) header.findViewById(R.id.album_art);
        songtitle = (TextView) header.findViewById(R.id.song_title);
        songartist = (TextView) header.findViewById(R.id.song_artist);

        setPanelSlideListeners(panelLayout);

        navDrawerRunnable.postDelayed(new Runnable() {
            @Override
            public void run() {
                setupDrawerContent(navigationView);
                setupNavigationIcons(navigationView);
            }
        }, 700);


        if (TimberUtils.isMarshmallow()) {
            checkPermissionAndThenLoad();
            //checkWritePermissions();
        } else {
            loadEverything();
        }

        addBackstackListener();

        if(Intent.ACTION_VIEW.equals(action)) {
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.clearQueue();
                    MusicPlayer.openFile(getIntent().getData().getPath());
                    MusicPlayer.playOrPause();
                    navigateNowplaying.run();
                }
            }, 350);
        }

        if (!panelLayout.isPanelHidden() && MusicPlayer.getTrackName() == null ) {
            panelLayout.hidePanel();
        }

        if (playServicesAvailable) {

            final FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(
                    FrameLayout.LayoutParams.WRAP_CONTENT,
                    FrameLayout.LayoutParams.WRAP_CONTENT);
            params.gravity = Gravity.BOTTOM;

            FrameLayout contentRoot = findViewById(R.id.content_root);
            contentRoot.addView(LayoutInflater.from(this)
                    .inflate(R.layout.fragment_cast_mini_controller, null), params);

            findViewById(R.id.castMiniController).setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    startActivity(new Intent(MainActivity.this, ExpandedControllerActivity.class));
                }
            });
        }

    }

    private void loadEverything() {
        Runnable navigation = navigationMap.get(action);
        if (navigation != null) {
            navigation.run();
        } else {
            navigateLibrary.run();
        }

        new initQuickControls().execute("");
    }

    private void checkPermissionAndThenLoad() {
        //check for permission
        if (Nammu.checkPermission(Manifest.permission.READ_EXTERNAL_STORAGE) && Nammu.checkPermission(Manifest.permission.WR
            loadEverything();
        } else {
            if (Nammu.shouldShowRequestPermissionRationale(this, Manifest.permission.READ_EXTERNAL_STORAGE)) {
                Snackbar.make(panelLayout, "Timber will need to read external storage to display songs on your device.",
                        Snackbar.LENGTH_INDEFINITE)
                        .setAction("OK", new View.OnClickListener() {
                            @Override
                            public void onClick(View view) {
                                Nammu.askForPermission(MainActivity.this, new String[]{Manifest.permission.READ_EXTERNAL_STO
                            }
                        }).show();
            } else {
                Nammu.askForPermission(this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE, Manifest.permission.WRI
            }
        }
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);

        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home: {
                if (isNavigatingMain()) {
                    mDrawerLayout.openDrawer(GravityCompat.START);
                } else super.onBackPressed();
                return true;
            }
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onBackPressed() {
        if (panelLayout.isPanelExpanded()) {
            panelLayout.collapsePanel();
        } else if (mDrawerLayout.isDrawerOpen(GravityCompat.START)) {
            mDrawerLayout.closeDrawer(GravityCompat.START);
        } else {
            super.onBackPressed();
        }
    }

    private void setupDrawerContent(NavigationView navigationView) {
        navigationView.setNavigationItemSelectedListener(
                new NavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(final MenuItem menuItem) {
                        updatePosition(menuItem);
                        return true;

                    }
                });
    }

    private void setupNavigationIcons(NavigationView navigationView) {

        //material-icon-lib currently doesn't work with navigationview of design support library 22.2.0+
        //set icons manually for now
        //https://github.com/code-mc/material-icon-lib/issues/15

        if (!isDarkTheme) {
            navigationView.getMenu().findItem(R.id.nav_library).setIcon(R.drawable.library_music);
            navigationView.getMenu().findItem(R.id.nav_playlists).setIcon(R.drawable.playlist_play);
            navigationView.getMenu().findItem(R.id.nav_queue).setIcon(R.drawable.music_note);
            navigationView.getMenu().findItem(R.id.nav_folders).setIcon(R.drawable.ic_folder_open_black_24dp);
            navigationView.getMenu().findItem(R.id.nav_nowplaying).setIcon(R.drawable.bookmark_music);
            navigationView.getMenu().findItem(R.id.nav_settings).setIcon(R.drawable.settings);
            navigationView.getMenu().findItem(R.id.nav_about).setIcon(R.drawable.information);
            navigationView.getMenu().findItem(R.id.nav_donate).setIcon(R.drawable.payment_black);
        } else {
            navigationView.getMenu().findItem(R.id.nav_library).setIcon(R.drawable.library_music_white);
            navigationView.getMenu().findItem(R.id.nav_playlists).setIcon(R.drawable.playlist_play_white);
            navigationView.getMenu().findItem(R.id.nav_queue).setIcon(R.drawable.music_note_white);
            navigationView.getMenu().findItem(R.id.nav_folders).setIcon(R.drawable.ic_folder_open_white_24dp);
            navigationView.getMenu().findItem(R.id.nav_nowplaying).setIcon(R.drawable.bookmark_music_white);
            navigationView.getMenu().findItem(R.id.nav_settings).setIcon(R.drawable.settings_white);
            navigationView.getMenu().findItem(R.id.nav_about).setIcon(R.drawable.information_white);
            navigationView.getMenu().findItem(R.id.nav_donate).setIcon(R.drawable.payment_white);
        }

        try {
            if (!BillingProcessor.isIabServiceAvailable(this)) {
                navigationView.getMenu().removeItem(R.id.nav_donate);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private void updatePosition(final MenuItem menuItem) {
        runnable = null;

        switch (menuItem.getItemId()) {
            case R.id.nav_library:
                runnable = navigateLibrary;

                break;
            case R.id.nav_playlists:
                runnable = navigatePlaylist;

                break;
            case R.id.nav_folders:
                runnable = navigateFolder;

                break;
            case R.id.nav_nowplaying:
                if (getCastSession() != null) {
                    startActivity(new Intent(MainActivity.this, ExpandedControlsActivity.class));
                } else {
                    NavigationUtils.navigateToNowplaying(MainActivity.this, false);
                }
                break;
            case R.id.nav_queue:
                runnable = navigateQueue;

                break;
            case R.id.nav_settings:
                NavigationUtils.navigateToSettings(MainActivity.this);
                break;
            case R.id.nav_about:
                mDrawerLayout.closeDrawers();
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        Helpers.showAbout(MainActivity.this);
                    }
                }, 350);

                break;
            case R.id.nav_donate:
                startActivity(new Intent(MainActivity.this, DonateActivity.class));
                break;
        }

        if (runnable != null) {
            menuItem.setChecked(true);
            mDrawerLayout.closeDrawers();
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    runnable.run();
                }
            }, 350);
        }
    }

    public void setDetailsToHeader() {
        String name = MusicPlayer.getTrackName();
        String artist = MusicPlayer.getArtistName();

        if (name != null && artist != null) {
            songtitle.setText(name);
            songartist.setText(artist);
        }
        ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(MusicPlayer.getCurrentAlbumId()).toString(), album
                new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnFail(R.drawable.ic_empty_music2)
                        .resetViewBeforeLoading(true)
                        .build());
    }

    @Override
    public void onMetaChanged() {
        super.onMetaChanged();
        setDetailsToHeader();

        if (panelLayout.isPanelHidden() && MusicPlayer.getTrackName() != null) {
            panelLayout.showPanel();
        }
    }

    @Override
    public void onRequestPermissionsResult(
            int requestCode, String[] permissions, int[] grantResults) {
        Nammu.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }

    private boolean isNavigatingMain() {
        Fragment currentFragment = getSupportFragmentManager().findFragmentById(R.id.fragment_container);
        return (currentFragment instanceof MainFragment || currentFragment instanceof QueueFragment
                || currentFragment instanceof PlaylistFragment || currentFragment instanceof FoldersFragment);
    }

    private void addBackstackListener() {
        getSupportFragmentManager().addOnBackStackChangedListener(new FragmentManager.OnBackStackChangedListener() {
            @Override
            public void onBackStackChanged() {
                getSupportFragmentManager().findFragmentById(R.id.fragment_container).onResume();
            }
        });
    }


    @Override
    public int getActivityTheme() {
        return isDarkTheme ? R.style.AppThemeNormalDark : R.style.AppThemeNormalLight;
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        getSupportFragmentManager().findFragmentById(R.id.fragment_container).onActivityResult(requestCode, resultCode, data
    }


    @Override
    public void showCastMiniController() {
        findViewById(R.id.castMiniController).setVisibility(View.VISIBLE);
        findViewById(R.id.quickcontrols_container).setVisibility(View.GONE);
        panelLayout.hidePanel();
    }

    @Override
    public void hideCastMiniController() {

        findViewById(R.id.castMiniController).setVisibility(View.GONE);
        findViewById(R.id.quickcontrols_container).setVisibility(View.VISIBLE);

        panelLayout.showPanel();
    }
}
```

```java
package com.naman14.timber.activities;

import android.content.Context;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.annotation.StyleRes;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;

import com.afollestad.appthemeengine.Config;
import com.afollestad.appthemeengine.customizers.ATEActivityThemeCustomizer;
import com.afollestad.appthemeengine.customizers.ATEStatusBarCustomizer;
import com.afollestad.appthemeengine.customizers.ATEToolbarCustomizer;
import com.naman14.timber.R;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;

/**
 * Created by naman on 01/01/16.
 */
public class NowPlayingActivity extends BaseActivity implements ATEActivityThemeCustomizer, ATEToolbarCustomizer, ATEStatusB

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nowplaying);
        SharedPreferences prefs = getSharedPreferences(Constants.FRAGMENT_ID, Context.MODE_PRIVATE);
        String fragmentID = prefs.getString(Constants.NOWPLAYING_FRAGMENT_ID, Constants.TIMBER3);

        Fragment fragment = NavigationUtils.getFragmentForNowplayingID(fragmentID);
        FragmentManager fragmentManager = getSupportFragmentManager();

        fragmentManager.beginTransaction()
                .replace(R.id.container, fragment).commit();

    }

    @StyleRes
    @Override
    public int getActivityTheme() {
        return PreferenceManager.getDefaultSharedPreferences(this).getBoolean("dark_theme", false) ? R.style.AppTheme_FullSc
    }

    @Override
    public int getLightToolbarMode() {
        return Config.LIGHT_TOOLBAR_AUTO;
    }

    @Override
    public int getLightStatusBarMode() {
        return Config.LIGHT_STATUS_BAR_OFF;
    }

    @Override
    public int getToolbarColor() {
        return Color.TRANSPARENT;
    }

    @Override
    public int getStatusBarColor() {
        return Color.TRANSPARENT;
    }

    @Override
    public void onResume() {
        super.onResume();
        if (PreferencesUtility.getInstance(this).didNowplayingThemeChanged()) {
            PreferencesUtility.getInstance(this).setNowPlayingThemeChanged(false);
            recreate();
        }
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.activities;

import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.provider.MediaStore;
import android.support.annotation.NonNull;
import android.support.annotation.StyleRes;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.transition.Transition;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.afollestad.appthemeengine.customizers.ATEActivityThemeCustomizer;
import com.afollestad.appthemeengine.customizers.ATEToolbarCustomizer;
import com.afollestad.materialdialogs.DialogAction;
import com.afollestad.materialdialogs.MaterialDialog;
import com.naman14.timber.R;
import com.naman14.timber.adapters.SongsListAdapter;
import com.naman14.timber.dataloaders.LastAddedLoader;
import com.naman14.timber.dataloaders.PlaylistLoader;
import com.naman14.timber.dataloaders.PlaylistSongLoader;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.dataloaders.TopTracksLoader;
import com.naman14.timber.listeners.SimplelTransitionListener;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.DragSortRecycler;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.util.HashMap;
import java.util.List;

public class PlaylistDetailActivity extends BaseActivity implements ATEActivityThemeCustomizer, ATEToolbarCustomizer {

    private String action;
    private long playlistID;
    private HashMap<String, Runnable> playlistsMap = new HashMap<>();
    private AppCompatActivity mContext = PlaylistDetailActivity.this;
    private SongsListAdapter mAdapter;
    private RecyclerView recyclerView;
    private ImageView blurFrame;
    private TextView playlistname;
    private View foreground;
    private boolean animate;

    private Runnable playlistLastAdded = new Runnable() {
        public void run() {
            new loadLastAdded().execute("");
        }
    };
    private Runnable playlistRecents = new Runnable() {
        @Override
        public void run() {
            new loadRecentlyPlayed().execute("");

        }
    };
    private Runnable playlistToptracks = new Runnable() {
        @Override
        public void run() {
            new loadTopTracks().execute("");
        }
    };
    private Runnable playlistUsercreated = new Runnable() {
        @Override
        public void run() {
            new loadUserCreatedPlaylist().execute("");

        }
    };

    @TargetApi(21)
    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_playlist_detail);

        action = getIntent().getAction();

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);
        getSupportActionBar().setTitle("");

        playlistsMap.put(Constants.NAVIGATE_PLAYLIST_LASTADDED, playlistLastAdded);
        playlistsMap.put(Constants.NAVIGATE_PLAYLIST_RECENT, playlistRecents);
        playlistsMap.put(Constants.NAVIGATE_PLAYLIST_TOPTRACKS, playlistToptracks);
        playlistsMap.put(Constants.NAVIGATE_PLAYLIST_USERCREATED, playlistUsercreated);

        recyclerView = (RecyclerView) findViewById(R.id.recyclerview);
        blurFrame = (ImageView) findViewById(R.id.blurFrame);
        playlistname = (TextView) findViewById(R.id.name);
        foreground = findViewById(R.id.foreground);

        recyclerView.setLayoutManager(new LinearLayoutManager(this));

        setAlbumart();

        animate = getIntent().getBooleanExtra(Constants.ACTIVITY_TRANSITION, false);
        if (animate && TimberUtils.isLollipop()) {
            getWindow().getEnterTransition().addListener(new EnterTransitionListener());
        } else {
            setUpSongs();
        }
    }

    private void setAlbumart() {
        playlistname.setText(getIntent().getExtras().getString(Constants.PLAYLIST_NAME));
        foreground.setBackgroundColor(getIntent().getExtras().getInt(Constants.PLAYLIST_FOREGROUND_COLOR));
        loadBitmap(TimberUtils.getAlbumArtUri(getIntent().getExtras().getLong(Constants.ALBUM_ID)).toString());
    }

    private void setUpSongs() {
        Runnable navigation = playlistsMap.get(action);
        if (navigation != null) {
            navigation.run();

            DragSortRecycler dragSortRecycler = new DragSortRecycler();
            dragSortRecycler.setViewHandleId(R.id.reorder);

            dragSortRecycler.setOnItemMovedListener(new DragSortRecycler.OnItemMovedListener() {
                @Override
                public void onItemMoved(int from, int to) {
                    Log.d("playlist", "onItemMoved " + from + " to " + to);
                    Song song = mAdapter.getSongAt(from);
                    mAdapter.removeSongAt(from);
                    mAdapter.addSongTo(to, song);
                    mAdapter.notifyDataSetChanged();
                    MediaStore.Audio.Playlists.Members.moveItem(getContentResolver(),
                            playlistID, from, to);
                }
            });

            recyclerView.addItemDecoration(dragSortRecycler);
            recyclerView.addOnItemTouchListener(dragSortRecycler);
            recyclerView.addOnScrollListener(dragSortRecycler.getScrollListener());

        } else {
            Log.d("PlaylistDetail", "mo action specified");
        }
    }

    private void loadBitmap(String uri) {
        ImageLoader.getInstance().displayImage(uri, blurFrame,
                new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnFail(R.drawable.ic_empty_music2)
                        .resetViewBeforeLoading(true)
                        .build());
    }

    private void setRecyclerViewAapter() {
        recyclerView.setAdapter(mAdapter);
        if (animate && TimberUtils.isLollipop()) {
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    recyclerView.addItemDecoration(new DividerItemDecoration(mContext, DividerItemDecoration.VERTICAL_LIST,
                }
            }, 250);
        } else
            recyclerView.addItemDecoration(new DividerItemDecoration(mContext, DividerItemDecoration.VERTICAL_LIST, R.drawab
    }

    @StyleRes
    @Override
    public int getActivityTheme() {
        return PreferenceManager.getDefaultSharedPreferences(this).getBoolean("dark_theme", false) ? R.style.AppTheme_FullSc

    }

    private class loadLastAdded extends AsyncTask<String, Void, String> {

        @Override
```

```java
        protected String doInBackground(String... params) {
            List<Song> lastadded = LastAddedLoader.getLastAddedSongs(mContext);
            mAdapter = new SongsListAdapter(mContext, lastadded, true, animate);
            mAdapter.setPlaylistId(playlistID);
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            setRecyclerViewAapter();
        }

        @Override
        protected void onPreExecute() {
        }
    }

    private class loadRecentlyPlayed extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            TopTracksLoader loader = new TopTracksLoader(mContext, TopTracksLoader.QueryType.RecentSongs);
            List<Song> recentsongs = SongLoader.getSongsForCursor(TopTracksLoader.getCursor());
            mAdapter = new SongsListAdapter(mContext, recentsongs, true, animate);
            mAdapter.setPlaylistId(playlistID);
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            setRecyclerViewAapter();

        }

        @Override
        protected void onPreExecute() {
        }
    }

    private class loadTopTracks extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            TopTracksLoader loader = new TopTracksLoader(mContext, TopTracksLoader.QueryType.TopTracks);
            List<Song> toptracks = SongLoader.getSongsForCursor(TopTracksLoader.getCursor());
            mAdapter = new SongsListAdapter(mContext, toptracks, true, animate);
            mAdapter.setPlaylistId(playlistID);
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            setRecyclerViewAapter();
        }

        @Override
        protected void onPreExecute() {
        }
    }

    private class loadUserCreatedPlaylist extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            playlistID = getIntent().getExtras().getLong(Constants.PLAYLIST_ID);
            List<Song> playlistsongs = PlaylistSongLoader.getSongsInPlaylist(mContext, playlistID);
            mAdapter = new SongsListAdapter(mContext, playlistsongs, true, animate);
            mAdapter.setPlaylistId(playlistID);
            return "Executed";
        }
        @Override
        protected void onPostExecute(String result) {
            setRecyclerViewAapter();
        }

        @Override
        protected void onPreExecute() {
        }
    }

    private class EnterTransitionListener extends SimplelTransitionListener {

        @TargetApi(21)
        public void onTransitionEnd(Transition paramTransition) {
            setUpSongs();
        }

        public void onTransitionStart(Transition paramTransition) {
        }

    }

    @Override
    public boolean onCreateOptionsMenu(final Menu menu) {

        getMenuInflater().inflate(R.menu.menu_playlist_detail, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        if (action.equals(Constants.NAVIGATE_PLAYLIST_USERCREATED)) {
            menu.findItem(R.id.action_delete_playlist).setVisible(true);
            menu.findItem(R.id.action_clear_auto_playlist).setVisible(false);
        } else {
            menu.findItem(R.id.action_delete_playlist).setVisible(false);
            menu.findItem(R.id.action_clear_auto_playlist).setTitle("Clear " + playlistname.getText().toString());
        }

        return super.onPrepareOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                super.onBackPressed();
                return true;
            case R.id.action_delete_playlist:
                showDeletePlaylistDialog();
                break;
            case R.id.action_clear_auto_playlist:
                clearAutoPlaylists();
                break;
            default:
                break;
        }
        return super.onOptionsItemSelected(item);
    }

    private void showDeletePlaylistDialog() {
        new MaterialDialog.Builder(this)
                .title("Delete playlist?")
                .content("Are you sure you want to delete playlist " + playlistname.getText().toString() + " ?")
                .positiveText("Delete")
                .negativeText("Cancel")
                .onPositive(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        PlaylistLoader.deletePlaylists(PlaylistDetailActivity.this, playlistID);
                        Intent returnIntent = new Intent();
                        setResult(Activity.RESULT_OK, returnIntent);
                        finish();
                    }
                })
                .onNegative(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        dialog.dismiss();
                    }
                })
                .show();
    }

    private void clearAutoPlaylists() {
        switch (action) {
            case Constants.NAVIGATE_PLAYLIST_LASTADDED:
                TimberUtils.clearLastAdded(this);
                break;
            case Constants.NAVIGATE_PLAYLIST_RECENT:
                TimberUtils.clearRecent(this);
                break;
            case Constants.NAVIGATE_PLAYLIST_TOPTRACKS:
                TimberUtils.clearTopTracks(this);
                break;
        }
        Intent returnIntent = new Intent();
        setResult(Activity.RESULT_OK, returnIntent);
        finish();
    }

    @Override
    public void onMetaChanged() {
        super.onMetaChanged();
        if (mAdapter != null)
            mAdapter.notifyDataSetChanged();
    }

    @Override
    public int getToolbarColor() {
        return Color.TRANSPARENT;
    }

    @Override
    public int getLightToolbarMode() {
        return Config.LIGHT_TOOLBAR_AUTO;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\activities\SearchActivity.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.activities;

import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.view.MenuItemCompat;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.SearchView;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.view.inputmethod.InputMethodManager;

import com.naman14.timber.R;
import com.naman14.timber.adapters.SearchAdapter;
import com.naman14.timber.dataloaders.AlbumLoader;
import com.naman14.timber.dataloaders.ArtistLoader;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.models.Album;
import com.naman14.timber.models.Artist;
import com.naman14.timber.models.Song;
import com.naman14.timber.provider.SearchHistory;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

public class SearchActivity extends BaseActivity implements SearchView.OnQueryTextListener, View.OnTouchListener {

    private final Executor mSearchExecutor = Executors.newSingleThreadExecutor();
    @Nullable
    private AsyncTask mSearchTask = null;

    private SearchView mSearchView;
    private InputMethodManager mImm;
    private String queryString;

    private SearchAdapter adapter;
    private RecyclerView recyclerView;

    private List<Object> searchResults = Collections.emptyList();

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_search);

        mImm = (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        recyclerView = (RecyclerView) findViewById(R.id.recyclerview);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        adapter = new SearchAdapter(this);
        recyclerView.setAdapter(adapter);
    }


    @Override
    public boolean onCreateOptionsMenu(final Menu menu) {

        getMenuInflater().inflate(R.menu.menu_search, menu);

        mSearchView = (SearchView) MenuItemCompat.getActionView(menu.findItem(R.id.menu_search));

        mSearchView.setOnQueryTextListener(this);
        mSearchView.setQueryHint(getString(R.string.search_library));

        mSearchView.setIconifiedByDefault(false);
        mSearchView.setIconified(false);

        MenuItemCompat.setOnActionExpandListener(menu.findItem(R.id.menu_search), new MenuItemCompat.OnActionExpandListener(
            @Override
            public boolean onMenuItemActionExpand(MenuItem item) {
                return true;
            }

            @Override
            public boolean onMenuItemActionCollapse(MenuItem item) {
                finish();
                return false;
            }
        });

        menu.findItem(R.id.menu_search).expandActionView();

        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        MenuItem item = menu.findItem(R.id.action_search);
        item.setVisible(false);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                finish();
                return true;
            default:
                break;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public boolean onQueryTextSubmit(final String query) {
        onQueryTextChange(query);
        hideInputManager();

        return true;
    }

    @Override
```

```java
    public boolean onQueryTextChange(final String newText) {

        if (newText.equals(queryString)) {
            return true;
        }
        if (mSearchTask != null) {
            mSearchTask.cancel(false);
            mSearchTask = null;
        }
        queryString = newText;
        if (queryString.trim().equals("")) {
            searchResults.clear();
            adapter.updateSearchResults(searchResults);
            adapter.notifyDataSetChanged();
        } else {
            mSearchTask = new SearchTask().executeOnExecutor(mSearchExecutor, queryString);
        }

        return true;
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        hideInputManager();
        return false;
    }

    @Override
    protected void onDestroy() {
        if (mSearchTask != null && mSearchTask.getStatus() != AsyncTask.Status.FINISHED) {
            mSearchTask.cancel(false);
        }
        super.onDestroy();
    }

    public void hideInputManager() {
        if (mSearchView != null) {
            if (mImm != null) {
                mImm.hideSoftInputFromWindow(mSearchView.getWindowToken(), 0);
            }
            mSearchView.clearFocus();

            SearchHistory.getInstance(this).addSearchString(queryString);
        }
    }

    private class SearchTask extends AsyncTask<String,Void,ArrayList<Object>> {

        @Override
        protected ArrayList<Object> doInBackground(String... params) {
            ArrayList<Object> results = new ArrayList<>(27);
            List<Song> songList = SongLoader.searchSongs(SearchActivity.this, params[0], 10);
            if (!songList.isEmpty()) {
                results.add(getString(R.string.songs));
                results.addAll(songList);
            }

            if (isCancelled()) {
                return null;
            }
            List<Album> albumList = AlbumLoader.getAlbums(SearchActivity.this, params[0], 7);
            if (!albumList.isEmpty()) {
                results.add(getString(R.string.albums));
                results.addAll(albumList);
            }

            if (isCancelled()) {
                return null;
            }
            List<Artist> artistList = ArtistLoader.getArtists(SearchActivity.this, params[0], 7);
            if (!artistList.isEmpty()) {
                results.add(getString(R.string.artists));
                results.addAll(artistList);
            }
            if (results.size() == 0) {
                results.add(getString(R.string.nothing_found));
            }
            return results;
        }

        @Override
        protected void onPostExecute(ArrayList<Object> objects) {
            super.onPostExecute(objects);
            mSearchTask = null;
            if (objects != null) {
                adapter.updateSearchResults(objects);
                adapter.notifyDataSetChanged();
            }
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\activities\SettingsActivity.jav

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.activities;

import android.os.Bundle;
import android.preference.PreferenceFragment;
import android.preference.PreferenceManager;
import android.support.annotation.ColorInt;
import android.support.annotation.NonNull;
import android.support.annotation.StyleRes;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v7.widget.Toolbar;
import android.view.MenuItem;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.Config;
import com.afollestad.appthemeengine.customizers.ATEActivityThemeCustomizer;
import com.afollestad.materialdialogs.color.ColorChooserDialog;
import com.naman14.timber.R;
import com.naman14.timber.fragments.SettingsFragment;
import com.naman14.timber.subfragments.StyleSelectorFragment;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.PreferencesUtility;

public class SettingsActivity extends BaseThemedActivity implements ColorChooserDialog.ColorCallback, ATEActivityThemeCustom

    private String action;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        if (PreferencesUtility.getInstance(this).getTheme().equals("dark"))
            setTheme(R.style.AppThemeNormalDark);
        else if (PreferencesUtility.getInstance(this).getTheme().equals("black"))
            setTheme(R.style.AppThemeNormalBlack);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        action = getIntent().getAction();

        if (action.equals(Constants.SETTINGS_STYLE_SELECTOR)) {
            getSupportActionBar().setTitle(R.string.now_playing);
            String what = getIntent().getExtras().getString(Constants.SETTINGS_STYLE_SELECTOR_WHAT);
            Fragment fragment = StyleSelectorFragment.newInstance(what);
            FragmentManager fragmentManager = getSupportFragmentManager();
            fragmentManager.beginTransaction()
                    .add(R.id.fragment_container, fragment).commit();
        } else {
            getSupportActionBar().setTitle(R.string.settings);
            PreferenceFragment fragment = new SettingsFragment();
            android.app.FragmentManager fragmentManager = getFragmentManager();
            fragmentManager.beginTransaction()
```

```
                    .replace(R.id.fragment_container, fragment).commit();
        }

    }

    @Override
    public boolean onOptionsItemSelected(final MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                finish();
                return true;
            default:
                break;
        }
        return super.onOptionsItemSelected(item);
    }

    @StyleRes
    @Override
    public int getActivityTheme() {
        return PreferenceManager.getDefaultSharedPreferences(this).getBoolean("dark_theme", false) ?
                R.style.AppThemeDark : R.style.AppThemeLight;
    }

    @Override
    public void onColorSelection(@NonNull ColorChooserDialog dialog, @ColorInt int selectedColor) {
        final Config config = ATE.config(this, getATEKey());
        switch (dialog.getTitle()) {
            case R.string.primary_color:
                config.primaryColor(selectedColor);
                break;
            case R.string.accent_color:
                config.accentColor(selectedColor);
                break;
        }
        config.commit();
        recreate(); // recreation needed to reach the checkboxes in the preferences layout
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\adapters\AlbumAdapter.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.graphics.Bitmap;
import android.support.v7.graphics.Palette;
import android.support.v7.widget.RecyclerView;
import android.util.Pair;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.R;
import com.naman14.timber.models.Album;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.display.FadeInBitmapDisplayer;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import java.util.List;

public class AlbumAdapter extends RecyclerView.Adapter<AlbumAdapter.ItemHolder> {

    private List<Album> arraylist;
    private Activity mContext;
    private boolean isGrid;

    public AlbumAdapter(Activity context, List<Album> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.isGrid = PreferencesUtility.getInstance(mContext).isAlbumsInGrid();

    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        if (isGrid) {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_album_grid, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        } else {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_album_list, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        }
    }

    @Override
    public void onBindViewHolder(final ItemHolder itemHolder, int i) {
        Album localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.title);
        itemHolder.artist.setText(localItem.artistName);

        ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(localItem.id).toString(), itemHolder.albumArt,
                new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnLoading(R.drawable.ic_empty_music2)
                        .resetViewBeforeLoading(true)
                        .displayer(new FadeInBitmapDisplayer(400))
                        .build(), new SimpleImageLoadingListener() {
                    @Override
                    public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                        if (isGrid) {
                            new Palette.Builder(loadedImage).generate(new Palette.PaletteAsyncListener() {
                                @Override
                                public void onGenerated(Palette palette) {
                                    Palette.Swatch swatch = palette.getVibrantSwatch();
                                    if (swatch != null) {
                                        int color = swatch.getRgb();
                                        itemHolder.footer.setBackgroundColor(color);
                                        int textColor = TimberUtils.getBlackWhiteColor(swatch.getTitleTextColor());
                                        itemHolder.title.setTextColor(textColor);
                                        itemHolder.artist.setTextColor(textColor);
                                    } else {
                                        Palette.Swatch mutedSwatch = palette.getMutedSwatch();
                                        if (mutedSwatch != null) {
                                            int color = mutedSwatch.getRgb();
                                            itemHolder.footer.setBackgroundColor(color);
                                            int textColor = TimberUtils.getBlackWhiteColor(mutedSwatch.getTitleTextColor());
                                            itemHolder.title.setTextColor(textColor);
                                            itemHolder.artist.setTextColor(textColor);
                                        }
                                    }


                                }
                            });
                        }

                    }

                    @Override
                    public void onLoadingFailed(String imageUri, View view, FailReason failReason) {
                        if (isGrid) {
                            itemHolder.footer.setBackgroundColor(0);
                            if (mContext != null) {
                                int textColorPrimary = Config.textColorPrimary(mContext, Helpers.getATEKey(mContext));
                                itemHolder.title.setTextColor(textColorPrimary);
                                itemHolder.artist.setTextColor(textColorPrimary);
                            }
                        }
                    }
                });

        if (TimberUtils.isLollipop())
            itemHolder.albumArt.setTransitionName("transition_album_art" + i);

    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    public void updateDataSet(List<Album> arraylist) {
        this.arraylist = arraylist;
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, artist;
        protected ImageView albumArt;
        protected View footer;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.album_title);
            this.artist = (TextView) view.findViewById(R.id.album_artist);
            this.albumArt = (ImageView) view.findViewById(R.id.album_art);
            this.footer = view.findViewById(R.id.footer);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            NavigationUtils.navigateToAlbum(mContext, arraylist.get(getAdapterPosition()).id,
                    new Pair<View, String>(albumArt, "transition_album_art" + getAdapterPosition()));
        }

    }


}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.PopupMenu;
import android.widget.TextView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;

import java.util.List;

public class AlbumSongsAdapter extends BaseSongAdapter<AlbumSongsAdapter.ItemHolder> {

    private List<Song> arraylist;
    private Activity mContext;
    private long albumID;
    private long[] songIDs;

    public AlbumSongsAdapter(Activity context, List<Song> arraylist, long albumID) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.songIDs = getSongIds();
        this.albumID = albumID;
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {

        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_album_song, null);
        ItemHolder ml = new ItemHolder(v);
        return ml;


    }

    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {

        Song localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.title);
        itemHolder.duration.setText(TimberUtils.makeShortTimeString(mContext, (localItem.duration) / 1000));
        int tracknumber = localItem.trackNumber;
        if (tracknumber == 0) {
            itemHolder.trackNumber.setText("-");
        } else itemHolder.trackNumber.setText(String.valueOf(tracknumber));

        setOnPopupMenuListener(itemHolder, i);


    }

    private void setOnPopupMenuListener(ItemHolder itemHolder, final int position) {

        itemHolder.menu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                final PopupMenu menu = new PopupMenu(mContext, v);
                menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        switch (item.getItemId()) {
                            case R.id.popup_song_play:
                                MusicPlayer.playAll(mContext, songIDs, position, -1, TimberUtils.IdType.NA, false);
                                break;
                            case R.id.popup_song_play_next:
                                long[] ids = new long[1];
                                ids[0] = arraylist.get(position).id;
                                MusicPlayer.playNext(mContext, ids, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_goto_album:
                                NavigationUtils.goToAlbum(mContext, arraylist.get(position).albumId);
                                break;
                            case R.id.popup_song_goto_artist:
                                NavigationUtils.goToArtist(mContext, arraylist.get(position).artistId);
                                break;
                            case R.id.popup_song_addto_queue:
                                long[] id = new long[1];
                                id[0] = arraylist.get(position).id;
                                MusicPlayer.addToQueue(mContext, id, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_addto_playlist:
                                AddPlaylistDialog.newInstance(arraylist.get(position)).show(((AppCompatActivity) mContext).g
                                break;
                            case R.id.popup_song_share:
                                TimberUtils.shareTrack(mContext, arraylist.get(position).id);
                                break;
                            case R.id.popup_song_delete:
                                long[] deleteIds = {arraylist.get(position).id};
                                TimberUtils.showDeleteDialog(mContext,arraylist.get(position).title, deleteIds, AlbumSongsAd
                                break;
                        }
                        return false;
                    }
                });
                menu.inflate(R.menu.popup_song);
                menu.show();
            }
        });
    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    public long[] getSongIds() {
        long[] ret = new long[getItemCount()];
        for (int i = 0; i < getItemCount(); i++) {
            ret[i] = arraylist.get(i).id;
        }

        return ret;
    }

    @Override
    public void updateDataSet(List<Song> arraylist) {
        this.arraylist = arraylist;
        this.songIDs = getSongIds();
    }

    @Override
    public void removeSongAt(int i){
        arraylist.remove(i);
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, duration, trackNumber;
        protected ImageView menu;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.song_title);
            this.duration = (TextView) view.findViewById(R.id.song_duration);
            this.trackNumber = (TextView) view.findViewById(R.id.trackNumber);
            this.menu = (ImageView) view.findViewById(R.id.popup_menu);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    playAll(mContext, songIDs, getAdapterPosition(), albumID,
                            TimberUtils.IdType.Album, false,
                            arraylist.get(getAdapterPosition()), true);
                }
            }, 100);

        }

    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\adapters\ArtistAdapter.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.support.annotation.ColorInt;
import android.support.v7.graphics.Palette;
import android.support.v7.widget.RecyclerView;
import android.util.Pair;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.R;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.ArtistInfoListener;
import com.naman14.timber.lastfmapi.models.ArtistQuery;
import com.naman14.timber.lastfmapi.models.LastfmArtist;
import com.naman14.timber.models.Artist;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.BubbleTextGetter;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.display.FadeInBitmapDisplayer;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import java.util.List;

public class ArtistAdapter extends RecyclerView.Adapter<ArtistAdapter.ItemHolder> implements BubbleTextGetter {

    private List<Artist> arraylist;
    private Activity mContext;
    private boolean isGrid;

    public ArtistAdapter(Activity context, List<Artist> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.isGrid = PreferencesUtility.getInstance(mContext).isArtistsInGrid();
    }

    public static int getOpaqueColor(@ColorInt int paramInt) {
        return 0xFF000000 | paramInt;
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        if (isGrid) {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_artist_grid, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        } else {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_artist, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        }
    }

    @Override
    public void onBindViewHolder(final ItemHolder itemHolder, int i) {
        final Artist localItem = arraylist.get(i);

        itemHolder.name.setText(localItem.name);
        String albumNmber = TimberUtils.makeLabel(mContext, R.plurals.Nalbums, localItem.albumCount);
        String songCount = TimberUtils.makeLabel(mContext, R.plurals.Nsongs, localItem.songCount);
        itemHolder.albums.setText(TimberUtils.makeCombinedString(mContext, albumNmber, songCount));


        LastFmClient.getInstance(mContext).getArtistInfo(new ArtistQuery(localItem.name), new ArtistInfoListener() {
            @Override
            public void artistInfoSucess(LastfmArtist artist) {
                if (artist != null && artist.mArtwork != null) {
                    if (isGrid) {
                        ImageLoader.getInstance().displayImage(artist.mArtwork.get(2).mUrl, itemHolder.artistImage,
                                new DisplayImageOptions.Builder().cacheInMemory(true)
                                        .cacheOnDisk(true)
                                        .showImageOnLoading(R.drawable.ic_empty_music2)
                                        .resetViewBeforeLoading(true)
                                        .displayer(new FadeInBitmapDisplayer(400))
                                        .build(), new SimpleImageLoadingListener() {
                                    @Override
                                    public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                                        if (isGrid && loadedImage != null) {
                                            new Palette.Builder(loadedImage).generate(new Palette.PaletteAsyncListener() {
                                                @Override
                                                public void onGenerated(Palette palette) {
                                                    int color = palette.getVibrantColor(Color.parseColor("#66000000"));
                                                    itemHolder.footer.setBackgroundColor(color);
                                                    Palette.Swatch swatch = palette.getVibrantSwatch();
                                                    int textColor;
                                                    if (swatch != null) {
                                                        textColor = getOpaqueColor(swatch.getTitleTextColor());
                                                    } else textColor = Color.parseColor("#ffffff");

                                                    itemHolder.name.setTextColor(textColor);
                                                    itemHolder.albums.setTextColor(textColor);
                                                }
                                            });
                                        }

                                    }

                                    @Override
                                    public void onLoadingFailed(String imageUri, View view, FailReason failReason) {
                                        if (isGrid) {
                                            itemHolder.footer.setBackgroundColor(0);
                                            if (mContext != null) {
                                                int textColorPrimary = Config.textColorPrimary(mContext, Helpers.getATEKey(m
                                                itemHolder.name.setTextColor(textColorPrimary);
                                                itemHolder.albums.setTextColor(textColorPrimary);
                                            }
                                        }
                                    }
                                });
                    } else {
                        ImageLoader.getInstance().displayImage(artist.mArtwork.get(1).mUrl, itemHolder.artistImage,
                                new DisplayImageOptions.Builder().cacheInMemory(true)
                                        .cacheOnDisk(true)
                                        .showImageOnLoading(R.drawable.ic_empty_music2)
                                        .resetViewBeforeLoading(true)
                                        .displayer(new FadeInBitmapDisplayer(400))
                                        .build());
                    }
                }
            }

            @Override
            public void artistInfoFailed() {

            }
        });

        if (TimberUtils.isLollipop())
            itemHolder.artistImage.setTransitionName("transition_artist_art" + i);

    }

    @Override
    public long getItemId(int position) {
        return arraylist.get(position).id;
    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    @Override
    public String getTextToShowInBubble(final int pos) {
        if (arraylist == null || arraylist.size() == 0)
            return "";
        return Character.toString(arraylist.get(pos).name.charAt(0));
    }

    public void updateDataSet(List<Artist> arrayList) {
        this.arraylist = arrayList;
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView name, albums;
        protected ImageView artistImage;
        protected View footer;

        public ItemHolder(View view) {
            super(view);
            this.name = (TextView) view.findViewById(R.id.artist_name);
            this.albums = (TextView) view.findViewById(R.id.album_song_count);
            this.artistImage = (ImageView) view.findViewById(R.id.artistImage);
            this.footer = view.findViewById(R.id.footer);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            NavigationUtils.navigateToArtist(mContext, arraylist.get(getAdapterPosition()).id,
                    new Pair<View, String>(artistImage, "transition_artist_art" + getAdapterPosition()));
        }

    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.support.v7.widget.CardView;
import android.support.v7.widget.RecyclerView;
import android.util.Pair;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.naman14.timber.R;
import com.naman14.timber.models.Album;
import com.naman14.timber.utils.ImageUtils;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;

import java.util.List;

public class ArtistAlbumAdapter extends RecyclerView.Adapter<ArtistAlbumAdapter.ItemHolder> {

    private List<Album> arraylist;
    private Activity mContext;

    public ArtistAlbumAdapter(Activity context, List<Album> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;

    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_artist_album, null);
        ItemHolder ml = new ItemHolder(v);
        return ml;
    }

    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {

        Album localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.title);
        String songCount = TimberUtils.makeLabel(mContext, R.plurals.Nsongs, localItem.songCount);
        itemHolder.details.setText(songCount);

        ImageUtils.loadAlbumArtIntoView(localItem.id, itemHolder.albumArt);

        if (TimberUtils.isLollipop())
            itemHolder.albumArt.setTransitionName("transition_album_art" + i);

    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }


    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, details;
        protected ImageView albumArt;
        protected CardView rootView;

        public ItemHolder(View view) {
            super(view);
            this.rootView = (CardView) view.findViewById(R.id.root_view);
            this.title = (TextView) view.findViewById(R.id.album_title);
            this.details = (TextView) view.findViewById(R.id.album_details);
            this.albumArt = (ImageView) view.findViewById(R.id.album_art);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            NavigationUtils.navigateToAlbum(mContext, arraylist.get(getAdapterPosition()).id,
                    new Pair<View, String>(albumArt, "transition_album_art" + getAdapterPosition()));
        }

    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\adapters\ArtistSongAdapter.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.graphics.Rect;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.PopupMenu;
import android.widget.TextView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dataloaders.ArtistAlbumLoader;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.util.ArrayList;
import java.util.List;

public class ArtistSongAdapter extends BaseSongAdapter<ArtistSongAdapter.ItemHolder> {

    private List<Song> arraylist;
    private Activity mContext;
    private long artistID;
    private long[] songIDs;

    public ArtistSongAdapter(Activity context, List<Song> arraylist, long artistID) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.artistID = artistID;
        this.songIDs = getSongIds();
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        if (viewType == 0) {
            View v0 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.artist_detail_albums_header, null);
            ItemHolder ml = new ItemHolder(v0);
            return ml;
        } else {
            View v2 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_artist_song, null);
            ItemHolder ml = new ItemHolder(v2);
            return ml;
        }
    }
    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {

        if (getItemViewType(i) == 0) {
            //nothing
            setUpAlbums(itemHolder.albumsRecyclerView);
        } else {
            Song localItem = arraylist.get(i);
            itemHolder.title.setText(localItem.title);
            itemHolder.album.setText(localItem.albumName);

            ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(localItem.albumId).toString(),
                    itemHolder.albumArt, new DisplayImageOptions.Builder()
                            .cacheInMemory(true).showImageOnLoading(R.drawable.ic_empty_music2).resetViewBeforeLoading(true)
            setOnPopupMenuListener(itemHolder, i - 1);
        }

    }

    @Override
    public void onViewRecycled(ItemHolder itemHolder) {

        if (itemHolder.getItemViewType() == 0)
            clearExtraSpacingBetweenCards(itemHolder.albumsRecyclerView);

    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    private void setOnPopupMenuListener(ItemHolder itemHolder, final int position) {

        itemHolder.menu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                final PopupMenu menu = new PopupMenu(mContext, v);
                menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        switch (item.getItemId()) {
                            case R.id.popup_song_play:
                                MusicPlayer.playAll(mContext, songIDs, position + 1, -1, TimberUtils.IdType.NA, false);
                                break;
                            case R.id.popup_song_play_next:
                                long[] ids = new long[1];
                                ids[0] = arraylist.get(position + 1).id;
                                MusicPlayer.playNext(mContext, ids, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_goto_album:
                                NavigationUtils.goToAlbum(mContext, arraylist.get(position + 1).albumId);
                                break;
                            case R.id.popup_song_goto_artist:
                                NavigationUtils.goToArtist(mContext, arraylist.get(position + 1).artistId);
                                break;
                            case R.id.popup_song_addto_queue:
                                long[] id = new long[1];
                                id[0] = arraylist.get(position + 1).id;
                                MusicPlayer.addToQueue(mContext, id, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_addto_playlist:
                                AddPlaylistDialog.newInstance(arraylist.get(position + 1)).show(((AppCompatActivity) mContex
                                break;
                            case R.id.popup_song_share:
                                TimberUtils.shareTrack(mContext, arraylist.get(position + 1).id);
                                break;
                            case R.id.popup_song_delete:
                                long[] deleteIds = {arraylist.get(position + 1).id};
                                TimberUtils.showDeleteDialog(mContext,arraylist.get(position + 1).title, deleteIds, ArtistSo
                                break;
                        }
                        return false;
                    }
                });
                menu.inflate(R.menu.popup_song);
                menu.show();
            }
        });
    }

    private void setUpAlbums(RecyclerView albumsRecyclerview) {

        albumsRecyclerview.setLayoutManager(new LinearLayoutManager(mContext, LinearLayoutManager.HORIZONTAL, false));
        albumsRecyclerview.setHasFixedSize(true);

        //to add spacing between cards
        int spacingInPixels = mContext.getResources().getDimensionPixelSize(R.dimen.spacing_card);
        albumsRecyclerview.addItemDecoration(new SpacesItemDecoration(spacingInPixels));
        albumsRecyclerview.setNestedScrollingEnabled(false);


        ArtistAlbumAdapter mAlbumAdapter = new ArtistAlbumAdapter(mContext, ArtistAlbumLoader.getAlbumsForArtist(mContext, a
        albumsRecyclerview.setAdapter(mAlbumAdapter);
    }

    private void clearExtraSpacingBetweenCards(RecyclerView albumsRecyclerview) {
        //to clear any extra spacing between cards
        int spacingInPixelstoClear = -(mContext.getResources().getDimensionPixelSize(R.dimen.spacing_card));
        albumsRecyclerview.addItemDecoration(new SpacesItemDecoration(spacingInPixelstoClear));

    }

    public long[] getSongIds() {
        List<Song> actualArraylist = new ArrayList<Song>(arraylist);
        //actualArraylist.remove(0);
        long[] ret = new long[actualArraylist.size()];
        for (int i = 0; i < actualArraylist.size(); i++) {
            ret[i] = actualArraylist.get(i).id;
        }
        return ret;
    }

    @Override
    public void removeSongAt(int i){
        arraylist.remove(i);
        updateDataSet(arraylist);
    }

    @Override
    public void updateDataSet(List<Song> arraylist) {
        this.arraylist = arraylist;
        this.songIDs = getSongIds();
    }

    @Override
    public int getItemViewType(int position) {
        int viewType;
        if (position == 0) {
            viewType = 0;
        } else viewType = 1;
        return viewType;
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, album;
        protected ImageView albumArt, menu;
        protected RecyclerView albumsRecyclerView;
```

```java
        public ItemHolder(View view) {
            super(view);

            this.albumsRecyclerView = (RecyclerView) view.findViewById(R.id.recycler_view_album);

            this.title = (TextView) view.findViewById(R.id.song_title);
            this.album = (TextView) view.findViewById(R.id.song_album);
            this.albumArt = (ImageView) view.findViewById(R.id.albumArt);
            this.menu = (ImageView) view.findViewById(R.id.popup_menu);


            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    playAll(mContext, songIDs, getAdapterPosition(), artistID,
                            TimberUtils.IdType.Artist, false,
                            arraylist.get(getAdapterPosition()), true);
                }
            }, 100);

        }

    }

    public class SpacesItemDecoration extends RecyclerView.ItemDecoration {
        private int space;

        public SpacesItemDecoration(int space) {
            this.space = space;
        }

        @Override
        public void getItemOffsets(Rect outRect, View view,
                                   RecyclerView parent, RecyclerView.State state) {

            //the padding from left
            outRect.left = space;


        }
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.PopupMenu;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.MusicVisualizer;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.util.List;

public class BaseQueueAdapter extends RecyclerView.Adapter<BaseQueueAdapter.ItemHolder> {

    public static int currentlyPlayingPosition;
    private List<Song> arraylist;
    private AppCompatActivity mContext;
    private String ateKey;

    public BaseQueueAdapter(AppCompatActivity context, List<Song> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;
        currentlyPlayingPosition = MusicPlayer.getQueuePosition();
        this.ateKey = Helpers.getATEKey(context);
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_song_timber1, null);
        ItemHolder ml = new ItemHolder(v);
        return ml;
    }

    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {
        Song localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.title);
        itemHolder.artist.setText(localItem.artistName);

        if (MusicPlayer.getCurrentAudioId() == localItem.id) {
            itemHolder.title.setTextColor(Config.accentColor(mContext, ateKey));
            if (MusicPlayer.isPlaying()) {
                itemHolder.visualizer.setColor(Config.accentColor(mContext, ateKey));
                itemHolder.visualizer.setVisibility(View.VISIBLE);
            } else {
                itemHolder.visualizer.setVisibility(View.GONE);
            }
        } else {
            itemHolder.title.setTextColor(Config.textColorPrimary(mContext, ateKey));
            itemHolder.visualizer.setVisibility(View.GONE);
        }
        ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(localItem.albumId).toString(),
                itemHolder.albumArt, new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnLoading(R.drawable.ic_empty_music2).resetViewBeforeLoading(true).build());
        setOnPopupMenuListener(itemHolder, i);
    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    private void setOnPopupMenuListener(ItemHolder itemHolder, final int position) {

        itemHolder.popupMenu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                final PopupMenu menu = new PopupMenu(mContext, v);
                menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        switch (item.getItemId()) {
                            case R.id.popup_song_play:
                                MusicPlayer.playAll(mContext, getSongIds(), position, -1, TimberUtils.IdType.NA, false);
                                break;
                            case R.id.popup_song_play_next:
                                long[] ids = new long[1];
                                ids[0] = arraylist.get(position).id;
                                MusicPlayer.playNext(mContext, ids, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_goto_album:
                                NavigationUtils.goToAlbum(mContext, arraylist.get(position).albumId);
                                break;
                            case R.id.popup_song_goto_artist:
                                NavigationUtils.goToArtist(mContext, arraylist.get(position).artistId);
                                break;
                            case R.id.popup_song_addto_queue:
                                long[] id = new long[1];
                                id[0] = arraylist.get(position).id;
                                MusicPlayer.addToQueue(mContext, id, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_addto_playlist:
                                AddPlaylistDialog.newInstance(arraylist.get(position)).show(mContext.getSupportFragmentManag
                                break;
                            case R.id.popup_song_share:
                                TimberUtils.shareTrack(mContext, arraylist.get(position).id);
                                break;
                            case R.id.popup_song_delete:
                                long[] deleteIds = {arraylist.get(position).id};
                                TimberUtils.showDeleteDialog(mContext,arraylist.get(position).title, deleteIds, BaseQueueAda
                                break;
                        }
                        return false;
                    }
                });
                menu.inflate(R.menu.popup_song);
                menu.show();
            }
        });
    }

    public long[] getSongIds() {
        long[] ret = new long[getItemCount()];
        for (int i = 0; i < getItemCount(); i++) {
            ret[i] = arraylist.get(i).id;
        }

        return ret;
    }

    public void removeSongAt(int i){
        arraylist.remove(i);
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, artist;
        protected ImageView albumArt, popupMenu;
        private MusicVisualizer visualizer;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.song_title);
            this.artist = (TextView) view.findViewById(R.id.song_artist);
            this.albumArt = (ImageView) view.findViewById(R.id.albumArt);
            this.popupMenu = (ImageView) view.findViewById(R.id.popup_menu);
            visualizer = (MusicVisualizer) view.findViewById(R.id.visualizer);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            final Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.setQueuePosition(getAdapterPosition());
                    Handler handler1 = new Handler();
                    handler1.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            notifyItemChanged(currentlyPlayingPosition);
                            notifyItemChanged(getAdapterPosition());
                        }
                    }, 50);
                }
            }, 100);

        }

    }

}
```

```java
package com.naman14.timber.adapters;

import android.app.Activity;
import android.net.Uri;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.View;
import android.view.ViewGroup;

import com.google.android.gms.cast.MediaInfo;
import com.google.android.gms.cast.MediaMetadata;
import com.google.android.gms.cast.framework.CastSession;
import com.google.android.gms.cast.framework.media.RemoteMediaClient;
import com.google.android.gms.common.images.WebImage;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.activities.BaseActivity;
import com.naman14.timber.activities.MainActivity;
import com.naman14.timber.cast.TimberCastHelper;
import com.naman14.timber.cast.WebServer;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;

import java.io.IOException;
import java.util.List;

/**
 * Created by naman on 7/12/17.
 */

public class BaseSongAdapter<V extends RecyclerView.ViewHolder> extends RecyclerView.Adapter<V> {

    @Override
    public V onCreateViewHolder(ViewGroup parent, int viewType) {
        return null;
    }

    @Override
    public void onBindViewHolder(V holder, int position) {

    }

    @Override
    public int getItemCount() {
        return 0;
    }

    @Override
    public int getItemViewType(int position) {
        return super.getItemViewType(position);
    }

    public class ItemHolder extends RecyclerView.ViewHolder {

        public ItemHolder(View view) {
            super(view);
        }

    }

    public void playAll(final Activity context, final long[] list, int position,
                        final long sourceId, final TimberUtils.IdType sourceType,
                        final boolean forceShuffle, final Song currentSong, boolean navigateNowPlaying) {

        if (context instanceof BaseActivity) {
            CastSession castSession = ((BaseActivity) context).getCastSession();
            if (castSession != null) {
                navigateNowPlaying = false;
                TimberCastHelper.startCasting(castSession, currentSong);
            } else {
                MusicPlayer.playAll(context, list, position, -1, TimberUtils.IdType.NA, false);
            }
        } else {
            MusicPlayer.playAll(context, list, position, -1, TimberUtils.IdType.NA, false);
        }

        if (navigateNowPlaying) {
            NavigationUtils.navigateToNowplaying(context, true);
        }


    }
    public void removeSongAt(int i){}
    public void updateDataSet(List<Song> arraylist) {}

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\adapters\FolderAdapter.java

```java
package com.naman14.timber.adapters;

import android.app.Activity;
import android.graphics.Color;
import android.graphics.ColorFilter;
import android.graphics.PorterDuff;
import android.graphics.PorterDuffColorFilter;
import android.graphics.drawable.Drawable;
import android.os.AsyncTask;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.support.v4.content.ContextCompat;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dataloaders.FolderLoader;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.BubbleTextGetter;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

/**
 * Created by nv95 on 10.11.16.
 */

public class FolderAdapter extends BaseSongAdapter<FolderAdapter.ItemHolder> implements BubbleTextGetter {

    @NonNull
    private List<File> mFileSet;
    private List<Song> mSongs;
    private File mRoot;
    private Activity mContext;
    private final Drawable[] mIcons;
    private boolean mBusy = false;


    public FolderAdapter(Activity context, File root) {
        mContext = context;
        mIcons = new Drawable[]{
                ContextCompat.getDrawable(context, R.drawable.ic_folder_open_black_24dp),
                ContextCompat.getDrawable(context, R.drawable.ic_folder_parent_dark),
                ContextCompat.getDrawable(context, R.drawable.ic_file_music_dark),
                ContextCompat.getDrawable(context, R.drawable.ic_timer_wait)
        };
        mSongs = new ArrayList<>();
        updateDataSet(root);
    }

    public void applyTheme(boolean dark) {
        ColorFilter cf = new PorterDuffColorFilter(Color.WHITE, PorterDuff.Mode.SRC_ATOP);
        for (Drawable d : mIcons) {
            if (dark) {
                d.setColorFilter(cf);
            } else {
                d.clearColorFilter();
            }
        }
    }

    @Override
    public FolderAdapter.ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_folder_list, viewGroup, false);
        return new ItemHolder(v);
    }

    @Override
    public void onBindViewHolder(final FolderAdapter.ItemHolder itemHolder, int i) {
        File localItem = mFileSet.get(i);
        Song song = mSongs.get(i);
        itemHolder.title.setText(localItem.getName());
        if (localItem.isDirectory()) {
            itemHolder.albumArt.setImageDrawable("..".equals(localItem.getName()) ? mIcons[1] : mIcons[0]);
        } else {
            ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(song.albumId).toString(),
                    itemHolder.albumArt,
                    new DisplayImageOptions.Builder().
                            cacheInMemory(true).showImageOnFail(mIcons[2])
                            .resetViewBeforeLoading(true).build());
        }
    }

    @Override
    public int getItemCount() {
        return mFileSet.size();
    }

    @Deprecated
    public void updateDataSet(File newRoot) {
        if (mBusy) {
            return;
        }
        if ("..".equals(newRoot.getName())) {
            goUp();
            return;
        }
        mRoot = newRoot;
        mFileSet = FolderLoader.getMediaFiles(newRoot, true);
        getSongsForFiles(mFileSet);
    }

    @Deprecated
    public boolean goUp() {
        if (mRoot == null || mBusy) {
            return false;
        }
        File parent = mRoot.getParentFile();
        if (parent != null && parent.canRead()) {
            updateDataSet(parent);
            return true;
        } else {
            return false;
        }
    }

    public boolean goUpAsync() {
        if (mRoot == null || mBusy) {
            return false;
        }
        File parent = mRoot.getParentFile();
        if (parent != null && parent.canRead()) {
            return updateDataSetAsync(parent);
        } else {
            return false;
        }
    }

    public boolean updateDataSetAsync(File newRoot) {
        if (mBusy) {
            return false;
        }
        if ("..".equals(newRoot.getName())) {
            goUpAsync();
            return false;
        }
        mRoot = newRoot;
        new NavigateTask().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, mRoot);
        return true;
    }

    @Override
    public String getTextToShowInBubble(int pos) {
        if (mBusy || mFileSet.size() == 0)
            return "";
        try {
            File f = mFileSet.get(pos);
            if (f.isDirectory()) {
                return String.valueOf(f.getName().charAt(0));
            } else {
                return Character.toString(f.getName().charAt(0));
            }
        } catch (Exception e) {
            return "";
        }
    }

    private void getSongsForFiles(List<File> files) {
        mSongs.clear();
        for (File file : files) {
            mSongs.add(SongLoader.getSongFromPath(file.getAbsolutePath(), mContext));
        }
    }


    private class NavigateTask extends AsyncTask<File, Void, List<File>> {

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            mBusy = true;
        }

        @Override
        protected List<File> doInBackground(File... params) {
            List<File> files = FolderLoader.getMediaFiles(params[0], true);
            getSongsForFiles(files);
            return files;
        }

        @Override
        protected void onPostExecute(List<File> files) {
            super.onPostExecute(files);
            mFileSet = files;
            notifyDataSetChanged();
            mBusy = false;
            PreferencesUtility.getInstance(mContext).storeLastFolder(mRoot.getPath());
        }
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {

        protected TextView title;
        protected ImageView albumArt;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.folder_title);
            this.albumArt = (ImageView) view.findViewById(R.id.album_art);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            if (mBusy) {
                return;
            }
            final File f = mFileSet.get(getAdapterPosition());

            if (f.isDirectory() && updateDataSetAsync(f)) {
                albumArt.setImageDrawable(mIcons[3]);
            } else if (f.isFile()) {

                final Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        int current = -1;
                        long songId = SongLoader.getSongFromPath(mFileSet.get(getAdapterPosition()).getAbsolutePath(),mConte
                        int count = 0;
                        for (Song song : mSongs) {
                            if (song.id != -1) {
                                count++;
                            }
                        }
                        long[] ret = new long[count];
                        int j = 0;
                        for (int i = 0; i < getItemCount(); i++) {
                            if (mSongs.get(i).id != -1) {
                                ret[j] = mSongs.get(i).id;
                                if (mSongs.get(i).id == songId) {
                                    current = j;
                                }
                                j++;
                            }
                        }
                        playAll(mContext, ret, current, -1, TimberUtils.IdType.NA,
                                false, mSongs.get(getAdapterPosition()), false);
                    }
                }, 100);


            }
        }

    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\adapters\PlayingQueueAdapter.ja

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.PopupMenu;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.MusicVisualizer;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.util.List;

public class PlayingQueueAdapter extends RecyclerView.Adapter<PlayingQueueAdapter.ItemHolder> {
    private static final String TAG = "PlayingQueueAdapter";

    public int currentlyPlayingPosition;
    private List<Song> arraylist;
    private Activity mContext;
    private String ateKey;

    public PlayingQueueAdapter(Activity context, List<Song> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.currentlyPlayingPosition = MusicPlayer.getQueuePosition();
        this.ateKey = Helpers.getATEKey(context);
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_playing_queue, null);
        ItemHolder ml = new ItemHolder(v);
        return ml;
    }

    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {
        Song localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.title);
        itemHolder.artist.setText(localItem.artistName);

        if (MusicPlayer.getCurrentAudioId() == localItem.id) {
            itemHolder.title.setTextColor(Config.accentColor(mContext, ateKey));
            if (MusicPlayer.isPlaying()) {
                itemHolder.visualizer.setColor(Config.accentColor(mContext, ateKey));
                itemHolder.visualizer.setVisibility(View.VISIBLE);
            } else {
                itemHolder.visualizer.setVisibility(View.GONE);
            }
        } else {
            itemHolder.title.setTextColor(Config.textColorPrimary(mContext, ateKey));
            itemHolder.visualizer.setVisibility(View.GONE);
        }
        ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(localItem.albumId).toString(),
                itemHolder.albumArt, new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnLoading(R.drawable.ic_empty_music2).resetViewBeforeLoading(true).build());
        setOnPopupMenuListener(itemHolder, i);
    }

    private void setOnPopupMenuListener(ItemHolder itemHolder, final int position) {

        itemHolder.menu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                final PopupMenu menu = new PopupMenu(mContext, v);
                menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        switch (item.getItemId()) {
                            case R.id.popup_song_remove_queue:
                                Log.v(TAG,"Removing " + position);
                                MusicPlayer.removeTrackAtPosition(getSongAt(position).id, position);
                                removeSongAt(position);
                                notifyItemRemoved(position);
                                break;
                            case R.id.popup_song_play:
                                MusicPlayer.playAll(mContext, getSongIds(), position, -1, TimberUtils.IdType.NA, false);
                                break;
                            case R.id.popup_song_goto_album:
                                NavigationUtils.goToAlbum(mContext, arraylist.get(position).albumId);
                                break;
                            case R.id.popup_song_goto_artist:
                                NavigationUtils.goToArtist(mContext, arraylist.get(position).artistId);
                                break;
                            case R.id.popup_song_addto_playlist:
                                AddPlaylistDialog.newInstance(arraylist.get(position)).show(((AppCompatActivity) mContext).g
                                break;
                        }
                        return false;
                    }
                });
                menu.inflate(R.menu.popup_playing_queue);
                menu.show();
            }
        });
    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    public long[] getSongIds() {
        long[] ret = new long[getItemCount()];
        for (int i = 0; i < getItemCount(); i++) {
            ret[i] = arraylist.get(i).id;
        }
        return ret;
    }

    public Song getSongAt(int i) {
        return arraylist.get(i);
    }

    public void addSongTo(int i, Song song) {
        arraylist.add(i, song);
    }

    public void removeSongAt(int i) {
        arraylist.remove(i);
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, artist;
        protected ImageView albumArt, reorder, menu;
        private MusicVisualizer visualizer;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.song_title);
            this.artist = (TextView) view.findViewById(R.id.song_artist);
            this.albumArt = (ImageView) view.findViewById(R.id.albumArt);
            this.menu = (ImageView) view.findViewById(R.id.popup_menu);
            this.reorder = (ImageView) view.findViewById(R.id.reorder);
            visualizer = (MusicVisualizer) view.findViewById(R.id.visualizer);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            final Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.setQueuePosition(getAdapterPosition());
                    Handler handler1 = new Handler();
                    handler1.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            notifyItemChanged(currentlyPlayingPosition);
                            notifyItemChanged(getAdapterPosition());
                        }
                    }, 50);
                }
            }, 100);

        }

    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\adapters\PlaylistAdapter.java

```java
package com.naman14.timber.adapters;

import android.app.Activity;
import android.graphics.Bitmap;
import android.support.v7.graphics.Palette;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.R;
import com.naman14.timber.dataloaders.LastAddedLoader;
import com.naman14.timber.dataloaders.PlaylistSongLoader;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.dataloaders.TopTracksLoader;
import com.naman14.timber.models.Playlist;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import java.util.List;
import java.util.Random;

/**
 * Created by naman on 31/10/16.
 */
public class PlaylistAdapter extends RecyclerView.Adapter<PlaylistAdapter.ItemHolder> {

    private List<Playlist> arraylist;
    private Activity mContext;
    private boolean isGrid;
    private boolean showAuto;
    private int songCountInt;
    private long totalRuntime;
    private long firstAlbumID = -1;
    private int foregroundColor;
    int[] foregroundColors = {R.color.pink_transparent, R.color.green_transparent, R.color.blue_transparent, R.color.red_tra

    public PlaylistAdapter(Activity context, List<Playlist> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.isGrid = PreferencesUtility.getInstance(mContext).getPlaylistView() == Constants.PLAYLIST_VIEW_GRID;
        this.showAuto = PreferencesUtility.getInstance(mContext).showAutoPlaylist();
        Random random = new Random();
        int rndInt = random.nextInt(foregroundColors.length);
        foregroundColor = foregroundColors[rndInt];

    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        if (isGrid) {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_album_grid, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        } else {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_album_list, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        }
    }

    @Override
    public void onBindViewHolder(final ItemHolder itemHolder, int i) {
        final Playlist localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.name);

        String s = getAlbumArtUri(i, localItem.id);
        itemHolder.albumArt.setTag(firstAlbumID);
        ImageLoader.getInstance().displayImage(s, itemHolder.albumArt,
                new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnFail(R.drawable.ic_empty_music2)
                        .resetViewBeforeLoading(true)
                        .build(), new SimpleImageLoadingListener() {
                    @Override
                    public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                        if (isGrid) {
                            new Palette.Builder(loadedImage).generate(new Palette.PaletteAsyncListener() {
                                @Override
                                public void onGenerated(Palette palette) {
                                    Palette.Swatch swatch = palette.getVibrantSwatch();
                                    if (swatch != null) {
                                        int color = swatch.getRgb();
                                        itemHolder.footer.setBackgroundColor(color);
                                        int textColor = TimberUtils.getBlackWhiteColor(swatch.getTitleTextColor());
                                        itemHolder.title.setTextColor(textColor);
                                        itemHolder.artist.setTextColor(textColor);
                                    } else {
                                        Palette.Swatch mutedSwatch = palette.getMutedSwatch();
                                        if (mutedSwatch != null) {
                                            int color = mutedSwatch.getRgb();
                                            itemHolder.footer.setBackgroundColor(color);
                                            int textColor = TimberUtils.getBlackWhiteColor(mutedSwatch.getTitleTextColor());
                                            itemHolder.title.setTextColor(textColor);
                                            itemHolder.artist.setTextColor(textColor);
                                        }
                                    }


                                }
                            });
                        }

                    }

                    @Override
                    public void onLoadingFailed(String imageUri, View view, FailReason failReason) {
                        if (isGrid) {
                            itemHolder.footer.setBackgroundColor(0);
                            if (mContext != null) {
                                int textColorPrimary = Config.textColorPrimary(mContext, Helpers.getATEKey(mContext));
                                itemHolder.title.setTextColor(textColorPrimary);
                                itemHolder.artist.setTextColor(textColorPrimary);
                            }
                        }
                    }
                });
        itemHolder.artist.setText(" " + String.valueOf(songCountInt) + " " + mContext.getString(R.string.songs) + " - " + Ti

        if (TimberUtils.isLollipop())
            itemHolder.albumArt.setTransitionName("transition_album_art" + i);

    }

    private String getAlbumArtUri(int position, long id) {
        if (mContext != null) {
            firstAlbumID = -1;
            if (showAuto) {
                switch (position) {
```

```
                case 0:
                    List<Song> lastAddedSongs = LastAddedLoader.getLastAddedSongs(mContext);
                    songCountInt = lastAddedSongs.size();
                    totalRuntime = 0;
                    for(Song song : lastAddedSongs){
                            totalRuntime += song.duration / 1000; //for some reason default playlists have songs with du
                    }

                    if (songCountInt != 0) {
                        firstAlbumID = lastAddedSongs.get(0).albumId;
                        return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                    } else return "nosongs";
                case 1:
                    TopTracksLoader recentloader = new TopTracksLoader(mContext, TopTracksLoader.QueryType.RecentSongs);
                    List<Song> recentsongs = SongLoader.getSongsForCursor(TopTracksLoader.getCursor());
                    songCountInt = recentsongs.size();
                    totalRuntime = 0;
                    for(Song song : recentsongs){
                        totalRuntime += song.duration / 1000; //for some reason default playlists have songs with durati
                    }

                    if (songCountInt != 0) {
                        firstAlbumID = recentsongs.get(0).albumId;
                        return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                    } else return "nosongs";
                case 2:
                    TopTracksLoader topTracksLoader = new TopTracksLoader(mContext, TopTracksLoader.QueryType.TopTracks)
                    List<Song> topsongs = SongLoader.getSongsForCursor(TopTracksLoader.getCursor());
                    songCountInt = topsongs.size();
                    totalRuntime = 0;
                    for(Song song : topsongs){
                        totalRuntime += song.duration / 1000; //for some reason default playlists have songs with durati
                    }

                    if (songCountInt != 0) {
                        firstAlbumID = topsongs.get(0).albumId;
                        return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                    } else return "nosongs";
                default:
                    List<Song> playlistsongs = PlaylistSongLoader.getSongsInPlaylist(mContext, id);
                    songCountInt = playlistsongs.size();
                    totalRuntime = 0;
                    for(Song song : playlistsongs){
                        totalRuntime += song.duration;
                    }

                    if (songCountInt != 0) {
                        firstAlbumID = playlistsongs.get(0).albumId;
                        return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                    } else return "nosongs";

            }
        } else {
            List<Song> playlistsongs = PlaylistSongLoader.getSongsInPlaylist(mContext, id);
            songCountInt = playlistsongs.size();
            totalRuntime = 0;
            for(Song song : playlistsongs){
                totalRuntime += song.duration;
            }

            if (songCountInt != 0) {
                firstAlbumID = playlistsongs.get(0).albumId;
                return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
            } else return "nosongs";
        }
    }
    return null;
}

@Override
```

```java
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    public void updateDataSet(List<Playlist> arraylist) {
        this.arraylist.clear();
        this.arraylist.addAll(arraylist);
        notifyDataSetChanged();
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, artist;
        protected ImageView albumArt;
        protected View footer;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.album_title);
            this.artist = (TextView) view.findViewById(R.id.album_artist);
            this.albumArt = (ImageView) view.findViewById(R.id.album_art);
            this.footer = view.findViewById(R.id.footer);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            NavigationUtils.navigateToPlaylistDetail(mContext, getPlaylistType(getAdapterPosition()), (long) albumArt.getTag

        }

    }

    private String getPlaylistType(int position) {
        if (showAuto) {
            switch (position) {
                case 0:
                    return Constants.NAVIGATE_PLAYLIST_LASTADDED;
                case 1:
                    return Constants.NAVIGATE_PLAYLIST_RECENT;
                case 2:
                    return Constants.NAVIGATE_PLAYLIST_TOPTRACKS;
                default:
                    return Constants.NAVIGATE_PLAYLIST_USERCREATED;
            }
        } else return Constants.NAVIGATE_PLAYLIST_USERCREATED;
    }


}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.PopupMenu;
import android.widget.TextView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.ArtistInfoListener;
import com.naman14.timber.lastfmapi.models.ArtistQuery;
import com.naman14.timber.lastfmapi.models.LastfmArtist;
import com.naman14.timber.models.Album;
import com.naman14.timber.models.Artist;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.display.FadeInBitmapDisplayer;

import java.util.Collections;
import java.util.List;

public class SearchAdapter extends BaseSongAdapter<SearchAdapter.ItemHolder> {

    private Activity mContext;
    private List searchResults = Collections.emptyList();

    public SearchAdapter(Activity context) {
        this.mContext = context;

    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        switch (viewType) {
            case 0:
                View v0 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_song, null);
                ItemHolder ml0 = new ItemHolder(v0);
                return ml0;
            case 1:
                View v1 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_album_search, null);
                ItemHolder ml1 = new ItemHolder(v1);
                return ml1;
            case 2:
                View v2 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_artist, null);
                ItemHolder ml2 = new ItemHolder(v2);
                return ml2;
            case 10:
                View v10 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.search_section_header, null);
                ItemHolder ml10 = new ItemHolder(v10);
                return ml10;
            default:
                View v3 = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_song, null);
                ItemHolder ml3 = new ItemHolder(v3);
                return ml3;
        }
    }

    @Override
    public void onBindViewHolder(final ItemHolder itemHolder, int i) {
        switch (getItemViewType(i)) {
            case 0:
                Song song = (Song) searchResults.get(i);
                itemHolder.title.setText(song.title);
                itemHolder.songartist.setText(song.albumName);
                ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(song.albumId).toString(), itemHolder.album
                        new DisplayImageOptions.Builder().cacheInMemory(true)
                                .cacheOnDisk(true)
                                .showImageOnFail(R.drawable.ic_empty_music2)
                                .resetViewBeforeLoading(true)
                                .displayer(new FadeInBitmapDisplayer(400))
                                .build());
                setOnPopupMenuListener(itemHolder, i);
                break;
            case 1:
                Album album = (Album) searchResults.get(i);
                itemHolder.albumtitle.setText(album.title);
                itemHolder.albumartist.setText(album.artistName);
                ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(album.id).toString(), itemHolder.albumArt,
                        new DisplayImageOptions.Builder().cacheInMemory(true)
                                .cacheOnDisk(true)
                                .showImageOnFail(R.drawable.ic_empty_music2)
                                .resetViewBeforeLoading(true)
                                .displayer(new FadeInBitmapDisplayer(400))
                                .build());
                break;
            case 2:
                Artist artist = (Artist) searchResults.get(i);
                itemHolder.artisttitle.setText(artist.name);
                String albumNmber = TimberUtils.makeLabel(mContext, R.plurals.Nalbums, artist.albumCount);
                String songCount = TimberUtils.makeLabel(mContext, R.plurals.Nsongs, artist.songCount);
                itemHolder.albumsongcount.setText(TimberUtils.makeCombinedString(mContext, albumNmber, songCount));
                LastFmClient.getInstance(mContext).getArtistInfo(new ArtistQuery(artist.name), new ArtistInfoListener() {
                    @Override
                    public void artistInfoSucess(LastfmArtist artist) {
                        if (artist != null && itemHolder.artistImage != null) {
                            ImageLoader.getInstance().displayImage(artist.mArtwork.get(1).mUrl, itemHolder.artistImage,
                                    new DisplayImageOptions.Builder().cacheInMemory(true)
                                            .cacheOnDisk(true)
                                            .showImageOnFail(R.drawable.ic_empty_music2)
                                            .resetViewBeforeLoading(true)
                                            .displayer(new FadeInBitmapDisplayer(400))
                                            .build());
                        }
                    }

                    @Override
                    public void artistInfoFailed() {

                    }
                });
                break;
            case 10:
                itemHolder.sectionHeader.setText((String) searchResults.get(i));
            case 3:
```

```java
                break;
        }
    }

    @Override
    public void onViewRecycled(ItemHolder itemHolder) {

    }

    @Override
    public int getItemCount() {
        return searchResults.size();
    }

    private void setOnPopupMenuListener(ItemHolder itemHolder, final int position) {

        itemHolder.menu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                final PopupMenu menu = new PopupMenu(mContext, v);
                menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {
                        long[] song = new long[1];
                        song[0] = ((Song) searchResults.get(position)).id;
                        switch (item.getItemId()) {
                            case R.id.popup_song_play:
                                MusicPlayer.playAll(mContext, song, 0, -1, TimberUtils.IdType.NA, false);
                                break;
                            case R.id.popup_song_play_next:
                                MusicPlayer.playNext(mContext, song, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_goto_album:
                                NavigationUtils.navigateToAlbum(mContext, ((Song) searchResults.get(position)).albumId, null
                                break;
                            case R.id.popup_song_goto_artist:
                                NavigationUtils.navigateToArtist(mContext, ((Song) searchResults.get(position)).artistId, nu
                                break;
                            case R.id.popup_song_addto_queue:
                                MusicPlayer.addToQueue(mContext, song, -1, TimberUtils.IdType.NA);
                                break;
                            case R.id.popup_song_addto_playlist:
                                AddPlaylistDialog.newInstance(((Song) searchResults.get(position))).show(((AppCompatActivity
                                break;
                        }
                        return false;
                    }
                });
                menu.inflate(R.menu.popup_song);
                //Hide these because they aren't implemented
                menu.getMenu().findItem(R.id.popup_song_delete).setVisible(false);
                menu.getMenu().findItem(R.id.popup_song_share).setVisible(false);
                menu.show();
            }
        });
    }

    @Override
    public int getItemViewType(int position) {
        if (searchResults.get(position) instanceof Song)
            return 0;
        if (searchResults.get(position) instanceof Album)
            return 1;
        if (searchResults.get(position) instanceof Artist)
            return 2;
        if (searchResults.get(position) instanceof String)
            return 10;
        return 3;
    }
    public void updateSearchResults(List searchResults) {
        this.searchResults = searchResults;
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, songartist, albumtitle, artisttitle, albumartist, albumsongcount, sectionHeader;
        protected ImageView albumArt, artistImage, menu;

        public ItemHolder(View view) {
            super(view);

            this.title = (TextView) view.findViewById(R.id.song_title);
            this.songartist = (TextView) view.findViewById(R.id.song_artist);
            this.albumsongcount = (TextView) view.findViewById(R.id.album_song_count);
            this.artisttitle = (TextView) view.findViewById(R.id.artist_name);
            this.albumtitle = (TextView) view.findViewById(R.id.album_title);
            this.albumartist = (TextView) view.findViewById(R.id.album_artist);
            this.albumArt = (ImageView) view.findViewById(R.id.albumArt);
            this.artistImage = (ImageView) view.findViewById(R.id.artistImage);
            this.menu = (ImageView) view.findViewById(R.id.popup_menu);

            this.sectionHeader = (TextView) view.findViewById(R.id.section_header);


            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            switch (getItemViewType()) {
                case 0:
                    final Handler handler = new Handler();
                    handler.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            long[] ret = new long[1];
                            ret[0] = ((Song) searchResults.get(getAdapterPosition())).id;
                            playAll(mContext, ret, 0, -1, TimberUtils.IdType.NA,
                                    false, (Song) searchResults.get(getAdapterPosition()), false);
                        }
                    }, 100);

                    break;
                case 1:
                    NavigationUtils.goToAlbum(mContext, ((Album) searchResults.get(getAdapterPosition())).id);
                    break;
                case 2:
                    NavigationUtils.goToArtist(mContext, ((Artist) searchResults.get(getAdapterPosition())).id);
                    break;
                case 3:
                    break;
                case 10:
                    break;
            }
        }

    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.app.Activity;
import android.os.Handler;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.util.List;

public class SlidingQueueAdapter extends RecyclerView.Adapter<SlidingQueueAdapter.ItemHolder> {

    public static int currentlyPlayingPosition;
    private List<Song> arraylist;
    private Activity mContext;
    private int lastPosition = -1;

    public SlidingQueueAdapter(Activity context, List<Song> arraylist) {
        this.arraylist = arraylist;
        this.mContext = context;
        currentlyPlayingPosition = MusicPlayer.getQueuePosition();
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_song_sliding_queue, null);
        ItemHolder ml = new ItemHolder(v);
        return ml;
    }

    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {

//        setAnimation(itemHolder.itemView, i);
        Song localItem = arraylist.get(i);

        ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(localItem.albumId).toString(),
                itemHolder.albumArt, new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnLoading(R.drawable.ic_empty_music2).resetViewBeforeLoading(true).build());

    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    public long[] getSongIds() {
        long[] ret = new long[getItemCount()];
        for (int i = 0; i < getItemCount(); i++) {
            ret[i] = arraylist.get(i).id;
        }

        return ret;
    }

    private void setAnimation(View viewToAnimate, int position) {
        // If the bound view wasn't previously displayed on screen, it's animated
        if (position > lastPosition) {
            Animation animation = AnimationUtils.loadAnimation(mContext, R.anim.scale);
            viewToAnimate.startAnimation(animation);
            lastPosition = position;
        }
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected ImageView albumArt;

        public ItemHolder(View view) {
            super(view);
            this.albumArt = (ImageView) view.findViewById(R.id.album_art);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            final Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.setQueuePosition(getAdapterPosition());
                    Handler handler1 = new Handler();
                    handler1.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            notifyItemChanged(currentlyPlayingPosition);
                            notifyItemChanged(getAdapterPosition());
                            Handler handler2 = new Handler();
                            handler2.postDelayed(new Runnable() {
                                @Override
                                public void run() {
                                }
                            }, 50);
                        }
                    }, 50);
                }
            }, 100);

        }

    }


}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.adapters;

import android.graphics.Color;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
import android.widget.PopupMenu;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.BubbleTextGetter;
import com.naman14.timber.widgets.MusicVisualizer;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;

import java.util.List;

public class SongsListAdapter extends BaseSongAdapter<SongsListAdapter.ItemHolder> implements BubbleTextGetter {

    public int currentlyPlayingPosition;
    private List<Song> arraylist;
    private AppCompatActivity mContext;
    private long[] songIDs;
    private boolean isPlaylist;
    private boolean animate;
    private int lastPosition = -1;
    private String ateKey;
    private long playlistId;

    public SongsListAdapter(AppCompatActivity context, List<Song> arraylist, boolean isPlaylistSong, boolean animate) {
        this.arraylist = arraylist;
        this.mContext = context;
        this.isPlaylist = isPlaylistSong;
        this.songIDs = getSongIds();
        this.ateKey = Helpers.getATEKey(context);
        this.animate = animate;
    }

    @Override
    public ItemHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
        if (isPlaylist) {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_song_playlist, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        } else {
            View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_song, null);
            ItemHolder ml = new ItemHolder(v);
            return ml;
        }
    }

    @Override
    public void onBindViewHolder(ItemHolder itemHolder, int i) {
        Song localItem = arraylist.get(i);

        itemHolder.title.setText(localItem.title);
        itemHolder.artist.setText(localItem.artistName);

        ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(localItem.albumId).toString(),
                itemHolder.albumArt, new DisplayImageOptions.Builder().cacheInMemory(true)
                        .showImageOnLoading(R.drawable.ic_empty_music2)
                        .resetViewBeforeLoading(true).build());

        if (MusicPlayer.getCurrentAudioId() == localItem.id) {
            itemHolder.title.setTextColor(Config.accentColor(mContext, ateKey));
            if (MusicPlayer.isPlaying()) {
                itemHolder.visualizer.setColor(Config.accentColor(mContext, ateKey));
                itemHolder.visualizer.setVisibility(View.VISIBLE);
            } else {
                itemHolder.visualizer.setVisibility(View.GONE);
            }
        } else {
            itemHolder.visualizer.setVisibility(View.GONE);
            if (isPlaylist) {
                itemHolder.title.setTextColor(Color.WHITE);
            } else {
                itemHolder.title.setTextColor(Config.textColorPrimary(mContext, ateKey));
            }
        }


        if (animate && isPlaylist) {
            if (TimberUtils.isLollipop())
                setAnimation(itemHolder.itemView, i);
            else {
                if (i > 10)
                    setAnimation(itemHolder.itemView, i);
            }
        }


        setOnPopupMenuListener(itemHolder, i);

    }

    public void setPlaylistId(long playlistId) {
        this.playlistId = playlistId;
    }

    @Override
    public int getItemCount() {
        return (null != arraylist ? arraylist.size() : 0);
    }

    private void setOnPopupMenuListener(ItemHolder itemHolder, final int position) {

        itemHolder.popupMenu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
            final PopupMenu menu = new PopupMenu(mContext, v);

            menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
                @Override
                public boolean onMenuItemClick(MenuItem item) {
                    switch (item.getItemId()) {
                        case R.id.popup_song_remove_playlist:
                            TimberUtils.removeFromPlaylist(mContext, arraylist.get(position).id, playlistId);
                            removeSongAt(position);
                            notifyItemRemoved(position);
                            break;
                        case R.id.popup_song_play:
                            MusicPlayer.playAll(mContext, songIDs, position, -1, TimberUtils.IdType.NA, false);
                            break;
                        case R.id.popup_song_play_next:
                            long[] ids = new long[1];
                            ids[0] = arraylist.get(position).id;
                            MusicPlayer.playNext(mContext, ids, -1, TimberUtils.IdType.NA);
                            break;
                        case R.id.popup_song_goto_album:
                            NavigationUtils.goToAlbum(mContext, arraylist.get(position).albumId);
                            break;
                        case R.id.popup_song_goto_artist:
                            NavigationUtils.goToArtist(mContext, arraylist.get(position).artistId);
                            break;
                        case R.id.popup_song_addto_queue:
                            long[] id = new long[1];
                            id[0] = arraylist.get(position).id;
                            MusicPlayer.addToQueue(mContext, id, -1, TimberUtils.IdType.NA);
                            break;
                        case R.id.popup_song_addto_playlist:
                            AddPlaylistDialog.newInstance(arraylist.get(position)).show(mContext.getSupportFragmentManag
                            break;
                        case R.id.popup_song_share:
                           TimberUtils.shareTrack(mContext, arraylist.get(position).id);
                            break;
                        case R.id.popup_song_delete:
                            long[] deleteIds = {arraylist.get(position).id};
                            TimberUtils.showDeleteDialog(mContext,arraylist.get(position).title, deleteIds, SongsListAda
                            break;
                    }
                    return false;
                }
            });
            menu.inflate(R.menu.popup_song);
            menu.show();
            if (isPlaylist)
                menu.getMenu().findItem(R.id.popup_song_remove_playlist).setVisible(true);
        }
    });
}

public long[] getSongIds() {
    long[] ret = new long[getItemCount()];
    for (int i = 0; i < getItemCount(); i++) {
        ret[i] = arraylist.get(i).id;
    }

    return ret;
}

@Override
public String getTextToShowInBubble(final int pos) {
    if (arraylist == null || arraylist.size() == 0)
        return "";
    Character ch = arraylist.get(pos).title.charAt(0);
    if (Character.isDigit(ch)) {
        return "#";
    } else
        return Character.toString(ch);
```

```java
    }

    private void setAnimation(View viewToAnimate, int position) {
        // If the bound view wasn't previously displayed on screen, it's animated
        if (position > lastPosition) {
            Animation animation = AnimationUtils.loadAnimation(mContext, R.anim.abc_slide_in_bottom);
            viewToAnimate.startAnimation(animation);
            lastPosition = position;
        }
    }

    @Override
    public void updateDataSet(List<Song> arraylist) {
        this.arraylist = arraylist;
        this.songIDs = getSongIds();
    }

    public class ItemHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        protected TextView title, artist;
        protected ImageView albumArt, popupMenu;
        private MusicVisualizer visualizer;

        public ItemHolder(View view) {
            super(view);
            this.title = (TextView) view.findViewById(R.id.song_title);
            this.artist = (TextView) view.findViewById(R.id.song_artist);
            this.albumArt = (ImageView) view.findViewById(R.id.albumArt);
            this.popupMenu = (ImageView) view.findViewById(R.id.popup_menu);
            visualizer = (MusicVisualizer) view.findViewById(R.id.visualizer);
            view.setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            final Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    playAll(mContext, songIDs, getAdapterPosition(), -1,
                            TimberUtils.IdType.NA, false,
                            arraylist.get(getAdapterPosition()), false);
                    Handler handler1 = new Handler();
                    handler1.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            notifyItemChanged(currentlyPlayingPosition);
                            notifyItemChanged(getAdapterPosition());
                        }
                    }, 50);
                }
            }, 100);


        }

    }

    public Song getSongAt(int i) {
        return arraylist.get(i);
    }

    public void addSongTo(int i, Song song) {
        arraylist.add(i, song);
    }

    @Override
    public void removeSongAt(int i) {
        arraylist.remove(i);
        updateDataSet(arraylist);
    }
```

```
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\cast\CastOptionsProvider.java

```java
package com.naman14.timber.cast;

import android.content.Context;

import com.google.android.gms.cast.framework.CastOptions;
import com.google.android.gms.cast.framework.OptionsProvider;
import com.google.android.gms.cast.framework.SessionProvider;
import com.google.android.gms.cast.framework.media.CastMediaOptions;
import com.google.android.gms.cast.framework.media.MediaIntentReceiver;
import com.google.android.gms.cast.framework.media.NotificationOptions;
import com.naman14.timber.R;

import java.util.ArrayList;
import java.util.List;

public class CastOptionsProvider implements OptionsProvider {

    @Override
    public CastOptions getCastOptions(Context appContext) {

        List<String> buttonActions = new ArrayList<>();
        buttonActions.add(MediaIntentReceiver.ACTION_TOGGLE_PLAYBACK);
        buttonActions.add(MediaIntentReceiver.ACTION_STOP_CASTING);
        int[] compatButtonActionsIndicies = new int[]{ 0, 1 };

        NotificationOptions notificationOptions = new NotificationOptions.Builder()
                .setActions(buttonActions, compatButtonActionsIndicies)
                .setTargetActivityClassName(ExpandedControlsActivity.class.getName())
                .build();

        CastMediaOptions mediaOptions = new CastMediaOptions.Builder()
                .setNotificationOptions(notificationOptions)
                .setExpandedControllerActivityClassName(ExpandedControlsActivity.class.getName())
                .build();

        CastOptions castOptions = new CastOptions.Builder()
            .setReceiverApplicationId(appContext.getString(R.string.cast_app_id))
                .setCastMediaOptions(mediaOptions)
                .build();

        return castOptions;
    }
    @Override
    public List<SessionProvider> getAdditionalSessionProviders(Context context) {
        return null;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\cast\ExpandedControlsActivity.j

```java
package com.naman14.timber.cast;

import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.WindowManager;

import com.google.android.gms.cast.framework.CastButtonFactory;
import com.google.android.gms.cast.framework.media.widget.ExpandedControllerActivity;
import com.naman14.timber.R;

public class ExpandedControlsActivity extends ExpandedControllerActivity {

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        getMenuInflater().inflate(R.menu.menu_expanded_controller, menu);
        CastButtonFactory.setUpMediaRouteButton(this, menu, R.id.media_route_menu_item);
        return true;
    }

    @Override
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        getWindow().getDecorView().setSystemUiVisibility(View.SYSTEM_UI_FLAG_VISIBLE);

    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\cast\SimpleSessionManagerListen

```java
package com.naman14.timber.cast;

import com.google.android.gms.cast.framework.Session;
import com.google.android.gms.cast.framework.SessionManagerListener;

/**
 * Created by naman on 7/12/17.
 */

public class SimpleSessionManagerListener implements SessionManagerListener {

    public void onSessionStarted(Session session, String sessionId) {
    }

    public void onSessionResumed(Session session, boolean wasSuspended) {
    }

    public void onSessionEnded(Session session, int error) {
    }

    public void onSessionSuspended(Session session, int i) {

    }

    public void onSessionStarting(Session session) {

    }

    public void onSessionEnding(Session session) {

    }

    public void onSessionResuming(Session session, String s) {

    }

    public void onSessionResumeFailed(Session session, int i) {

    }

    public void onSessionStartFailed(Session session, int i) {

    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\cast\TimberCastHelper.java

```java
package com.naman14.timber.cast;

import android.net.Uri;
import android.util.Log;

import com.google.android.gms.cast.MediaInfo;
import com.google.android.gms.cast.MediaMetadata;
import com.google.android.gms.cast.TextTrackStyle;
import com.google.android.gms.cast.framework.CastSession;
import com.google.android.gms.cast.framework.media.RemoteMediaClient;
import com.google.android.gms.common.images.WebImage;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.TimberUtils;

import java.net.MalformedURLException;
import java.net.URL;

/**
 * Created by naman on 2/12/17.
 */

public class TimberCastHelper  {

    public static void startCasting(CastSession castSession, Song song) {

        String ipAddress = TimberUtils.getIPAddress(true);
        URL baseUrl;
        try {
            baseUrl = new URL("http", ipAddress, Constants.CAST_SERVER_PORT, "" );
        } catch (MalformedURLException e) {
            e.printStackTrace();
            return;
        }

        String songUrl = baseUrl.toString() + "/song?id=" + song.id;
        String albumArtUrl = baseUrl.toString() + "/albumart?id=" + song.albumId;

        MediaMetadata musicMetadata = new MediaMetadata(MediaMetadata.MEDIA_TYPE_MUSIC_TRACK);

        musicMetadata.putString(MediaMetadata.KEY_TITLE, song.title);
        musicMetadata.putString(MediaMetadata.KEY_ARTIST, song.artistName);
        musicMetadata.putString(MediaMetadata.KEY_ALBUM_TITLE, song.albumName);
        musicMetadata.putInt(MediaMetadata.KEY_TRACK_NUMBER, song.trackNumber);
        musicMetadata.addImage(new WebImage(Uri.parse(albumArtUrl)));

        try {
            MediaInfo mediaInfo = new MediaInfo.Builder(songUrl)
                    .setStreamType(MediaInfo.STREAM_TYPE_BUFFERED)
                    .setContentType("audio/mpeg")
                    .setMetadata(musicMetadata)
                    .setStreamDuration(song.duration)
                    .build();
            RemoteMediaClient remoteMediaClient = castSession.getRemoteMediaClient();
            remoteMediaClient.load(mediaInfo, true, 0);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\cast\WebServer.java

```java
package com.naman14.timber.cast;

import android.content.Context;
import android.net.Uri;

import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.TimberUtils;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.Map;

import fi.iki.elonen.NanoHTTPD;

public class WebServer extends NanoHTTPD {

    private Context context;
    private Uri songUri, albumArtUri;

    public WebServer(Context context) {
        super(Constants.CAST_SERVER_PORT);
        this.context = context;
    }

    @Override
    public Response serve(String uri, Method method,
                          Map<String, String> header,
                          Map<String, String> parameters,
                          Map<String, String> files) {
        if (uri.contains("albumart")) {
            //serve the picture

            String albumId = parameters.get("id");
            this.albumArtUri = TimberUtils.getAlbumArtUri(Long.parseLong(albumId));

            if (albumArtUri != null) {
                String mediasend = "image/jpg";
                InputStream fisAlbumArt = null;
                try {
                    fisAlbumArt = context.getContentResolver().openInputStream(albumArtUri);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
                Response.Status st = Response.Status.OK;

                //serve the song
                return newChunkedResponse(st, mediasend, fisAlbumArt);
            }

        } else if (uri.contains("song")) {

            String songId = parameters.get("id");
            this.songUri = TimberUtils.getSongUri(context, Long.parseLong(songId));

            if (songUri != null) {
                String mediasend = "audio/mp3";
                FileInputStream fisSong = null;
                File song = new File(songUri.getPath());
                try {
                    fisSong = new FileInputStream(song);
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
                Response.Status st = Response.Status.OK;

                //serve the song
                return newFixedLengthResponse(st, mediasend, fisSong, song.length());
            }
        }
        return newFixedLengthResponse("Error");
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\AlbumLoader.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.database.Cursor;
import android.provider.MediaStore;

import com.naman14.timber.models.Album;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;
import java.util.List;

public class AlbumLoader {


    public static Album getAlbum(Cursor cursor) {
        Album album = new Album();
        if (cursor != null) {
            if (cursor.moveToFirst())
                album = new Album(cursor.getLong(0), cursor.getString(1), cursor.getString(2), cursor.getLong(3), cursor.get
        }
        if (cursor != null)
            cursor.close();
        return album;
    }


    public static List<Album> getAlbumsForCursor(Cursor cursor) {
        ArrayList arrayList = new ArrayList();
        if ((cursor != null) && (cursor.moveToFirst()))
            do {
                arrayList.add(new Album(cursor.getLong(0), cursor.getString(1), cursor.getString(2), cursor.getLong(3), curs
            }
            while (cursor.moveToNext());
        if (cursor != null)
            cursor.close();
        return arrayList;
    }

    public static List<Album> getAllAlbums(Context context) {
        return getAlbumsForCursor(makeAlbumCursor(context, null, null));
    }

    public static Album getAlbum(Context context, long id) {
        return getAlbum(makeAlbumCursor(context, "_id=?", new String[]{String.valueOf(id)}));
    }

    public static List<Album> getAlbums(Context context, String paramString, int limit) {
        List<Album> result = getAlbumsForCursor(makeAlbumCursor(context, "album LIKE ?", new String[]{paramString + "%"}));
        if (result.size() < limit) {
            result.addAll(getAlbumsForCursor(makeAlbumCursor(context, "album LIKE ?", new String[]{"%_" + paramString + "%"}
        }
        return result.size() < limit ? result : result.subList(0, limit);
    }
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\AlbumLoader.java

```
    public static Cursor makeAlbumCursor(Context context, String selection, String[] paramArrayOfString) {
        final String albumSortOrder = PreferencesUtility.getInstance(context).getAlbumSortOrder();
        Cursor cursor = context.getContentResolver().query(MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI, new String[]{"_id",

        return cursor;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\AlbumSongLoader.jav

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.ContentResolver;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.provider.MediaStore;

import com.naman14.timber.models.Song;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;

public class AlbumSongLoader {

    private static final long[] sEmptyList = new long[0];

    public static ArrayList<Song> getSongsForAlbum(Context context, long albumID) {

        Cursor cursor = makeAlbumSongCursor(context, albumID);
        ArrayList arrayList = new ArrayList();
        if ((cursor != null) && (cursor.moveToFirst()))
            do {
                long id = cursor.getLong(0);
                String title = cursor.getString(1);
                String artist = cursor.getString(2);
                String album = cursor.getString(3);
                int duration = cursor.getInt(4);
                int trackNumber = cursor.getInt(5);
                /*This fixes bug where some track numbers displayed as 100 or 200*/
                while (trackNumber >= 1000) {
                    trackNumber -= 1000; //When error occurs the track numbers have an extra 1000 or 2000 added, so decrease
                }
                long artistId = cursor.getInt(6);
                long albumId = albumID;

                arrayList.add(new Song(id, albumId, artistId, title, artist, album, duration, trackNumber));
            }
            while (cursor.moveToNext());
        if (cursor != null)
            cursor.close();
        return arrayList;
    }

    public static Cursor makeAlbumSongCursor(Context context, long albumID) {
        ContentResolver contentResolver = context.getContentResolver();
        final String albumSongSortOrder = PreferencesUtility.getInstance(context).getAlbumSongSortOrder();
        Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
        String string = "is_music=1 AND title != '' AND album_id=" + albumID;
        Cursor cursor = contentResolver.query(uri, new String[]{"_id", "title", "artist", "album", "duration", "track", "art
        return cursor;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\ArtistAlbumLoader.j

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.database.Cursor;
import android.provider.MediaStore;

import com.naman14.timber.models.Album;

import java.util.ArrayList;

public class ArtistAlbumLoader {

    public static ArrayList<Album> getAlbumsForArtist(Context context, long artistID) {

        ArrayList albumList = new ArrayList();
        Cursor cursor = makeAlbumForArtistCursor(context, artistID);

        if (cursor != null) {
            if (cursor.moveToFirst())
                do {

                    Album album = new Album(cursor.getLong(0), cursor.getString(1), cursor.getString(2), artistID, cursor.ge
                    albumList.add(album);
                }
                while (cursor.moveToNext());

        }
        if (cursor != null)
            cursor.close();
        return albumList;
    }


    public static Cursor makeAlbumForArtistCursor(Context context, long artistID) {

        if (artistID == -1)
            return null;

        return context.getContentResolver()
                      .query(MediaStore.Audio.Artists.Albums.getContentUri("external", artistID),
                             new String[]{"_id", "album", "artist", "numsongs", "minyear"},
                             null,
                             null,
                             MediaStore.Audio.Albums.FIRST_YEAR);
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\ArtistLoader.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.database.Cursor;
import android.provider.MediaStore;

import com.naman14.timber.models.Artist;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;
import java.util.List;

public class ArtistLoader {

    public static Artist getArtist(Cursor cursor) {
        Artist artist = new Artist();
        if (cursor != null) {
            if (cursor.moveToFirst())
                artist = new Artist(cursor.getLong(0), cursor.getString(1), cursor.getInt(2), cursor.getInt(3));
        }
        if (cursor != null)
            cursor.close();
        return artist;
    }

    public static List<Artist> getArtistsForCursor(Cursor cursor) {
        ArrayList arrayList = new ArrayList();
        if ((cursor != null) && (cursor.moveToFirst()))
            do {
                arrayList.add(new Artist(cursor.getLong(0), cursor.getString(1), cursor.getInt(2), cursor.getInt(3)));
            }
            while (cursor.moveToNext());
        if (cursor != null)
            cursor.close();
        return arrayList;
    }

    public static List<Artist> getAllArtists(Context context) {
        return getArtistsForCursor(makeArtistCursor(context, null, null));
    }

    public static Artist getArtist(Context context, long id) {
        return getArtist(makeArtistCursor(context, "_id=?", new String[]{String.valueOf(id)}));
    }

    public static List<Artist> getArtists(Context context, String paramString, int limit) {
        List<Artist> result = getArtistsForCursor(makeArtistCursor(context, "artist LIKE ?", new String[]{paramString + "%"}
        if (result.size() < limit) {
            result.addAll(getArtistsForCursor(makeArtistCursor(context, "artist LIKE ?", new String[]{"%_" + paramString + "
        }
        return result.size() < limit ? result : result.subList(0, limit);
    }


    public static Cursor makeArtistCursor(Context context, String selection, String[] paramArrayOfString) {
        final String artistSortOrder = PreferencesUtility.getInstance(context).getArtistSortOrder();
        Cursor cursor = context.getContentResolver().query(MediaStore.Audio.Artists.EXTERNAL_CONTENT_URI, new String[]{"_id"
        return cursor;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\ArtistSongLoader.ja

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.ContentResolver;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.provider.MediaStore;

import com.naman14.timber.models.Song;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;

public class ArtistSongLoader {

    public static ArrayList<Song> getSongsForArtist(Context context, long artistID) {
        Cursor cursor = makeArtistSongCursor(context, artistID);
        ArrayList songsList = new ArrayList();
        if ((cursor != null) && (cursor.moveToFirst()))
            do {
                long id = cursor.getLong(0);
                String title = cursor.getString(1);
                String artist = cursor.getString(2);
                String album = cursor.getString(3);
                int duration = cursor.getInt(4);
                int trackNumber = cursor.getInt(5);
                long albumId = cursor.getInt(6);
                long artistId = artistID;

                songsList.add(new Song(id, albumId, artistID, title, artist, album, duration, trackNumber));
            }
            while (cursor.moveToNext());
        if (cursor != null)
            cursor.close();
        return songsList;
    }


    public static Cursor makeArtistSongCursor(Context context, long artistID) {
        ContentResolver contentResolver = context.getContentResolver();
        final String artistSongSortOrder = PreferencesUtility.getInstance(context).getArtistSongSortOrder();
        Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
        String string = "is_music=1 AND title != '' AND artist_id=" + artistID;
        return contentResolver.query(uri, new String[]{"_id", "title", "artist", "album", "duration", "track", "album_id"},
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\FolderLoader.java

```java
package com.naman14.timber.dataloaders;

import android.text.TextUtils;

import java.io.File;
import java.io.FileFilter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * Created by nv95 on 10.11.16.
 */

public class FolderLoader {

    private static final String[] SUPPORTED_EXT = new String[] {
            "mp3",
            "mp4",
            "m4a",
            "aac",
            "ogg",
            "wav"
    };

    public static List<File> getMediaFiles(File dir, final boolean acceptDirs) {
        ArrayList<File> list = new ArrayList<>();
        list.add(new File(dir, ".."));
        if (dir.isDirectory()) {
            List<File> files = Arrays.asList(dir.listFiles(new FileFilter() {

                @Override
                public boolean accept(File file) {
                    if (file.isFile()) {
                        String name = file.getName();
                        return !".nomedia".equals(name) && checkFileExt(name);
                    } else if (file.isDirectory()) {
                        return acceptDirs && checkDir(file);
                    } else
                        return false;
                }
            }));
            Collections.sort(files, new FilenameComparator());
            Collections.sort(files, new DirFirstComparator());
            list.addAll(files);
        }

        return list;
    }

    public static boolean isMediaFile(File file) {
        return file.exists() && file.canRead() && checkFileExt(file.getName());
    }

    private static boolean checkDir(File dir) {
        return dir.exists() && dir.canRead() && !".".equals(dir.getName()) && dir.listFiles(new FileFilter() {
            @Override
            public boolean accept(File pathname) {
                String name = pathname.getName();
                return !".".equals(name) && !"..".equals(name) && pathname.canRead() && (pathname.isDirectory()  || (pathnam
            }

        }).length != 0;
    }

    private static boolean checkFileExt(String name) {
        if (TextUtils.isEmpty(name)) {
            return false;
        }
        int p = name.lastIndexOf(".") + 1;
        if (p < 1) {
            return false;
        }
        String ext = name.substring(p).toLowerCase();
        for (String o : SUPPORTED_EXT) {
            if (o.equals(ext)) {
                return true;
            }
        }
        return false;
    }

    private static class FilenameComparator implements Comparator<File> {
        @Override
        public int compare(File f1, File f2) {
            return f1.getName().compareTo(f2.getName());
        }
    }

    private static class DirFirstComparator implements Comparator<File> {
        @Override
        public int compare(File f1, File f2) {
            if (f1.isDirectory() == f2.isDirectory())
                return 0;
            else if (f1.isDirectory() && !f2.isDirectory())
                return -1;
            else
                return 1;
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\LastAddedLoader.jav

```java
/*
 * Copyright (C) 2012 Andrew Neal
 * Copyright (C) 2014 The CyanogenMod Project
 * Copyright (C) 2015 Naman Dwivedi
 * Licensed under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.database.Cursor;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.AudioColumns;

import com.naman14.timber.models.Song;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;
import java.util.List;

public class LastAddedLoader {

    private static Cursor mCursor;

    public static List<Song> getLastAddedSongs(Context context) {

        ArrayList<Song> mSongList = new ArrayList<>();
        mCursor = makeLastAddedCursor(context);

        if (mCursor != null && mCursor.moveToFirst()) {
            do {
                long id = mCursor.getLong(0);
                String title = mCursor.getString(1);
                String artist = mCursor.getString(2);
                String album = mCursor.getString(3);
                int duration = mCursor.getInt(4);
                int trackNumber = mCursor.getInt(5);
                long artistId = mCursor.getInt(6);
                long albumId = mCursor.getLong(7);

                final Song song = new Song(id, albumId, artistId, title, artist, album, duration, trackNumber);

                mSongList.add(song);
            } while (mCursor.moveToNext());
        }
        if (mCursor != null) {
            mCursor.close();
            mCursor = null;
        }
        return mSongList;
    }

    public static final Cursor makeLastAddedCursor(final Context context) {
        //four weeks ago
        long fourWeeksAgo = (System.currentTimeMillis() / 1000) - (4 * 3600 * 24 * 7);
        long cutoff = PreferencesUtility.getInstance(context).getLastAddedCutoff();
        // use the most recent of the two timestamps
        if (cutoff < fourWeeksAgo) {
            cutoff = fourWeeksAgo;
        }

        final StringBuilder selection = new StringBuilder();
        selection.append(AudioColumns.IS_MUSIC + "=1");
        selection.append(" AND " + AudioColumns.TITLE + " != ''");
        selection.append(" AND " + MediaStore.Audio.Media.DATE_ADDED + ">");
        selection.append(cutoff);

        return context.getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                new String[]{"_id", "title", "artist", "album", "duration", "track", "artist_id", "album_id"}, selection.toS
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\NowPlayingCursor.ja

```java
/*
 * Copyright (C) 2012 Andrew Neal
 * Copyright (C) 2014 The CyanogenMod Project
 * Licensed under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.database.AbstractCursor;
import android.database.Cursor;
import android.os.RemoteException;
import android.provider.BaseColumns;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.AudioColumns;
import android.util.Log;

import com.naman14.timber.MusicPlayer;

import java.util.Arrays;

import static com.naman14.timber.MusicPlayer.mService;


public class NowPlayingCursor extends AbstractCursor {

    private static final String[] PROJECTION = new String[]{

            BaseColumns._ID,

            AudioColumns.TITLE,

            AudioColumns.ARTIST,

            AudioColumns.ALBUM_ID,

            AudioColumns.ALBUM,

            AudioColumns.DURATION,

            AudioColumns.TRACK,

            AudioColumns.ARTIST_ID,

            AudioColumns.TRACK,
    };

    private final Context mContext;

    private long[] mNowPlaying;

    private long[] mCursorIndexes;

    private int mSize;

    private int mCurPos;

    private Cursor mQueueCursor;


    public NowPlayingCursor(final Context context) {
        mContext = context;
        makeNowPlayingCursor();
    }


    @Override
    public int getCount() {
        return mSize;
    }


    @Override
    public boolean onMove(final int oldPosition, final int newPosition) {
        if (oldPosition == newPosition) {
            return true;
        }

        if (mNowPlaying == null || mCursorIndexes == null || newPosition >= mNowPlaying.length) {
            return false;
        }

        final long id = mNowPlaying[newPosition];
        final int cursorIndex = Arrays.binarySearch(mCursorIndexes, id);
        mQueueCursor.moveToPosition(cursorIndex);
        mCurPos = newPosition;
        return true;
    }

    @Override
    public String getString(final int column) {
        try {
            return mQueueCursor.getString(column);
        } catch (final Exception ignored) {
            onChange(true);
            return "";
        }
    }


    @Override
    public short getShort(final int column) {
        return mQueueCursor.getShort(column);
    }


    @Override
    public int getInt(final int column) {
        try {
            return mQueueCursor.getInt(column);
        } catch (final Exception ignored) {
            onChange(true);
            return 0;
        }
    }


    @Override
    public long getLong(final int column) {
        try {
            return mQueueCursor.getLong(column);
        } catch (final Exception ignored) {
            onChange(true);
            return 0;
        }
    }


    @Override
    public float getFloat(final int column) {
        return mQueueCursor.getFloat(column);
    }
    @Override
    public double getDouble(final int column) {
        return mQueueCursor.getDouble(column);
    }


    @Override
    public int getType(final int column) {
        return mQueueCursor.getType(column);
    }

    @Override
    public boolean isNull(final int column) {
        return mQueueCursor.isNull(column);
    }


    @Override
    public String[] getColumnNames() {
        return PROJECTION;
    }


    @SuppressWarnings("deprecation")
    @Override
    public void deactivate() {
        if (mQueueCursor != null) {
            mQueueCursor.deactivate();
        }
    }

    @Override
    public boolean requery() {
        makeNowPlayingCursor();
        return true;
    }


    @Override
    public void close() {
        try {
            if (mQueueCursor != null) {
                mQueueCursor.close();
                mQueueCursor = null;
            }
        } catch (final Exception close) {
        }
        super.close();
    }


    private void makeNowPlayingCursor() {
        mQueueCursor = null;
        mNowPlaying = MusicPlayer.getQueue();
        Log.d("lol1", mNowPlaying.toString() + "   " + mNowPlaying.length);
        mSize = mNowPlaying.length;
        if (mSize == 0) {
            return;
        }

        final StringBuilder selection = new StringBuilder();
        selection.append(MediaStore.Audio.Media._ID + " IN (");
        for (int i = 0; i < mSize; i++) {
            selection.append(mNowPlaying[i]);
            if (i < mSize - 1) {
                selection.append(",");
            }
        }
        selection.append(")");
        mQueueCursor = mContext.getContentResolver().query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, PROJECTION, selection.toString(),
                null, MediaStore.Audio.Media._ID);

        if (mQueueCursor == null) {
            mSize = 0;
            return;
        }

        final int playlistSize = mQueueCursor.getCount();
        mCursorIndexes = new long[playlistSize];
        mQueueCursor.moveToFirst();
        final int columnIndex = mQueueCursor.getColumnIndexOrThrow(MediaStore.Audio.Media._ID);
        for (int i = 0; i < playlistSize; i++) {
            mCursorIndexes[i] = mQueueCursor.getLong(columnIndex);
            mQueueCursor.moveToNext();
        }
        mQueueCursor.moveToFirst();
        mCurPos = -1;

        int removed = 0;
        for (int i = mNowPlaying.length - 1; i >= 0; i--) {
            final long trackId = mNowPlaying[i];
            final int cursorIndex = Arrays.binarySearch(mCursorIndexes, trackId);
            if (cursorIndex < 0) {
                removed += MusicPlayer.removeTrack(trackId);
            }
        }
        if (removed > 0) {
            mNowPlaying = MusicPlayer.getQueue();
            mSize = mNowPlaying.length;
            if (mSize == 0) {
                mCursorIndexes = null;
                return;
            }
        }
    }


    public boolean removeItem(final int which) {
        try {
            if (mService.removeTracks(which, which) == 0) {
                return false;
            }
            int i = which;
            mSize--;
            while (i < mSize) {
                mNowPlaying[i] = mNowPlaying[i + 1];
                i++;
            }
            onMove(-1, mCurPos);
        } catch (final RemoteException ignored) {
        }
        return true;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\PlaylistLoader.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.content.res.Resources;
import android.database.Cursor;
import android.net.Uri;
import android.provider.BaseColumns;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.PlaylistsColumns;

import com.naman14.timber.models.Playlist;
import com.naman14.timber.utils.TimberUtils;

import java.util.ArrayList;
import java.util.List;

public class PlaylistLoader {

    static ArrayList<Playlist> mPlaylistList;
    private static Cursor mCursor;

    public static List<Playlist> getPlaylists(Context context, boolean defaultIncluded) {

        mPlaylistList = new ArrayList<>();

        if (defaultIncluded)
            makeDefaultPlaylists(context);

        mCursor = makePlaylistCursor(context);

        if (mCursor != null && mCursor.moveToFirst()) {
            do {

                final long id = mCursor.getLong(0);

                final String name = mCursor.getString(1);

                final int songCount = TimberUtils.getSongCountForPlaylist(context, id);

                final Playlist playlist = new Playlist(id, name, songCount);

                mPlaylistList.add(playlist);
            } while (mCursor.moveToNext());
        }
        if (mCursor != null) {
            mCursor.close();
            mCursor = null;
        }
        return mPlaylistList;
    }

    private static void makeDefaultPlaylists(Context context) {
        final Resources resources = context.getResources();

        /* Last added list */
        final Playlist lastAdded = new Playlist(TimberUtils.PlaylistType.LastAdded.mId,
                resources.getString(TimberUtils.PlaylistType.LastAdded.mTitleId), -1);
        mPlaylistList.add(lastAdded);

        /* Recently Played */
        final Playlist recentlyPlayed = new Playlist(TimberUtils.PlaylistType.RecentlyPlayed.mId,
                resources.getString(TimberUtils.PlaylistType.RecentlyPlayed.mTitleId), -1);
        mPlaylistList.add(recentlyPlayed);

        /* Top Tracks */
        final Playlist topTracks = new Playlist(TimberUtils.PlaylistType.TopTracks.mId,
                resources.getString(TimberUtils.PlaylistType.TopTracks.mTitleId), -1);
        mPlaylistList.add(topTracks);
    }


    public static final Cursor makePlaylistCursor(final Context context) {
        return context.getContentResolver().query(MediaStore.Audio.Playlists.EXTERNAL_CONTENT_URI,
                new String[]{
                        BaseColumns._ID,
                        PlaylistsColumns.NAME
                }, null, null, MediaStore.Audio.Playlists.DEFAULT_SORT_ORDER);
    }

    public static void deletePlaylists(Context context, long playlistId) {
        Uri localUri = MediaStore.Audio.Playlists.EXTERNAL_CONTENT_URI;
        StringBuilder localStringBuilder = new StringBuilder();
        localStringBuilder.append("_id IN (");
        localStringBuilder.append((playlistId));
        localStringBuilder.append(")");
        context.getContentResolver().delete(localUri, localStringBuilder.toString(), null);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\PlaylistSongLoader.

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.ContentProviderOperation;
import android.content.Context;
import android.content.OperationApplicationException;
import android.database.Cursor;
import android.net.Uri;
import android.os.RemoteException;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.AudioColumns;
import android.provider.MediaStore.Audio.Playlists;

import com.naman14.timber.models.Song;

import java.util.ArrayList;
import java.util.List;

public class PlaylistSongLoader {

    private static Cursor mCursor;

    private static long mPlaylistID;
    private static Context context;


    public static List<Song> getSongsInPlaylist(Context mContext, long playlistID) {
        ArrayList<Song> mSongList = new ArrayList<>();

        context = mContext;
        mPlaylistID = playlistID;

        final int playlistCount = countPlaylist(context, mPlaylistID);

        mCursor = makePlaylistSongCursor(context, mPlaylistID);

        if (mCursor != null) {
            boolean runCleanup = false;
            if (mCursor.getCount() != playlistCount) {
                runCleanup = true;
            }

            if (!runCleanup && mCursor.moveToFirst()) {
                final int playOrderCol = mCursor.getColumnIndexOrThrow(Playlists.Members.PLAY_ORDER);

                int lastPlayOrder = -1;
                do {
                    int playOrder = mCursor.getInt(playOrderCol);
                    if (playOrder == lastPlayOrder) {
                        runCleanup = true;
                        break;
                    }
                    lastPlayOrder = playOrder;
                } while (mCursor.moveToNext());
            }

            if (runCleanup) {
                cleanupPlaylist(context, mPlaylistID, mCursor);

                mCursor.close();
                mCursor = makePlaylistSongCursor(context, mPlaylistID);
                if (mCursor != null) {
                }
            }
        }

        if (mCursor != null && mCursor.moveToFirst()) {
            do {

                final long id = mCursor.getLong(mCursor
                        .getColumnIndexOrThrow(MediaStore.Audio.Playlists.Members.AUDIO_ID));

                final String songName = mCursor.getString(mCursor
                        .getColumnIndexOrThrow(AudioColumns.TITLE));

                final String artist = mCursor.getString(mCursor
                        .getColumnIndexOrThrow(AudioColumns.ARTIST));

                final long albumId = mCursor.getLong(mCursor
                        .getColumnIndexOrThrow(AudioColumns.ALBUM_ID));

                final long artistId = mCursor.getLong(mCursor
                        .getColumnIndexOrThrow(AudioColumns.ARTIST_ID));

                final String album = mCursor.getString(mCursor
                        .getColumnIndexOrThrow(AudioColumns.ALBUM));

                final long duration = mCursor.getLong(mCursor
                        .getColumnIndexOrThrow(AudioColumns.DURATION));

                final int durationInSecs = (int) duration / 1000;

                final int tracknumber = mCursor.getInt(mCursor
                        .getColumnIndexOrThrow(AudioColumns.TRACK));

                final Song song = new Song(id, albumId, artistId, songName, artist, album, durationInSecs, tracknumber);

                mSongList.add(song);
            } while (mCursor.moveToNext());
        }
        // Close the cursor
        if (mCursor != null) {
            mCursor.close();
            mCursor = null;
        }
        return mSongList;
    }

    private static void cleanupPlaylist(final Context context, final long playlistId,
                                        final Cursor cursor) {
        final int idCol = cursor.getColumnIndexOrThrow(MediaStore.Audio.Playlists.Members.AUDIO_ID);
        final Uri uri = MediaStore.Audio.Playlists.Members.getContentUri("external", playlistId);

        ArrayList<ContentProviderOperation> ops = new ArrayList<ContentProviderOperation>();

        ops.add(ContentProviderOperation.newDelete(uri).build());

        final int YIELD_FREQUENCY = 100;

        if (cursor.moveToFirst() && cursor.getCount() > 0) {
            do {
                final ContentProviderOperation.Builder builder =
                        ContentProviderOperation.newInsert(uri)
                                .withValue(Playlists.Members.PLAY_ORDER, cursor.getPosition())
                                .withValue(Playlists.Members.AUDIO_ID, cursor.getLong(idCol));
                if ((cursor.getPosition() + 1) % YIELD_FREQUENCY == 0) {
                    builder.withYieldAllowed(true);
                }
                ops.add(builder.build());
            } while (cursor.moveToNext());
        }

        try {
            context.getContentResolver().applyBatch(MediaStore.AUTHORITY, ops);
        } catch (RemoteException e) {
        } catch (OperationApplicationException e) {
        }
    }


    private static int countPlaylist(final Context context, final long playlistId) {
        Cursor c = null;
        try {
            c = context.getContentResolver().query(
                    MediaStore.Audio.Playlists.Members.getContentUri("external", playlistId),
                    new String[]{
                            MediaStore.Audio.Playlists.Members.AUDIO_ID,
                    }, null, null,
                    MediaStore.Audio.Playlists.Members.DEFAULT_SORT_ORDER);

            if (c != null) {
                return c.getCount();
            }
        } finally {
            if (c != null) {
                c.close();
                c = null;
            }
        }

        return 0;
    }


    public static final Cursor makePlaylistSongCursor(final Context context, final Long playlistID) {
        final StringBuilder mSelection = new StringBuilder();
        mSelection.append(AudioColumns.IS_MUSIC + "=1");
        mSelection.append(" AND " + AudioColumns.TITLE + " != ''");
        return context.getContentResolver().query(
                MediaStore.Audio.Playlists.Members.getContentUri("external", playlistID),
                new String[]{
                        MediaStore.Audio.Playlists.Members._ID,
                        MediaStore.Audio.Playlists.Members.AUDIO_ID,
                        AudioColumns.TITLE,
                        AudioColumns.ARTIST,
                        AudioColumns.ALBUM_ID,
                        AudioColumns.ARTIST_ID,
                        AudioColumns.ALBUM,
                        AudioColumns.DURATION,
                        AudioColumns.TRACK,
                        Playlists.Members.PLAY_ORDER,
                }, mSelection.toString(), null,
                MediaStore.Audio.Playlists.Members.DEFAULT_SORT_ORDER);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dataloaders\QueueLoader.java

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.Context;

import com.naman14.timber.models.Song;

import java.util.ArrayList;
import java.util.List;


public class QueueLoader {


    private static NowPlayingCursor mCursor;

    public static List<Song> getQueueSongs(Context context) {

        final ArrayList<Song> mSongList = new ArrayList<>();
        mCursor = new NowPlayingCursor(context);

        if (mCursor != null && mCursor.moveToFirst()) {
            do {

                final long id = mCursor.getLong(0);

                final String songName = mCursor.getString(1);

                final String artist = mCursor.getString(2);

                final long albumId = mCursor.getLong(3);

                final String album = mCursor.getString(4);

                final int duration = mCursor.getInt(5);

                final long artistid = mCursor.getInt(7);

                final int tracknumber = mCursor.getInt(6);

                final Song song = new Song(id, albumId, artistid, songName, artist, album, duration, tracknumber);

                mSongList.add(song);
            } while (mCursor.moveToNext());
        }
        if (mCursor != null) {
            mCursor.close();
            mCursor = null;
        }
        return mSongList;
    }



}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.dataloaders;

import android.content.ContentResolver;
import android.content.Context;
import android.database.Cursor;
import android.media.MediaMetadataRetriever;
import android.net.Uri;
import android.provider.BaseColumns;
import android.provider.MediaStore;
import android.text.TextUtils;

import com.naman14.timber.models.Song;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;
import java.util.List;

public class SongLoader {

    private static final long[] sEmptyList = new long[0];

    public static ArrayList<Song> getSongsForCursor(Cursor cursor) {
        ArrayList arrayList = new ArrayList();
        if ((cursor != null) && (cursor.moveToFirst()))
            do {
                long id = cursor.getLong(0);
                String title = cursor.getString(1);
                String artist = cursor.getString(2);
                String album = cursor.getString(3);
                int duration = cursor.getInt(4);
                int trackNumber = cursor.getInt(5);
                long artistId = cursor.getInt(6);
                long albumId = cursor.getLong(7);

                arrayList.add(new Song(id, albumId, artistId, title, artist, album, duration, trackNumber));
            }
            while (cursor.moveToNext());
        if (cursor != null)
            cursor.close();
        return arrayList;
    }

    public static Song getSongForCursor(Cursor cursor) {
        Song song = new Song();
        if ((cursor != null) && (cursor.moveToFirst())) {
            long id = cursor.getLong(0);
            String title = cursor.getString(1);
            String artist = cursor.getString(2);
            String album = cursor.getString(3);
            int duration = cursor.getInt(4);
            int trackNumber = cursor.getInt(5);
            long artistId = cursor.getInt(6);
            long albumId = cursor.getLong(7);

            song = new Song(id, albumId, artistId, title, artist, album, duration, trackNumber);
        }
        if (cursor != null)
            cursor.close();
        return song;
    }

    public static final long[] getSongListForCursor(Cursor cursor) {
        if (cursor == null) {
            return sEmptyList;
        }
        final int len = cursor.getCount();
        final long[] list = new long[len];
        cursor.moveToFirst();
        int columnIndex = -1;
        try {
            columnIndex = cursor.getColumnIndexOrThrow(MediaStore.Audio.Playlists.Members.AUDIO_ID);
        } catch (final IllegalArgumentException notaplaylist) {
            columnIndex = cursor.getColumnIndexOrThrow(BaseColumns._ID);
        }
        for (int i = 0; i < len; i++) {
            list[i] = cursor.getLong(columnIndex);
            cursor.moveToNext();
        }
        cursor.close();
        cursor = null;
        return list;
    }

    public static Song getSongFromPath(String songPath, Context context) {
        ContentResolver cr = context.getContentResolver();

        Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
        String selection = MediaStore.Audio.Media.DATA;
        String[] selectionArgs = {songPath};
        String[] projection = new String[]{"_id", "title", "artist", "album", "duration", "track", "artist_id", "album_id"};
        String sortOrder = MediaStore.Audio.Media.TITLE + " ASC";

        Cursor cursor = cr.query(uri, projection, selection + "=?", selectionArgs, sortOrder);

        if (cursor != null && cursor.getCount() > 0) {
            Song song = getSongForCursor(cursor);
            cursor.close();
            return song;
        }
        else return new Song();
    }

    public static ArrayList<Song> getAllSongs(Context context) {
        return getSongsForCursor(makeSongCursor(context, null, null));
    }

    public static long[] getSongListInFolder(Context context, String path) {
        String[] whereArgs = new String[]{path + "%"};
        return getSongListForCursor(makeSongCursor(context, MediaStore.Audio.Media.DATA + " LIKE ?", whereArgs, null));
    }

    public static Song getSongForID(Context context, long id) {
        return getSongForCursor(makeSongCursor(context, "_id=" + String.valueOf(id), null));
    }

    public static List<Song> searchSongs(Context context, String searchString, int limit) {
        ArrayList<Song> result = getSongsForCursor(makeSongCursor(context, "title LIKE ?", new String[]{searchString + "%"})
        if (result.size() < limit) {
            result.addAll(getSongsForCursor(makeSongCursor(context, "title LIKE ?", new String[]{"%_" + searchString + "%"})
        }
        return result.size() < limit ? result : result.subList(0, limit);
    }


    public static Cursor makeSongCursor(Context context, String selection, String[] paramArrayOfString) {
        final String songSortOrder = PreferencesUtility.getInstance(context).getSongSortOrder();
        return makeSongCursor(context, selection, paramArrayOfString, songSortOrder);
    }

    private static Cursor makeSongCursor(Context context, String selection, String[] paramArrayOfString, String sortOrder) {
        String selectionStatement = "is_music=1 AND title != ''";

        if (!TextUtils.isEmpty(selection)) {
            selectionStatement = selectionStatement + " AND " + selection;
        }
        return context.getContentResolver().query(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, new String[]{"_id", "title",

    }

    public static Song songFromFile(String filePath) {
        MediaMetadataRetriever mmr = new MediaMetadataRetriever();
        mmr.setDataSource(filePath);
        return new Song(
                -1,
                -1,
                -1,
                mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_TITLE),
                mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ARTIST),
                mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ALBUM),
                Integer.parseInt(mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_DURATION)),
                0
        );
    }

}
```

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
package com.naman14.timber.dataloaders;

import android.database.AbstractCursor;
import android.database.Cursor;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;

/**
 * This cursor basically wraps a song cursor and is given a list of the order of the ids of the
 * contents of the cursor. It wraps the Cursor and simulates the internal cursor being sorted
 * by moving the point to the appropriate spot
 */
public class SortedCursor extends AbstractCursor {
    // cursor to wrap
    private final Cursor mCursor;
    // the map of external indices to internal indices
    private ArrayList<Integer> mOrderedPositions;
    // this contains the ids that weren't found in the underlying cursor
    private ArrayList<Long> mMissingIds;
    // this contains the mapped cursor positions and afterwards the extra ids that weren't found
    private HashMap<Long, Integer> mMapCursorPositions;
    // extra we want to store with the cursor
    private ArrayList<Object> mExtraData;

    /**
     * @param cursor     to wrap
     * @param order      the list of unique ids in sorted order to display
     * @param columnName the column name of the id to look up in the internal cursor
     */
    public SortedCursor(final Cursor cursor, final long[] order, final String columnName,
                        final List<? extends Object> extraData) {
        if (cursor == null) {
            throw new IllegalArgumentException("Non-null cursor is needed");
        }

        mCursor = cursor;
        mMissingIds = buildCursorPositionMapping(order, columnName, extraData);
    }

    /**
     * This function populates mOrderedPositions with the cursor positions in the order based
     * on the order passed in
     *
     * @param order     the target order of the internal cursor
     * @param extraData Extra data we want to add to the cursor
     * @return returns the ids that aren't found in the underlying cursor
     */
    private ArrayList<Long> buildCursorPositionMapping(final long[] order,
                                                       final String columnName, final List<? extends Object> extraData) {
        ArrayList<Long> missingIds = new ArrayList<Long>();

        mOrderedPositions = new ArrayList<Integer>(mCursor.getCount());
        mExtraData = new ArrayList<Object>();

        mMapCursorPositions = new HashMap<Long, Integer>(mCursor.getCount());
        final int idPosition = mCursor.getColumnIndex(columnName);

        if (mCursor.moveToFirst()) {
            // first figure out where each of the ids are in the cursor
            do {
                mMapCursorPositions.put(mCursor.getLong(idPosition), mCursor.getPosition());
            } while (mCursor.moveToNext());

            // now create the ordered positions to map to the internal cursor given the
            // external sort order
            for (int i = 0; order != null && i < order.length; i++) {
                final long id = order[i];
                if (mMapCursorPositions.containsKey(id)) {
                    mOrderedPositions.add(mMapCursorPositions.get(id));
                    mMapCursorPositions.remove(id);
                    if (extraData != null) {
                        mExtraData.add(extraData.get(i));
                    }
                } else {
                    missingIds.add(id);
                }
            }

            mCursor.moveToFirst();
        }

        return missingIds;
    }

    /**
     * @return the list of ids that weren't found in the underlying cursor
     */
    public ArrayList<Long> getMissingIds() {
        return mMissingIds;
    }

    /**
     * @return the list of ids that were in the underlying cursor but not part of the ordered list
     */
    public Collection<Long> getExtraIds() {
        return mMapCursorPositions.keySet();
    }

    /**
     * @return the extra object data that was passed in to be attached to the current row
     */
    public Object getExtraData() {
        int position = getPosition();
        return position < mExtraData.size() ? mExtraData.get(position) : null;
    }

    @Override
    public void close() {
        mCursor.close();

        super.close();
    }

    @Override
    public int getCount() {
        return mOrderedPositions.size();
    }

    @Override
    public String[] getColumnNames() {
        return mCursor.getColumnNames();
    }
    @Override
    public String getString(int column) {
        return mCursor.getString(column);
    }

    @Override
    public short getShort(int column) {
        return mCursor.getShort(column);
    }

    @Override
    public int getInt(int column) {
        return mCursor.getInt(column);
    }

    @Override
    public long getLong(int column) {
        return mCursor.getLong(column);
    }

    @Override
    public float getFloat(int column) {
        return mCursor.getFloat(column);
    }

    @Override
    public double getDouble(int column) {
        return mCursor.getDouble(column);
    }

    @Override
    public boolean isNull(int column) {
        return mCursor.isNull(column);
    }

    @Override
    public boolean onMove(int oldPosition, int newPosition) {
        if (newPosition >= 0 && newPosition < getCount()) {
            mCursor.moveToPosition(mOrderedPositions.get(newPosition));
            return true;
        }

        return false;
    }
}
```

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.dataloaders;

import android.content.Context;
import android.database.Cursor;
import android.provider.BaseColumns;

import com.naman14.timber.provider.RecentStore;
import com.naman14.timber.provider.SongPlayCount;

import java.util.ArrayList;

public class TopTracksLoader extends SongLoader {

    public static final int NUMBER_OF_SONGS = 99;
    protected static QueryType mQueryType;
    private static Context mContext;

    public TopTracksLoader(final Context context, QueryType type) {
        mContext = context;
        mQueryType = type;
    }

    public static Cursor getCursor() {
        SortedCursor retCursor = null;
        if (mQueryType == QueryType.TopTracks) {
            retCursor = makeTopTracksCursor(mContext);
        } else if (mQueryType == QueryType.RecentSongs) {
            retCursor = makeRecentTracksCursor(mContext);
        }

        if (retCursor != null) {
            ArrayList<Long> missingIds = retCursor.getMissingIds();
            if (missingIds != null && missingIds.size() > 0) {
                for (long id : missingIds) {
                    if (mQueryType == QueryType.TopTracks) {
                        SongPlayCount.getInstance(mContext).removeItem(id);
                    } else if (mQueryType == QueryType.RecentSongs) {
                        RecentStore.getInstance(mContext).removeItem(id);
                    }
                }
            }
        }

        return retCursor;
    }

    public static final SortedCursor makeTopTracksCursor(final Context context) {

        Cursor songs = SongPlayCount.getInstance(context).getTopPlayedResults(NUMBER_OF_SONGS);

        try {
            return makeSortedCursor(context, songs,
                    songs.getColumnIndex(SongPlayCount.SongPlayCountColumns.ID));
        } finally {
            if (songs != null) {
                songs.close();
                songs = null;
            }
        }
    }

    public static final SortedCursor makeRecentTracksCursor(final Context context) {

        Cursor songs = RecentStore.getInstance(context).queryRecentIds(null);

        try {
            return makeSortedCursor(context, songs,
                    songs.getColumnIndex(SongPlayCount.SongPlayCountColumns.ID));
        } finally {
            if (songs != null) {
                songs.close();
                songs = null;
            }
        }
    }

    public static final SortedCursor makeSortedCursor(final Context context, final Cursor cursor,
                                                      final int idColumn) {
        if (cursor != null && cursor.moveToFirst()) {

            StringBuilder selection = new StringBuilder();
            selection.append(BaseColumns._ID);
            selection.append(" IN (");

            long[] order = new long[cursor.getCount()];

            long id = cursor.getLong(idColumn);
            selection.append(id);
            order[cursor.getPosition()] = id;

            while (cursor.moveToNext()) {
                selection.append(",");

                id = cursor.getLong(idColumn);
                order[cursor.getPosition()] = id;
                selection.append(String.valueOf(id));
            }

            selection.append(")");

            Cursor songCursor = makeSongCursor(context, selection.toString(), null);
            if (songCursor != null) {
                return new SortedCursor(songCursor, order, BaseColumns._ID, null);
            }
        }

        return null;
    }


    public enum QueryType {
        TopTracks,
        RecentSongs,
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dialogs\AddPlaylistDialog.java

```java
package com.naman14.timber.dialogs;

import android.app.Dialog;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.DialogFragment;
import android.view.View;

import com.afollestad.materialdialogs.MaterialDialog;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.dataloaders.PlaylistLoader;
import com.naman14.timber.models.Playlist;
import com.naman14.timber.models.Song;

import java.util.List;

/**
 * Created by naman on 20/12/15.
 */
public class AddPlaylistDialog extends DialogFragment {

    public static AddPlaylistDialog newInstance(Song song) {
        long[] songs = new long[1];
        songs[0] = song.id;
        return newInstance(songs);
    }

    public static AddPlaylistDialog newInstance(long[] songList) {
        AddPlaylistDialog dialog = new AddPlaylistDialog();
        Bundle bundle = new Bundle();
        bundle.putLongArray("songs", songList);
        dialog.setArguments(bundle);
        return dialog;
    }

    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final List<Playlist> playlists = PlaylistLoader.getPlaylists(getActivity(), false);
        CharSequence[] chars = new CharSequence[playlists.size() + 1];
        chars[0] = "Create new playlist";

        for (int i = 0; i < playlists.size(); i++) {
            chars[i + 1] = playlists.get(i).name;
        }
        return new MaterialDialog.Builder(getActivity()).title("Add to playlist").items(chars).itemsCallback(new MaterialDia
            @Override
            public void onSelection(MaterialDialog dialog, View itemView, int which, CharSequence text) {
                long[] songs = getArguments().getLongArray("songs");
                if (which == 0) {
                    CreatePlaylistDialog.newInstance(songs).show(getActivity().getSupportFragmentManager(), "CREATE_PLAYLIST
                    return;
                }

                MusicPlayer.addToPlaylist(getActivity(), songs, playlists.get(which - 1).id);
                dialog.dismiss();

            }
        }).build();
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dialogs\CreatePlaylistDialog.ja

```java
package com.naman14.timber.dialogs;

import android.app.Dialog;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.v4.app.DialogFragment;
import android.widget.Toast;

import com.afollestad.materialdialogs.MaterialDialog;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.fragments.PlaylistFragment;
import com.naman14.timber.models.Song;

/**
 * Created by naman on 20/12/15.
 */
public class CreatePlaylistDialog extends DialogFragment {

    public static CreatePlaylistDialog newInstance() {
        return newInstance((Song) null);
    }

    public static CreatePlaylistDialog newInstance(Song song) {
        long[] songs;
        if (song == null) {
            songs = new long[0];
        } else {
            songs = new long[1];
            songs[0] = song.id;
        }
        return newInstance(songs);
    }

    public static CreatePlaylistDialog newInstance(long[] songList) {
        CreatePlaylistDialog dialog = new CreatePlaylistDialog();
        Bundle bundle = new Bundle();
        bundle.putLongArray("songs", songList);
        dialog.setArguments(bundle);
        return dialog;
    }

    @NonNull
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new MaterialDialog.Builder(getActivity()).positiveText("Create").negativeText("Cancel").input("Enter playlist
            @Override
            public void onInput(@NonNull MaterialDialog dialog, CharSequence input) {

                long[] songs = getArguments().getLongArray("songs");
                long playistId = MusicPlayer.createPlaylist(getActivity(), input.toString());

                if (playistId != -1) {
                    if (songs != null && songs.length != 0)
                        MusicPlayer.addToPlaylist(getActivity(), songs, playistId);
                    else
                        Toast.makeText(getActivity(), "Created playlist", Toast.LENGTH_SHORT).show();
                    if (getParentFragment() instanceof PlaylistFragment) {
                        ((PlaylistFragment) getParentFragment()).updatePlaylists(playistId);
                    }
                } else {
                    Toast.makeText(getActivity(), "Unable to create playlist", Toast.LENGTH_SHORT).show();
                }

            }
        }).build();
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dialogs\LastFmLoginDialog.java

```java
package com.naman14.timber.dialogs;

import android.app.Dialog;
import android.app.DialogFragment;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.widget.EditText;
import android.widget.Toast;

import com.afollestad.materialdialogs.DialogAction;
import com.afollestad.materialdialogs.MaterialDialog;
import com.naman14.timber.R;
import com.naman14.timber.fragments.SettingsFragment;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.UserListener;
import com.naman14.timber.lastfmapi.models.UserLoginQuery;
import com.naman14.timber.utils.PreferencesUtility;

/**
 * Created by christoph on 17.07.16.
 */
public class LastFmLoginDialog extends DialogFragment {
    public static final String FRAGMENT_NAME = "LastFMLogin";

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new MaterialDialog.Builder(getActivity()).
                positiveText("Login").
                negativeText(getString(R.string.cancel)).
                title(getString(R.string.lastfm_login)).
                customView(R.layout.dialog_lastfm_login, false).
                onPositive(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        String username = ((EditText) dialog.findViewById(R.id.lastfm_username)).getText().toString();
                        String password = ((EditText) dialog.findViewById(R.id.lastfm_password)).getText().toString();
                        if (username.length() == 0 || password.length() == 0) return;
                        final ProgressDialog progressDialog = new ProgressDialog(getActivity());
                        progressDialog.setMessage("Logging in..");
                        progressDialog.show();
                        LastFmClient.getInstance(getActivity()).getUserLoginInfo(new UserLoginQuery(username, password), new

                            @Override
                            public void userSuccess() {
                                progressDialog.dismiss();
                                if (getTargetFragment() instanceof SettingsFragment) {
                                    ((SettingsFragment) getTargetFragment()).updateLastFM();
                                }
                            }

                            @Override
                            public void userInfoFailed() {
                                progressDialog.dismiss();
                                Toast.makeText(getTargetFragment().getActivity(), getString(R.string.lastfm_login_failture),
                            }
                        });
                    }
                }).build();
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\dialogs\StorageSelectDialog.jav

```java
package com.naman14.timber.dialogs;

import android.content.Context;
import android.content.DialogInterface;
import android.os.Environment;
import android.support.v7.app.AlertDialog;

import com.naman14.timber.R;

import java.io.File;
import java.io.FileFilter;

/**
 * Created by nv95 on 06.12.16.
 */

public class StorageSelectDialog implements DialogInterface.OnClickListener {

    private final AlertDialog mDialog;
    private final File[] mStorages;
    private OnDirSelectListener mDirSelectListener;

    public StorageSelectDialog(final Context context) {
        mStorages = getAvailableStorages(context);
        String[] names = new String[mStorages.length];
        for (int i=0;i<mStorages.length;i++) {
            names[i] = mStorages[i].getName();
        }
        mDialog = new AlertDialog.Builder(context)
                .setItems(names, this)
                .setNegativeButton(android.R.string.cancel, null)
                .setNeutralButton(R.string.menu_show_as_entry_default, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        mDirSelectListener.onDirSelected(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY
                    }
                })
                .setCancelable(true)
                .setTitle(R.string.select_storage)
                .create();
    }

    public StorageSelectDialog setDirSelectListener(OnDirSelectListener dirSelectListener) {
        this.mDirSelectListener = dirSelectListener;
        return this;
    }

    public void show() {
        mDialog.show();
    }

    @Override
    public void onClick(DialogInterface dialogInterface, int position) {
        File dir = mStorages[position];
        mDirSelectListener.onDirSelected(dir);
    }


    private static File[] getAvailableStorages(Context context) {
        File storageRoot = new File("/storage");
        return storageRoot.listFiles(new FileFilter() {
            @Override
            public boolean accept(File file) {
                return file.canRead();
            }
        });
    }

    public interface OnDirSelectListener {
        void onDirSelected(File dir);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\AlbumDetailFragment.j

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */
package com.naman14.timber.fragments;

import android.annotation.TargetApi;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.support.design.widget.AppBarLayout;
import android.support.design.widget.CollapsingToolbarLayout;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.graphics.Palette;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.transition.Transition;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.activities.MainActivity;
import com.naman14.timber.adapters.AlbumSongsAdapter;
import com.naman14.timber.dataloaders.AlbumLoader;
import com.naman14.timber.dataloaders.AlbumSongLoader;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.listeners.SimplelTransitionListener;
import com.naman14.timber.models.Album;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.ATEUtils;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.FabAnimationUtils;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.ImageUtils;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.SortOrder;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.listener.ImageLoadingListener;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

import java.util.List;
```

```java
public class AlbumDetailFragment extends Fragment {

    private long albumID = -1;

    private ImageView albumArt, artistArt;
    private TextView albumTitle, albumDetails;
    private AppCompatActivity mContext;

    private RecyclerView recyclerView;
    private AlbumSongsAdapter mAdapter;

    private Toolbar toolbar;

    private Album album;

    private CollapsingToolbarLayout collapsingToolbarLayout;
    private AppBarLayout appBarLayout;
    private FloatingActionButton fab;

    private boolean loadFailed = false;

    private PreferencesUtility mPreferences;
    private Context context;
    private int primaryColor = -1;

    public static AlbumDetailFragment newInstance(long id, boolean useTransition, String transitionName) {
        AlbumDetailFragment fragment = new AlbumDetailFragment();
        Bundle args = new Bundle();
        args.putLong(Constants.ALBUM_ID, id);
        args.putBoolean("transition", useTransition);
        if (useTransition)
            args.putString("transition_name", transitionName);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            albumID = getArguments().getLong(Constants.ALBUM_ID);
        }
        context = getActivity();
        mContext = (AppCompatActivity) context;
        mPreferences = PreferencesUtility.getInstance(context);
    }

    @TargetApi(21)
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        final View rootView = inflater.inflate(
                R.layout.fragment_album_detail, container, false);

        albumArt = (ImageView) rootView.findViewById(R.id.album_art);
        artistArt = (ImageView) rootView.findViewById(R.id.artist_art);
        albumTitle = (TextView) rootView.findViewById(R.id.album_title);
        albumDetails = (TextView) rootView.findViewById(R.id.album_details);

        toolbar = (Toolbar) rootView.findViewById(R.id.toolbar);

        fab = (FloatingActionButton) rootView.findViewById(R.id.fab);

        if (getArguments().getBoolean("transition")) {
            albumArt.setTransitionName(getArguments().getString("transition_name"));
        }
        recyclerView = (RecyclerView) rootView.findViewById(R.id.recyclerview);
        collapsingToolbarLayout = (CollapsingToolbarLayout) rootView.findViewById(R.id.collapsing_toolbar);
        appBarLayout = (AppBarLayout) rootView.findViewById(R.id.app_bar);
        recyclerView.setEnabled(false);
        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        album = AlbumLoader.getAlbum(getActivity(), albumID);

        setAlbumart();

        setUpEverything();


        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        AlbumSongsAdapter adapter = (AlbumSongsAdapter) recyclerView.getAdapter();
                        MusicPlayer.playAll(getActivity(), adapter.getSongIds(), 0, albumID, TimberUtils.IdType.Album, true)
                        NavigationUtils.navigateToNowplaying(getActivity(), false);
                    }
                }, 150);
            }
        });

        return rootView;
    }

    private void setupToolbar() {

        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);
        final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setDisplayHomeAsUpEnabled(true);
        collapsingToolbarLayout.setTitle(album.title);

    }

    private void setAlbumart() {
        ImageUtils.loadAlbumArtIntoView(album.id, albumArt, new ImageLoadingListener() {
                    @Override
                    public void onLoadingStarted(String imageUri, View view) {
                    }

                    @Override
                    public void onLoadingFailed(String imageUri, View view, FailReason failReason) {
                        loadFailed = true;
                        MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(context)
                                .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                                .setColor(TimberUtils.getBlackWhiteColor(Config.accentColor(context, Helpers.getATEKey(conte
                        ATEUtils.setFabBackgroundTint(fab, Config.accentColor(context, Helpers.getATEKey(context)));
                        fab.setImageDrawable(builder.build());
                    }

                    @Override
                    public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                        try {
                            new Palette.Builder(loadedImage).generate(new Palette.PaletteAsyncListener() {
                                                                          @Override
                                                                          public void onGenerated(Palette palette) {
                                                                              Palette.Swatch swatch = palette.getVibrantSwat
                                                                              if (swatch != null) {
                                                                                  primaryColor = swatch.getRgb();
                                                                                  collapsingToolbarLayout.setContentScrimCol
                                                                                  if (getActivity() != null)
                                                                                      ATEUtils.setStatusBarColor(getActivity
                                                                              } else {
                                                                                  Palette.Swatch swatchMuted = palette.getMu
                                                                                  if (swatchMuted != null) {
                                                                                      primaryColor = swatchMuted.getRgb();
                                                                                      collapsingToolbarLayout.setContentScri
                                                                                      if (getActivity() != null)
```

```
                                                                                    ATEUtils.setStatusBarColor(getActi
                                                                            }
                                                                        }

                                                                        if (getActivity() != null) {
                                                                            MaterialDrawableBuilder builder = Material
                                                                                    .setIcon(MaterialDrawableBuilder.I
                                                                                    .setSizeDp(30);
                                                                            if (primaryColor != -1) {
                                                                                builder.setColor(TimberUtils.getBlackW
                                                                                ATEUtils.setFabBackgroundTint(fab, pri
                                                                                fab.setImageDrawable(builder.build());
                                                                            } else {
                                                                                if (context != null) {
                                                                                    ATEUtils.setFabBackgroundTint(fab,
                                                                                    builder.setColor(TimberUtils.getBl
                                                                                    fab.setImageDrawable(builder.build
                                                                                }
                                                                            }
                                                                        }
                                                                    }
                                                                }

                            );
                        } catch (
                                Exception ignored
                                )

                        {

                        }
                    }

                    @Override
                    public void onLoadingCancelled(String imageUri, View view) {
                    }

                }

        );
    }

    private void setAlbumDetails() {

        String songCount = TimberUtils.makeLabel(getActivity(), R.plurals.Nsongs, album.songCount);

        String year = (album.year != 0) ? (" - " + String.valueOf(album.year)) : "";

        albumTitle.setText(album.title);
        albumDetails.setText(album.artistName + " - " + songCount + year);


    }

    private void setUpAlbumSongs() {

        List<Song> songList = AlbumSongLoader.getSongsForAlbum(getActivity(), albumID);
        mAdapter = new AlbumSongsAdapter(getActivity(), songList, albumID);
        recyclerView.addItemDecoration(new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST));
        recyclerView.setAdapter(mAdapter);

    }

    private void setUpEverything() {
        setupToolbar();
        setAlbumDetails();
        setUpAlbumSongs();
    }

    private void reloadAdapter() {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(final Void... unused) {
                List<Song> songList = AlbumSongLoader.getSongsForAlbum(getActivity(), albumID);
                mAdapter.updateDataSet(songList);
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                mAdapter.notifyDataSetChanged();
            }
        }.execute();
    }

    @Override
    public void onActivityCreated(final Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.album_detail, menu);
        if (getActivity() != null)
            ATE.applyMenu(getActivity(), "dark_theme", menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_go_to_artist:
                NavigationUtils.goToArtist(getContext(), album.artistId);
                break;
            case R.id.popup_song_addto_queue:
                MusicPlayer.addToQueue(context, mAdapter.getSongIds(), -1, TimberUtils.IdType.NA);
                break;
            case R.id.popup_song_addto_playlist:
                AddPlaylistDialog.newInstance(mAdapter.getSongIds()).show(mContext.getSupportFragmentManager(), "ADD_PLAYLIS
                break;
            case R.id.menu_sort_by_az:
                mPreferences.setAlbumSongSortOrder(SortOrder.AlbumSongSortOrder.SONG_A_Z);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_za:
                mPreferences.setAlbumSongSortOrder(SortOrder.AlbumSongSortOrder.SONG_Z_A);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_year:
                mPreferences.setAlbumSongSortOrder(SortOrder.AlbumSongSortOrder.SONG_YEAR);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_duration:
                mPreferences.setAlbumSongSortOrder(SortOrder.AlbumSongSortOrder.SONG_DURATION);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_track_number:
                mPreferences.setAlbumSongSortOrder(SortOrder.AlbumSongSortOrder.SONG_TRACK_LIST);
                reloadAdapter();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onResume() {
        super.onResume();
        String ateKey = Helpers.getATEKey(getActivity());
        toolbar.setBackgroundColor(Color.TRANSPARENT);
        if (primaryColor != -1 && getActivity() != null) {
            collapsingToolbarLayout.setContentScrimColor(primaryColor);
            ATEUtils.setFabBackgroundTint(fab, primaryColor);
            ATEUtils.setStatusBarColor(getActivity(), ateKey, primaryColor);
        }

    }

    private class EnterTransitionListener extends SimplelTransitionListener {

        @TargetApi(21)
        public void onTransitionEnd(Transition paramTransition) {
            FabAnimationUtils.scaleIn(fab);
        }

        public void onTransitionStart(Transition paramTransition) {
            FabAnimationUtils.scaleOut(fab, 0, null);
        }

    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\AlbumFragment.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.graphics.Rect;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.adapters.AlbumAdapter;
import com.naman14.timber.dataloaders.AlbumLoader;
import com.naman14.timber.models.Album;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.SortOrder;
import com.naman14.timber.widgets.BaseRecyclerView;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.FastScroller;

import java.util.List;

public class AlbumFragment extends Fragment {

    private AlbumAdapter mAdapter;
    private BaseRecyclerView recyclerView;
    private FastScroller fastScroller;
    private GridLayoutManager layoutManager;
    private RecyclerView.ItemDecoration itemDecoration;
    private PreferencesUtility mPreferences;
    private boolean isGrid;

    @Override
    public void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = PreferencesUtility.getInstance(getActivity());
        isGrid = mPreferences.isAlbumsInGrid();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_recyclerview, container, false);

        recyclerView = rootView.findViewById(R.id.recyclerview);
        fastScroller = rootView.findViewById(R.id.fastscroller);

        recyclerView.setEmptyView(getActivity(), rootView.findViewById(R.id.list_empty), "No media found");

        setLayoutManager();
        if (getActivity() != null)
            new loadAlbums().execute("");
        return rootView;
    }

    private void setLayoutManager() {
        if (isGrid) {
            layoutManager = new GridLayoutManager(getActivity(), 2);
            fastScroller.setVisibility(View.GONE);
        } else {
            layoutManager = new GridLayoutManager(getActivity(), 1);
            fastScroller.setVisibility(View.VISIBLE);
            fastScroller.setRecyclerView(recyclerView);
        }
        recyclerView.setLayoutManager(layoutManager);
    }

    private void setItemDecoration() {
        if (isGrid) {
            int spacingInPixels = getActivity().getResources().getDimensionPixelSize(R.dimen.spacing_card_album_grid);
            itemDecoration = new SpacesItemDecoration(spacingInPixels);
        } else {
            itemDecoration = new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST);
        }
        recyclerView.addItemDecoration(itemDecoration);
    }

    private void updateLayoutManager(int column) {
        recyclerView.removeItemDecoration(itemDecoration);
        recyclerView.setAdapter(new AlbumAdapter(getActivity(), AlbumLoader.getAllAlbums(getActivity())));
        layoutManager.setSpanCount(column);
        layoutManager.requestLayout();
        setItemDecoration();
    }

    private void reloadAdapter() {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(final Void... unused) {
                List<Album> albumList = AlbumLoader.getAllAlbums(getActivity());
                mAdapter.updateDataSet(albumList);
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                mAdapter.notifyDataSetChanged();
            }
        }.execute();
    }

    @Override
    public void onActivityCreated(final Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.album_sort_by, menu);
        inflater.inflate(R.menu.menu_show_as, menu);

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_sort_by_az:
                mPreferences.setAlbumSortOrder(SortOrder.AlbumSortOrder.ALBUM_A_Z);
```

```java
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_za:
                mPreferences.setAlbumSortOrder(SortOrder.AlbumSortOrder.ALBUM_Z_A);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_year:
                mPreferences.setAlbumSortOrder(SortOrder.AlbumSortOrder.ALBUM_YEAR);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_artist:
                mPreferences.setAlbumSortOrder(SortOrder.AlbumSortOrder.ALBUM_ARTIST);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_number_of_songs:
                mPreferences.setAlbumSortOrder(SortOrder.AlbumSortOrder.ALBUM_NUMBER_OF_SONGS);
                reloadAdapter();
                return true;
            case R.id.menu_show_as_list:
                mPreferences.setAlbumsInGrid(false);
                isGrid = false;
                updateLayoutManager(1);
                return true;
            case R.id.menu_show_as_grid:
                mPreferences.setAlbumsInGrid(true);
                isGrid = true;
                updateLayoutManager(2);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    public class SpacesItemDecoration extends RecyclerView.ItemDecoration {
        private int space;

        public SpacesItemDecoration(int space) {
            this.space = space;
        }

        @Override
        public void getItemOffsets(Rect outRect, View view,
                                   RecyclerView parent, RecyclerView.State state) {


            outRect.left = space;
            outRect.top = space;
            outRect.right = space;
            outRect.bottom = space;

        }
    }

    private class loadAlbums extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            if (getActivity() != null)
                mAdapter = new AlbumAdapter(getActivity(), AlbumLoader.getAllAlbums(getActivity()));
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            recyclerView.setAdapter(mAdapter);
            //to add spacing between cards
            if (getActivity() != null) {
                setItemDecoration();
            }

        }
        @Override
        protected void onPreExecute() {
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\ArtistBioFragment.jav

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentStatePagerAdapter;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.dataloaders.ArtistLoader;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.ArtistInfoListener;
import com.naman14.timber.lastfmapi.models.ArtistQuery;
import com.naman14.timber.lastfmapi.models.LastfmArtist;
import com.naman14.timber.models.Artist;
import com.naman14.timber.subfragments.ArtistTagFragment;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.widgets.MultiViewPager;

public class ArtistBioFragment extends Fragment {

    long artistID = -1;

    public static ArtistBioFragment newInstance(long id) {
        ArtistBioFragment fragment = new ArtistBioFragment();
        Bundle args = new Bundle();
        args.putLong(Constants.ARTIST_ID, id);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            artistID = getArguments().getLong(Constants.ARTIST_ID);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_artist_bio, container, false);

        Artist artist = ArtistLoader.getArtist(getActivity(), artistID);

        LastFmClient.getInstance(getActivity()).getArtistInfo(new ArtistQuery(artist.name), new ArtistInfoListener() {
            @Override
            public void artistInfoSucess(LastfmArtist artist) {

            }

            @Override
            public void artistInfoFailed() {
            }
        });

        final MultiViewPager pager = (MultiViewPager) rootView.findViewById(R.id.tagspager);

        final FragmentStatePagerAdapter adapter = new FragmentStatePagerAdapter(getActivity().getSupportFragmentManager()) {

            @Override
            public int getCount() {
                return 20;
            }

            @Override
            public Fragment getItem(int position) {
                return ArtistTagFragment.newInstance(position);
            }

        };
        pager.setAdapter(adapter);

        return rootView;

    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\ArtistDetailFragment.

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.support.design.widget.AppBarLayout;
import android.support.design.widget.CollapsingToolbarLayout;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.graphics.Palette;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

import com.afollestad.appthemeengine.ATE;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.adapters.ArtistSongAdapter;
import com.naman14.timber.dataloaders.ArtistLoader;
import com.naman14.timber.dataloaders.ArtistSongLoader;
import com.naman14.timber.dialogs.AddPlaylistDialog;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.ArtistInfoListener;
import com.naman14.timber.lastfmapi.models.ArtistQuery;
import com.naman14.timber.lastfmapi.models.LastfmArtist;
import com.naman14.timber.models.Artist;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.ATEUtils;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.ImageUtils;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import java.util.List;

public class ArtistDetailFragment extends Fragment {

    private long artistID = -1;
    private ImageView artistArt;
    private Toolbar toolbar;
    private CollapsingToolbarLayout collapsingToolbarLayout;
    private AppBarLayout appBarLayout;
    private boolean largeImageLoaded = false;
    private int primaryColor = -1;
    private ArtistSongAdapter mAdapter;

    public static ArtistDetailFragment newInstance(long id, boolean useTransition, String transitionName) {
        ArtistDetailFragment fragment = new ArtistDetailFragment();
        Bundle args = new Bundle();
        args.putLong(Constants.ARTIST_ID, id);
        args.putBoolean("transition", useTransition);
        if (useTransition)
            args.putString("transition_name", transitionName);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            artistID = getArguments().getLong(Constants.ARTIST_ID);
        }
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_artist_detail, container, false);

        artistArt = (ImageView) rootView.findViewById(R.id.artist_art);

        collapsingToolbarLayout = (CollapsingToolbarLayout) rootView.findViewById(R.id.collapsing_toolbar);
        appBarLayout = (AppBarLayout) rootView.findViewById(R.id.app_bar);

        if (getArguments().getBoolean("transition")) {
            artistArt.setTransitionName(getArguments().getString("transition_name"));
        }

        toolbar = (Toolbar) rootView.findViewById(R.id.toolbar);
        setupToolbar();
        setUpArtistDetails();

        getChildFragmentManager().beginTransaction().replace(R.id.container, ArtistMusicFragment.newInstance(artistID)).comm


        return rootView;
    }

    private void setupToolbar() {

        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);

        final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setDisplayShowTitleEnabled(false);
        ab.setDisplayHomeAsUpEnabled(true);
    }

    private void setUpArtistDetails() {

        final Artist artist = ArtistLoader.getArtist(getActivity(), artistID);
        List<Song> songList = ArtistSongLoader.getSongsForArtist(getActivity(), artistID);
        mAdapter = new ArtistSongAdapter(getActivity(), songList, artistID);

        collapsingToolbarLayout.setTitle(artist.name);

        LastFmClient.getInstance(getActivity()).getArtistInfo(new ArtistQuery(artist.name), new ArtistInfoListener() {
            @Override
            public void artistInfoSucess(final LastfmArtist artist) {
                if (artist != null) {

                    ImageLoader.getInstance().displayImage(artist.mArtwork.get(4).mUrl, artistArt,
                            new DisplayImageOptions.Builder().cacheInMemory(true)
```

```
                                .cacheOnDisk(true)
                                .showImageOnFail(R.drawable.ic_empty_music2)
                                .build(), new SimpleImageLoadingListener() {
                            @Override
                            public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                                largeImageLoaded = true;
                                try {
                                    new Palette.Builder(loadedImage).generate(new Palette.PaletteAsyncListener() {
                                        @Override
                                        public void onGenerated(Palette palette) {
                                            Palette.Swatch swatch = palette.getVibrantSwatch();
                                            if (swatch != null) {
                                                primaryColor = swatch.getRgb();
                                                collapsingToolbarLayout.setContentScrimColor(primaryColor);
                                                if (getActivity() != null)
                                                    ATEUtils.setStatusBarColor(getActivity(), Helpers.getATEKey(getActiv
                                            } else {
                                                Palette.Swatch swatchMuted = palette.getMutedSwatch();
                                                if (swatchMuted != null) {
                                                    primaryColor = swatchMuted.getRgb();
                                                    collapsingToolbarLayout.setContentScrimColor(primaryColor);
                                                    if (getActivity() != null)
                                                        ATEUtils.setStatusBarColor(getActivity(), Helpers.getATEKey(getA
                                                }
                                            }

                                        }
                                    });
                                } catch (Exception ignored) {

                                }
                            }
                        });
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        setBlurredPlaceholder(artist);
                    }
                }, 100);

            }
        }

        @Override
        public void artistInfoFailed() {

        }
    });

}

private void setBlurredPlaceholder(LastfmArtist artist) {
    ImageLoader.getInstance().loadImage(artist.mArtwork.get(1).mUrl, new SimpleImageLoadingListener() {
        @Override
        public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
            if (getActivity() != null && !largeImageLoaded)
                new setBlurredAlbumArt().execute(loadedImage);

        }
    });
}

@Override
public void onActivityCreated(final Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    setHasOptionsMenu(true);
}

@Override
```

```
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.artist_detail, menu);
        if (getActivity() != null)
            ATE.applyMenu(getActivity(), "dark_theme", menu);
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.popup_song_addto_queue:
                MusicPlayer.addToQueue(getContext(), mAdapter.getSongIds(), -1, TimberUtils.IdType.NA);
                break;
            case R.id.popup_song_addto_playlist:
                AddPlaylistDialog.newInstance(mAdapter.getSongIds()).show(getActivity().getSupportFragmentManager(), "ADD_PL
                break;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onResume() {
        super.onResume();
        toolbar.setBackgroundColor(Color.TRANSPARENT);
        if (primaryColor != -1 && getActivity() != null) {
            collapsingToolbarLayout.setContentScrimColor(primaryColor);
            String ateKey = Helpers.getATEKey(getActivity());
            ATEUtils.setStatusBarColor(getActivity(), ateKey, primaryColor);
        }

    }

    private class setBlurredAlbumArt extends AsyncTask<Bitmap, Void, Drawable> {

        @Override
        protected Drawable doInBackground(Bitmap... loadedImage) {
            Drawable drawable = null;
            try {
                drawable = ImageUtils.createBlurredImageFromBitmap(loadedImage[0], getActivity(), 3);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return drawable;
        }

        @Override
        protected void onPostExecute(Drawable result) {
            if (result != null && !largeImageLoaded) {
                artistArt.setImageDrawable(result);
            }
        }

        @Override
        protected void onPreExecute() {
        }
    }

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.graphics.Rect;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.adapters.ArtistAdapter;
import com.naman14.timber.dataloaders.ArtistLoader;
import com.naman14.timber.models.Artist;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.SortOrder;
import com.naman14.timber.widgets.BaseRecyclerView;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.FastScroller;

import java.util.List;

public class ArtistFragment extends Fragment {

    private ArtistAdapter mAdapter;
    private BaseRecyclerView recyclerView;
    private GridLayoutManager layoutManager;
    private RecyclerView.ItemDecoration itemDecoration;
    private PreferencesUtility mPreferences;
    private boolean isGrid;

    @Override
    public void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = PreferencesUtility.getInstance(getActivity());
        isGrid = mPreferences.isArtistsInGrid();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_recyclerview, container, false);

        recyclerView =  rootView.findViewById(R.id.recyclerview);
        FastScroller fastScroller = rootView.findViewById(R.id.fastscroller);
        fastScroller.setRecyclerView(recyclerView);
        recyclerView.setEmptyView(getActivity(), rootView.findViewById(R.id.list_empty), "No media found");

        setLayoutManager();

        if (getActivity() != null)
```

```java
            new loadArtists().execute("");
        return rootView;
    }

    private void setLayoutManager() {
        if (isGrid) {
            layoutManager = new GridLayoutManager(getActivity(), 2);
        } else {
            layoutManager = new GridLayoutManager(getActivity(), 1);
        }
        recyclerView.setLayoutManager(layoutManager);
    }

    private void setItemDecoration() {
        if (isGrid) {
            int spacingInPixels = getActivity().getResources().getDimensionPixelSize(R.dimen.spacing_card_album_grid);
            itemDecoration = new SpacesItemDecoration(spacingInPixels);
        } else {
            itemDecoration = new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST);
        }
        recyclerView.addItemDecoration(itemDecoration);
    }

    private void updateLayoutManager(int column) {
        recyclerView.removeItemDecoration(itemDecoration);
        recyclerView.setAdapter(new ArtistAdapter(getActivity(), ArtistLoader.getAllArtists(getActivity())));
        layoutManager.setSpanCount(column);
        layoutManager.requestLayout();
        setItemDecoration();
    }

    private void reloadAdapter() {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(final Void... unused) {
                List<Artist> artistList = ArtistLoader.getAllArtists(getActivity());
                mAdapter.updateDataSet(artistList);
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                mAdapter.notifyDataSetChanged();
            }
        }.execute();
    }

    @Override
    public void onActivityCreated(final Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.artist_sort_by, menu);
        inflater.inflate(R.menu.menu_show_as, menu);

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_sort_by_az:
                mPreferences.setArtistSortOrder(SortOrder.ArtistSortOrder.ARTIST_A_Z);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_za:
                mPreferences.setArtistSortOrder(SortOrder.ArtistSortOrder.ARTIST_Z_A);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_number_of_songs:
                mPreferences.setArtistSortOrder(SortOrder.ArtistSortOrder.ARTIST_NUMBER_OF_SONGS);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_number_of_albums:
                mPreferences.setArtistSortOrder(SortOrder.ArtistSortOrder.ARTIST_NUMBER_OF_ALBUMS);
                reloadAdapter();
                return true;
            case R.id.menu_show_as_list:
                mPreferences.setArtistsInGrid(false);
                isGrid = false;
                updateLayoutManager(1);
                return true;
            case R.id.menu_show_as_grid:
                mPreferences.setArtistsInGrid(true);
                isGrid = true;
                updateLayoutManager(2);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private class loadArtists extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            if (getActivity() != null)
                mAdapter = new ArtistAdapter(getActivity(), ArtistLoader.getAllArtists(getActivity()));
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            if (mAdapter != null) {
                mAdapter.setHasStableIds(true);
                recyclerView.setAdapter(mAdapter);
            }
            if (getActivity() != null) {
                setItemDecoration();
            }
        }

        @Override
        protected void onPreExecute() {
        }
    }

    public class SpacesItemDecoration extends RecyclerView.ItemDecoration {
        private int space;

        public SpacesItemDecoration(int space) {
            this.space = space;
        }

        @Override
        public void getItemOffsets(Rect outRect, View view,
                                   RecyclerView parent, RecyclerView.State state) {
            outRect.left = space;
            outRect.top = space;
            outRect.right = space;
            outRect.bottom = space;

        }
    }


}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.adapters.ArtistSongAdapter;
import com.naman14.timber.dataloaders.ArtistSongLoader;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.widgets.DividerItemDecoration;

import java.util.ArrayList;

public class ArtistMusicFragment extends Fragment {

    public static RecyclerView songsRecyclerview;
    private long artistID = -1;
    private ArtistSongAdapter mSongAdapter;

    public static ArtistMusicFragment newInstance(long id) {
        ArtistMusicFragment fragment = new ArtistMusicFragment();
        Bundle args = new Bundle();
        args.putLong(Constants.ARTIST_ID, id);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            artistID = getArguments().getLong(Constants.ARTIST_ID);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_artist_music, container, false);

        songsRecyclerview = (RecyclerView) rootView.findViewById(R.id.recycler_view_songs);

        setUpSongs();


        return rootView;
    }


    private void setUpSongs() {
        songsRecyclerview.setLayoutManager(new LinearLayoutManager(getActivity()));

        ArrayList<Song> songList;
        songList = ArtistSongLoader.getSongsForArtist(getActivity(), artistID);

        // adding one dummy song to top of arraylist
        //there will be albums header at this position in recyclerview
        songList.add(0, new Song(-1, -1, -1, "dummy", "dummy", "dummy", -1, -1));

        mSongAdapter = new ArtistSongAdapter(getActivity(), songList, artistID);
        songsRecyclerview.addItemDecoration(new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST));
        songsRecyclerview.setAdapter(mSongAdapter);
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\FoldersFragment.java

```java
package com.naman14.timber.fragments;

import android.app.Activity;
import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ProgressBar;

import com.afollestad.appthemeengine.ATE;
import com.naman14.timber.R;
import com.naman14.timber.adapters.FolderAdapter;
import com.naman14.timber.dialogs.StorageSelectDialog;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.FastScroller;

import java.io.File;

/**
 * Created by nv95 on 10.11.16.
 */

public class FoldersFragment extends Fragment implements StorageSelectDialog.OnDirSelectListener {

    private FolderAdapter mAdapter;
    private RecyclerView recyclerView;
    private FastScroller fastScroller;
    private ProgressBar mProgressBar;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_folders, container, false);

        Toolbar toolbar = (Toolbar) rootView.findViewById(R.id.toolbar);
        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);

        ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setHomeAsUpIndicator(R.drawable.ic_menu);
        ab.setDisplayHomeAsUpEnabled(true);
        ab.setTitle(R.string.folders);

        recyclerView = (RecyclerView) rootView.findViewById(R.id.recyclerview);
        fastScroller = (FastScroller) rootView.findViewById(R.id.fastscroller);
        mProgressBar = (ProgressBar) rootView.findViewById(R.id.progressBar);

        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        if (getActivity() != null)
            new loadFolders().execute("");
        return rootView;
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        boolean dark = PreferenceManager.getDefaultSharedPreferences(getActivity()).getBoolean("dark_theme", false);
        if (dark) {
            ATE.apply(this, "dark_theme");
        } else {
            ATE.apply(this, "light_theme");
        }
        if (mAdapter != null) {
            mAdapter.applyTheme(dark);
            mAdapter.notifyDataSetChanged();
        }
    }

    private void setItemDecoration() {
        recyclerView.addItemDecoration(new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST));
    }

    @Override
    public void onActivityCreated(final Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.menu_folders, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.action_storages) {
            new StorageSelectDialog(getActivity())
                    .setDirSelectListener(this)
                    .show();
        }
        return super.onOptionsItemSelected(item);
    }

    public void updateTheme() {
        Context context = getActivity();
        if (context != null) {
            boolean dark = PreferenceManager.getDefaultSharedPreferences(context).getBoolean("dark_theme", false);
            mAdapter.applyTheme(dark);
        }
    }

    @Override
    public void onDirSelected(File dir) {
        mAdapter.updateDataSetAsync(dir);
    }

    private class loadFolders extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            Activity activity = getActivity();
            if (activity != null) {
                mAdapter = new FolderAdapter(activity, new File(PreferencesUtility.getInstance(activity).getLastFolder()));
                updateTheme();
            }
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            recyclerView.setAdapter(mAdapter);
            //to add spacing between cards
            if (getActivity() != null) {
                setItemDecoration();
            }
            mAdapter.notifyDataSetChanged();
            mProgressBar.setVisibility(View.GONE);
            fastScroller.setVisibility(View.VISIBLE);
            fastScroller.setRecyclerView(recyclerView);
        }

        @Override
        protected void onPreExecute() {
        }
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.design.widget.TabLayout;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.Config;
import com.naman14.timber.R;
import com.naman14.timber.utils.ATEUtils;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.PreferencesUtility;

import java.util.ArrayList;
import java.util.List;

public class MainFragment extends Fragment {

    private PreferencesUtility mPreferences;
    private ViewPager viewPager;

    @Override
    public void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = PreferencesUtility.getInstance(getActivity());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_main, container, false);

        Toolbar toolbar = (Toolbar) rootView.findViewById(R.id.toolbar);
        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);

        final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setHomeAsUpIndicator(R.drawable.ic_menu);
        ab.setDisplayHomeAsUpEnabled(true);


        viewPager = (ViewPager) rootView.findViewById(R.id.viewpager);
        if (viewPager != null) {
            setupViewPager(viewPager);
            viewPager.setOffscreenPageLimit(2);
        }
        TabLayout tabLayout = (TabLayout) rootView.findViewById(R.id.tabs);
        tabLayout.setupWithViewPager(viewPager);

        return rootView;

    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        if (PreferenceManager.getDefaultSharedPreferences(getActivity()).getBoolean("dark_theme", false)) {
            ATE.apply(this, "dark_theme");
        } else {
            ATE.apply(this, "light_theme");
        }
        viewPager.setCurrentItem(mPreferences.getStartPageIndex());
    }

    private void setupViewPager(ViewPager viewPager) {
        Adapter adapter = new Adapter(getChildFragmentManager());
        adapter.addFragment(new SongsFragment(), this.getString(R.string.songs));
        adapter.addFragment(new AlbumFragment(), this.getString(R.string.albums));
        adapter.addFragment(new ArtistFragment(), this.getString(R.string.artists));
        viewPager.setAdapter(adapter);
    }

    @Override
    public void onPause() {
        super.onPause();
        if (mPreferences.lastOpenedIsStartPagePreference()) {
            mPreferences.setStartPageIndex(viewPager.getCurrentItem());
        }
    }

    @Override
    public void onResume() {
        super.onResume();
        String ateKey = Helpers.getATEKey(getActivity());
        ATEUtils.setStatusBarColor(getActivity(), ateKey, Config.primaryColor(getActivity(), ateKey));

    }

    @Override
    public void onStart() {
        super.onStart();
    }

    static class Adapter extends FragmentPagerAdapter {
        private final List<Fragment> mFragments = new ArrayList<>();
        private final List<String> mFragmentTitles = new ArrayList<>();

        public Adapter(FragmentManager fm) {
            super(fm);
        }

        public void addFragment(Fragment fragment, String title) {
            mFragments.add(fragment);
            mFragmentTitles.add(title);
        }

        @Override
        public Fragment getItem(int position) {
            return mFragments.get(position);
        }

        @Override
        public int getCount() {
            return mFragments.size();
        }
        @Override
        public CharSequence getPageTitle(int position) {
            return mFragmentTitles.get(position);
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\PlaylistFragment.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Rect;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentStatePagerAdapter;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.GridLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

import com.afollestad.appthemeengine.ATE;
import com.naman14.timber.R;
import com.naman14.timber.adapters.PlaylistAdapter;
import com.naman14.timber.dataloaders.PlaylistLoader;
import com.naman14.timber.dialogs.CreatePlaylistDialog;
import com.naman14.timber.models.Playlist;
import com.naman14.timber.subfragments.PlaylistPagerFragment;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.MultiViewPager;

import java.util.ArrayList;
import java.util.List;

public class PlaylistFragment extends Fragment {

    private int playlistcount;
    private FragmentStatePagerAdapter adapter;
    private MultiViewPager pager;
    private RecyclerView recyclerView;
    private GridLayoutManager layoutManager;
    private RecyclerView.ItemDecoration itemDecoration;

    private PreferencesUtility mPreferences;
    private boolean isGrid;
    private boolean isDefault;
    private boolean showAuto;
    private PlaylistAdapter mAdapter;

    private List<Playlist> playlists = new ArrayList<>();

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = PreferencesUtility.getInstance(getActivity());
        isGrid = mPreferences.getPlaylistView() == Constants.PLAYLIST_VIEW_GRID;
        isDefault = mPreferences.getPlaylistView() == Constants.PLAYLIST_VIEW_DEFAULT;
        showAuto = mPreferences.showAutoPlaylist();

    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_playlist, container, false);

        Toolbar toolbar = (Toolbar) rootView.findViewById(R.id.toolbar);
        pager = (MultiViewPager) rootView.findViewById(R.id.playlistpager);
        recyclerView = (RecyclerView) rootView.findViewById(R.id.recyclerview);


        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);

        final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setHomeAsUpIndicator(R.drawable.ic_menu);
        ab.setDisplayHomeAsUpEnabled(true);
        ab.setTitle(R.string.playlists);

        playlists = PlaylistLoader.getPlaylists(getActivity(), showAuto);
        playlistcount = playlists.size();

        if (isDefault) {
            initPager();
        } else {
            initRecyclerView();
        }

        return rootView;

    }


    private void initPager() {
        pager.setVisibility(View.VISIBLE);
        recyclerView.setVisibility(View.GONE);
        recyclerView.setAdapter(null);
        adapter = new FragmentStatePagerAdapter(getChildFragmentManager()) {

            @Override
            public int getCount() {
                return playlistcount;
            }

            @Override
            public Fragment getItem(int position) {
                return PlaylistPagerFragment.newInstance(position);
            }

        };
        pager.setAdapter(adapter);
        pager.setOffscreenPageLimit(3);
    }

    private void initRecyclerView() {
        recyclerView.setVisibility(View.VISIBLE);
        pager.setVisibility(View.GONE);
        setLayoutManager();
        mAdapter = new PlaylistAdapter(getActivity(), playlists);

        recyclerView.setAdapter(mAdapter);
        //to add spacing between cards
        if (getActivity() != null) {
```

```java
            setItemDecoration();
        }
    }


    private void setLayoutManager() {
        if (isGrid) {
            layoutManager = new GridLayoutManager(getActivity(), 2);
        } else {
            layoutManager = new GridLayoutManager(getActivity(), 1);
        }
        recyclerView.setLayoutManager(layoutManager);
    }

    private void setItemDecoration() {
        if (isGrid) {
            int spacingInPixels = getActivity().getResources().getDimensionPixelSize(R.dimen.spacing_card_album_grid);
            itemDecoration = new SpacesItemDecoration(spacingInPixels);
        } else {
            itemDecoration = new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST);
        }
        recyclerView.addItemDecoration(itemDecoration);
    }

    private void updateLayoutManager(int column) {
        recyclerView.removeItemDecoration(itemDecoration);
        recyclerView.setAdapter(new PlaylistAdapter(getActivity(), PlaylistLoader.getPlaylists(getActivity(), showAuto)));
        layoutManager.setSpanCount(column);
        layoutManager.requestLayout();
        setItemDecoration();
    }


    public class SpacesItemDecoration extends RecyclerView.ItemDecoration {
        private int space;

        public SpacesItemDecoration(int space) {
            this.space = space;
        }

        @Override
        public void getItemOffsets(Rect outRect, View view,
                                   RecyclerView parent, RecyclerView.State state) {


            outRect.left = space;
            outRect.top = space;
            outRect.right = space;
            outRect.bottom = space;

        }
    }


    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        if (PreferenceManager.getDefaultSharedPreferences(getActivity()).getBoolean("dark_theme", false)) {
            ATE.apply(this, "dark_theme");
        } else {
            ATE.apply(this, "light_theme");
        }
    }

    @Override
    public void onActivityCreated(final Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.menu_playlist, menu);

    }

    @Override
    public void onPrepareOptionsMenu(Menu menu) {
        super.onPrepareOptionsMenu(menu);
        if (showAuto) {
            menu.findItem(R.id.action_view_auto_playlists).setTitle("Hide auto playlists");
        } else menu.findItem(R.id.action_view_auto_playlists).setTitle("Show auto playlists");
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_new_playlist:
                CreatePlaylistDialog.newInstance().show(getChildFragmentManager(), "CREATE_PLAYLIST");
                return true;
            case R.id.menu_show_as_list:
                mPreferences.setPlaylistView(Constants.PLAYLIST_VIEW_LIST);
                isGrid = false;
                isDefault = false;
                initRecyclerView();
                updateLayoutManager(1);
                return true;
            case R.id.menu_show_as_grid:
                mPreferences.setPlaylistView(Constants.PLAYLIST_VIEW_GRID);
                isGrid = true;
                isDefault = false;
                initRecyclerView();
                updateLayoutManager(2);
                return true;
            case R.id.menu_show_as_default:
                mPreferences.setPlaylistView(Constants.PLAYLIST_VIEW_DEFAULT);
                isDefault = true;
                initPager();
                return true;
            case R.id.action_view_auto_playlists:
                if (showAuto) {
                    showAuto = false;
                    mPreferences.setToggleShowAutoPlaylist(false);
                } else {
                    showAuto = true;
                    mPreferences.setToggleShowAutoPlaylist(true);
                }
                reloadPlaylists();
                getActivity().invalidateOptionsMenu();
                break;

        }
        return super.onOptionsItemSelected(item);
    }

    public void updatePlaylists(final long id) {
        playlists = PlaylistLoader.getPlaylists(getActivity(), showAuto);
        playlistcount = playlists.size();

        if (isDefault) {
            adapter.notifyDataSetChanged();
            if (id != -1) {
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        for (int i = 0; i < playlists.size(); i++) {
                            long playlistid = playlists.get(i).id;
                            if (playlistid == id) {
                                pager.setCurrentItem(i);
                                break;
                            }
                        }
                    }
                }, 200);
            }

        } else {
            mAdapter.updateDataSet(playlists);
        }
    }

    public void reloadPlaylists() {
        playlists = PlaylistLoader.getPlaylists(getActivity(), showAuto);
        playlistcount = playlists.size();

        if (isDefault) {
            initPager();
        } else {
            initRecyclerView();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == Constants.ACTION_DELETE_PLAYLIST) {
            if (resultCode == Activity.RESULT_OK) {
                reloadPlaylists();
            }

        }
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.os.AsyncTask;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.afollestad.appthemeengine.ATE;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.activities.BaseActivity;
import com.naman14.timber.adapters.PlayingQueueAdapter;
import com.naman14.timber.dataloaders.QueueLoader;
import com.naman14.timber.listeners.MusicStateListener;
import com.naman14.timber.models.Song;
import com.naman14.timber.widgets.BaseRecyclerView;
import com.naman14.timber.widgets.DragSortRecycler;

public class QueueFragment extends Fragment implements MusicStateListener {

    private PlayingQueueAdapter mAdapter;
    private BaseRecyclerView recyclerView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_queue, container, false);

        Toolbar toolbar = rootView.findViewById(R.id.toolbar);
        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);

        final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setHomeAsUpIndicator(R.drawable.ic_menu);
        ab.setDisplayHomeAsUpEnabled(true);
        ab.setTitle(R.string.playing_queue);

        recyclerView = rootView.findViewById(R.id.recyclerview);
        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
        recyclerView.setItemAnimator(null);
        recyclerView.setEmptyView(getActivity(), rootView.findViewById(R.id.list_empty), "No songs in queue");

        new loadQueueSongs().execute("");
        ((BaseActivity) getActivity()).setMusicStateListenerListener(this);

        return rootView;
    }
    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        if (PreferenceManager.getDefaultSharedPreferences(getActivity()).getBoolean("dark_theme", false)) {
            ATE.apply(this, "dark_theme");
        } else {
            ATE.apply(this, "light_theme");
        }
    }

    public void restartLoader() {

    }

    public void onPlaylistChanged() {

    }

    public void onMetaChanged() {
        if (mAdapter != null)
            mAdapter.notifyDataSetChanged();
    }

    private class loadQueueSongs extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            mAdapter = new PlayingQueueAdapter(getActivity(), QueueLoader.getQueueSongs(getActivity()));
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            recyclerView.setAdapter(mAdapter);
            DragSortRecycler dragSortRecycler = new DragSortRecycler();
            dragSortRecycler.setViewHandleId(R.id.reorder);

            dragSortRecycler.setOnItemMovedListener(new DragSortRecycler.OnItemMovedListener() {
                @Override
                public void onItemMoved(int from, int to) {
                    Log.d("queue", "onItemMoved " + from + " to " + to);
                    Song song = mAdapter.getSongAt(from);
                    mAdapter.removeSongAt(from);
                    mAdapter.addSongTo(to, song);
                    mAdapter.notifyDataSetChanged();
                    MusicPlayer.moveQueueItem(from, to);
                }
            });

            recyclerView.addItemDecoration(dragSortRecycler);
            recyclerView.addOnItemTouchListener(dragSortRecycler);
            recyclerView.addOnScrollListener(dragSortRecycler.getScrollListener());

            recyclerView.getLayoutManager().scrollToPosition(mAdapter.currentlyPlayingPosition);

        }

        @Override
        protected void onPreExecute() {
        }
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\SettingsFragment.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.os.Bundle;
import android.preference.ListPreference;
import android.preference.Preference;
import android.preference.PreferenceFragment;
import android.preference.SwitchPreference;
import android.view.View;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.Config;
import com.afollestad.appthemeengine.prefs.ATECheckBoxPreference;
import com.afollestad.appthemeengine.prefs.ATEColorPreference;
import com.afollestad.materialdialogs.color.ColorChooserDialog;
import com.naman14.timber.R;
import com.naman14.timber.activities.DonateActivity;
import com.naman14.timber.activities.SettingsActivity;
import com.naman14.timber.dialogs.LastFmLoginDialog;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;

public class SettingsFragment extends PreferenceFragment implements SharedPreferences.OnSharedPreferenceChangeListener {

    private static final String NOW_PLAYING_SELECTOR = "now_playing_selector";
    private static final String LASTFM_LOGIN = "lastfm_login";

    private static final String LOCKSCREEN = "show_albumart_lockscreen";
    private static final String XPOSED = "toggle_xposed_trackselector";

    private static final String KEY_ABOUT = "preference_about";
    private static final String KEY_SOURCE = "preference_source";
    private static final String KEY_THEME = "theme_preference";
    private static final String TOGGLE_ANIMATIONS = "toggle_animations";
    private static final String TOGGLE_SYSTEM_ANIMATIONS = "toggle_system_animations";
    private static final String KEY_START_PAGE = "start_page_preference";
    private boolean lastFMlogedin;

    private Preference nowPlayingSelector,  lastFMlogin, lockscreen, xposed;

    private SwitchPreference toggleAnimations;
    private ListPreference themePreference, startPagePreference;
    private PreferencesUtility mPreferences;
    private String mAteKey;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.preferences);

        mPreferences = PreferencesUtility.getInstance(getActivity());
        lockscreen = findPreference(LOCKSCREEN);
        nowPlayingSelector = findPreference(NOW_PLAYING_SELECTOR);

        xposed = findPreference(XPOSED);

        lastFMlogin = findPreference(LASTFM_LOGIN);
        updateLastFM();
//        themePreference = (ListPreference) findPreference(KEY_THEME);
        startPagePreference = (ListPreference) findPreference(KEY_START_PAGE);

        nowPlayingSelector.setIntent(NavigationUtils.getNavigateToStyleSelectorIntent(getActivity(), Constants.SETTINGS_STYL

        setPreferenceClickListeners();

    }

    @Override
    public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
                                          String key) {
    }

    private void setPreferenceClickListeners() {

//        themePreference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
//            @Override
//            public boolean onPreferenceChange(Preference preference, Object newValue) {
//                Intent i = getActivity().getBaseContext().getPackageManager().getLaunchIntentForPackage(getActivity().getB
//                i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
//                startActivity(i);
//                return true;
//            }
//        });

        startPagePreference.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                switch ((String) newValue) {
                    case "last_opened":
                        mPreferences.setLastOpenedAsStartPagePreference(true);
                        break;
                    case "songs":
                        mPreferences.setLastOpenedAsStartPagePreference(false);
                        mPreferences.setStartPageIndex(0);
                        break;
                    case "albums":
                        mPreferences.setLastOpenedAsStartPagePreference(false);
                        mPreferences.setStartPageIndex(1);
                        break;
                    case "artists":
                        mPreferences.setLastOpenedAsStartPagePreference(false);
                        mPreferences.setStartPageIndex(2);
                        break;
                }
                return true;
            }
        });

        Intent restoreIntent = new Intent(getActivity(), DonateActivity.class);
        restoreIntent.putExtra("title", "Restoring purchases..");
        restoreIntent.setAction("restore");

        findPreference("support_development").setIntent(new Intent(getActivity(), DonateActivity.class));
        findPreference("restore_purchases").setIntent(restoreIntent);

        lockscreen.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                Bundle extras = new Bundle();
                extras.putBoolean("lockscreen",(boolean)newValue);
                mPreferences.updateService(extras);
                return true;
            }
        });

        xposed.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                Bundle extras = new Bundle();
                extras.putBoolean("xtrack",(boolean)newValue);
                mPreferences.updateService(extras);
                return true;
            }
        });

        lastFMlogin.setOnPreferenceClickListener(new Preference.OnPreferenceClickListener() {
            @Override
            public boolean onPreferenceClick(Preference preference) {
                if (lastFMlogedin) {
                    LastFmClient.getInstance(getActivity()).logout();
                    Bundle extras = new Bundle();
                    extras.putString("lf_token","logout");
                    extras.putString("lf_user",null);
                    mPreferences.updateService(extras);
                    updateLastFM();
                } else {
                    LastFmLoginDialog lastFmLoginDialog = new LastFmLoginDialog();
                    lastFmLoginDialog.setTargetFragment(SettingsFragment.this, 0);
                    lastFmLoginDialog.show(getFragmentManager(), LastFmLoginDialog.FRAGMENT_NAME);

                }
                return true;
            }
        });

    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        invalidateSettings();
        ATE.apply(view, mAteKey);
    }

    public void invalidateSettings() {
        mAteKey = ((SettingsActivity) getActivity()).getATEKey();

        ATEColorPreference primaryColorPref = (ATEColorPreference) findPreference("primary_color");
        primaryColorPref.setColor(Config.primaryColor(getActivity(), mAteKey), Color.BLACK);
        primaryColorPref.setOnPreferenceClickListener(new Preference.OnPreferenceClickListener() {
            @Override
            public boolean onPreferenceClick(Preference preference) {
                new ColorChooserDialog.Builder((SettingsActivity) getActivity(), R.string.primary_color)
                        .preselect(Config.primaryColor(getActivity(), mAteKey))
                        .show();
                return true;
            }
        });

        ATEColorPreference accentColorPref = (ATEColorPreference) findPreference("accent_color");
        accentColorPref.setColor(Config.accentColor(getActivity(), mAteKey), Color.BLACK);
        accentColorPref.setOnPreferenceClickListener(new Preference.OnPreferenceClickListener() {
            @Override
            public boolean onPreferenceClick(Preference preference) {
                new ColorChooserDialog.Builder((SettingsActivity) getActivity(), R.string.accent_color)
                        .preselect(Config.accentColor(getActivity(), mAteKey))
                        .show();
                return true;
            }
        });
        findPreference("dark_theme").setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                // Marks both theme configs as changed so MainActivity restarts itself on return
                Config.markChanged(getActivity(), "light_theme");
                Config.markChanged(getActivity(), "dark_theme");
                // The dark_theme preference value gets saved by Android in the default PreferenceManager.
                // It's used in getATEKey() of both the Activities.
                getActivity().recreate();
                return true;
            }
        });

        final ATECheckBoxPreference statusBarPref = (ATECheckBoxPreference) findPreference("colored_status_bar");
        final ATECheckBoxPreference navBarPref = (ATECheckBoxPreference) findPreference("colored_nav_bar");

        statusBarPref.setChecked(Config.coloredStatusBar(getActivity(), mAteKey));
        statusBarPref.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                ATE.config(getActivity(), mAteKey)
                        .coloredStatusBar((Boolean) newValue)
                        .apply(getActivity());
                return true;
            }
        });


        navBarPref.setChecked(Config.coloredNavigationBar(getActivity(), mAteKey));
        navBarPref.setOnPreferenceChangeListener(new Preference.OnPreferenceChangeListener() {
            @Override
            public boolean onPreferenceChange(Preference preference, Object newValue) {
                ATE.config(getActivity(), mAteKey)
                        .coloredNavigationBar((Boolean) newValue)
                        .apply(getActivity());
                return true;
            }
        });

    }


    public void updateLastFM() {
        String username = LastFmClient.getInstance(getActivity()).getUsername();
        if (username != null) {
            lastFMlogedin = true;
            lastFMlogin.setTitle("Logout");
            lastFMlogin.setSummary(String.format(getString(R.string.lastfm_loged_in),username));
        } else {
            lastFMlogedin = false;
            lastFMlogin.setTitle("Login");
            lastFMlogin.setSummary(getString(R.string.lastfm_pref));
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\fragments\SimilarArtistFragment

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.dataloaders.ArtistLoader;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.ArtistInfoListener;
import com.naman14.timber.lastfmapi.models.ArtistQuery;
import com.naman14.timber.lastfmapi.models.LastfmArtist;
import com.naman14.timber.models.Artist;
import com.naman14.timber.utils.Constants;

public class SimilarArtistFragment extends Fragment {

    private long artistID = -1;

    public static SimilarArtistFragment newInstance(long id) {
        SimilarArtistFragment fragment = new SimilarArtistFragment();
        Bundle args = new Bundle();
        args.putLong(Constants.ARTIST_ID, id);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            artistID = getArguments().getLong(Constants.ARTIST_ID);
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_similar_artists, container, false);

        Artist artist = ArtistLoader.getArtist(getActivity(), artistID);

        LastFmClient.getInstance(getActivity()).getArtistInfo(new ArtistQuery(artist.name), new ArtistInfoListener() {
            @Override
            public void artistInfoSucess(LastfmArtist artist) {

            }

            @Override
            public void artistInfoFailed() {
            }
        });

        return rootView;
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.fragments;

import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.activities.BaseActivity;
import com.naman14.timber.adapters.SongsListAdapter;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.listeners.MusicStateListener;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.SortOrder;
import com.naman14.timber.widgets.BaseRecyclerView;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.FastScroller;

import java.util.List;

public class SongsFragment extends Fragment implements MusicStateListener {

    private SongsListAdapter mAdapter;
    private BaseRecyclerView recyclerView;
    private PreferencesUtility mPreferences;

    @Override
    public void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = PreferencesUtility.getInstance(getActivity());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_recyclerview, container, false);

        recyclerView = rootView.findViewById(R.id.recyclerview);
        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
        recyclerView.setEmptyView(getActivity(), rootView.findViewById(R.id.list_empty), "No media found");
        FastScroller fastScroller =  rootView.findViewById(R.id.fastscroller);
        fastScroller.setRecyclerView(recyclerView);

        new loadSongs().execute("");
        ((BaseActivity) getActivity()).setMusicStateListenerListener(this);

        return rootView;
    }

    public void restartLoader() {

    }

    public void onPlaylistChanged() {

    }

    public void onMetaChanged() {
        if (mAdapter != null)
            mAdapter.notifyDataSetChanged();
    }

    private void reloadAdapter() {
        new AsyncTask<Void, Void, Void>() {
            @Override
            protected Void doInBackground(final Void... unused) {
                List<Song> songList = SongLoader.getAllSongs(getActivity());
                mAdapter.updateDataSet(songList);
                return null;
            }

            @Override
            protected void onPostExecute(Void aVoid) {
                mAdapter.notifyDataSetChanged();
            }
        }.execute();
    }

    @Override
    public void onActivityCreated(final Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.song_sort_by, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_sort_by_az:
                mPreferences.setSongSortOrder(SortOrder.SongSortOrder.SONG_A_Z);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_za:
                mPreferences.setSongSortOrder(SortOrder.SongSortOrder.SONG_Z_A);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_artist:
                mPreferences.setSongSortOrder(SortOrder.SongSortOrder.SONG_ARTIST);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_album:
                mPreferences.setSongSortOrder(SortOrder.SongSortOrder.SONG_ALBUM);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_year:
                mPreferences.setSongSortOrder(SortOrder.SongSortOrder.SONG_YEAR);
                reloadAdapter();
                return true;
            case R.id.menu_sort_by_duration:
                mPreferences.setSongSortOrder(SortOrder.SongSortOrder.SONG_DURATION);
                reloadAdapter();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    private class loadSongs extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            if (getActivity() != null)
                mAdapter = new SongsListAdapter((AppCompatActivity) getActivity(), SongLoader.getAllSongs(getActivity()), fa
            return "Executed";
        }

        @Override
        protected void onPostExecute(String result) {
            recyclerView.setAdapter(mAdapter);
            if (getActivity() != null)
                recyclerView.addItemDecoration(new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_LIST)

        }

        @Override
        protected void onPreExecute() {
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\helpers\MediaButtonIntentReceiv

```java
/*
 * Copyright (C) 2007 The Android Open Source Project Licensed under the Apache
 * License, Version 2.0 (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber.helpers;

import android.content.Context;
import android.content.Intent;
import android.media.AudioManager;
import android.os.Handler;
import android.os.Message;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.support.v4.content.WakefulBroadcastReceiver;
import android.util.Log;
import android.view.KeyEvent;

import com.naman14.timber.MusicService;
import com.naman14.timber.activities.MainActivity;
import com.naman14.timber.utils.PreferencesUtility;

/**
 * Used to control headset playback.
 * Single press: pause/resume
 * Double press: next track
 * Triple press: previous track
 * Long press: voice search
 */
public class MediaButtonIntentReceiver extends WakefulBroadcastReceiver {
    private static final boolean DEBUG = false;
    private static final String TAG = "ButtonIntentReceiver";

    private static final int MSG_LONGPRESS_TIMEOUT = 1;
    private static final int MSG_HEADSET_DOUBLE_CLICK_TIMEOUT = 2;

    private static final int LONG_PRESS_DELAY = 1000;
    private static final int DOUBLE_CLICK = 800;

    private static WakeLock mWakeLock = null;
    private static int mClickCounter = 0;
    private static long mLastClickTime = 0;
    private static boolean mDown = false;
    private static boolean mLaunched = false;

    private static Handler mHandler = new Handler() {

        /**
         * {@inheritDoc}
         */
        @Override
        public void handleMessage(final Message msg) {
            switch (msg.what) {
                case MSG_LONGPRESS_TIMEOUT:
                    if (DEBUG) Log.v(TAG, "Handling longpress timeout, launched " + mLaunched);
                    if (!mLaunched) {
                        final Context context = (Context) msg.obj;
                        final Intent i = new Intent();
                        i.setClass(context, MainActivity.class);
                        i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        context.startActivity(i);
                        mLaunched = true;
                    }
                    break;
                case MSG_HEADSET_DOUBLE_CLICK_TIMEOUT:
                    final int clickCount = msg.arg1;
                    final String command;

                    if (DEBUG) Log.v(TAG, "Handling headset click, count = " + clickCount);
                    switch (clickCount) {
                        case 1:
                            command = MusicService.CMDTOGGLEPAUSE;
                            break;
                        case 2:
                            command = MusicService.CMDNEXT;
                            break;
                        case 3:
                            command = MusicService.CMDPREVIOUS;
                            break;
                        default:
                            command = null;
                            break;
                    }

                    if (command != null) {
                        final Context context = (Context) msg.obj;
                        startService(context, command);
                    }
                    break;
            }
            releaseWakeLockIfHandlerIdle();
        }
    };

    private static void startService(Context context, String command) {
        final Intent i = new Intent(context, MusicService.class);
        i.setAction(MusicService.SERVICECMD);
        i.putExtra(MusicService.CMDNAME, command);
        i.putExtra(MusicService.FROM_MEDIA_BUTTON, true);
        startWakefulService(context, i);
    }

    private static void acquireWakeLockAndSendMessage(Context context, Message msg, long delay) {
        if (mWakeLock == null) {
            Context appContext = context.getApplicationContext();
            PowerManager pm = (PowerManager) appContext.getSystemService(Context.POWER_SERVICE);
            mWakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "Timber headset button");
            mWakeLock.setReferenceCounted(false);
        }
        if (DEBUG) Log.v(TAG, "Acquiring wake lock and sending " + msg.what);
        // Make sure we don't indefinitely hold the wake lock under any circumstances
        mWakeLock.acquire(10000);

        mHandler.sendMessageDelayed(msg, delay);
    }

    private static void releaseWakeLockIfHandlerIdle() {
        if (mHandler.hasMessages(MSG_LONGPRESS_TIMEOUT)
                || mHandler.hasMessages(MSG_HEADSET_DOUBLE_CLICK_TIMEOUT)) {
            if (DEBUG) Log.v(TAG, "Handler still has messages pending, not releasing wake lock");
            return;
        }

        if (mWakeLock != null) {
            if (DEBUG) Log.v(TAG, "Releasing wake lock");
            mWakeLock.release();
            mWakeLock = null;
        }
    }

    @Override
    public void onReceive(final Context context, final Intent intent) {
        final String intentAction = intent.getAction();
```

```java
        if (AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(intentAction)) {
            if (PreferencesUtility.getInstance(context).pauseEnabledOnDetach())
                startService(context, MusicService.CMDPAUSE);
        } else if (Intent.ACTION_MEDIA_BUTTON.equals(intentAction)) {
            final KeyEvent event = intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
            if (event == null) {
                return;
            }

            final int keycode = event.getKeyCode();
            final int action = event.getAction();
            final long eventtime = event.getEventTime();

            String command = null;
            switch (keycode) {
                case KeyEvent.KEYCODE_MEDIA_STOP:
                    command = MusicService.CMDSTOP;
                    break;
                case KeyEvent.KEYCODE_HEADSETHOOK:
                case KeyEvent.KEYCODE_MEDIA_PLAY_PAUSE:
                    command = MusicService.CMDTOGGLEPAUSE;
                    break;
                case KeyEvent.KEYCODE_MEDIA_NEXT:
                    command = MusicService.CMDNEXT;
                    break;
                case KeyEvent.KEYCODE_MEDIA_PREVIOUS:
                    command = MusicService.CMDPREVIOUS;
                    break;
                case KeyEvent.KEYCODE_MEDIA_PAUSE:
                    command = MusicService.CMDPAUSE;
                    break;
                case KeyEvent.KEYCODE_MEDIA_PLAY:
                    command = MusicService.CMDPLAY;
                    break;
            }
            if (command != null) {
                if (action == KeyEvent.ACTION_DOWN) {
                    if (mDown) {
                        if (MusicService.CMDTOGGLEPAUSE.equals(command)
                                || MusicService.CMDPLAY.equals(command)) {
                            if (mLastClickTime != 0
                                    && eventtime - mLastClickTime > LONG_PRESS_DELAY) {
                                acquireWakeLockAndSendMessage(context,
                                        mHandler.obtainMessage(MSG_LONGPRESS_TIMEOUT, context), 0);
                            }
                        }
                    } else if (event.getRepeatCount() == 0) {

                        if (keycode == KeyEvent.KEYCODE_HEADSETHOOK) {
                            if (eventtime - mLastClickTime >= DOUBLE_CLICK) {
                                mClickCounter = 0;
                            }

                            mClickCounter++;
                            if (DEBUG) Log.v(TAG, "Got headset click, count = " + mClickCounter);
                            mHandler.removeMessages(MSG_HEADSET_DOUBLE_CLICK_TIMEOUT);

                            Message msg = mHandler.obtainMessage(
                                    MSG_HEADSET_DOUBLE_CLICK_TIMEOUT, mClickCounter, 0, context);

                            long delay = mClickCounter < 3 ? DOUBLE_CLICK : 0;
                            if (mClickCounter >= 3) {
                                mClickCounter = 0;
                            }
                            mLastClickTime = eventtime;
                            acquireWakeLockAndSendMessage(context, msg, delay);
                        } else {
                            startService(context, command);
                        }
                        mLaunched = false;
                        mDown = true;
                    }
                } else {
                    mHandler.removeMessages(MSG_LONGPRESS_TIMEOUT);
                    mDown = false;
                }
                if (isOrderedBroadcast()) {
                    abortBroadcast();
                }
                releaseWakeLockIfHandlerIdle();
            }
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\helpers\MusicPlaybackTrack.java

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.helpers;

import android.os.Parcel;
import android.os.Parcelable;

import com.naman14.timber.utils.TimberUtils;

/**
 * This is used by the music playback service to track the music tracks it is playing
 * It has extra meta data to determine where the track came from so that we can show the appropriate
 * song playing indicator
 */
public class MusicPlaybackTrack implements Parcelable {

    public static final Creator<MusicPlaybackTrack> CREATOR = new Creator<MusicPlaybackTrack>() {
        @Override
        public MusicPlaybackTrack createFromParcel(Parcel source) {
            return new MusicPlaybackTrack(source);
        }

        @Override
        public MusicPlaybackTrack[] newArray(int size) {
            return new MusicPlaybackTrack[size];
        }
    };
    public long mId;
    public long mSourceId;
    public TimberUtils.IdType mSourceType;
    public int mSourcePosition;

    public MusicPlaybackTrack(long id, long sourceId, TimberUtils.IdType type, int sourcePosition) {
        mId = id;
        mSourceId = sourceId;
        mSourceType = type;
        mSourcePosition = sourcePosition;
    }

    public MusicPlaybackTrack(Parcel in) {
        mId = in.readLong();
        mSourceId = in.readLong();
        mSourceType = TimberUtils.IdType.getTypeById(in.readInt());
        mSourcePosition = in.readInt();
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeLong(mId);
        dest.writeLong(mSourceId);
        dest.writeInt(mSourceType.mId);
        dest.writeInt(mSourcePosition);
    }

    @Override
    public boolean equals(Object o) {
        if (o instanceof MusicPlaybackTrack) {
            MusicPlaybackTrack other = (MusicPlaybackTrack) o;
            if (other != null) {
                return mId == other.mId
                        && mSourceId == other.mSourceId
                        && mSourceType == other.mSourceType
                        && mSourcePosition == other.mSourcePosition;

            }
        }

        return super.equals(o);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\LastFmClient.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.util.Log;

import com.naman14.timber.lastfmapi.callbacks.AlbumInfoListener;
import com.naman14.timber.lastfmapi.callbacks.ArtistInfoListener;
import com.naman14.timber.lastfmapi.callbacks.UserListener;
import com.naman14.timber.lastfmapi.models.AlbumInfo;
import com.naman14.timber.lastfmapi.models.AlbumQuery;
import com.naman14.timber.lastfmapi.models.ArtistInfo;
import com.naman14.timber.lastfmapi.models.ArtistQuery;
import com.naman14.timber.lastfmapi.models.LastfmUserSession;
import com.naman14.timber.lastfmapi.models.ScrobbleInfo;
import com.naman14.timber.lastfmapi.models.ScrobbleQuery;
import com.naman14.timber.lastfmapi.models.UserLoginInfo;
import com.naman14.timber.lastfmapi.models.UserLoginQuery;
import com.naman14.timber.utils.PreferencesUtility;

import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.HashSet;
import java.util.Map;
import java.util.TreeMap;

import retrofit.Callback;
import retrofit.RetrofitError;
import retrofit.client.Response;

public class LastFmClient {

    //TODO update the api keys
    public static final String API_KEY = "62ac1851456e4558bef1c41747b1aec2";
    public static final String API_SECRET = "b4ae8965723d67fb18e35d207014d6f3";

    public static final String JSON = "json";

    public static final String BASE_API_URL = "http://ws.audioscrobbler.com/2.0";
    public static final String BASE_SECURE_API_URL = "https://ws.audioscrobbler.com/2.0";

    public static final String PREFERENCES_NAME = "Lastfm";
    static final String PREFERENCE_CACHE_NAME = "Cache";

    private static LastFmClient sInstance;
    private LastFmRestService mRestService;
    private LastFmUserRestService mUserRestService;

    private HashSet<String> queries;
    private boolean isUploading = false;

    private Context context;

    private LastfmUserSession mUserSession;
    private static final Object sLock = new Object();

    public static LastFmClient getInstance(Context context) {
        synchronized (sLock) {
            if (sInstance == null) {
                sInstance = new LastFmClient();
                sInstance.context = context;
                sInstance.mRestService = RestServiceFactory.createStatic(context, BASE_API_URL, LastFmRestService.class);
                sInstance.mUserRestService = RestServiceFactory.create(context, BASE_SECURE_API_URL, LastFmUserRestService.c
                sInstance.mUserSession = LastfmUserSession.getSession(context);

            }
            return sInstance;
        }
    }

    private static String generateMD5(String in) {
        try {
            byte[] bytesOfMessage = in.getBytes("UTF-8");
            MessageDigest md = MessageDigest.getInstance("MD5");
            byte[] digest = md.digest(bytesOfMessage);
            String out = "";
            for (byte symbol : digest) {
                out += String.format("%02X", symbol);
            }
            return out;
        } catch (UnsupportedEncodingException | NoSuchAlgorithmException ignored) {
            return null;
        }


    }

    public void getAlbumInfo(AlbumQuery albumQuery, final AlbumInfoListener listener) {
        mRestService.getAlbumInfo(albumQuery.mArtist, albumQuery.mALbum, new Callback<AlbumInfo>() {
            @Override
            public void success(AlbumInfo albumInfo, Response response) {
                listener.albumInfoSuccess(albumInfo.mAlbum);
            }

            @Override
            public void failure(RetrofitError error) {
                listener.albumInfoFailed();
                error.printStackTrace();
            }
        });
    }

    public void getArtistInfo(ArtistQuery artistQuery, final ArtistInfoListener listener) {
        mRestService.getArtistInfo(artistQuery.mArtist, new Callback<ArtistInfo>() {
            @Override
            public void success(ArtistInfo artistInfo, Response response) {
                listener.artistInfoSucess(artistInfo.mArtist);
            }

            @Override
            public void failure(RetrofitError error) {
                listener.artistInfoFailed();
                error.printStackTrace();
            }
        });
    }

    public void getUserLoginInfo(UserLoginQuery userLoginQuery, final UserListener listener) {
        mUserRestService.getUserLoginInfo(UserLoginQuery.Method, JSON, API_KEY, generateMD5(userLoginQuery.getSignature()),
            @Override
            public void success(UserLoginInfo userLoginInfo, Response response) {
                Log.d("Logedin", userLoginInfo.mSession.mToken + " " + userLoginInfo.mSession.mUsername);
                Bundle extras = new Bundle();
                extras.putString("lf_token",userLoginInfo.mSession.mToken);
                extras.putString("lf_user",userLoginInfo.mSession.mUsername);
                PreferencesUtility.getInstance(context).updateService(extras);
                mUserSession = userLoginInfo.mSession;
                mUserSession.update(context);
                listener.userSuccess();
            }

            @Override
            public void failure(RetrofitError error) {
                listener.userInfoFailed();
            }
        });
    }

    public void Scrobble(final ScrobbleQuery scrobbleQuery) {
        if (mUserSession.isLogedin())
            new ScrobbleUploader(scrobbleQuery);
    }

    private class ScrobbleUploader {
        boolean cachedirty = false;
        ScrobbleQuery newquery;
        SharedPreferences preferences = context.getSharedPreferences(PREFERENCES_NAME, Context.MODE_PRIVATE);

        ScrobbleUploader(ScrobbleQuery query) {
            if (queries == null) {
                queries = new HashSet<>();
                queries.addAll(preferences.getStringSet(PREFERENCE_CACHE_NAME, new HashSet<String>()));
            }
            if (query != null) {
                synchronized (sLock) {
                    if (isUploading) {
                        cachedirty = true;
                        queries.add(query.toString());
                        save();
                        return;
                    }
                }
                newquery = query;
            }
            upload();
        }

        void upload() {
            synchronized (sLock) {
                isUploading = true;
            }
            int size = queries.size();
            if (size == 0 && newquery == null) return;
            //Max 50 Scrobbles per Request (restriction by LastFM)
            if (size > 50) size = 50;
            if (newquery != null && size > 49) size = 49;
            final String currentqueries[] = new String[size];
            int n = 0;
            for (String t : queries) {
                currentqueries[n++] = t;
                if (n >= size) break;
            }

            TreeMap<String, String> fields = new TreeMap<>();
            fields.put("method", ScrobbleQuery.Method);
            fields.put("api_key", API_KEY);
            fields.put("sk", mUserSession.mToken);

            int i = 0;
            for (String squery : currentqueries) {
                ScrobbleQuery query = new ScrobbleQuery(squery);
                fields.put("artist[" + i + ']', query.mArtist);
                fields.put("track[" + i + ']', query.mTrack);
                fields.put("timestamp[" + i + ']', Long.toString(query.mTimestamp));
                i++;
            }
            if (newquery != null) {
                fields.put("artist[" + i + ']', newquery.mArtist);
                fields.put("track[" + i + ']', newquery.mTrack);
                fields.put("timestamp[" + i + ']', Long.toString(newquery.mTimestamp));
            }
            String sig = "";
            for (Map.Entry<String, String> ent : fields.entrySet()) {
                sig += ent.getKey() + ent.getValue();
            }
            sig += API_SECRET;
            mUserRestService.getScrobbleInfo(generateMD5(sig), JSON, fields, new Callback<ScrobbleInfo>() {
                @Override
                public void success(ScrobbleInfo scrobbleInfo, Response response) {
                    synchronized (sLock) {
                        isUploading = false;
                        cachedirty = true;
                        if (newquery != null) newquery = null;

                        for (String squery : currentqueries) {
                            queries.remove(squery);
                        }
                        if (queries.size() > 0)
                            upload();
                        else
                            save();

                    }
                }

                @Override
                public void failure(RetrofitError error) {
                    synchronized (sLock) {
                        isUploading = false;
                        //Max 500 scrobbles in Cache
                        if (newquery != null && queries.size() <= 500)
                            queries.add(newquery.toString());

                        if (cachedirty)
                            save();
                    }
                }
            });


        }

        void save() {
            if (!cachedirty) return;
            SharedPreferences.Editor editor = preferences.edit();
            editor.putStringSet(PREFERENCE_CACHE_NAME, queries);
            editor.apply();
        }

    }

    public void logout() {
        this.mUserSession.mToken = null;
        this.mUserSession.mUsername = null;
        SharedPreferences preferences = context.getSharedPreferences(PREFERENCES_NAME, Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        editor.clear();
        editor.apply();
    }

    public String getUsername() {
        if (mUserSession != null) return mUserSession.mUsername;
        return null;
    }
```

```
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi;

import com.naman14.timber.lastfmapi.models.AlbumInfo;
import com.naman14.timber.lastfmapi.models.ArtistInfo;

import retrofit.Callback;
import retrofit.http.GET;
import retrofit.http.Headers;
import retrofit.http.Query;

public interface LastFmRestService {

    String BASE_PARAMETERS_ALBUM = "/?method=album.getinfo&api_key=fdb3a51437d4281d4d64964d333531d4&format=json";
    String BASE_PARAMETERS_ARTIST = "/?method=artist.getinfo&api_key=fdb3a51437d4281d4d64964d333531d4&format=json";

    @Headers("Cache-Control: public")
    @GET(BASE_PARAMETERS_ALBUM)
    void getAlbumInfo(@Query("artist") String artist, @Query("album") String album, Callback<AlbumInfo> callback);

    @Headers("Cache-Control: public")
    @GET(BASE_PARAMETERS_ARTIST)
    void getArtistInfo(@Query("artist") String artist, Callback<ArtistInfo> callback);

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\LastFmUserRestService

```java
package com.naman14.timber.lastfmapi;

import com.naman14.timber.lastfmapi.models.ScrobbleInfo;
import com.naman14.timber.lastfmapi.models.UserLoginInfo;

import java.util.Map;

import retrofit.Callback;
import retrofit.http.Field;
import retrofit.http.FieldMap;
import retrofit.http.FormUrlEncoded;
import retrofit.http.POST;

/**
 * Created by christoph on 17.07.16.
 */
public interface LastFmUserRestService {

    String BASE = "/";

    @POST(BASE)
    @FormUrlEncoded
    void getUserLoginInfo(@Field("method") String method, @Field("format") String format, @Field("api_key") String apikey, @

    @POST(BASE)
    @FormUrlEncoded
    void getScrobbleInfo(@Field("api_sig") String apisig, @Field("format") String format, @FieldMap Map<String, String> fiel

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi;

import android.content.Context;

import com.naman14.timber.utils.PreferencesUtility;
import com.squareup.okhttp.Cache;
import com.squareup.okhttp.OkHttpClient;

import java.util.concurrent.TimeUnit;

import retrofit.RequestInterceptor;
import retrofit.RestAdapter;
import retrofit.client.OkClient;

public class RestServiceFactory {
    private static final String TAG_OK_HTTP = "OkHttp";
    private static final long CACHE_SIZE = 1024 * 1024;

    public static <T> T createStatic(final Context context, String baseUrl, Class<T> clazz) {
        final OkHttpClient okHttpClient = new OkHttpClient();

        okHttpClient.setCache(new Cache(context.getApplicationContext().getCacheDir(),
                CACHE_SIZE));
        okHttpClient.setConnectTimeout(40, TimeUnit.SECONDS);

        RequestInterceptor interceptor = new RequestInterceptor() {
            PreferencesUtility prefs = PreferencesUtility.getInstance(context);

            @Override
            public void intercept(RequestFacade request) {
                //7-days cache
                request.addHeader("Cache-Control",
                        String.format("max-age=%d,%smax-stale=%d",
                                Integer.valueOf(60 * 60 * 24 * 7),
                                prefs.loadArtistAndAlbumImages() ? "" : "only-if-cached,", Integer.valueOf(31536000)));
                request.addHeader("Connection", "keep-alive");
            }
        };

        RestAdapter.Builder builder = new RestAdapter.Builder()
                .setEndpoint(baseUrl)
                .setRequestInterceptor(interceptor)
                .setClient(new OkClient(okHttpClient));

        return builder
                .build()
                .create(clazz);

    }

    public static <T> T create(final Context context, String baseUrl, Class<T> clazz) {

        RestAdapter.Builder builder = new RestAdapter.Builder()
                .setEndpoint(baseUrl);

        return builder
```

```
                .build()
                .create(clazz);

    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\callbacks\AlbumInfoLi

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.callbacks;

import com.naman14.timber.lastfmapi.models.LastfmAlbum;

public interface AlbumInfoListener {

    void albumInfoSuccess(LastfmAlbum album);

    void albumInfoFailed();

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.callbacks;

import com.naman14.timber.lastfmapi.models.LastfmArtist;

public interface ArtistInfoListener {

    void artistInfoSucess(LastfmArtist artist);

    void artistInfoFailed();

}
```

```java
package com.naman14.timber.lastfmapi.callbacks;



/**
 * Created by christoph on 17.07.16.
 */
public interface UserListener {
    void userSuccess();

    void userInfoFailed();

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\AlbumBio.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

public class AlbumBio {
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\AlbumInfo.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class AlbumInfo {

    private static final String ALBUM = "album";


    @SerializedName(ALBUM)
    public LastfmAlbum mAlbum;
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\AlbumQuery.jav

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class AlbumQuery {

    private static final String ALBUM_NAME = "album";
    private static final String ARTIST_NAME = "artist";

    @SerializedName(ALBUM_NAME)
    public String mALbum;

    @SerializedName(ARTIST_NAME)
    public String mArtist;

    public AlbumQuery(String album, String artist) {
        this.mALbum = album;
        this.mArtist = artist;
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\AlbumTracks.ja

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

public class AlbumTracks {
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\ArtistBio.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class ArtistBio {

    private static final String PUBLISHED = "published";
    private static final String SUMMARY = "summary";
    private static final String CONTENT = "content";
    private static final String YEARFORMED = "yearformed";

    @SerializedName(PUBLISHED)
    public String mPublished;

    @SerializedName(SUMMARY)
    public String mSummary;

    @SerializedName(CONTENT)
    public String mContent;

    @SerializedName(YEARFORMED)
    public String mYearFormed;

}
```

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class ArtistInfo {

    private static final String ARTIST = "artist";

    @SerializedName(ARTIST)
    public LastfmArtist mArtist;

}
```

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class ArtistQuery {

    private static final String ARTIST_NAME = "artist";

    @SerializedName(ARTIST_NAME)
    public String mArtist;

    public ArtistQuery(String artist) {
        this.mArtist = artist;
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\ArtistTag.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class ArtistTag {

    private static final String NAME = "name";

    @SerializedName(NAME)
    public String mName;
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\Artwork.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

public class Artwork {

    private static final String URL = "#text";
    private static final String SIZE = "size";

    @SerializedName(URL)
    public String mUrl;

    @SerializedName(SIZE)
    public String mSize;
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\LastfmAlbum.ja

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

import java.util.List;

public class LastfmAlbum {
    private static final String IMAGE = "image";

    @SerializedName(IMAGE)
    public List<Artwork> mArtwork;

    // Only needed fields have been defined. See https://www.last.fm/api/show/album.getInfo
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\LastfmArtist.j

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;

import java.util.List;

public class LastfmArtist {

    private static final String NAME = "name";
    private static final String IMAGE = "image";
    private static final String SIMILAR = "similar";
    private static final String TAGS = "tags";
    private static final String BIO = "bio";

    @SerializedName(NAME)
    public String mName;

    @SerializedName(IMAGE)
    public List<Artwork> mArtwork;

    @SerializedName(SIMILAR)
    public SimilarArtist mSimilarArtist;

    @SerializedName(TAGS)
    public ArtistTag mArtistTags;

    @SerializedName(BIO)
    public ArtistBio mArtistBio;


    public class SimilarArtist {

        public static final String ARTIST = "artist";

        @SerializedName(ARTIST)
        public List<LastfmArtist> mSimilarArtist;
    }

    public class ArtistTag {

        public static final String TAG = "tag";

        @SerializedName(TAG)
        public List<com.naman14.timber.lastfmapi.models.ArtistTag> mTags;
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\LastfmUserSess

```
package com.naman14.timber.lastfmapi.models;

import android.content.Context;
import android.content.SharedPreferences;

import com.google.gson.annotations.SerializedName;
import com.naman14.timber.lastfmapi.LastFmClient;

/**
 * Created by christoph on 17.07.16.
 */
public class LastfmUserSession {
    private static final String USERNAME = "name";
    private static final String TOKEN = "key";
    private static LastfmUserSession session;


    public static LastfmUserSession getSession(Context context) {
        if (session != null) return session;
        SharedPreferences preferences = context.getSharedPreferences(LastFmClient.PREFERENCES_NAME, Context.MODE_PRIVATE);
        session = new LastfmUserSession();
        session.mToken = preferences.getString(TOKEN, null);
        session.mUsername = preferences.getString(USERNAME, null);
        return session;
    }

    public boolean isLogedin(){
        return session.mToken != null && session.mUsername != null;
    }

    public void update(Context context) {
        SharedPreferences preferences = context.getSharedPreferences(LastFmClient.PREFERENCES_NAME, Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        if (this.mToken == null || this.mUsername == null) {
            editor.clear();
        } else {
            editor.putString(TOKEN, this.mToken);
            editor.putString(USERNAME, this.mUsername);
        }
        editor.apply();
    }

    @SerializedName(USERNAME)
    public String mUsername;

    @SerializedName(TOKEN)
    public String mToken;
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\ScrobbleInfo.j

```java
package com.naman14.timber.lastfmapi.models;

/**
 * Created by christoph on 17.07.16.
 */
public class ScrobbleInfo {
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\ScrobbleQuery.

```java
package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;
import com.naman14.timber.lastfmapi.LastFmClient;

import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.net.URLEncoder;

/**
 * Created by christoph on 17.07.16.
 */
public class ScrobbleQuery {
    private static final String ARTIST_NAME = "artist";
    private static final String TRACK_NAME = "track";
    private static final String TIMESTAMP_NAME = "timestamp";

    @SerializedName(ARTIST_NAME)
    public String mArtist;

    @SerializedName(TRACK_NAME)
    public String mTrack;

    @SerializedName(TIMESTAMP_NAME)
    public long mTimestamp;

    public static final String Method = "track.scrobble";

    public ScrobbleQuery(String in) {
        String[] arr = in.split(",");
        try {
            this.mArtist = URLDecoder.decode(arr[0],"UTF-8");
            this.mTrack = URLDecoder.decode(arr[1],"UTF-8");
            this.mTimestamp = Long.parseLong(arr[2],16);
        } catch (UnsupportedEncodingException ignored) { }
    }

    public ScrobbleQuery(String artist, String track, long timestamp) {
        this.mArtist = artist;
        this.mTrack = track;
        this.mTimestamp = timestamp;
    }

    @Override
    public String toString(){
        try {
            return URLEncoder.encode(mArtist,"UTF-8")+','+URLEncoder.encode(mTrack,"UTF-8")+','+Long.toHexString(mTimestamp)
        } catch (UnsupportedEncodingException ignored) {
            return "";
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\UserLoginInfo.

```
package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;
import com.naman14.timber.lastfmapi.LastFmClient;

/**
 * Created by christoph on 17.07.16.
 */
public class UserLoginInfo {
    private static final String SESSION = "session";

    @SerializedName(SESSION)
    public LastfmUserSession mSession;


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\lastfmapi\models\UserLoginQuery

```java
package com.naman14.timber.lastfmapi.models;

import com.google.gson.annotations.SerializedName;
import com.naman14.timber.lastfmapi.LastFmClient;

/**
 * Created by christoph on 17.07.16.
 */
public class UserLoginQuery {
    private static final String USERNAME_NAME = "username";
    private static final String PASSWORD_NAME = "password";

    @SerializedName(USERNAME_NAME)
    public String mUsername;

    @SerializedName(PASSWORD_NAME)
    public String mPassword;

    public static final String Method = "auth.getMobileSession";

    public UserLoginQuery(String username, String password) {
        this.mUsername = username;
        this.mPassword = password;
    }

    public String getSignature() {
        return "api_key" + LastFmClient.API_KEY  + "method" + Method + "password" + mPassword + "username" + mUsername + Las
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\listeners\MusicStateListener.ja

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.listeners;

/**
 * Listens for playback changes to send the the fragments bound to this activity
 */
public interface MusicStateListener {

    /**
     * Called when {@link com.naman14.timber.MusicService#REFRESH} is invoked
     */
    void restartLoader();

    /**
     * Called when {@link com.naman14.timber.MusicService#PLAYLIST_CHANGED} is invoked
     */
    void onPlaylistChanged();

    /**
     * Called when {@link com.naman14.timber.MusicService#META_CHANGED} is invoked
     */
    void onMetaChanged();

}
```

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.listeners;

import android.annotation.TargetApi;
import android.transition.Transition;

@TargetApi(21)
public class SimplelTransitionListener
        implements Transition.TransitionListener {
    public void onTransitionCancel(Transition paramTransition) {
    }

    public void onTransitionEnd(Transition paramTransition) {
    }

    public void onTransitionPause(Transition paramTransition) {
    }

    public void onTransitionResume(Transition paramTransition) {
    }

    public void onTransitionStart(Transition paramTransition) {
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\models\Album.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.models;

public class Album {
    public final long artistId;
    public final String artistName;
    public final long id;
    public final int songCount;
    public final String title;
    public final int year;

    public Album() {
        this.id = -1;
        this.title = "";
        this.artistName = "";
        this.artistId = -1;
        this.songCount = -1;
        this.year = -1;
    }

    public Album(long _id, String _title, String _artistName, long _artistId, int _songCount, int _year) {
        this.id = _id;
        this.title = _title;
        this.artistName = _artistName;
        this.artistId = _artistId;
        this.songCount = _songCount;
        this.year = _year;
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\models\Artist.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.models;

public class Artist {

    public final int albumCount;
    public final long id;
    public final String name;
    public final int songCount;

    public Artist() {
        this.id = -1;
        this.name = "";
        this.songCount = -1;
        this.albumCount = -1;
    }

    public Artist(long _id, String _name, int _albumCount, int _songCount) {
        this.id = _id;
        this.name = _name;
        this.songCount = _songCount;
        this.albumCount = _albumCount;
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\models\Playlist.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.models;

public class Playlist {

    public final long id;
    public final String name;
    public final int songCount;

    public Playlist() {
        this.id = -1;
        this.name = "";
        this.songCount = -1;
    }

    public Playlist(long _id, String _name, int _songCount) {
        this.id = _id;
        this.name = _name;
        this.songCount = _songCount;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\models\Song.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.models;

public class Song {

    public final long albumId;
    public final String albumName;
    public final long artistId;
    public final String artistName;
    public final int duration;
    public final long id;
    public final String title;
    public final int trackNumber;

    public Song() {
        this.id = -1;
        this.albumId = -1;
        this.artistId = -1;
        this.title = "";
        this.artistName = "";
        this.albumName = "";
        this.duration = -1;
        this.trackNumber = -1;
    }

    public Song(long _id, long _albumId, long _artistId, String _title, String _artistName, String _albumName, int _duration
        this.id = _id;
        this.albumId = _albumId;
        this.artistId = _artistId;
        this.title = _title;
        this.artistName = _artistName;
        this.albumName = _albumName;
        this.duration = _duration;
        this.trackNumber = _trackNumber;
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.nowplaying;

import android.animation.ObjectAnimator;
import android.graphics.Bitmap;
import android.graphics.PorterDuff;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.support.annotation.Nullable;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.app.Fragment;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.activities.BaseActivity;
import com.naman14.timber.adapters.BaseQueueAdapter;
import com.naman14.timber.adapters.SlidingQueueAdapter;
import com.naman14.timber.dataloaders.QueueLoader;
import com.naman14.timber.listeners.MusicStateListener;
import com.naman14.timber.timely.TimelyView;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.SlideTrackSwitcher;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.CircularSeekBar;
import com.naman14.timber.widgets.DividerItemDecoration;
import com.naman14.timber.widgets.PlayPauseButton;
import com.naman14.timber.widgets.PlayPauseDrawable;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;
import net.steamcrafted.materialiconlib.MaterialIconView;
import java.security.InvalidParameterException;

public class BaseNowplayingFragment extends Fragment implements MusicStateListener {

    private MaterialIconView previous, next;
    private PlayPauseButton mPlayPause;
    private PlayPauseDrawable playPauseDrawable = new PlayPauseDrawable();
    private FloatingActionButton playPauseFloating;
    private View playPauseWrapper;

    private String ateKey;
    private int overflowcounter = 0;
    private TextView songtitle, songalbum, songartist, songduration, elapsedtime;
    private SeekBar mProgress;
    boolean fragmentPaused = false;

    private CircularSeekBar mCircularProgress;
    private BaseQueueAdapter mAdapter;
    private SlidingQueueAdapter slidingQueueAdapter;

    private TimelyView timelyView11, timelyView12, timelyView13, timelyView14, timelyView15;
    private TextView hourColon;
    private int[] timeArr = new int[]{0, 0, 0, 0, 0};
    private Handler mElapsedTimeHandler;
    private boolean duetoplaypause = false;

    public ImageView albumart, shuffle, repeat;
    public int accentColor;
    public RecyclerView recyclerView;

    //seekbar
    public Runnable mUpdateProgress = new Runnable() {

        @Override
        public void run() {

            long position = MusicPlayer.position();
            if (mProgress != null) {
                mProgress.setProgress((int) position);
                if (elapsedtime != null && getActivity() != null)
                    elapsedtime.setText(TimberUtils.makeShortTimeString(getActivity(), position / 1000));
            }
            overflowcounter--;
            int delay = 250; //not sure why this delay was so high before
            if (overflowcounter < 0 && !fragmentPaused) {
                    overflowcounter++;
                    mProgress.postDelayed(mUpdateProgress, delay); //delay
            }
        }
    };

    //circular seekbar
    public Runnable mUpdateCircularProgress = new Runnable() {

        @Override
        public void run() {
            long position = MusicPlayer.position();
            if (mCircularProgress != null) {
                mCircularProgress.setProgress((int) position);
                if (elapsedtime != null && getActivity() != null)
                    elapsedtime.setText(TimberUtils.makeShortTimeString(getActivity(), position / 1000));

            }
            overflowcounter--;
            if (MusicPlayer.isPlaying()) {
                int delay = (int) (1500 - (position % 1000));
                if (overflowcounter < 0 && !fragmentPaused) {
                    overflowcounter++;
                    mCircularProgress.postDelayed(mUpdateCircularProgress, delay);
                }
            }

        }
    };

    public Runnable mUpdateElapsedTime = new Runnable() {
        @Override
        public void run() {
            if (getActivity() != null) {
                String time = TimberUtils.makeShortTimeString(getActivity(), MusicPlayer.position() / 1000);
                if (time.length() < 5) {
                    timelyView11.setVisibility(View.GONE);
                    timelyView12.setVisibility(View.GONE);
                    hourColon.setVisibility(View.GONE);
                    tv13(time.charAt(0) - '0');
                    tv14(time.charAt(2) - '0');
                    tv15(time.charAt(3) - '0');
                } else if (time.length() == 5) {
                    timelyView12.setVisibility(View.VISIBLE);
                    tv12(time.charAt(0) - '0');
                    tv13(time.charAt(1) - '0');
                    tv14(time.charAt(3) - '0');
                    tv15(time.charAt(4) - '0');
                } else {
                    timelyView11.setVisibility(View.VISIBLE);
                    hourColon.setVisibility(View.VISIBLE);
                    tv11(time.charAt(0) - '0');
                    tv12(time.charAt(2) - '0');
                    tv13(time.charAt(3) - '0');
                    tv14(time.charAt(5) - '0');
                    tv15(time.charAt(6) - '0');
                }
                mElapsedTimeHandler.postDelayed(this, 600);
            }

        }
    };

    private final View.OnClickListener mButtonListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            duetoplaypause = true;
            if (!mPlayPause.isPlayed()) {
                mPlayPause.setPlayed(true);
                mPlayPause.startAnimation();
            } else {
                mPlayPause.setPlayed(false);
                mPlayPause.startAnimation();
            }
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.playOrPause();
                    if (recyclerView != null && recyclerView.getAdapter() != null)
                        recyclerView.getAdapter().notifyDataSetChanged();
                }
            }, 200);


        }
    };

    private final View.OnClickListener mFLoatingButtonListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            duetoplaypause = true;
            if(MusicPlayer.getCurrentTrack() == null) {
                Toast.makeText(getContext(), getString(R.string.now_playing_no_track_selected), Toast.LENGTH_SHORT).show();
            } else {
                playPauseDrawable.transformToPlay(true);
                playPauseDrawable.transformToPause(true);
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        MusicPlayer.playOrPause();
                        if (recyclerView != null && recyclerView.getAdapter() != null)
                            recyclerView.getAdapter().notifyDataSetChanged();
                    }
                }, 250);
            }



        }
    };

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ateKey = Helpers.getATEKey(getActivity());
        accentColor = Config.accentColor(getActivity(), ateKey);
    }

    @Override
    public void onActivityCreated(@Nullable Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.now_playing, menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_go_to_album:
                NavigationUtils.goToAlbum(getContext(), MusicPlayer.getCurrentAlbumId());
                break;
            case R.id.menu_go_to_artist:
                NavigationUtils.goToArtist(getContext(), MusicPlayer.getCurrentArtistId());
                break;
            case R.id.action_lyrics:
                NavigationUtils.goToLyrics(getContext());
                break;
        }
        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onPause() {
        super.onPause();
        fragmentPaused = true;
    }

    @Override
    public void onResume() {
        super.onResume();
        fragmentPaused = false;
        if (mProgress != null)
            mProgress.postDelayed(mUpdateProgress, 10);

        if (mCircularProgress != null)
            mCircularProgress.postDelayed(mUpdateCircularProgress, 10);
    }
    public void setSongDetails(View view) {

        albumart = (ImageView) view.findViewById(R.id.album_art);
        shuffle = (ImageView) view.findViewById(R.id.shuffle);
        repeat = (ImageView) view.findViewById(R.id.repeat);
        next = (MaterialIconView) view.findViewById(R.id.next);
        previous = (MaterialIconView) view.findViewById(R.id.previous);
        mPlayPause = (PlayPauseButton) view.findViewById(R.id.playpause);
        playPauseFloating = (FloatingActionButton) view.findViewById(R.id.playpausefloating);
        playPauseWrapper = view.findViewById(R.id.playpausewrapper);

        songtitle = (TextView) view.findViewById(R.id.song_title);
        songalbum = (TextView) view.findViewById(R.id.song_album);
        songartist = (TextView) view.findViewById(R.id.song_artist);
        songduration = (TextView) view.findViewById(R.id.song_duration);
        elapsedtime = (TextView) view.findViewById(R.id.song_elapsed_time);

        timelyView11 = (TimelyView) view.findViewById(R.id.timelyView11);
        timelyView12 = (TimelyView) view.findViewById(R.id.timelyView12);
        timelyView13 = (TimelyView) view.findViewById(R.id.timelyView13);
        timelyView14 = (TimelyView) view.findViewById(R.id.timelyView14);
        timelyView15 = (TimelyView) view.findViewById(R.id.timelyView15);
        hourColon = (TextView) view.findViewById(R.id.hour_colon);

        mProgress = (SeekBar) view.findViewById(R.id.song_progress);
        mCircularProgress = (CircularSeekBar) view.findViewById(R.id.song_progress_circular);

        recyclerView = (RecyclerView) view.findViewById(R.id.queue_recyclerview);


        songtitle.setSelected(true);


        Toolbar toolbar = (Toolbar) view.findViewById(R.id.toolbar);
        if (toolbar != null) {
            ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);
            final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
            ab.setDisplayHomeAsUpEnabled(true);
            ab.setTitle("");
        }
        if (mPlayPause != null && getActivity() != null) {
            mPlayPause.setColor(ContextCompat.getColor(getContext(), android.R.color.white));
        }

        if (playPauseFloating != null) {
            playPauseDrawable.setColorFilter(TimberUtils.getBlackWhiteColor(accentColor), PorterDuff.Mode.MULTIPLY);
            playPauseFloating.setImageDrawable(playPauseDrawable);
            if (MusicPlayer.isPlaying())
                playPauseDrawable.transformToPause(false);
            else playPauseDrawable.transformToPlay(false);
        }

        if (mCircularProgress != null) {
            mCircularProgress.setCircleProgressColor(accentColor);
            mCircularProgress.setPointerColor(accentColor);
            mCircularProgress.setPointerHaloColor(accentColor);
        }

        if (timelyView11 != null) {
            String time = TimberUtils.makeShortTimeString(getActivity(), MusicPlayer.position() / 1000);
            if (time.length() < 5) {
                timelyView11.setVisibility(View.GONE);
                timelyView12.setVisibility(View.GONE);
                hourColon.setVisibility(View.GONE);

                changeDigit(timelyView13, time.charAt(0) - '0');
                changeDigit(timelyView14, time.charAt(2) - '0');
                changeDigit(timelyView15, time.charAt(3) - '0');

            } else if (time.length() == 5) {
                timelyView12.setVisibility(View.VISIBLE);
                changeDigit(timelyView12, time.charAt(0) - '0');
                changeDigit(timelyView13, time.charAt(1) - '0');
                changeDigit(timelyView14, time.charAt(3) - '0');
                changeDigit(timelyView15, time.charAt(4) - '0');
            } else {
                timelyView11.setVisibility(View.VISIBLE);
                hourColon.setVisibility(View.VISIBLE);
                changeDigit(timelyView11, time.charAt(0) - '0');
                changeDigit(timelyView12, time.charAt(2) - '0');
                changeDigit(timelyView13, time.charAt(3) - '0');
                changeDigit(timelyView14, time.charAt(5) - '0');
                changeDigit(timelyView15, time.charAt(6) - '0');
            }
        }

        setSongDetails();

    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        if (PreferenceManager.getDefaultSharedPreferences(getActivity()).getBoolean("dark_theme", false)) {
            ATE.apply(this, "dark_theme");
        } else {
            ATE.apply(this, "light_theme");
        }
    }

    private void setSongDetails() {
        updateSongDetails();

        if (recyclerView != null)
            setQueueSongs();

        setSeekBarListener();

        if (next != null) {
            next.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Handler handler = new Handler();
                    handler.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            MusicPlayer.next();
                            notifyPlayingDrawableChange();
                        }
                    }, 200);

                }
            });
        }
        if (previous != null) {
            previous.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Handler handler = new Handler();
                    handler.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            MusicPlayer.previous(getActivity(), false);
                            notifyPlayingDrawableChange();
                        }
                    }, 200);

                }
            });
        }
```

```java
        if (playPauseWrapper != null)
            playPauseWrapper.setOnClickListener(mButtonListener);

        if (playPauseFloating != null)
            playPauseFloating.setOnClickListener(mFLoatingButtonListener);

        updateShuffleState();
        updateRepeatState();

    }

    public void updateShuffleState() {
        if (shuffle != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                    .setSizeDp(30);

            if (getActivity() != null) {
                if (MusicPlayer.getShuffleMode() == 0) {
                    builder.setColor(Config.textColorPrimary(getActivity(), ateKey));
                } else builder.setColor(Config.accentColor(getActivity(), ateKey));
            }

            shuffle.setImageDrawable(builder.build());
            shuffle.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleShuffle();
                    updateShuffleState();
                    updateRepeatState();
                }
            });
        }
    }

    public void updateRepeatState() {
        if (repeat != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setSizeDp(30);

                if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_NONE) {
                    builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
                    builder.setColor(Config.textColorPrimary(getActivity(), ateKey));
                } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_CURRENT) {
                    builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT_ONCE);
                    builder.setColor(Config.accentColor(getActivity(), ateKey));
                } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_ALL) {
                    builder.setColor(Config.accentColor(getActivity(), ateKey));
                    builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
                }


            repeat.setImageDrawable(builder.build());
            repeat.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleRepeat();
                    updateRepeatState();
                    updateShuffleState();
                }
            });
        }
    }

    private void setSeekBarListener() {
        if (mProgress != null)
            mProgress.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
                @Override
                public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
```

```
                if (b) {
                    MusicPlayer.seek((long) i);
                }
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
            }
        });
    if (mCircularProgress != null) {
        mCircularProgress.setOnSeekBarChangeListener(new CircularSeekBar.OnCircularSeekBarChangeListener() {
            @Override
            public void onProgressChanged(CircularSeekBar circularSeekBar, int progress, boolean fromUser) {
                if (fromUser) {
                    MusicPlayer.seek((long) progress);
                }
            }

            @Override
            public void onStopTrackingTouch(CircularSeekBar seekBar) {

            }

            @Override
            public void onStartTrackingTouch(CircularSeekBar seekBar) {

            }
        });
    }
}

public void updateSongDetails() {
    //do not reload image if it was a play/pause change
    if (!duetoplaypause) {
        if (albumart != null) {
            ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(MusicPlayer.getCurrentAlbumId()).toString(
                    new DisplayImageOptions.Builder().cacheInMemory(true)
                            .showImageOnFail(R.drawable.ic_empty_music2)
                            .build(), new SimpleImageLoadingListener() {

                        @Override
                        public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                            doAlbumArtStuff(loadedImage);
                        }

                        @Override
                        public void onLoadingFailed(String imageUri, View view, FailReason failReason) {
                            Bitmap failedBitmap = ImageLoader.getInstance().loadImageSync("drawable://" + R.drawable.ic_
                            doAlbumArtStuff(failedBitmap);
                        }

                    });
        }
        if (songtitle != null && MusicPlayer.getTrackName() != null) {
                songtitle.setText(MusicPlayer.getTrackName());
                if(MusicPlayer.getTrackName().length() <= 23){
                    songtitle.setTextSize(25);
                }
                else if(MusicPlayer.getTrackName().length() >= 30){
                    songtitle.setTextSize(18);
                }
                else{
                    songtitle.setTextSize(18 + (MusicPlayer.getTrackName().length() - 24));
                }
                Log.v("BaseNowPlayingFrag", "Title Text Size: " + songtitle.getTextSize());
        }
```

```
            if (songartist != null) {
                songartist.setText(MusicPlayer.getArtistName());
                songartist.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        NavigationUtils.goToArtist(getContext(), MusicPlayer.getCurrentArtistId());
                    }
                });
            }
            if (songalbum != null)
                songalbum.setText(MusicPlayer.getAlbumName());

        }
        duetoplaypause = false;

        if (mPlayPause != null)
            updatePlayPauseButton();

        if (playPauseFloating != null)
            updatePlayPauseFloatingButton();

        if (songduration != null && getActivity() != null)
            songduration.setText(TimberUtils.makeShortTimeString(getActivity(), MusicPlayer.duration() / 1000));

        if (mProgress != null) {
            mProgress.setMax((int) MusicPlayer.duration());
            if (mUpdateProgress != null) {
                mProgress.removeCallbacks(mUpdateProgress);
            }
            mProgress.postDelayed(mUpdateProgress, 10);
        }
        if (mCircularProgress != null) {
            mCircularProgress.setMax((int) MusicPlayer.duration());
            if (mUpdateCircularProgress != null) {
                mCircularProgress.removeCallbacks(mUpdateCircularProgress);
            }
            mCircularProgress.postDelayed(mUpdateCircularProgress, 10);
        }

        if (timelyView11 != null) {
            mElapsedTimeHandler = new Handler();
            mElapsedTimeHandler.postDelayed(mUpdateElapsedTime, 600);
        }
    }

    public void setQueueSongs() {
        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
        //load queue songs in asynctask
        if (getActivity() != null)
            new loadQueueSongs().execute("");

    }

    public void updatePlayPauseButton() {
        if (MusicPlayer.isPlaying()) {
            if (!mPlayPause.isPlayed()) {
                mPlayPause.setPlayed(true);
                mPlayPause.startAnimation();
            }
        } else {
            if (mPlayPause.isPlayed()) {
                mPlayPause.setPlayed(false);
                mPlayPause.startAnimation();
            }
        }
    }

    public void updatePlayPauseFloatingButton() {
        if (MusicPlayer.isPlaying()) {
            playPauseDrawable.transformToPause(false);
        } else {
            playPauseDrawable.transformToPlay(false);
        }
    }

    public void notifyPlayingDrawableChange() {
        int position = MusicPlayer.getQueuePosition();
        BaseQueueAdapter.currentlyPlayingPosition = position;
    }

    public void restartLoader() {

    }

    public void onPlaylistChanged() {

    }

    public void onMetaChanged() {
        updateSongDetails();

        if (recyclerView != null && recyclerView.getAdapter() != null)
            recyclerView.getAdapter().notifyDataSetChanged();
    }

    public void setMusicStateListener() {
        ((BaseActivity) getActivity()).setMusicStateListenerListener(this);
    }

    public void doAlbumArtStuff(Bitmap loadedImage) {

    }

    public void changeDigit(TimelyView tv, int end) {
        ObjectAnimator obja = tv.animate(end);
        obja.setDuration(400);
        obja.start();
    }

    public void changeDigit(TimelyView tv, int start, int end) {
        try {
            ObjectAnimator obja = tv.animate(start, end);
            obja.setDuration(400);
            obja.start();
        } catch (InvalidParameterException e) {
            e.printStackTrace();
        }
    }

    public void tv11(int a) {
        if (a != timeArr[0]) {
            changeDigit(timelyView11, timeArr[0], a);
            timeArr[0] = a;
        }
    }

    public void tv12(int a) {
        if (a != timeArr[1]) {
            changeDigit(timelyView12, timeArr[1], a);
            timeArr[1] = a;
        }
    }

    public void tv13(int a) {
        if (a != timeArr[2]) {
            changeDigit(timelyView13, timeArr[2], a);
            timeArr[2] = a;
        }
    }
    public void tv14(int a) {
        if (a != timeArr[3]) {
            changeDigit(timelyView14, timeArr[3], a);
            timeArr[3] = a;
        }
    }

    public void tv15(int a) {
        if (a != timeArr[4]) {
            changeDigit(timelyView15, timeArr[4], a);
            timeArr[4] = a;
        }
    }

    protected void initGestures(View v) {
        if (PreferencesUtility.getInstance(v.getContext()).isGesturesEnabled()) {
            new SlideTrackSwitcher() {
                @Override
                public void onSwipeBottom() {
                    getActivity().finish();
                }
            }.attach(v);
        }
    }

    private class loadQueueSongs extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            if (getActivity() != null) {
                mAdapter = new BaseQueueAdapter((AppCompatActivity) getActivity(), QueueLoader.getQueueSongs(getActivity()))
                return "Executed";
            } else return null;
        }

        @Override
        protected void onPostExecute(String result) {
            if (result != null) {
                recyclerView.setAdapter(mAdapter);
                if (getActivity() != null)
                    recyclerView.addItemDecoration(new DividerItemDecoration(getActivity(), DividerItemDecoration.VERTICAL_L
                recyclerView.scrollToPosition(MusicPlayer.getQueuePosition() - 1);
            }

        }

        @Override
        protected void onPreExecute() {
        }
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.nowplaying;

import android.os.Bundle;
import android.os.Handler;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.utils.TimberUtils;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

public class Timber1 extends BaseNowplayingFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_timber1, container, false);

        setMusicStateListener();
        setSongDetails(rootView);
        initGestures(rootView.findViewById(R.id.album_art));

        return rootView;
    }

    @Override
    public void updateShuffleState() {
        if (shuffle != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                    .setSizeDp(30);

            builder.setColor(TimberUtils.getBlackWhiteColor(accentColor));

            shuffle.setImageDrawable(builder.build());
            shuffle.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Handler handler = new Handler();
                    handler.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            MusicPlayer.setShuffleMode(MusicService.SHUFFLE_NORMAL);
                            MusicPlayer.next();
                            recyclerView.scrollToPosition(MusicPlayer.getQueuePosition());
                        }
                    }, 150);

                }
            });
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\nowplaying\Timber2.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.nowplaying;

import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.TransitionDrawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.utils.ImageUtils;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

public class Timber2 extends BaseNowplayingFragment {

    ImageView mBlurredArt;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_timber2, container, false);

        setMusicStateListener();
        setSongDetails(rootView);
        mBlurredArt = (ImageView) rootView.findViewById(R.id.album_art_blurred);

        initGestures(mBlurredArt);

        return rootView;
    }

    @Override
    public void updateShuffleState() {
        if (shuffle != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                    .setSizeDp(30);

            if (MusicPlayer.getShuffleMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            shuffle.setImageDrawable(builder.build());
            shuffle.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleShuffle();
                    updateShuffleState();
                    updateRepeatState();
                }
            });
        }
    }

    @Override
    public void updateRepeatState() {
        if (repeat != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setSizeDp(30);

            if (MusicPlayer.getRepeatMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_NONE) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
                builder.setColor(Color.WHITE);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_CURRENT) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT_ONCE);
                builder.setColor(accentColor);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_ALL) {
                builder.setColor(accentColor);
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
            }

            repeat.setImageDrawable(builder.build());
            repeat.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleRepeat();
                    updateRepeatState();
                    updateShuffleState();
                }
            });
        }
    }

    @Override
    public void doAlbumArtStuff(Bitmap loadedImage) {
        setBlurredAlbumArt blurredAlbumArt = new setBlurredAlbumArt();
        blurredAlbumArt.execute(loadedImage);
    }

    private class setBlurredAlbumArt extends AsyncTask<Bitmap, Void, Drawable> {

        @Override
        protected Drawable doInBackground(Bitmap... loadedImage) {
            Drawable drawable = null;
            try {
                drawable = ImageUtils.createBlurredImageFromBitmap(loadedImage[0], getActivity(), 6);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return drawable;
        }

        @Override
        protected void onPostExecute(Drawable result) {
            if (result != null) {
                if (mBlurredArt.getDrawable() != null) {
                    final TransitionDrawable td =
                            new TransitionDrawable(new Drawable[]{
                                    mBlurredArt.getDrawable(),
                                    result
                            });
                    mBlurredArt.setImageDrawable(td);
                    td.startTransition(200);
                } else {
                    mBlurredArt.setImageDrawable(result);
                }
            }
        }

        @Override
        protected void onPreExecute() {
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\nowplaying\Timber3.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.nowplaying;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;

public class Timber3 extends BaseNowplayingFragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_timber3, container, false);

        setMusicStateListener();
        setSongDetails(rootView);

        initGestures(rootView.findViewById(R.id.album_art));

        return rootView;
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\nowplaying\Timber4.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.nowplaying;

import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.TransitionDrawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.adapters.SlidingQueueAdapter;
import com.naman14.timber.dataloaders.QueueLoader;
import com.naman14.timber.utils.ImageUtils;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

public class Timber4 extends BaseNowplayingFragment {

    ImageView mBlurredArt;
    RecyclerView horizontalRecyclerview;
    SlidingQueueAdapter horizontalAdapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_timber4, container, false);

        setMusicStateListener();
        setSongDetails(rootView);

        mBlurredArt = (ImageView) rootView.findViewById(R.id.album_art_blurred);
        horizontalRecyclerview = (RecyclerView) rootView.findViewById(R.id.queue_recyclerview_horizontal);

        setupHorizontalQueue();
        initGestures(mBlurredArt);

        return rootView;
    }

    @Override
    public void updateShuffleState() {
        if (shuffle != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                    .setSizeDp(30);

            if (MusicPlayer.getShuffleMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            shuffle.setImageDrawable(builder.build());
            shuffle.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleShuffle();
                    updateShuffleState();
                    updateRepeatState();
                }
            });
        }
    }

    @Override
    public void updateRepeatState() {
        if (repeat != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setSizeDp(30);

            if (MusicPlayer.getRepeatMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_NONE) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
                builder.setColor(Color.WHITE);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_CURRENT) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT_ONCE);
                builder.setColor(accentColor);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_ALL) {
                builder.setColor(accentColor);
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
            }
            repeat.setImageDrawable(builder.build());
            repeat.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleRepeat();
                    updateRepeatState();
                    updateShuffleState();
                }
            });
        }
    }

    @Override
    public void doAlbumArtStuff(Bitmap loadedImage) {
        setBlurredAlbumArt blurredAlbumArt = new setBlurredAlbumArt();
        blurredAlbumArt.execute(loadedImage);
    }

    private void setupHorizontalQueue() {
        horizontalRecyclerview.setLayoutManager(new LinearLayoutManager(getActivity(), LinearLayoutManager.HORIZONTAL, false
        horizontalAdapter = new SlidingQueueAdapter(getActivity(), QueueLoader.getQueueSongs(getActivity()));
        horizontalRecyclerview.setAdapter(horizontalAdapter);
        horizontalRecyclerview.scrollToPosition(MusicPlayer.getQueuePosition() - 3);
    }

    private class setBlurredAlbumArt extends AsyncTask<Bitmap, Void, Drawable> {

        @Override
        protected Drawable doInBackground(Bitmap... loadedImage) {
            Drawable drawable = null;
            try {
                drawable = ImageUtils.createBlurredImageFromBitmap(loadedImage[0], getActivity(), 6);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return drawable;
        }

        @Override
        protected void onPostExecute(Drawable result) {
            if (result != null) {
                if (mBlurredArt.getDrawable() != null) {
                    final TransitionDrawable td =
                            new TransitionDrawable(new Drawable[]{
                                    mBlurredArt.getDrawable(),
                                    result
                            });
                    mBlurredArt.setImageDrawable(td);
                    td.startTransition(200);

                } else {
                    mBlurredArt.setImageDrawable(result);
                }
            }
        }

        @Override
        protected void onPreExecute() {
        }
    }


}
```

```java
package com.naman14.timber.nowplaying;

import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.TransitionDrawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.adapters.SlidingQueueAdapter;
import com.naman14.timber.dataloaders.QueueLoader;
import com.naman14.timber.utils.ImageUtils;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

/**
 * Created by naman on 22/02/17.
 */

public class Timber5 extends BaseNowplayingFragment {

    ImageView mBlurredArt;
    RecyclerView recyclerView;
    SlidingQueueAdapter adapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_timber5, container, false);

        setMusicStateListener();
        setSongDetails(rootView);

        mBlurredArt = (ImageView) rootView.findViewById(R.id.album_art_blurred);
        recyclerView = (RecyclerView) rootView.findViewById(R.id.queue_recyclerview_horizontal) ;
        initGestures(mBlurredArt);
        setupSlidingQueue();

        return rootView;
    }

    @Override
    public void updateShuffleState() {
        if (shuffle != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                    .setSizeDp(30);

            if (MusicPlayer.getShuffleMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            shuffle.setImageDrawable(builder.build());
            shuffle.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleShuffle();
                    updateShuffleState();
                    updateRepeatState();
                }
            });
        }
    }

    @Override
    public void updateRepeatState() {
        if (repeat != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setSizeDp(30);

            if (MusicPlayer.getRepeatMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_NONE) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
                builder.setColor(Color.WHITE);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_CURRENT) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT_ONCE);
                builder.setColor(accentColor);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_ALL) {
                builder.setColor(accentColor);
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
            }

            repeat.setImageDrawable(builder.build());
            repeat.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleRepeat();
                    updateRepeatState();
                    updateShuffleState();
                }
            });
        }
    }

    @Override
    public void doAlbumArtStuff(Bitmap loadedImage) {
        setBlurredAlbumArt blurredAlbumArt = new setBlurredAlbumArt();
        blurredAlbumArt.execute(loadedImage);
    }

    private void setupSlidingQueue() {
        recyclerView.setLayoutManager(new LinearLayoutManager(getActivity(), LinearLayoutManager.HORIZONTAL, false));
        adapter = new SlidingQueueAdapter((AppCompatActivity) getActivity(), QueueLoader.getQueueSongs(getActivity()));
        recyclerView.setAdapter(adapter);
        recyclerView.scrollToPosition(MusicPlayer.getQueuePosition() - 3);
    }


    private class setBlurredAlbumArt extends AsyncTask<Bitmap, Void, Drawable> {

        @Override
        protected Drawable doInBackground(Bitmap... loadedImage) {
            Drawable drawable = null;
            try {
                drawable = ImageUtils.createBlurredImageFromBitmap(loadedImage[0], getActivity(), 12);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return drawable;
        }

        @Override
        protected void onPostExecute(Drawable result) {
            if (result != null) {
                if (mBlurredArt.getDrawable() != null) {
                    final TransitionDrawable td =
                            new TransitionDrawable(new Drawable[]{
                                    mBlurredArt.getDrawable(),
                                    result
                            });
                    mBlurredArt.setImageDrawable(td);
                    td.startTransition(200);

                } else {
                    mBlurredArt.setImageDrawable(result);
                }
            }
        }

        @Override
        protected void onPreExecute() {
        }
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\nowplaying\Timber6.java

```java
package com.naman14.timber.nowplaying;

import android.graphics.Color;
import android.graphics.PorterDuff;
import android.graphics.PorterDuffColorFilter;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.SeekBar;
import android.widget.TextView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.CircleImageView;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

/**
 * Created by naman on 22/02/17.
 */

public class Timber6 extends BaseNowplayingFragment {

    TextView nextSong;
    CircleImageView nextArt;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(
                R.layout.fragment_timber6, container, false);

        setMusicStateListener();
        setSongDetails(rootView);

        initGestures(rootView.findViewById(R.id.album_art));

        ((SeekBar) rootView.findViewById(R.id.song_progress)).getProgressDrawable().setColorFilter(new PorterDuffColorFilter
        ((SeekBar) rootView.findViewById(R.id.song_progress)).getThumb().setColorFilter(new PorterDuffColorFilter(Color.WHIT

        nextSong = (TextView) rootView.findViewById(R.id.title_next);
        nextArt = (CircleImageView) rootView.findViewById(R.id.album_art_next);

        rootView.findViewById(R.id.nextView).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                MusicPlayer.next();
            }
        });

        return rootView;
    }

    @Override
    public void updateShuffleState() {
        if (shuffle != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setIcon(MaterialDrawableBuilder.IconValue.SHUFFLE)
                    .setSizeDp(30);

            if (MusicPlayer.getShuffleMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            shuffle.setImageDrawable(builder.build());
            shuffle.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleShuffle();
                    updateShuffleState();
                    updateRepeatState();
                }
            });
        }
    }

    @Override
    public void updateRepeatState() {
        if (repeat != null && getActivity() != null) {
            MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(getActivity())
                    .setSizeDp(30);

            if (MusicPlayer.getRepeatMode() == 0) {
                builder.setColor(Color.WHITE);
            } else builder.setColor(accentColor);

            if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_NONE) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
                builder.setColor(Color.WHITE);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_CURRENT) {
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT_ONCE);
                builder.setColor(accentColor);
            } else if (MusicPlayer.getRepeatMode() == MusicService.REPEAT_ALL) {
                builder.setColor(accentColor);
                builder.setIcon(MaterialDrawableBuilder.IconValue.REPEAT);
            }

            repeat.setImageDrawable(builder.build());
            repeat.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    MusicPlayer.cycleRepeat();
                    updateRepeatState();
                    updateShuffleState();
                }
            });
        }
    }

    @Override
    public void onMetaChanged() {
        super.onMetaChanged();
        if (getActivity() != null) {
            long nextId = MusicPlayer.getNextAudioId();
            Song next = SongLoader.getSongForID(getActivity(), nextId);
            nextSong.setText(next.title);
            nextArt.setImageURI(TimberUtils.getAlbumArtUri(next.albumId));
        }
    }
}
```

```java
/*
* The MIT License (MIT)

* Copyright (c) 2015 Michal Tajchert

* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*/

package com.naman14.timber.permissions;

import android.Manifest;
import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.os.Build;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

/**
 * Created by Michal Tajchert on 2015-06-04.
 */
public class Nammu {
    private static final String TAG = Nammu.class.getSimpleName();
    private static final String KEY_PREV_PERMISSIONS = "previous_permissions";
    private static final String KEY_IGNORED_PERMISSIONS = "ignored_permissions";
    private static Context context;
    private static SharedPreferences sharedPreferences;
    private static ArrayList<PermissionRequest> permissionRequests = new ArrayList<PermissionRequest>();

    public static void init(Context context) {
        sharedPreferences = context.getSharedPreferences("pl.tajchert.runtimepermissionhelper", Context.MODE_PRIVATE);
        Nammu.context = context;
    }

    /**
     * Check that all given permissions have been granted by verifying that each entry in the
     * given array is of the value {@link PackageManager#PERMISSION_GRANTED}.
     */
    public static boolean verifyPermissions(int[] grantResults) {
        for (int result : grantResults) {
            if (result != PackageManager.PERMISSION_GRANTED) {
                return false;
            }
        }
        return true;
    }

    /**
     * Returns true if the Activity has access to given permissions.
     */
    public static boolean hasPermission(Activity activity, String permission) {
        return activity.checkSelfPermission(permission) == PackageManager.PERMISSION_GRANTED;
    }

    /**
     * Returns true if the Activity has access to a all given permission.
     */
    public static boolean hasPermission(Activity activity, String[] permissions) {
        for (String permission : permissions) {
            if (activity.checkSelfPermission(permission) != PackageManager.PERMISSION_GRANTED) {
                return false;
            }
        }
        return true;
    }

    /*
     * If we override other methods, lets do it as well, and keep name same as it is already weird enough.
     * Returns true if we should show explanation why we need this permission.
     */
    public static boolean shouldShowRequestPermissionRationale(Activity activity, String permissions) {
        return activity.shouldShowRequestPermissionRationale(permissions);
    }

    public static void askForPermission(Activity activity, String permission, PermissionCallback permissionCallback) {
        askForPermission(activity, new String[]{permission}, permissionCallback);
    }

    public static void askForPermission(Activity activity, String[] permissions, PermissionCallback permissionCallback) {
        if (permissionCallback == null) {
            return;
        }
        if (hasPermission(activity, permissions)) {
            permissionCallback.permissionGranted();
            return;
        }
        PermissionRequest permissionRequest = new PermissionRequest(new ArrayList<String>(Arrays.asList(permissions)), permi
        permissionRequests.add(permissionRequest);

        activity.requestPermissions(permissions, permissionRequest.getRequestCode());
    }

    public static void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        PermissionRequest requestResult = new PermissionRequest(requestCode);
        if (permissionRequests.contains(requestResult)) {
            PermissionRequest permissionRequest = permissionRequests.get(permissionRequests.indexOf(requestResult));
            if (verifyPermissions(grantResults)) {
                //Permission has been granted
                permissionRequest.getPermissionCallback().permissionGranted();
            } else {
                permissionRequest.getPermissionCallback().permissionRefused();

            }
            permissionRequests.remove(requestResult);
        }
        refreshMonitoredList();
    }


    //Permission monitoring part below

    /**
     * Get list of currently granted permissions, without saving it inside Nammu
     *
     * @return currently granted permissions
     */
    public static ArrayList<String> getGrantedPermissions() {
        if (context == null) {
            throw new RuntimeException("Must call init() earlier");
        }
        ArrayList<String> permissions = new ArrayList<String>();
        ArrayList<String> permissionsGranted = new ArrayList<String>();
        //Group location
        permissions.add(Manifest.permission.ACCESS_FINE_LOCATION);
        permissions.add(Manifest.permission.ACCESS_COARSE_LOCATION);
        //Group Calendar
        permissions.add(Manifest.permission.WRITE_CALENDAR);
        permissions.add(Manifest.permission.READ_CALENDAR);
        //Group Camera
        permissions.add(Manifest.permission.CAMERA);
        //Group Contacts
        permissions.add(Manifest.permission.WRITE_CONTACTS);
        permissions.add(Manifest.permission.READ_CONTACTS);
        permissions.add(Manifest.permission.GET_ACCOUNTS);
        //Group Microphone
        permissions.add(Manifest.permission.RECORD_AUDIO);
        //Group Phone
        permissions.add(Manifest.permission.CALL_PHONE);
        permissions.add(Manifest.permission.READ_PHONE_STATE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
            permissions.add(Manifest.permission.READ_CALL_LOG);
        }
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
            permissions.add(Manifest.permission.WRITE_CALL_LOG);
        }
        permissions.add(Manifest.permission.ADD_VOICEMAIL);
        permissions.add(Manifest.permission.USE_SIP);
        permissions.add(Manifest.permission.PROCESS_OUTGOING_CALLS);
        //Group Body sensors
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT_WATCH) {
            permissions.add(Manifest.permission.BODY_SENSORS);
        }
        //Group SMS
        permissions.add(Manifest.permission.SEND_SMS);
        permissions.add(Manifest.permission.READ_SMS);
        permissions.add(Manifest.permission.RECEIVE_SMS);
        permissions.add(Manifest.permission.RECEIVE_WAP_PUSH);
        permissions.add(Manifest.permission.RECEIVE_MMS);
        //Group Storage
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
            permissions.add(Manifest.permission.READ_EXTERNAL_STORAGE);
        }
        permissions.add(Manifest.permission.WRITE_EXTERNAL_STORAGE);
        for (String permission : permissions) {
            if (context.checkSelfPermission(permission) == PackageManager.PERMISSION_GRANTED) {
                permissionsGranted.add(permission);
            }
        }
        return permissionsGranted;
    }

    /**
     * Refresh currently granted permission list, and save it for later comparing using @permissionCompare()
     */
    public static void refreshMonitoredList() {
        ArrayList<String> permissions = getGrantedPermissions();
        Set<String> set = new HashSet<String>();
        for (String perm : permissions) {
            set.add(perm);
        }
        sharedPreferences.edit().putStringSet(KEY_PREV_PERMISSIONS, set).apply();
    }

    /**
     * Get list of previous Permissions, from last refreshMonitoredList() call and they may be outdated,
     * use getGrantedPermissions() to get current
     */
    public static ArrayList<String> getPreviousPermissions() {
        ArrayList<String> prevPermissions = new ArrayList<String>();
        prevPermissions.addAll(sharedPreferences.getStringSet(KEY_PREV_PERMISSIONS, new HashSet<String>()));
        return prevPermissions;
    }

    public static ArrayList<String> getIgnoredPermissions() {
        ArrayList<String> ignoredPermissions = new ArrayList<String>();
        ignoredPermissions.addAll(sharedPreferences.getStringSet(KEY_IGNORED_PERMISSIONS, new HashSet<String>()));
        return ignoredPermissions;
    }

    /**
     * Lets see if we already ignore this permission
     */
    public static boolean isIgnoredPermission(String permission) {
        if (permission == null) {
            return false;
        }
        return getIgnoredPermissions().contains(permission);
    }

    /**
     * Use to ignore to particular Permission - even if user will deny or add it we won't receive a callback.
     *
     * @param permission Permission to ignore
     */
    public static void ignorePermission(String permission) {
        if (!isIgnoredPermission(permission)) {
            ArrayList<String> ignoredPermissions = getIgnoredPermissions();
            ignoredPermissions.add(permission);
            Set<String> set = new HashSet<String>();
            set.addAll(ignoredPermissions);
            sharedPreferences.edit().putStringSet(KEY_IGNORED_PERMISSIONS, set).apply();
        }
    }

    /**
     * Used to trigger comparing process - @permissionListener will be called each time Permission was revoked, or added (bu
     *
     * @param permissionListener Callback that handles all permission changes
     */
    public static void permissionCompare(PermissionListener permissionListener) {
        if (context == null) {
            throw new RuntimeException("Before comparing permissions you need to call Nammu.init(context)");

        }
        ArrayList<String> previouslyGranted = getPreviousPermissions();
        ArrayList<String> currentPermissions = getGrantedPermissions();
        ArrayList<String> ignoredPermissions = getIgnoredPermissions();
        for (String permission : ignoredPermissions) {
            if (previouslyGranted != null && !previouslyGranted.isEmpty()) {
                if (previouslyGranted.contains(permission)) {
                    previouslyGranted.remove(permission);
                }
            }

            if (currentPermissions != null && !currentPermissions.isEmpty()) {
                if (currentPermissions.contains(permission)) {
                    currentPermissions.remove(permission);
                }
            }
        }
        for (String permission : currentPermissions) {
            if (previouslyGranted.contains(permission)) {
                //All is fine, was granted and still is
                previouslyGranted.remove(permission);
            } else {
                //We didn't have it last time
                if (permissionListener != null) {
                    permissionListener.permissionsChanged(permission);
                    permissionListener.permissionsGranted(permission);
                }
            }
        }
        if (previouslyGranted != null && !previouslyGranted.isEmpty()) {
            //Something was granted and removed
            for (String permission : previouslyGranted) {
                if (permissionListener != null) {
                    permissionListener.permissionsChanged(permission);
                    permissionListener.permissionsRemoved(permission);
                }
            }
        }
        refreshMonitoredList();
    }

    /**
     * Not that needed method but if we override others it is good to keep same.
     */
    public static boolean checkPermission(String permissionName) {
        if (context == null) {
            throw new RuntimeException("Before comparing permissions you need to call Nammu.init(context)");
        }
        return PackageManager.PERMISSION_GRANTED == context.checkSelfPermission(permissionName);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\permissions\PermissionCallback.

```
/*
* The MIT License (MIT)

* Copyright (c) 2015 Michal Tajchert

* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*/

package com.naman14.timber.permissions;

public interface PermissionCallback {
    void permissionGranted();

    void permissionRefused();
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\permissions\PermissionListener.

```java
/*
* The MIT License (MIT)

* Copyright (c) 2015 Michal Tajchert

* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*/

package com.naman14.timber.permissions;

public interface PermissionListener {
    /**
     * Gets called each time we run Nammu.permissionCompare() and some Permission is revoke/granted to us
     *
     * @param permissionChanged
     */
    void permissionsChanged(String permissionChanged);

    /**
     * Gets called each time we run Nammu.permissionCompare() and some Permission is granted
     *
     * @param permissionGranted
     */
    void permissionsGranted(String permissionGranted);

    /**
     * Gets called each time we run Nammu.permissionCompare() and some Permission is removed
     *
     * @param permissionRemoved
     */
    void permissionsRemoved(String permissionRemoved);
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\permissions\PermissionRequest.j

```
package com.naman14.timber.permissions;

import java.util.ArrayList;
import java.util.Random;

public class PermissionRequest {
    private static Random random;
    private ArrayList<String> permissions;
    private int requestCode;
    private PermissionCallback permissionCallback;

    public PermissionRequest(int requestCode) {
        this.requestCode = requestCode;
    }

    public PermissionRequest(ArrayList<String> permissions, PermissionCallback permissionCallback) {
        this.permissions = permissions;
        this.permissionCallback = permissionCallback;
        if (random == null) {
            random = new Random();
        }
        this.requestCode = random.nextInt(32768);
    }

    public ArrayList<String> getPermissions() {
        return permissions;
    }

    public int getRequestCode() {
        return requestCode;
    }

    public PermissionCallback getPermissionCallback() {
        return permissionCallback;
    }

    public boolean equals(Object object) {
        if (object == null) {
            return false;
        }
        if (object instanceof PermissionRequest) {
            return ((PermissionRequest) object).requestCode == this.requestCode;
        }
        return false;
    }
    @Override
    public int hashCode() {
        return requestCode;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\provider\MusicDB.java

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.provider;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class MusicDB extends SQLiteOpenHelper {

    public static final String DATABASENAME = "musicdb.db";
    private static final int VERSION = 4;
    private static MusicDB sInstance = null;

    private final Context mContext;

    public MusicDB(final Context context) {
        super(context, DATABASENAME, null, VERSION);

        mContext = context;
    }

    public static final synchronized MusicDB getInstance(final Context context) {
        if (sInstance == null) {
            sInstance = new MusicDB(context.getApplicationContext());
        }
        return sInstance;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        MusicPlaybackState.getInstance(mContext).onCreate(db);
        RecentStore.getInstance(mContext).onCreate(db);
        SongPlayCount.getInstance(mContext).onCreate(db);
        SearchHistory.getInstance(mContext).onCreate(db);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        MusicPlaybackState.getInstance(mContext).onUpgrade(db, oldVersion, newVersion);
        RecentStore.getInstance(mContext).onUpgrade(db, oldVersion, newVersion);
        SongPlayCount.getInstance(mContext).onUpgrade(db, oldVersion, newVersion);
        SearchHistory.getInstance(mContext).onUpgrade(db, oldVersion, newVersion);
    }

    @Override
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        MusicPlaybackState.getInstance(mContext).onDowngrade(db, oldVersion, newVersion);
        RecentStore.getInstance(mContext).onDowngrade(db, oldVersion, newVersion);
        SongPlayCount.getInstance(mContext).onDowngrade(db, oldVersion, newVersion);
        SearchHistory.getInstance(mContext).onDowngrade(db, oldVersion, newVersion);
    }
}
```

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.provider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import com.naman14.timber.helpers.MusicPlaybackTrack;
import com.naman14.timber.utils.TimberUtils;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;

/**
 * This keeps track of the music playback and history state of the playback service
 */
public class MusicPlaybackState {
    private static MusicPlaybackState sInstance = null;

    private MusicDB mMusicDatabase = null;

    public MusicPlaybackState(final Context context) {
        mMusicDatabase = MusicDB.getInstance(context);
    }

    public static final synchronized MusicPlaybackState getInstance(final Context context) {
        if (sInstance == null) {
            sInstance = new MusicPlaybackState(context.getApplicationContext());
        }
        return sInstance;
    }

    public void onCreate(final SQLiteDatabase db) {
        StringBuilder builder = new StringBuilder();
        builder.append("CREATE TABLE IF NOT EXISTS ");
        builder.append(PlaybackQueueColumns.NAME);
        builder.append("(");

        builder.append(PlaybackQueueColumns.TRACK_ID);
        builder.append(" LONG NOT NULL,");

        builder.append(PlaybackQueueColumns.SOURCE_ID);
        builder.append(" LONG NOT NULL,");

        builder.append(PlaybackQueueColumns.SOURCE_TYPE);
        builder.append(" INT NOT NULL,");

        builder.append(PlaybackQueueColumns.SOURCE_POSITION);
        builder.append(" INT NOT NULL);");

        db.execSQL(builder.toString());

        builder = new StringBuilder();
        builder.append("CREATE TABLE IF NOT EXISTS ");
        builder.append(PlaybackHistoryColumns.NAME);
        builder.append("(");

        builder.append(PlaybackHistoryColumns.POSITION);
        builder.append(" INT NOT NULL);");

        db.execSQL(builder.toString());
    }

    public void onUpgrade(final SQLiteDatabase db, final int oldVersion, final int newVersion) {
        // this table was created in version 2 so call the onCreate method if we hit that scenario
        if (oldVersion < 2 && newVersion >= 2) {
            onCreate(db);
        }
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {

        db.execSQL("DROP TABLE IF EXISTS " + PlaybackQueueColumns.NAME);
        db.execSQL("DROP TABLE IF EXISTS " + PlaybackHistoryColumns.NAME);
        onCreate(db);
    }

    public synchronized void saveState(final ArrayList<MusicPlaybackTrack> queue,
                                       LinkedList<Integer> history) {
        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        database.beginTransaction();

        try {
            database.delete(PlaybackQueueColumns.NAME, null, null);
            database.delete(PlaybackHistoryColumns.NAME, null, null);
            database.setTransactionSuccessful();
        } finally {
            database.endTransaction();
        }

        final int NUM_PROCESS = 20;
        int position = 0;
        while (position < queue.size()) {
            database.beginTransaction();
            try {
                for (int i = position; i < queue.size() && i < position + NUM_PROCESS; i++) {
                    MusicPlaybackTrack track = queue.get(i);
                    ContentValues values = new ContentValues(4);

                    values.put(PlaybackQueueColumns.TRACK_ID, track.mId);
                    values.put(PlaybackQueueColumns.SOURCE_ID, track.mSourceId);
                    values.put(PlaybackQueueColumns.SOURCE_TYPE, track.mSourceType.mId);
                    values.put(PlaybackQueueColumns.SOURCE_POSITION, track.mSourcePosition);

                    database.insert(PlaybackQueueColumns.NAME, null, values);
                }
                database.setTransactionSuccessful();
            } finally {
                database.endTransaction();
                position += NUM_PROCESS;
            }
        }

        if (history != null) {
            Iterator<Integer> iter = history.iterator();
            while (iter.hasNext()) {
                database.beginTransaction();
                try {
                    for (int i = 0; iter.hasNext() && i < NUM_PROCESS; i++) {
                        ContentValues values = new ContentValues(1);
                        values.put(PlaybackHistoryColumns.POSITION, iter.next());

                        database.insert(PlaybackHistoryColumns.NAME, null, values);
                    }

                    database.setTransactionSuccessful();
                } finally {
                    database.endTransaction();
                }
            }
        }
    }

    public ArrayList<MusicPlaybackTrack> getQueue() {
        ArrayList<MusicPlaybackTrack> results = new ArrayList<>();

        Cursor cursor = null;
        try {
            cursor = mMusicDatabase.getReadableDatabase().query(PlaybackQueueColumns.NAME, null,
                    null, null, null, null, null);

            if (cursor != null && cursor.moveToFirst()) {
                results.ensureCapacity(cursor.getCount());

                do {
                    results.add(new MusicPlaybackTrack(cursor.getLong(0), cursor.getLong(1),
                            TimberUtils.IdType.getTypeById(cursor.getInt(2)), cursor.getInt(3)));
                } while (cursor.moveToNext());
            }

            return results;
        } finally {
            if (cursor != null) {
                cursor.close();
                cursor = null;
            }
        }
    }

    public LinkedList<Integer> getHistory(final int playlistSize) {
        LinkedList<Integer> results = new LinkedList<>();

        Cursor cursor = null;
        try {
            cursor = mMusicDatabase.getReadableDatabase().query(PlaybackHistoryColumns.NAME, null,
                    null, null, null, null, null);

            if (cursor != null && cursor.moveToFirst()) {
                do {
                    int pos = cursor.getInt(0);
                    if (pos >= 0 && pos < playlistSize) {
                        results.add(pos);
                    }
                } while (cursor.moveToNext());
            }

            return results;
        } finally {
            if (cursor != null) {
                cursor.close();
                cursor = null;
            }
        }
    }

    public class PlaybackQueueColumns {

        public static final String NAME = "playbackqueue";
        public static final String TRACK_ID = "trackid";
        public static final String SOURCE_ID = "sourceid";
        public static final String SOURCE_TYPE = "sourcetype";
        public static final String SOURCE_POSITION = "sourceposition";
    }
```

```
    public class PlaybackHistoryColumns {

        public static final String NAME = "playbackhistory";

        public static final String POSITION = "position";
    }
}
```

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.provider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

public class RecentStore {

    private static final int MAX_ITEMS_IN_DB = 100;

    private static RecentStore sInstance = null;

    private MusicDB mMusicDatabase = null;

    public RecentStore(final Context context) {
        mMusicDatabase = MusicDB.getInstance(context);
    }

    public static final synchronized RecentStore getInstance(final Context context) {
        if (sInstance == null) {
            sInstance = new RecentStore(context.getApplicationContext());
        }
        return sInstance;
    }

    public void onCreate(final SQLiteDatabase db) {
        db.execSQL("CREATE TABLE IF NOT EXISTS " + RecentStoreColumns.NAME + " ("
                + RecentStoreColumns.ID + " LONG NOT NULL," + RecentStoreColumns.TIMEPLAYED
                + " LONG NOT NULL);");
    }

    public void onUpgrade(final SQLiteDatabase db, final int oldVersion, final int newVersion) {
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + RecentStoreColumns.NAME);
        onCreate(db);
    }

    public void addSongId(final long songId) {
        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        database.beginTransaction();

        try {

            Cursor mostRecentItem = null;
            try {
                mostRecentItem = queryRecentIds("1");
                if (mostRecentItem != null && mostRecentItem.moveToFirst()) {
                    if (songId == mostRecentItem.getLong(0)) {
                        return;
                    }
                }
            } finally {
                if (mostRecentItem != null) {
                    mostRecentItem.close();
                    mostRecentItem = null;
                }
            }


            final ContentValues values = new ContentValues(2);
            values.put(RecentStoreColumns.ID, songId);
            values.put(RecentStoreColumns.TIMEPLAYED, System.currentTimeMillis());
            database.insert(RecentStoreColumns.NAME, null, values);

            Cursor oldest = null;
            try {
                oldest = database.query(RecentStoreColumns.NAME,
                        new String[]{RecentStoreColumns.TIMEPLAYED}, null, null, null, null,
                        RecentStoreColumns.TIMEPLAYED + " ASC");

                if (oldest != null && oldest.getCount() > MAX_ITEMS_IN_DB) {
                    oldest.moveToPosition(oldest.getCount() - MAX_ITEMS_IN_DB);
                    long timeOfRecordToKeep = oldest.getLong(0);

                    database.delete(RecentStoreColumns.NAME,
                            RecentStoreColumns.TIMEPLAYED + " < ?",
                            new String[]{String.valueOf(timeOfRecordToKeep)});

                }
            } finally {
                if (oldest != null) {
                    oldest.close();
                    oldest = null;
                }
            }
        } finally {
            database.setTransactionSuccessful();
            database.endTransaction();
        }
    }


    public void removeItem(final long songId) {
        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        database.delete(RecentStoreColumns.NAME, RecentStoreColumns.ID + " = ?", new String[]{
                String.valueOf(songId)
        });

    }

    public void deleteAll() {
        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        database.delete(RecentStoreColumns.NAME, null, null);
    }


    public Cursor queryRecentIds(final String limit) {
        final SQLiteDatabase database = mMusicDatabase.getReadableDatabase();
        return database.query(RecentStoreColumns.NAME,
                new String[]{RecentStoreColumns.ID}, null, null, null, null,
                RecentStoreColumns.TIMEPLAYED + " DESC", limit);
    }

    public interface RecentStoreColumns {
        /* Table name */
        String NAME = "recenthistory";

        /* Album IDs column */
        String ID = "songid";

        /* Time played column */
        String TIMEPLAYED = "timeplayed";
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\provider\SearchHistory.java

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.provider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

import java.util.ArrayList;

public class SearchHistory {

    private static final int MAX_ITEMS_IN_DB = 25;

    private static SearchHistory sInstance = null;

    private MusicDB mMusicDatabase = null;

    public SearchHistory(final Context context) {
        mMusicDatabase = MusicDB.getInstance(context);
    }

    public static final synchronized SearchHistory getInstance(final Context context) {
        if (sInstance == null) {
            sInstance = new SearchHistory(context.getApplicationContext());
        }
        return sInstance;
    }

    public void onCreate(final SQLiteDatabase db) {
        db.execSQL("CREATE TABLE IF NOT EXISTS " + SearchHistoryColumns.NAME + " ("
                + SearchHistoryColumns.SEARCHSTRING + " STRING NOT NULL,"
                + SearchHistoryColumns.TIMESEARCHED + " LONG NOT NULL);");
    }

    public void onUpgrade(final SQLiteDatabase db, final int oldVersion, final int newVersion) {
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + SearchHistoryColumns.NAME);
        onCreate(db);
    }

    public void addSearchString(final String searchString) {
        if (searchString == null) {
            return;
        }

        String trimmedString = searchString.trim();

        if (trimmedString.isEmpty()) {
            return;
        }

        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        database.beginTransaction();

        try {

            database.delete(SearchHistoryColumns.NAME,
                    SearchHistoryColumns.SEARCHSTRING + " = ? COLLATE NOCASE",
                    new String[]{trimmedString});

            final ContentValues values = new ContentValues(2);
            values.put(SearchHistoryColumns.SEARCHSTRING, trimmedString);
            values.put(SearchHistoryColumns.TIMESEARCHED, System.currentTimeMillis());
            database.insert(SearchHistoryColumns.NAME, null, values);

            Cursor oldest = null;
            try {
                database.query(SearchHistoryColumns.NAME,
                        new String[]{SearchHistoryColumns.TIMESEARCHED}, null, null, null, null,
                        SearchHistoryColumns.TIMESEARCHED + " ASC");

                if (oldest != null && oldest.getCount() > MAX_ITEMS_IN_DB) {
                    oldest.moveToPosition(oldest.getCount() - MAX_ITEMS_IN_DB);
                    long timeOfRecordToKeep = oldest.getLong(0);

                    database.delete(SearchHistoryColumns.NAME,
                            SearchHistoryColumns.TIMESEARCHED + " < ?",
                            new String[]{String.valueOf(timeOfRecordToKeep)});

                }
            } finally {
                if (oldest != null) {
                    oldest.close();
                    oldest = null;
                }
            }
        } finally {
            database.setTransactionSuccessful();
            database.endTransaction();
        }
    }


    public Cursor queryRecentSearches(final String limit) {
        final SQLiteDatabase database = mMusicDatabase.getReadableDatabase();
        return database.query(SearchHistoryColumns.NAME,
                new String[]{SearchHistoryColumns.SEARCHSTRING}, null, null, null, null,
                SearchHistoryColumns.TIMESEARCHED + " DESC", limit);
    }

    public ArrayList<String> getRecentSearches() {
        Cursor searches = queryRecentSearches(String.valueOf(MAX_ITEMS_IN_DB));

        ArrayList<String> results = new ArrayList<String>(MAX_ITEMS_IN_DB);

        try {
            if (searches != null && searches.moveToFirst()) {
                int colIdx = searches.getColumnIndex(SearchHistoryColumns.SEARCHSTRING);

                do {
                    results.add(searches.getString(colIdx));
                } while (searches.moveToNext());
            }
        } finally {
            if (searches != null) {
                searches.close();
                searches = null;
            }
        }

        return results;
    }
    public interface SearchHistoryColumns {
        /* Table name */
        String NAME = "searchhistory";

        /* What was searched */
        String SEARCHSTRING = "searchstring";

        /* Time of search */
        String TIMESEARCHED = "timesearched";
    }
}
```

```java
/*
* Copyright (C) 2014 The CyanogenMod Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
* http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.provider;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.view.animation.AccelerateInterpolator;
import android.view.animation.Interpolator;

import java.util.HashSet;
import java.util.Iterator;

/**
 * This database tracks the number of play counts for an individual song.  This is used to drive
 * the top played tracks as well as the playlist images
 */
public class SongPlayCount {
    // how many weeks worth of playback to track
    private static final int NUM_WEEKS = 52;
    private static SongPlayCount sInstance = null;
    // interpolator curve applied for measuring the curve
    private static Interpolator sInterpolator = new AccelerateInterpolator(1.5f);
    // how high to multiply the interpolation curve
    private static int INTERPOLATOR_HEIGHT = 50;
    // how high the base value is. The ratio of the Height to Base is what really matters
    private static int INTERPOLATOR_BASE = 25;
    private static int ONE_WEEK_IN_MS = 1000 * 60 * 60 * 24 * 7;
    private static String WHERE_ID_EQUALS = SongPlayCountColumns.ID + "=?";
    private MusicDB mMusicDatabase = null;
    // number of weeks since epoch time
    private int mNumberOfWeeksSinceEpoch;

    // used to track if we've walkd through the db and updated all the rows
    private boolean mDatabaseUpdated;

    /**
     * Constructor of <code>RecentStore</code>
     *
     * @param context The {@link android.content.Context} to use
     */
    public SongPlayCount(final Context context) {
        mMusicDatabase = MusicDB.getInstance(context);

        long msSinceEpoch = System.currentTimeMillis();
        mNumberOfWeeksSinceEpoch = (int) (msSinceEpoch / ONE_WEEK_IN_MS);

        mDatabaseUpdated = false;
    }

    /**
     * @param context The {@link android.content.Context} to use
     * @return A new instance of this class.
     */
    public static final synchronized SongPlayCount getInstance(final Context context) {
        if (sInstance == null) {
            sInstance = new SongPlayCount(context.getApplicationContext());
        }
        return sInstance;
    }

    /**
     * Calculates the score of the song given the play counts
     *
     * @param playCounts an array of the # of times a song has been played for each week
     *                   where playCounts[N] is the # of times it was played N weeks ago
     * @return the score
     */
    private static float calculateScore(final int[] playCounts) {
        if (playCounts == null) {
            return 0;
        }

        float score = 0;
        for (int i = 0; i < Math.min(playCounts.length, NUM_WEEKS); i++) {
            score += playCounts[i] * getScoreMultiplierForWeek(i);
        }

        return score;
    }

    /**
     * Gets the column name for each week #
     *
     * @param week number
     * @return the column name
     */
    private static String getColumnNameForWeek(final int week) {
        return SongPlayCountColumns.WEEK_PLAY_COUNT + String.valueOf(week);
    }

    /**
     * Gets the score multiplier for each week
     *
     * @param week number
     * @return the multiplier to apply
     */
    private static float getScoreMultiplierForWeek(final int week) {
        return sInterpolator.getInterpolation(1 - (week / (float) NUM_WEEKS)) * INTERPOLATOR_HEIGHT
                + INTERPOLATOR_BASE;
    }

    /**
     * For some performance gain, return a static value for the column index for a week
     * WARNIGN: This function assumes you have selected all columns for it to work
     *
     * @param week number
     * @return column index of that week
     */
    private static int getColumnIndexForWeek(final int week) {
        // ID, followed by the weeks columns
        return 1 + week;
    }

    public void onCreate(final SQLiteDatabase db) {
        // create the play count table
        // WARNING: If you change the order of these columns
        // please update getColumnIndexForWeek
        StringBuilder builder = new StringBuilder();
        builder.append("CREATE TABLE IF NOT EXISTS ");
        builder.append(SongPlayCountColumns.NAME);
        builder.append("(");
        builder.append(SongPlayCountColumns.ID);
        builder.append(" INT UNIQUE,");
        for (int i = 0; i < NUM_WEEKS; i++) {
            builder.append(getColumnNameForWeek(i));
            builder.append(" INT DEFAULT 0,");
        }

        builder.append(SongPlayCountColumns.LAST_UPDATED_WEEK_INDEX);
        builder.append(" INT NOT NULL,");

        builder.append(SongPlayCountColumns.PLAYCOUNTSCORE);
        builder.append(" REAL DEFAULT 0);");

        db.execSQL(builder.toString());
    }

    public void onUpgrade(final SQLiteDatabase db, final int oldVersion, final int newVersion) {
        // No upgrade path needed yet
    }

    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // If we ever have downgrade, drop the table to be safe
        db.execSQL("DROP TABLE IF EXISTS " + SongPlayCountColumns.NAME);
        onCreate(db);
    }

    /**
     * Increases the play count of a song by 1
     *
     * @param songId The song id to increase the play count
     */
    public void bumpSongCount(final long songId) {
        if (songId < 0) {
            return;
        }

        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        updateExistingRow(database, songId, true);
    }

    /**
     * This creates a new entry that indicates a song has been played once as well as its score
     *
     * @param database a writeable database
     * @param songId   the id of the track
     */
    private void createNewPlayedEntry(final SQLiteDatabase database, final long songId) {
        // no row exists, create a new one
        float newScore = getScoreMultiplierForWeek(0);
        int newPlayCount = 1;

        final ContentValues values = new ContentValues(3);
        values.put(SongPlayCountColumns.ID, songId);
        values.put(SongPlayCountColumns.PLAYCOUNTSCORE, newScore);
        values.put(SongPlayCountColumns.LAST_UPDATED_WEEK_INDEX, mNumberOfWeeksSinceEpoch);
        values.put(getColumnNameForWeek(0), newPlayCount);

        database.insert(SongPlayCountColumns.NAME, null, values);
    }

    /**
     * This function will take a song entry and update it to the latest week and increase the count
     * for the current week by 1 if necessary
     *
     * @param database  a writeable database
     * @param id        the id of the track to bump
     * @param bumpCount whether to bump the current's week play count by 1 and adjust the score
     */
    private void updateExistingRow(final SQLiteDatabase database, final long id, boolean bumpCount) {
        String stringId = String.valueOf(id);

        // begin the transaction
        database.beginTransaction();

        // get the cursor of this content inside the transaction
        final Cursor cursor = database.query(SongPlayCountColumns.NAME, null, WHERE_ID_EQUALS,
                new String[]{stringId}, null, null, null);

        // if we have a result
        if (cursor != null && cursor.moveToFirst()) {
            // figure how many weeks since we last updated
            int lastUpdatedIndex = cursor.getColumnIndex(SongPlayCountColumns.LAST_UPDATED_WEEK_INDEX);
            int lastUpdatedWeek = cursor.getInt(lastUpdatedIndex);
            int weekDiff = mNumberOfWeeksSinceEpoch - lastUpdatedWeek;

            // if it's more than the number of weeks we track, delete it and create a new entry
            if (Math.abs(weekDiff) >= NUM_WEEKS) {
                // this entry needs to be dropped since it is too outdated
                deleteEntry(database, stringId);
                if (bumpCount) {
                    createNewPlayedEntry(database, id);
                }
            } else if (weekDiff != 0) {
                // else, shift the weeks
                int[] playCounts = new int[NUM_WEEKS];

                if (weekDiff > 0) {
                    // time is shifted forwards
                    for (int i = 0; i < NUM_WEEKS - weekDiff; i++) {
                        playCounts[i + weekDiff] = cursor.getInt(getColumnIndexForWeek(i));
                    }
                } else if (weekDiff < 0) {
                    // time is shifted backwards (by user) - nor typical behavior but we
                    // will still handle it

                    // since weekDiff is -ve, NUM_WEEKS + weekDiff is the real # of weeks we have to
                    // transfer.  Then we transfer the old week i - weekDiff to week i
                    // for example if the user shifted back 2 weeks, ie -2, then for 0 to
                    // NUM_WEEKS + (-2) we set the new week i = old week i - (-2) or i+2
                    for (int i = 0; i < NUM_WEEKS + weekDiff; i++) {
                        playCounts[i] = cursor.getInt(getColumnIndexForWeek(i - weekDiff));
                    }
                }

                // bump the count
                if (bumpCount) {
                    playCounts[0]++;
                }

                float score = calculateScore(playCounts);

                // if the score is non-existant, then delete it
                if (score < .01f) {
                    deleteEntry(database, stringId);
                } else {
                    // create the content values
                    ContentValues values = new ContentValues(NUM_WEEKS + 2);
                    values.put(SongPlayCountColumns.LAST_UPDATED_WEEK_INDEX, mNumberOfWeeksSinceEpoch);
                    values.put(SongPlayCountColumns.PLAYCOUNTSCORE, score);

                    for (int i = 0; i < NUM_WEEKS; i++) {
                        values.put(getColumnNameForWeek(i), playCounts[i]);
                    }

                    // update the entry
                    database.update(SongPlayCountColumns.NAME, values, WHERE_ID_EQUALS,
                            new String[]{stringId});
                }
            } else if (bumpCount) {
                // else no shifting, just update the scores
                ContentValues values = new ContentValues(2);
                // increase the score by a single score amount
                int scoreIndex = cursor.getColumnIndex(SongPlayCountColumns.PLAYCOUNTSCORE);
                float score = cursor.getFloat(scoreIndex) + getScoreMultiplierForWeek(0);
                values.put(SongPlayCountColumns.PLAYCOUNTSCORE, score);

                // increase the play count by 1
                values.put(getColumnNameForWeek(0), cursor.getInt(getColumnIndexForWeek(0)) + 1);

                // update the entry
                database.update(SongPlayCountColumns.NAME, values, WHERE_ID_EQUALS,
                        new String[]{stringId});
            }

            cursor.close();
        } else if (bumpCount) {
            // if we have no existing results, create a new one
            createNewPlayedEntry(database, id);
        }

        database.setTransactionSuccessful();
        database.endTransaction();
    }

    public void deleteAll() {
        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        database.delete(SongPlayCountColumns.NAME, null, null);
    }

    /**
     * Gets a cursor containing the top songs played.  Note this only returns songs that have been
     * played at least once in the past NUM_WEEKS
     *
     * @param numResults number of results to limit by.  If <= 0 it returns all results
     * @return the top tracks
     */
    public Cursor getTopPlayedResults(int numResults) {
        updateResults();

        final SQLiteDatabase database = mMusicDatabase.getReadableDatabase();
        return database.query(SongPlayCountColumns.NAME, new String[]{SongPlayCountColumns.ID},
                null, null, null, null, SongPlayCountColumns.PLAYCOUNTSCORE + " DESC",
                (numResults <= 0 ? null : String.valueOf(numResults)));
    }

    /**
     * Given a list of ids, it sorts the results based on the most played results
     *
     * @param ids list
     * @return sorted list - this may be smaller than the list passed in for performance reasons
     */
    public long[] getTopPlayedResultsForList(long[] ids) {
        final int MAX_NUMBER_SONGS_TO_ANALYZE = 250;

        if (ids == null || ids.length == 0) {
            return null;
        }

        HashSet<Long> uniqueIds = new HashSet<Long>(ids.length);

        // create the list of ids to select against
        StringBuilder selection = new StringBuilder();
        selection.append(SongPlayCountColumns.ID);
        selection.append(" IN (");

        // add the first element to handle the separator case for the first element
        uniqueIds.add(ids[0]);
        selection.append(ids[0]);

        for (int i = 1; i < ids.length; i++) {
            // if the new id doesn't exist
            if (uniqueIds.add(ids[i])) {
                // append a separator
                selection.append(",");

                // append the id
                selection.append(ids[i]);

                // for performance reasons, only look at a certain number of songs
                // in case their playlist is ridiculously large
                if (uniqueIds.size() >= MAX_NUMBER_SONGS_TO_ANALYZE) {
                    break;
                }
            }
        }

        // close out the selection
        selection.append(")");

        long[] sortedList = new long[uniqueIds.size()];

        // now query for the songs
        final SQLiteDatabase database = mMusicDatabase.getReadableDatabase();
        Cursor topSongsCursor = null;
        int idx = 0;

        try {
            topSongsCursor = database.query(SongPlayCountColumns.NAME,
                    new String[]{SongPlayCountColumns.ID}, selection.toString(), null, null,
                    null, SongPlayCountColumns.PLAYCOUNTSCORE + " DESC");

            if (topSongsCursor != null && topSongsCursor.moveToFirst()) {
                do {
                    // for each id found, add it to the list and remove it from the unique ids
                    long id = topSongsCursor.getLong(0);
                    sortedList[idx++] = id;
                    uniqueIds.remove(id);
                } while (topSongsCursor.moveToNext());
            }
        } finally {
            if (topSongsCursor != null) {
                topSongsCursor.close();
                topSongsCursor = null;
            }
        }

        // append the remaining items - these are songs that haven't been played recently
        Iterator<Long> iter = uniqueIds.iterator();
        while (iter.hasNext()) {
            sortedList[idx++] = iter.next();
        }

        return sortedList;
    }

    /**
     * This updates all the results for the getTopPlayedResults so that we can get an
     * accurate list of the top played results
     */
    private synchronized void updateResults() {
        if (mDatabaseUpdated) {
            return;
        }

        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();

        database.beginTransaction();

        int oldestWeekWeCareAbout = mNumberOfWeeksSinceEpoch - NUM_WEEKS + 1;
        // delete rows we don't care about anymore
        database.delete(SongPlayCountColumns.NAME, SongPlayCountColumns.LAST_UPDATED_WEEK_INDEX
                + " < " + oldestWeekWeCareAbout, null);

        // get the remaining rows
        Cursor cursor = database.query(SongPlayCountColumns.NAME,
                new String[]{SongPlayCountColumns.ID},
                null, null, null, null, null);

        if (cursor != null && cursor.moveToFirst()) {
            // for each row, update it
            do {
                updateExistingRow(database, cursor.getLong(0), false);
            } while (cursor.moveToNext());

            cursor.close();
            cursor = null;
        }

        mDatabaseUpdated = true;
        database.setTransactionSuccessful();
        database.endTransaction();
    }

    /**
     * @param songId The song Id to remove.
     */
    public void removeItem(final long songId) {
        final SQLiteDatabase database = mMusicDatabase.getWritableDatabase();
        deleteEntry(database, String.valueOf(songId));
    }

    /**
     * Deletes the entry
     *
     * @param database database to use
     * @param stringId id to delete
     */
    private void deleteEntry(final SQLiteDatabase database, final String stringId) {
        database.delete(SongPlayCountColumns.NAME, WHERE_ID_EQUALS, new String[]{stringId});
    }

    public interface SongPlayCountColumns {

        /* Table name */
        String NAME = "songplaycount";

        /* Song IDs column */
        String ID = "songid";

        /* Week Play Count */
        String WEEK_PLAY_COUNT = "week";

        /* Weeks since Epoch */
        String LAST_UPDATED_WEEK_INDEX = "weekindex";

        /* Play count */
        String PLAYCOUNTSCORE = "playcountscore";
    }
}
```

```
package com.naman14.timber.slidinguppanel;

import android.annotation.SuppressLint;
import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.Rect;
import android.graphics.drawable.Drawable;
import android.os.Parcel;
import android.os.Parcelable;
import android.support.v4.content.ContextCompat;
import android.support.v4.view.MotionEventCompat;
import android.support.v4.view.ViewCompat;
import android.util.AttributeSet;
import android.view.Gravity;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.accessibility.AccessibilityEvent;

import com.naman14.timber.R;

public class SlidingUpPanelLayout extends ViewGroup {

    private static final String TAG = SlidingUpPanelLayout.class.getSimpleName();

    /**
     * Default peeking out panel height
     */
    private static final int DEFAULT_PANEL_HEIGHT = 68; // dp;

    /**
     * Default anchor point height
     */
    private static final float DEFAULT_ANCHOR_POINT = 1.0f; // In relative %
    /**
     * Default height of the shadow above the peeking out panel
     */
    private static final int DEFAULT_SHADOW_HEIGHT = 4; // dp;
    /**
     * If no fade color is given by default it will fade to 80% gray.
     */
    private static final int DEFAULT_FADE_COLOR = 0x99000000;
    /**
     * Whether we should hook up the drag view clickable state
     */
    private static final boolean DEFAULT_DRAG_VIEW_CLICKABLE = true;
    /**
     * Default Minimum velocity that will be detected as a fling
     */
    private static final int DEFAULT_MIN_FLING_VELOCITY = 400; // dips per second
    /**
     * Default is set to false because that is how it was written
     */
    private static final boolean DEFAULT_OVERLAY_FLAG = false;
    /**
     * Default attributes for layout
     */
    private static final int[] DEFAULT_ATTRS = new int[]{
            android.R.attr.gravity
    };
    /**
     * Default paralax length of the main view
     */
    private static final int DEFAULT_PARALAX_OFFSET = 0;
    /**
     * Default slide panel offset when collapsed
     */
    private static final int DEFAULT_SLIDE_PANEL_OFFSET = 0;
    /**
     * Default direct offset flag
     */
    private static final boolean DEFAULT_DIRECT_OFFSET_FLAG = false;
    /**
     * Default initial state for the component
     */
    private static SlideState DEFAULT_SLIDE_STATE = SlideState.COLLAPSED;
    /**
     * The paint used to dim the main layout when sliding
     */
    private final Paint mCoveredFadePaint = new Paint();
    /**
     * Drawable used to draw the shadow between panes.
     */
    private final Drawable mShadowDrawable;
    private final ViewDragHelper mDragHelper;
    private final Rect mTmpRect = new Rect();
    /**
     * Minimum velocity that will be detected as a fling
     */
    private int mMinFlingVelocity = DEFAULT_MIN_FLING_VELOCITY;
    /**
     * The fade color used for the panel covered by the slider. 0 = no fading.
     */
    private int mCoveredFadeColor = DEFAULT_FADE_COLOR;
    /**
     * The size of the overhang in pixels.
     */
    private int mPanelHeight = -1;
    /**
     * Determines how much to slide the panel off when expanded
     */
    private int mSlidePanelOffset = 0;
    /**
     * The size of the shadow in pixels.
     */
    private int mShadowHeight = -1;
    /**
     * Paralax offset
     */
    private int mParallaxOffset = -1;
    /**
     * Clamps the Main view to the slideable view
     */
    private boolean mDirectOffset = false;
    /**
     * True if the collapsed panel should be dragged up.
     */
    private boolean mIsSlidingUp;
    /**
     * Panel overlays the windows instead of putting it underneath it.
     */
    private boolean mOverlayContent = DEFAULT_OVERLAY_FLAG;
    /**
     * If provided, the panel can be dragged by only this view. Otherwise, the entire panel can be
     * used for dragging.
     */
    private View mDragView;
    /**
     * If provided, the panel can be dragged by only this view. Otherwise, the entire panel can be
     * used for dragging.
     */
    private int mDragViewResId = -1;
    /**
     * Whether clicking on the drag view will expand/collapse
     */
    private boolean mDragViewClickable = DEFAULT_DRAG_VIEW_CLICKABLE;
    /**
```

```
     * The child view that can slide, if any.
     */
    private View mSlideableView;
    /**
     * The main view
     */
    private View mMainView;
    /**
     * The background view
     */
    private View mBackgroundView;
    private SlideState mSlideState = SlideState.COLLAPSED;
    /**
     * How far the panel is offset from its expanded position.
     * range [0, 1] where 0 = collapsed, 1 = expanded.
     */
    private float mSlideOffset;
    /**
     * How far in pixels the slideable panel may move.
     */
    private int mSlideRange;
    /**
     * A panel view is locked into internal scrolling or another condition that
     * is preventing a drag.
     */
    private boolean mIsUnableToDrag;
    /**
     * Flag indicating that sliding feature is enabled\disabled
     */
    private boolean mIsSlidingEnabled;
    /**
     * Flag indicating if a drag view can have its own touch events.  If set
     * to true, a drag view can scroll horizontally and have its own click listener.
     * <p/>
     * Default is set to false.
     */
    private boolean mIsUsingDragViewTouchEvents;
    private float mInitialMotionX;
    private float mInitialMotionY;
    private float mAnchorPoint = 1.f;
    private PanelSlideListener mPanelSlideListener;
    /**
     * Stores whether or not the pane was expanded the last time it was slideable.
     * If expand/collapse operations are invoked this state is modified. Used by
     * instance state save/restore.
     */
    private boolean mFirstLayout = true;

    public SlidingUpPanelLayout(Context context) {
        this(context, null);
    }

    public SlidingUpPanelLayout(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public SlidingUpPanelLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);

        if (isInEditMode()) {
            mShadowDrawable = null;
            mDragHelper = null;
            return;
        }

        if (attrs != null) {
            TypedArray defAttrs = context.obtainStyledAttributes(attrs, DEFAULT_ATTRS);

            if (defAttrs != null) {
                int gravity = defAttrs.getInt(0, Gravity.NO_GRAVITY);
                if (gravity != Gravity.TOP && gravity != Gravity.BOTTOM) {
                    throw new IllegalArgumentException("gravity must be set to either top or bottom");
                }
                mIsSlidingUp = gravity == Gravity.BOTTOM;
            }

            defAttrs.recycle();

            TypedArray ta = context.obtainStyledAttributes(attrs, R.styleable.SlidingUpPanelLayout);

            if (ta != null) {
                mPanelHeight = ta.getDimensionPixelSize(R.styleable.SlidingUpPanelLayout_panelHeight, -1);
                mSlidePanelOffset = ta.getDimensionPixelSize(R.styleable.SlidingUpPanelLayout_slidePanelOffset, DEFAULT_SLID
                mShadowHeight = ta.getDimensionPixelSize(R.styleable.SlidingUpPanelLayout_shadowHeight, -1);
                mParallaxOffset = ta.getDimensionPixelSize(R.styleable.SlidingUpPanelLayout_paralaxOffset, -1);
                mDirectOffset = ta.getBoolean(R.styleable.SlidingUpPanelLayout_directOffset, DEFAULT_DIRECT_OFFSET_FLAG);

                mMinFlingVelocity = ta.getInt(R.styleable.SlidingUpPanelLayout_flingVelocity, DEFAULT_MIN_FLING_VELOCITY);
                mCoveredFadeColor = ta.getColor(R.styleable.SlidingUpPanelLayout_fadeColor, DEFAULT_FADE_COLOR);

                mDragViewResId = ta.getResourceId(R.styleable.SlidingUpPanelLayout_dragView, -1);
                mDragViewClickable = ta.getBoolean(R.styleable.SlidingUpPanelLayout_dragViewClickable, DEFAULT_DRAG_VIEW_CLI

                mOverlayContent = ta.getBoolean(R.styleable.SlidingUpPanelLayout_overlay, DEFAULT_OVERLAY_FLAG);

                mAnchorPoint = ta.getFloat(R.styleable.SlidingUpPanelLayout_anchorPoint, DEFAULT_ANCHOR_POINT);

                mSlideState = SlideState.values()[ta.getInt(R.styleable.SlidingUpPanelLayout_initialState, DEFAULT_SLIDE_STA
            }

            ta.recycle();
        }

        final float density = context.getResources().getDisplayMetrics().density;
        if (mPanelHeight == -1) {
            mPanelHeight = (int) (DEFAULT_PANEL_HEIGHT * density + 0.5f);
        }
        if (mShadowHeight == -1) {
            mShadowHeight = (int) (DEFAULT_SHADOW_HEIGHT * density + 0.5f);
        }
        if (mParallaxOffset == -1) {
            mParallaxOffset = (int) (DEFAULT_PARALAX_OFFSET * density);
        }
        // If the shadow height is zero, don't show the shadow
        if (mShadowHeight > 0) {
            if (mIsSlidingUp) {
                mShadowDrawable = ContextCompat.getDrawable(context, R.drawable.above_shadow);
            } else {
                mShadowDrawable = ContextCompat.getDrawable(context, R.drawable.below_shadow);
            }

        } else {
            mShadowDrawable = null;
        }

        setWillNotDraw(false);

        mDragHelper = ViewDragHelper.create(this, 0.5f, new DragHelperCallback());
        mDragHelper.setMinVelocity(mMinFlingVelocity * density);

        mIsSlidingEnabled = true;
    }

    private static boolean hasOpaqueBackground(View v) {
        final Drawable bg = v.getBackground();
        return bg != null && bg.getOpacity() == PixelFormat.OPAQUE;
    }

    /**
     * Set the Drag View after the view is inflated
     */
    @Override
    protected void onFinishInflate() {
        super.onFinishInflate();
        if (mDragViewResId != -1) {
            setDragView(findViewById(mDragViewResId));
        }
    }

    /**
     * @return The ARGB-packed color value used to fade the fixed pane
     */
    public int getCoveredFadeColor() {
        return mCoveredFadeColor;
    }

    /**
     * Set the color used to fade the pane covered by the sliding pane out when the pane
     * will become fully covered in the expanded state.
     *
     * @param color An ARGB-packed color value
     */
    public void setCoveredFadeColor(int color) {
        mCoveredFadeColor = color;
        invalidate();
    }

    public boolean isSlidingEnabled() {
        return mIsSlidingEnabled && mSlideableView != null;
    }

    /**
     * Set sliding enabled flag
     *
     * @param enabled flag value
     */
    public void setSlidingEnabled(boolean enabled) {
        mIsSlidingEnabled = enabled;
    }

    /**
     * @return The current collapsed panel height
     */
    public int getPanelHeight() {
        return mPanelHeight;
    }

    /**
     * Set the collapsed panel height in pixels
     *
     * @param val A height in pixels
     */
    public void setPanelHeight(int val) {
        mPanelHeight = val;
        requestLayout();
    }

    /**
     * Sets the panel offset when collapsed so you can exit
     * the boundaries of the top of the screen
     *
     * @param val Offset in pixels
     */
    public void setSlidePanelOffset(int val) {
        mSlidePanelOffset = val;
        requestLayout();
    }

    /**
     * @return The current paralax offset
```

```java
     */
    public int getCurrentParalaxOffset() {
        if (mParallaxOffset < 0) {
            return 0;
        }

        return (int) (mParallaxOffset * getDirectionalSlideOffset());
    }

    /**
     * @return The directional slide offset
     */
    protected float getDirectionalSlideOffset() {
        return mIsSlidingUp ? -mSlideOffset : mSlideOffset;
    }

    /**
     * Sets the panel slide listener
     *
     * @param listener
     */
    public void setPanelSlideListener(PanelSlideListener listener) {
        mPanelSlideListener = listener;
    }

    /**
     * Set the draggable view portion. Use to null, to allow the whole panel to be draggable
     *
     * @param dragView A view that will be used to drag the panel.
     */
    public void setDragView(View dragView) {
        if (mDragView != null && mDragViewClickable) {
            mDragView.setOnClickListener(null);
        }
        mDragView = dragView;
        if (mDragView != null) {
            mDragView.setClickable(true);
            mDragView.setFocusable(false);
            mDragView.setFocusableInTouchMode(false);
            if (mDragViewClickable) {
                mDragView.setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        if (!isEnabled()) return;
                        if (!isPanelExpanded() && !isPanelAnchored()) {
                            expandPanel(mAnchorPoint);
                        } else {
                            collapsePanel();
                        }
                    }
                });
            }
        }
    }

    /**
     * Gets the currently set anchor point
     *
     * @return the currently set anchor point
     */
    public float getAnchorPoint() {
        return mAnchorPoint;
    }

    /**
     * Set an anchor point where the panel can stop during sliding
     *
     * @param anchorPoint A value between 0 and 1, determining the position of the anchor point
     *                    starting from the top of the layout.
     */
    public void setAnchorPoint(float anchorPoint) {
        if (anchorPoint > 0 && anchorPoint <= 1) {
            mAnchorPoint = anchorPoint;
        }
    }

    /**
     * Check if the panel is set as an overlay.
     */
    public boolean isOverlayed() {
        return mOverlayContent;
    }

    /**
     * Sets whether or not the panel overlays the content
     *
     * @param overlayed
     */
    public void setOverlayed(boolean overlayed) {
        mOverlayContent = overlayed;
    }

    void dispatchOnPanelSlide(View panel) {
        if (mPanelSlideListener != null) {
            mPanelSlideListener.onPanelSlide(panel, mSlideOffset);
        }
    }

    void dispatchOnPanelExpanded(View panel) {
        if (mPanelSlideListener != null) {
            mPanelSlideListener.onPanelExpanded(panel);
        }
        sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
    }

    void dispatchOnPanelCollapsed(View panel) {
        if (mPanelSlideListener != null) {
            mPanelSlideListener.onPanelCollapsed(panel);
        }
        sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
    }

    void dispatchOnPanelAnchored(View panel) {
        if (mPanelSlideListener != null) {
            mPanelSlideListener.onPanelAnchored(panel);
        }
        sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
    }

    void dispatchOnPanelHidden(View panel) {
        if (mPanelSlideListener != null) {
            mPanelSlideListener.onPanelHidden(panel);
        }
        sendAccessibilityEvent(AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED);
    }

    void updateObscuredViewVisibility() {
        if (getChildCount() == 0) {
            return;
        }
        final int leftBound = getPaddingLeft();
        final int rightBound = getWidth() - getPaddingRight();
        final int topBound = getPaddingTop();
        final int bottomBound = getHeight() - getPaddingBottom();
        final int left;
        final int right;
        final int top;
        final int bottom;
        if (mSlideableView != null && hasOpaqueBackground(mSlideableView)) {
            left = mSlideableView.getLeft();
            right = mSlideableView.getRight();
            top = mSlideableView.getTop();
            bottom = mSlideableView.getBottom();
        } else {
            left = right = top = bottom = 0;
        }
        View child = mMainView;
        final int clampedChildLeft = Math.max(leftBound, child.getLeft());
        final int clampedChildTop = Math.max(topBound, child.getTop());
        final int clampedChildRight = Math.min(rightBound, child.getRight());
        final int clampedChildBottom = Math.min(bottomBound, child.getBottom());
        final int vis;
        if (clampedChildLeft >= left && clampedChildTop >= top &&
                clampedChildRight <= right && clampedChildBottom <= bottom) {
            vis = INVISIBLE;
        } else {
            vis = VISIBLE;
        }
        child.setVisibility(vis);
    }

    void setAllChildrenVisible() {
        for (int i = 0, childCount = getChildCount(); i < childCount; i++) {
            final View child = getChildAt(i);
            if (child.getVisibility() == INVISIBLE) {
                child.setVisibility(VISIBLE);
            }
        }
    }

    @Override
    protected void onAttachedToWindow() {
        super.onAttachedToWindow();
        mFirstLayout = true;
    }

    @Override
    protected void onDetachedFromWindow() {
        super.onDetachedFromWindow();
        mFirstLayout = true;
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        final int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        final int widthSize = MeasureSpec.getSize(widthMeasureSpec);
        final int heightMode = MeasureSpec.getMode(heightMeasureSpec);
        final int heightSize = MeasureSpec.getSize(heightMeasureSpec);

        if (widthMode != MeasureSpec.EXACTLY) {
            throw new IllegalStateException("Width must have an exact value or MATCH_PARENT");
        } else if (heightMode != MeasureSpec.EXACTLY) {
            throw new IllegalStateException("Height must have an exact value or MATCH_PARENT");
        }

        final int childCount = getChildCount();

        if (childCount != 2 && childCount != 3) {
            throw new IllegalStateException("Sliding up panel layout must have exactly 2 or 3 children!");
        }

        if (childCount == 2) {
            mMainView = getChildAt(0);
            mSlideableView = getChildAt(1);
        } else {
            mBackgroundView = getChildAt(0);
            mMainView = getChildAt(1);
            mSlideableView = getChildAt(2);
        }
        if (mDragView == null) {
            setDragView(mSlideableView);
        }

        // If the sliding panel is not visible, then put the whole view in the hidden state
        if (mSlideableView.getVisibility() == GONE) {
            mSlideState = SlideState.HIDDEN;
        }

        int layoutHeight = heightSize - getPaddingTop() - getPaddingBottom();

        // First pass. Measure based on child LayoutParams width/height.
        for (int i = 0; i < childCount; i++) {
            final View child = getChildAt(i);
            final LayoutParams lp = (LayoutParams) child.getLayoutParams();

            // We always measure the sliding panel in order to know it's height (needed for show panel)
            if (child.getVisibility() == GONE && child == mMainView) {
                continue;
            }

            int height = layoutHeight;
            if (child == mMainView && !mOverlayContent && mSlideState != SlideState.HIDDEN) {
                height -= mPanelHeight;
            }

            int childWidthSpec;
            if (lp.width == LayoutParams.WRAP_CONTENT) {
                childWidthSpec = MeasureSpec.makeMeasureSpec(widthSize, MeasureSpec.AT_MOST);
            } else if (lp.width == LayoutParams.MATCH_PARENT) {
                childWidthSpec = MeasureSpec.makeMeasureSpec(widthSize, MeasureSpec.EXACTLY);
            } else {
                childWidthSpec = MeasureSpec.makeMeasureSpec(lp.width, MeasureSpec.EXACTLY);
            }

            int childHeightSpec;
            if (lp.height == LayoutParams.WRAP_CONTENT) {
                childHeightSpec = MeasureSpec.makeMeasureSpec(height, MeasureSpec.AT_MOST);
            } else if (lp.height == LayoutParams.MATCH_PARENT) {
                childHeightSpec = MeasureSpec.makeMeasureSpec(height, MeasureSpec.EXACTLY);
            } else {
                childHeightSpec = MeasureSpec.makeMeasureSpec(lp.height, MeasureSpec.EXACTLY);
            }

            if (child == mSlideableView) {
                mSlideRange = MeasureSpec.getSize(childHeightSpec) - mPanelHeight + mSlidePanelOffset;
                childHeightSpec += mSlidePanelOffset;
            }

            child.measure(childWidthSpec, childHeightSpec);
        }

        setMeasuredDimension(widthSize, heightSize);
    }

    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        final int paddingLeft = getPaddingLeft();
        final int paddingTop = getPaddingTop();

        final int childCount = getChildCount();

        if (mFirstLayout) {
            switch (mSlideState) {
                case EXPANDED:
                    mSlideOffset = 1.0f;
                    break;
                case ANCHORED:
                    mSlideOffset = mAnchorPoint;
                    break;
                case HIDDEN:
                    int newTop = computePanelTopPosition(0.0f) + (mIsSlidingUp ? +mPanelHeight : -mPanelHeight);
                    mSlideOffset = computeSlideOffset(newTop);
                    break;
                default:
                    mSlideOffset = 0.f;
                    break;
            }
        }

        for (int i = 0; i < childCount; i++) {
            final View child = getChildAt(i);

            // Always layout the sliding view on the first layout
            if (child.getVisibility() == GONE && (child == mMainView || mFirstLayout)) {
                continue;
            }

            final int childHeight = child.getMeasuredHeight();
            int childTop = paddingTop;

            if (child == mSlideableView) {
                childTop = computePanelTopPosition(mSlideOffset);
            }

            if (!mIsSlidingUp) {
                if (child == mMainView && !mOverlayContent) {
                    childTop = computePanelTopPosition(mSlideOffset) + mSlideableView.getMeasuredHeight();
                }
            }
            final int childBottom = childTop + childHeight;
            final int childLeft = paddingLeft;
            final int childRight = childLeft + child.getMeasuredWidth();

            child.layout(childLeft, childTop, childRight, childBottom);
        }

        if (mFirstLayout) {
            updateObscuredViewVisibility();
        }

        mFirstLayout = false;
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        super.onSizeChanged(w, h, oldw, oldh);
        // Recalculate sliding panes and their details
        if (h != oldh) {
            mFirstLayout = true;
        }
    }

    /**
     * Set if the drag view can have its own touch events.  If set
     * to true, a drag view can scroll horizontally and have its own click listener.
     * <p/>
     * Default is set to false.
     */
    public void setEnableDragViewTouchEvents(boolean enabled) {
        mIsUsingDragViewTouchEvents = enabled;
    }

    @Override
    public void setEnabled(boolean enabled) {
        if (!enabled) {
            collapsePanel();
        }
        super.setEnabled(enabled);
    }
    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        final int action = MotionEventCompat.getActionMasked(ev);


        if (!isEnabled() || !mIsSlidingEnabled || (mIsUnableToDrag && action != MotionEvent.ACTION_DOWN)) {
            mDragHelper.cancel();
            return super.onInterceptTouchEvent(ev);
        }

        if (action == MotionEvent.ACTION_CANCEL || action == MotionEvent.ACTION_UP) {
            mDragHelper.cancel();
            return false;
        }

        final float x = ev.getX();
        final float y = ev.getY();

        switch (action) {
            case MotionEvent.ACTION_DOWN: {
                mIsUnableToDrag = false;
                mInitialMotionX = x;
                mInitialMotionY = y;
                break;
            }

            case MotionEvent.ACTION_MOVE: {
                final float adx = Math.abs(x - mInitialMotionX);
                final float ady = Math.abs(y - mInitialMotionY);
                final int dragSlop = mDragHelper.getTouchSlop();

                // Handle any horizontal scrolling on the drag view.
                if (mIsUsingDragViewTouchEvents && adx > dragSlop && ady < dragSlop) {
                    return super.onInterceptTouchEvent(ev);
                }

                if ((ady > dragSlop && adx > ady) || !isDragViewUnder((int) mInitialMotionX, (int) mInitialMotionY)) {
                    mDragHelper.cancel();
                    mIsUnableToDrag = true;
                    return false;
                }
                break;
            }
        }

        return mDragHelper.shouldInterceptTouchEvent(ev);
    }

    @Override
    public boolean onTouchEvent(MotionEvent ev) {
        if (!isSlidingEnabled()) {
            return super.onTouchEvent(ev);
        }
        mDragHelper.processTouchEvent(ev);
        return true;
    }

    private boolean isDragViewUnder(int x, int y) {
        if (mDragView == null) return false;
        int[] viewLocation = new int[2];
        mDragView.getLocationOnScreen(viewLocation);
        int[] parentLocation = new int[2];
        this.getLocationOnScreen(parentLocation);
        int screenX = parentLocation[0] + x;
        int screenY = parentLocation[1] + y;
        return screenX >= viewLocation[0] && screenX < viewLocation[0] + mDragView.getWidth() &&
                screenY >= viewLocation[1] && screenY < viewLocation[1] + mDragView.getHeight();
    }
    private boolean expandPanel(View pane, int initialVelocity, float mSlideOffset) {
        return mFirstLayout || smoothSlideTo(mSlideOffset, initialVelocity);
    }

    private boolean collapsePanel(View pane, int initialVelocity) {
        return mFirstLayout || smoothSlideTo(0.0f, initialVelocity);
    }

    /*
     * Computes the top position of the panel based on the slide offset.
     */
    private int computePanelTopPosition(float slideOffset) {
        int slidingViewHeight = mSlideableView != null ? mSlideableView.getMeasuredHeight() : 0;
        int slidePixelOffset = (int) (slideOffset * mSlideRange);
        // Compute the top of the panel if its collapsed
        return mIsSlidingUp
                ? getMeasuredHeight() - getPaddingBottom() - mPanelHeight - slidePixelOffset
                : getPaddingTop() - slidingViewHeight + mPanelHeight + slidePixelOffset;
    }

    /*
     * Computes the slide offset based on the top position of the panel
     */
    private float computeSlideOffset(int topPosition) {
        // Compute the panel top position if the panel is collapsed (offset 0)
        final int topBoundCollapsed = computePanelTopPosition(0);

        // Determine the new slide offset based on the collapsed top position and the new required
        // top position
        return (mIsSlidingUp
                ? (float) (topBoundCollapsed - topPosition) / mSlideRange
                : (float) (topPosition - topBoundCollapsed) / mSlideRange);
    }

    /**
     * Collapse the sliding pane if it is currently slideable. If first layout
     * has already completed this will animate.
     *
     * @return true if the pane was slideable and is now collapsed/in the process of collapsing
     */
    public boolean collapsePanel() {
        if (mFirstLayout) {
            mSlideState = SlideState.COLLAPSED;
            return true;
        } else {
            if (mSlideState == SlideState.HIDDEN || mSlideState == SlideState.COLLAPSED)
                return false;
            return collapsePanel(mSlideableView, 0);
        }
    }

    /**
     * Expand the sliding pane if it is currently slideable.
     *
     * @return true if the pane was slideable and is now expanded/in the process of expading
     */
    public boolean expandPanel() {
        if (mFirstLayout) {
            mSlideState = SlideState.EXPANDED;
            return true;
        } else {
            return expandPanel(1.0f);
        }
    }

    /**
     * Expand the sliding pane to the anchor point if it is currently slideable.
     *
     * @return true if the pane was slideable and is now expanded/in the process of expading
     */
    public boolean anchorPanel() {
        if (mFirstLayout) {
            mSlideState = SlideState.ANCHORED;
            return true;
        } else {
            return expandPanel(mAnchorPoint);
        }
    }

    /**
     * Partially expand the sliding panel up to a specific offset
     *
     * @param mSlideOffset Value between 0 and 1, where 0 is completely expanded.
     * @return true if the pane was slideable and is now expanded/in the process of expanding
     */
    public boolean expandPanel(float mSlideOffset) {
        if (mSlideableView == null || mSlideState == SlideState.EXPANDED) return false;
        mSlideableView.setVisibility(View.VISIBLE);
        return expandPanel(mSlideableView, 0, mSlideOffset);
    }

    /**
     * Check if the sliding panel in this layout is fully expanded.
     *
     * @return true if sliding panel is completely expanded
     */
    public boolean isPanelExpanded() {
        return mSlideState == SlideState.EXPANDED;
    }

    /**
     * Check if the sliding panel in this layout is anchored.
     *
     * @return true if sliding panel is anchored
     */
    public boolean isPanelAnchored() {
        return mSlideState == SlideState.ANCHORED;
    }

    /**
     * Check if the sliding panel in this layout is currently visible.
     *
     * @return true if the sliding panel is visible.
     */
    public boolean isPanelHidden() {
        return mSlideState == SlideState.HIDDEN;
    }

    /**
     * Shows the panel from the hidden state
     */
    public void showPanel() {
        if (mFirstLayout) {
            mSlideState = SlideState.COLLAPSED;
        } else {
            if (mSlideableView == null || mSlideState != SlideState.HIDDEN) return;
            mSlideableView.setVisibility(View.VISIBLE);
            requestLayout();
            smoothSlideTo(0, 0);
        }
    }

    /**
     * Hides the sliding panel entirely.
     */
    public void hidePanel() {
        if (mFirstLayout) {
            mSlideState = SlideState.HIDDEN;
        } else {
            if (mSlideState == SlideState.DRAGGING || mSlideState == SlideState.HIDDEN) return;
            int newTop = computePanelTopPosition(0.0f) + (mIsSlidingUp ? +mPanelHeight : -mPanelHeight);
            smoothSlideTo(computeSlideOffset(newTop), 0);
        }
    }

    @SuppressLint("NewApi")
    private void onPanelDragged(int newTop) {
        mSlideState = SlideState.DRAGGING;
        // Recompute the slide offset based on the new top position
        mSlideOffset = computeSlideOffset(newTop);
        // Update the parallax based on the new slide offset
        if ((mParallaxOffset > 0 || mDirectOffset) && mSlideOffset >= 0) {
            int mainViewOffset = 0;
            if (mParallaxOffset > 0) {
                mainViewOffset = getCurrentParalaxOffset();
            } else {
                mainViewOffset = (int) (getDirectionalSlideOffset() * mSlideRange);
            }

            mMainView.setTranslationY(mainViewOffset);
        }

        // Dispatch the slide event
        dispatchOnPanelSlide(mSlideableView);
        // If the slide offset is negative, and overlay is not on, we need to increase the
        // height of the main content
        if (mSlideOffset <= 0 && !mOverlayContent) {
            // expand the main view
            LayoutParams lp = (LayoutParams) mMainView.getLayoutParams();
            lp.height = mIsSlidingUp ? (newTop - getPaddingBottom()) : (getHeight() - getPaddingBottom() - mSlideableView.ge
            mMainView.requestLayout();
        }
    }

    @Override
    protected boolean drawChild(Canvas canvas, View child, long drawingTime) {
        boolean result;
        final int save = canvas.save(Canvas.CLIP_SAVE_FLAG);

        if (isSlidingEnabled() && mMainView == child) {
            // Clip against the slider; no sense drawing what will immediately be covered,
            // Unless the panel is set to overlay content
            if (!mOverlayContent) {
                canvas.getClipBounds(mTmpRect);
                if (mIsSlidingUp) {
                    mTmpRect.bottom = Math.min(mTmpRect.bottom, mSlideableView.getTop());
                } else {
                    mTmpRect.top = Math.max(mTmpRect.top, mSlideableView.getBottom());
                }
                canvas.clipRect(mTmpRect);
            }
        }

        result = super.drawChild(canvas, child, drawingTime);
        canvas.restoreToCount(save);

        if (mCoveredFadeColor != 0 && mSlideOffset > 0) {
            final int baseAlpha = (mCoveredFadeColor & 0xff000000) >>> 24;
            final int imag = (int) (baseAlpha * mSlideOffset);
            final int color = imag << 24 | (mCoveredFadeColor & 0xffffff);
            mCoveredFadePaint.setColor(color);
            canvas.drawRect(mTmpRect, mCoveredFadePaint);
        }

        return result;
    }

    /**
     * Smoothly animate mDraggingPane to the target X position within its range.
     *
     * @param slideOffset position to animate to
     * @param velocity    initial velocity in case of fling, or 0.
     */
    boolean smoothSlideTo(float slideOffset, int velocity) {
        if (!isSlidingEnabled()) {
            // Nothing to do.
            return false;
        }

        int panelTop = computePanelTopPosition(slideOffset);
        if (mDragHelper.smoothSlideViewTo(mSlideableView, mSlideableView.getLeft(), panelTop)) {
            setAllChildrenVisible();
            ViewCompat.postInvalidateOnAnimation(this);
            return true;
        }
        return false;
    }

    @Override
    public void computeScroll() {
        if (mDragHelper != null && mDragHelper.continueSettling(true)) {
            if (!isSlidingEnabled()) {
                mDragHelper.abort();
                return;
            }

            ViewCompat.postInvalidateOnAnimation(this);
        }
    }

    @Override
    public void draw(Canvas c) {
        super.draw(c);

        if (!isSlidingEnabled()) {
            // No need to draw a shadow if we don't have one.
            return;
        }

        final int right = mSlideableView.getRight();
        final int top;
        final int bottom;
        if (mIsSlidingUp) {
            top = mSlideableView.getTop() - mShadowHeight;
            bottom = mSlideableView.getTop();
        } else {
            top = mSlideableView.getBottom();
            bottom = mSlideableView.getBottom() + mShadowHeight;
        }
        final int left = mSlideableView.getLeft();

        if (mShadowDrawable != null) {
            mShadowDrawable.setBounds(left, top, right, bottom);
            mShadowDrawable.draw(c);
        }
    }

    /**
     * Tests scrollability within child views of v given a delta of dx.
     *
     * @param v      View to test for horizontal scrollability
     * @param checkV Whether the view v passed should itself be checked for scrollability (true),
     *               or just its children (false).
     * @param dx     Delta scrolled in pixels
     * @param x      X coordinate of the active touch point
     * @param y      Y coordinate of the active touch point
     * @return true if child views of v can be scrolled by delta of dx.
     */
    protected boolean canScroll(View v, boolean checkV, int dx, int x, int y) {
        if (v instanceof ViewGroup) {
            final ViewGroup group = (ViewGroup) v;
            final int scrollX = v.getScrollX();
            final int scrollY = v.getScrollY();
            final int count = group.getChildCount();
            // Count backwards - let topmost views consume scroll distance first.
            for (int i = count - 1; i >= 0; i--) {
                final View child = group.getChildAt(i);
                if (x + scrollX >= child.getLeft() && x + scrollX < child.getRight() &&
                        y + scrollY >= child.getTop() && y + scrollY < child.getBottom() &&
                        canScroll(child, true, dx, x + scrollX - child.getLeft(),
                                y + scrollY - child.getTop())) {
                    return true;
                }
            }
        }
        return checkV && ViewCompat.canScrollHorizontally(v, -dx);
    }

    @Override
    protected ViewGroup.LayoutParams generateDefaultLayoutParams() {
        return new LayoutParams();
    }

    @Override
    protected ViewGroup.LayoutParams generateLayoutParams(ViewGroup.LayoutParams p) {
        return p instanceof MarginLayoutParams
                ? new LayoutParams((MarginLayoutParams) p)
                : new LayoutParams(p);
    }

    @Override
    protected boolean checkLayoutParams(ViewGroup.LayoutParams p) {
        return p instanceof LayoutParams && super.checkLayoutParams(p);
    }

    @Override
    public ViewGroup.LayoutParams generateLayoutParams(AttributeSet attrs) {
        return new LayoutParams(getContext(), attrs);
    }

    @Override
    public Parcelable onSaveInstanceState() {
        Parcelable superState = super.onSaveInstanceState();

        SavedState ss = new SavedState(superState);
        ss.mSlideState = mSlideState;

        return ss;
    }

    @Override
    public void onRestoreInstanceState(Parcelable state) {
        SavedState ss = (SavedState) state;
        super.onRestoreInstanceState(ss.getSuperState());
        mSlideState = ss.mSlideState;
    }

    /**
     * Current state of the slideable view.
     */
    private enum SlideState {
        EXPANDED,
        COLLAPSED,
        ANCHORED,
        HIDDEN,
        DRAGGING
    }

    /**
     * Listener for monitoring events about sliding panes.
     */
    public interface PanelSlideListener {
        /**
         * Called when a sliding pane's position changes.
         *
         * @param panel       The child view that was moved
         * @param slideOffset The new offset of this sliding pane within its range, from 0-1
         */
        void onPanelSlide(View panel, float slideOffset);

        /**
         * Called when a sliding panel becomes slid completely collapsed.
         *
         * @param panel The child view that was slid to an collapsed position
         */
        void onPanelCollapsed(View panel);

        /**
         * Called when a sliding panel becomes slid completely expanded.
         *
         * @param panel The child view that was slid to a expanded position
         */
        void onPanelExpanded(View panel);

        /**
         * Called when a sliding panel becomes anchored.
         *
         * @param panel The child view that was slid to a anchored position
         */
        void onPanelAnchored(View panel);

        /**
         * Called when a sliding panel becomes completely hidden.
         *
         * @param panel The child view that was slid to a hidden position
         */
        void onPanelHidden(View panel);
    }

    /**
     * No-op stubs for {@link PanelSlideListener}. If you only want to implement a subset
     * of the listener methods you can extend this instead of implement the full interface.
     */
    public static class SimplePanelSlideListener implements PanelSlideListener {
        @Override
        public void onPanelSlide(View panel, float slideOffset) {
        }

        @Override
        public void onPanelCollapsed(View panel) {
        }

        @Override
        public void onPanelExpanded(View panel) {
        }

        @Override
        public void onPanelAnchored(View panel) {
        }

        @Override
        public void onPanelHidden(View panel) {
        }
    }

    public static class LayoutParams extends MarginLayoutParams {
        private static final int[] ATTRS = new int[]{
                android.R.attr.layout_weight
        };
        public LayoutParams() {
            super(MATCH_PARENT, MATCH_PARENT);
        }

        public LayoutParams(int width, int height) {
            super(width, height);
        }

        public LayoutParams(ViewGroup.LayoutParams source) {
            super(source);
        }

        public LayoutParams(MarginLayoutParams source) {
            super(source);
        }

        public LayoutParams(LayoutParams source) {
            super(source);
        }

        public LayoutParams(Context c, AttributeSet attrs) {
            super(c, attrs);

            final TypedArray a = c.obtainStyledAttributes(attrs, ATTRS);
            a.recycle();
        }

    }

    static class SavedState extends BaseSavedState {
        public static final Creator<SavedState> CREATOR =
                new Creator<SavedState>() {
                    @Override
                    public SavedState createFromParcel(Parcel in) {
                        return new SavedState(in);
                    }

                    @Override
                    public SavedState[] newArray(int size) {
                        return new SavedState[size];
                    }
                };
        SlideState mSlideState;

        SavedState(Parcelable superState) {
            super(superState);
        }

        private SavedState(Parcel in) {
            super(in);
            try {
                mSlideState = Enum.valueOf(SlideState.class, in.readString());
            } catch (IllegalArgumentException e) {
                mSlideState = SlideState.COLLAPSED;
            }
        }

        @Override
        public void writeToParcel(Parcel out, int flags) {
            super.writeToParcel(out, flags);
            out.writeString(mSlideState.toString());
        }
    }

    private class DragHelperCallback extends ViewDragHelper.Callback {

        @Override
        public boolean tryCaptureView(View child, int pointerId) {
            if (mIsUnableToDrag) {
                return false;
```

```java
            }

            return child == mSlideableView;
        }

        @Override
        public void onViewDragStateChanged(int state) {
            if (mDragHelper.getViewDragState() == ViewDragHelper.STATE_IDLE) {
                mSlideOffset = computeSlideOffset(mSlideableView.getTop());

                if (mSlideOffset == 1) {
                    if (mSlideState != SlideState.EXPANDED) {
                        updateObscuredViewVisibility();
                        mSlideState = SlideState.EXPANDED;
                        dispatchOnPanelExpanded(mSlideableView);
                    }
                } else if (mSlideOffset == 0) {
                    if (mSlideState != SlideState.COLLAPSED) {
                        mSlideState = SlideState.COLLAPSED;
                        dispatchOnPanelCollapsed(mSlideableView);
                    }
                } else if (mSlideOffset < 0) {
                    mSlideState = SlideState.HIDDEN;
                    mSlideableView.setVisibility(View.GONE);
                    dispatchOnPanelHidden(mSlideableView);
                } else if (mSlideState != SlideState.ANCHORED) {
                    updateObscuredViewVisibility();
                    mSlideState = SlideState.ANCHORED;
                    dispatchOnPanelAnchored(mSlideableView);
                }
            }
        }

        @Override
        public void onViewCaptured(View capturedChild, int activePointerId) {
            setAllChildrenVisible();
        }

        @Override
        public void onViewPositionChanged(View changedView, int left, int top, int dx, int dy) {
            onPanelDragged(top);
            invalidate();
        }

        @Override
        public void onViewReleased(View releasedChild, float xvel, float yvel) {
            int target = 0;

            // direction is always positive if we are sliding in the expanded direction
            float direction = mIsSlidingUp ? -yvel : yvel;

            if (direction > 0) {
                // swipe up -> expand
                target = computePanelTopPosition(1.0f);
            } else if (direction < 0) {
                // swipe down -> collapse
                target = computePanelTopPosition(0.0f);
            } else if (mAnchorPoint != 1 && mSlideOffset >= (1.f + mAnchorPoint) / 2) {
                // zero velocity, and far enough from anchor point => expand to the top
                target = computePanelTopPosition(1.0f);
            } else if (mAnchorPoint == 1 && mSlideOffset >= 0.5f) {
                // zero velocity, and far enough from anchor point => expand to the top
                target = computePanelTopPosition(1.0f);
            } else if (mAnchorPoint != 1 && mSlideOffset >= mAnchorPoint) {
                target = computePanelTopPosition(mAnchorPoint);
            } else if (mAnchorPoint != 1 && mSlideOffset >= mAnchorPoint / 2) {
                target = computePanelTopPosition(mAnchorPoint);
            } else {
                // settle at the bottom
                target = computePanelTopPosition(0.0f);
            }

            mDragHelper.settleCapturedViewAt(releasedChild.getLeft(), target);
            invalidate();
        }

        @Override
        public int getViewVerticalDragRange(View child) {
            return mSlideRange;
        }

        @Override
        public int clampViewPositionVertical(View child, int top, int dy) {
            final int collapsedTop = computePanelTopPosition(0.f);
            final int expandedTop = computePanelTopPosition(1.0f);
            if (mIsSlidingUp) {
                return Math.min(Math.max(top, expandedTop), collapsedTop);
            } else {
                return Math.min(Math.max(top, collapsedTop), expandedTop);
            }
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\slidinguppanel\ViewDragHelper.j

```java
/*
 * Copyright (C) 2013 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


package com.naman14.timber.slidinguppanel;

import android.content.Context;
import android.support.v4.view.MotionEventCompat;
import android.support.v4.view.VelocityTrackerCompat;
import android.support.v4.view.ViewCompat;
import android.support.v4.widget.ScrollerCompat;
import android.view.MotionEvent;
import android.view.VelocityTracker;
import android.view.View;
import android.view.ViewConfiguration;
import android.view.ViewGroup;
import android.view.animation.Interpolator;

import java.util.Arrays;

/**
 * ViewDragHelper is a utility class for writing custom ViewGroups. It offers a number
 * of useful operations and state tracking for allowing a user to drag and reposition
 * views within their parent ViewGroup.
 */
public class ViewDragHelper {
    /**
     * A null/invalid pointer ID.
     */
    public static final int INVALID_POINTER = -1;
    /**
     * A view is not currently being dragged or animating as a result of a fling/snap.
     */
    public static final int STATE_IDLE = 0;
    /**
     * A view is currently being dragged. The position is currently changing as a result
     * of user input or simulated user input.
     */
    public static final int STATE_DRAGGING = 1;
    /**
     * A view is currently settling into place as a result of a fling or
     * predefined non-interactive motion.
     */
    public static final int STATE_SETTLING = 2;
    /**
     * Edge flag indicating that the left edge should be affected.
     */
    public static final int EDGE_LEFT = 1 << 0;
    /**
     * Edge flag indicating that the right edge should be affected.
     */
    public static final int EDGE_RIGHT = 1 << 1;
    /**
     * Edge flag indicating that the top edge should be affected.
     */
    public static final int EDGE_TOP = 1 << 2;
    /**
```

```
     * Edge flag indicating that the bottom edge should be affected.
     */
    public static final int EDGE_BOTTOM = 1 << 3;
    /**
     * Edge flag set indicating all edges should be affected.
     */
    public static final int EDGE_ALL = EDGE_LEFT | EDGE_TOP | EDGE_RIGHT | EDGE_BOTTOM;
    /**
     * Indicates that a check should occur along the horizontal axis
     */
    public static final int DIRECTION_HORIZONTAL = 1 << 0;
    /**
     * Indicates that a check should occur along the vertical axis
     */
    public static final int DIRECTION_VERTICAL = 1 << 1;
    /**
     * Indicates that a check should occur along all axes
     */
    public static final int DIRECTION_ALL = DIRECTION_HORIZONTAL | DIRECTION_VERTICAL;
    private static final String TAG = "ViewDragHelper";
    private static final int EDGE_SIZE = 20; // dp

    private static final int BASE_SETTLE_DURATION = 256; // ms
    private static final int MAX_SETTLE_DURATION = 600; // ms
    /**
     * Interpolator defining the animation curve for mScroller
     */
    private static final Interpolator sInterpolator = new Interpolator() {
        public float getInterpolation(float t) {
            t -= 1.0f;
            return t * t * t * t * t + 1.0f;
        }
    };
    private final Callback mCallback;
    private final ViewGroup mParentView;
    // Current drag state; idle, dragging or settling
    private int mDragState;
    // Distance to travel before a drag may begin
    private int mTouchSlop;
    // Last known position/pointer tracking
    private int mActivePointerId = INVALID_POINTER;
    private float[] mInitialMotionX;
    private float[] mInitialMotionY;
    private float[] mLastMotionX;
    private float[] mLastMotionY;
    private int[] mInitialEdgesTouched;
    private int[] mEdgeDragsInProgress;
    private int[] mEdgeDragsLocked;
    private int mPointersDown;
    private VelocityTracker mVelocityTracker;
    private float mMaxVelocity;
    private float mMinVelocity;
    private int mEdgeSize;
    private int mTrackingEdges;
    private ScrollerCompat mScroller;
    private View mCapturedView;
    private final Runnable mSetIdleRunnable = new Runnable() {
        public void run() {
            setDragState(STATE_IDLE);
        }
    };
    private boolean mReleaseInProgress;

    /**
     * Apps should use ViewDragHelper.create() to get a new instance.
     * This will allow VDH to use internal compatibility implementations for different
     * platform versions.
     *
     * @param context   Context to initialize config-dependent params from
     * @param forParent Parent view to monitor
```

```java
     */
    private ViewDragHelper(Context context, ViewGroup forParent, Callback cb) {
        if (forParent == null) {
            throw new IllegalArgumentException("Parent view may not be null");
        }
        if (cb == null) {
            throw new IllegalArgumentException("Callback may not be null");
        }

        mParentView = forParent;
        mCallback = cb;

        final ViewConfiguration vc = ViewConfiguration.get(context);
        final float density = context.getResources().getDisplayMetrics().density;
        mEdgeSize = (int) (EDGE_SIZE * density + 0.5f);

        mTouchSlop = vc.getScaledTouchSlop();
        mMaxVelocity = vc.getScaledMaximumFlingVelocity();
        mMinVelocity = vc.getScaledMinimumFlingVelocity();
        mScroller = ScrollerCompat.create(context, sInterpolator);
    }

    /**
     * Factory method to create a new ViewDragHelper.
     *
     * @param forParent Parent view to monitor
     * @param cb        Callback to provide information and receive events
     * @return a new ViewDragHelper instance
     */
    public static ViewDragHelper create(ViewGroup forParent, Callback cb) {
        return new ViewDragHelper(forParent.getContext(), forParent, cb);
    }

    /**
     * Factory method to create a new ViewDragHelper.
     *
     * @param forParent   Parent view to monitor
     * @param sensitivity Multiplier for how sensitive the helper should be about detecting
     *                    the start of a drag. Larger values are more sensitive. 1.0f is normal.
     * @param cb          Callback to provide information and receive events
     * @return a new ViewDragHelper instance
     */
    public static ViewDragHelper create(ViewGroup forParent, float sensitivity, Callback cb) {
        final ViewDragHelper helper = create(forParent, cb);
        helper.mTouchSlop = (int) (helper.mTouchSlop * (1 / sensitivity));
        return helper;
    }

    /**
     * Return the currently configured minimum velocity. Any flings with a magnitude less
     * than this value in pixels per second. Callback methods accepting a velocity will receive
     * zero as a velocity value if the real detected velocity was below this threshold.
     *
     * @return the minimum velocity that will be detected
     */
    public float getMinVelocity() {
        return mMinVelocity;
    }

    /**
     * Set the minimum velocity that will be detected as having a magnitude greater than zero
     * in pixels per second. Callback methods accepting a velocity will be clamped appropriately.
     *
     * @param minVel Minimum velocity to detect
     */
    public void setMinVelocity(float minVel) {
        mMinVelocity = minVel;
    }

    /**
```

```java
     * Retrieve the current drag state of this helper. This will return one of
     * {@link #STATE_IDLE}, {@link #STATE_DRAGGING} or {@link #STATE_SETTLING}.
     *
     * @return The current drag state
     */
    public int getViewDragState() {
        return mDragState;
    }

    /**
     * Enable edge tracking for the selected edges of the parent view.
     * The callback's {@link Callback#onEdgeTouched(int, int)} and
     * {@link Callback#onEdgeDragStarted(int, int)} methods will only be invoked
     * for edges for which edge tracking has been enabled.
     *
     * @param edgeFlags Combination of edge flags describing the edges to watch
     * @see #EDGE_LEFT
     * @see #EDGE_TOP
     * @see #EDGE_RIGHT
     * @see #EDGE_BOTTOM
     */
    public void setEdgeTrackingEnabled(int edgeFlags) {
        mTrackingEdges = edgeFlags;
    }

    /**
     * Return the size of an edge. This is the range in pixels along the edges of this view
     * that will actively detect edge touches or drags if edge tracking is enabled.
     *
     * @return The size of an edge in pixels
     * @see #setEdgeTrackingEnabled(int)
     */
    public int getEdgeSize() {
        return mEdgeSize;
    }

    /**
     * Capture a specific child view for dragging within the parent. The callback will be notified
     * but {@link Callback#tryCaptureView(android.view.View, int)} will not be asked permission to
     * capture this view.
     *
     * @param childView       Child view to capture
     * @param activePointerId ID of the pointer that is dragging the captured child view
     */
    public void captureChildView(View childView, int activePointerId) {
        if (childView.getParent() != mParentView) {
            throw new IllegalArgumentException("captureChildView: parameter must be a descendant " +
                    "of the ViewDragHelper's tracked parent view (" + mParentView + ")");
        }

        mCapturedView = childView;
        mActivePointerId = activePointerId;
        mCallback.onViewCaptured(childView, activePointerId);
        setDragState(STATE_DRAGGING);
    }

    /**
     * @return The currently captured view, or null if no view has been captured.
     */
    public View getCapturedView() {
        return mCapturedView;
    }

    /**
     * @return The ID of the pointer currently dragging the captured view,
     * or {@link #INVALID_POINTER}.
     */
    public int getActivePointerId() {
        return mActivePointerId;
    }
    /**
     * @return The minimum distance in pixels that the user must travel to initiate a drag
     */
    public int getTouchSlop() {
        return mTouchSlop;
    }

    /**
     * The result of a call to this method is equivalent to
     * {@link #processTouchEvent(android.view.MotionEvent)} receiving an ACTION_CANCEL event.
     */
    public void cancel() {
        mActivePointerId = INVALID_POINTER;
        clearMotionHistory();

        if (mVelocityTracker != null) {
            mVelocityTracker.recycle();
            mVelocityTracker = null;
        }
    }

    /**
     * {@link #cancel()}, but also abort all motion in progress and snap to the end of any
     * animation.
     */
    public void abort() {
        cancel();
        if (mDragState == STATE_SETTLING) {
            final int oldX = mScroller.getCurrX();
            final int oldY = mScroller.getCurrY();
            mScroller.abortAnimation();
            final int newX = mScroller.getCurrX();
            final int newY = mScroller.getCurrY();
            mCallback.onViewPositionChanged(mCapturedView, newX, newY, newX - oldX, newY - oldY);
        }
        setDragState(STATE_IDLE);
    }

    /**
     * Animate the view <code>child</code> to the given (left, top) position.
     * If this method returns true, the caller should invoke {@link #continueSettling(boolean)}
     * on each subsequent frame to continue the motion until it returns false. If this method
     * returns false there is no further work to do to complete the movement.
     * <p/>
     * <p>This operation does not count as a capture event, though {@link #getCapturedView()}
     * will still report the sliding view while the slide is in progress.</p>
     *
     * @param child     Child view to capture and animate
     * @param finalLeft Final left position of child
     * @param finalTop  Final top position of child
     * @return true if animation should continue through {@link #continueSettling(boolean)} calls
     */
    public boolean smoothSlideViewTo(View child, int finalLeft, int finalTop) {
        mCapturedView = child;
        mActivePointerId = INVALID_POINTER;

        return forceSettleCapturedViewAt(finalLeft, finalTop, 0, 0);
    }

    /**
     * Settle the captured view at the given (left, top) position.
     * The appropriate velocity from prior motion will be taken into account.
     * If this method returns true, the caller should invoke {@link #continueSettling(boolean)}
     * on each subsequent frame to continue the motion until it returns false. If this method
     * returns false there is no further work to do to complete the movement.
     *
     * @param finalLeft Settled left edge position for the captured view
     * @param finalTop  Settled top edge position for the captured view
     * @return true if animation should continue through {@link #continueSettling(boolean)} calls
     */
    public boolean settleCapturedViewAt(int finalLeft, int finalTop) {
        if (!mReleaseInProgress) {
            throw new IllegalStateException("Cannot settleCapturedViewAt outside of a call to " +
                    "Callback#onViewReleased");
        }

        return forceSettleCapturedViewAt(finalLeft, finalTop,
                (int) VelocityTrackerCompat.getXVelocity(mVelocityTracker, mActivePointerId),
                (int) VelocityTrackerCompat.getYVelocity(mVelocityTracker, mActivePointerId));
    }

    /**
     * Settle the captured view at the given (left, top) position.
     *
     * @param finalLeft Target left position for the captured view
     * @param finalTop  Target top position for the captured view
     * @param xvel      Horizontal velocity
     * @param yvel      Vertical velocity
     * @return true if animation should continue through {@link #continueSettling(boolean)} calls
     */
    private boolean forceSettleCapturedViewAt(int finalLeft, int finalTop, int xvel, int yvel) {
        final int startLeft = mCapturedView.getLeft();
        final int startTop = mCapturedView.getTop();
        final int dx = finalLeft - startLeft;
        final int dy = finalTop - startTop;

        if (dx == 0 && dy == 0) {
            // Nothing to do. Send callbacks, be done.
            mScroller.abortAnimation();
            setDragState(STATE_IDLE);
            return false;
        }

        final int duration = computeSettleDuration(mCapturedView, dx, dy, xvel, yvel);
        mScroller.startScroll(startLeft, startTop, dx, dy, duration);

        setDragState(STATE_SETTLING);
        return true;
    }

    private int computeSettleDuration(View child, int dx, int dy, int xvel, int yvel) {
        xvel = clampMag(xvel, (int) mMinVelocity, (int) mMaxVelocity);
        yvel = clampMag(yvel, (int) mMinVelocity, (int) mMaxVelocity);
        final int absDx = Math.abs(dx);
        final int absDy = Math.abs(dy);
        final int absXVel = Math.abs(xvel);
        final int absYVel = Math.abs(yvel);
        final int addedVel = absXVel + absYVel;
        final int addedDistance = absDx + absDy;

        final float xweight = xvel != 0 ? (float) absXVel / addedVel :
                (float) absDx / addedDistance;
        final float yweight = yvel != 0 ? (float) absYVel / addedVel :
                (float) absDy / addedDistance;

        int xduration = computeAxisDuration(dx, xvel, mCallback.getViewHorizontalDragRange(child));
        int yduration = computeAxisDuration(dy, yvel, mCallback.getViewVerticalDragRange(child));

        return (int) (xduration * xweight + yduration * yweight);
    }

    private int computeAxisDuration(int delta, int velocity, int motionRange) {
        if (delta == 0) {
            return 0;
        }

        final int width = mParentView.getWidth();
        final int halfWidth = width / 2;
        final float distanceRatio = Math.min(1f, (float) Math.abs(delta) / width);
```

```java
        final float distance = halfWidth + halfWidth *
                distanceInfluenceForSnapDuration(distanceRatio);

        int duration;
        velocity = Math.abs(velocity);
        if (velocity > 0) {
            duration = 4 * Math.round(1000 * Math.abs(distance / velocity));
        } else {
            final float range = (float) Math.abs(delta) / motionRange;
            duration = (int) ((range + 1) * BASE_SETTLE_DURATION);
        }
        return Math.min(duration, MAX_SETTLE_DURATION);
    }

    /**
     * Clamp the magnitude of value for absMin and absMax.
     * If the value is below the minimum, it will be clamped to zero.
     * If the value is above the maximum, it will be clamped to the maximum.
     *
     * @param value  Value to clamp
     * @param absMin Absolute value of the minimum significant value to return
     * @param absMax Absolute value of the maximum value to return
     * @return The clamped value with the same sign as <code>value</code>
     */
    private int clampMag(int value, int absMin, int absMax) {
        final int absValue = Math.abs(value);
        if (absValue < absMin) return 0;
        if (absValue > absMax) return value > 0 ? absMax : -absMax;
        return value;
    }

    /**
     * Clamp the magnitude of value for absMin and absMax.
     * If the value is below the minimum, it will be clamped to zero.
     * If the value is above the maximum, it will be clamped to the maximum.
     *
     * @param value  Value to clamp
     * @param absMin Absolute value of the minimum significant value to return
     * @param absMax Absolute value of the maximum value to return
     * @return The clamped value with the same sign as <code>value</code>
     */
    private float clampMag(float value, float absMin, float absMax) {
        final float absValue = Math.abs(value);
        if (absValue < absMin) return 0;
        if (absValue > absMax) return value > 0 ? absMax : -absMax;
        return value;
    }

    private float distanceInfluenceForSnapDuration(float f) {
        f -= 0.5f; // center the values about 0.
        f *= 0.3f * Math.PI / 2.0f;
        return (float) Math.sin(f);
    }

    /**
     * Settle the captured view based on standard free-moving fling behavior.
     * The caller should invoke {@link #continueSettling(boolean)} on each subsequent frame
     * to continue the motion until it returns false.
     *
     * @param minLeft Minimum X position for the view's left edge
     * @param minTop  Minimum Y position for the view's top edge
     * @param maxLeft Maximum X position for the view's left edge
     * @param maxTop  Maximum Y position for the view's top edge
     */
    public void flingCapturedView(int minLeft, int minTop, int maxLeft, int maxTop) {
        if (!mReleaseInProgress) {
            throw new IllegalStateException("Cannot flingCapturedView outside of a call to " +
                    "Callback#onViewReleased");
        }
        mScroller.fling(mCapturedView.getLeft(), mCapturedView.getTop(),
                (int) VelocityTrackerCompat.getXVelocity(mVelocityTracker, mActivePointerId),
                (int) VelocityTrackerCompat.getYVelocity(mVelocityTracker, mActivePointerId),
                minLeft, maxLeft, minTop, maxTop);

        setDragState(STATE_SETTLING);
    }

    /**
     * Move the captured settling view by the appropriate amount for the current time.
     * If <code>continueSettling</code> returns true, the caller should call it again
     * on the next frame to continue.
     *
     * @param deferCallbacks true if state callbacks should be deferred via posted message.
     *                       Set this to true if you are calling this method from
     *                       {@link android.view.View#computeScroll()} or similar methods
     *                       invoked as part of layout or drawing.
     * @return true if settle is still in progress
     */
    public boolean continueSettling(boolean deferCallbacks) {
        if (mDragState == STATE_SETTLING) {
            boolean keepGoing = mScroller.computeScrollOffset();
            final int x = mScroller.getCurrX();
            final int y = mScroller.getCurrY();
            final int dx = x - mCapturedView.getLeft();
            final int dy = y - mCapturedView.getTop();

            if (dx != 0) {
                mCapturedView.offsetLeftAndRight(dx);
            }
            if (dy != 0) {
                mCapturedView.offsetTopAndBottom(dy);
            }

            if (dx != 0 || dy != 0) {
                mCallback.onViewPositionChanged(mCapturedView, x, y, dx, dy);
            }

            if (keepGoing && x == mScroller.getFinalX() && y == mScroller.getFinalY()) {
                // Close enough. The interpolator/scroller might think we're still moving
                // but the user sure doesn't.
                mScroller.abortAnimation();
                keepGoing = mScroller.isFinished();
            }

            if (!keepGoing) {
                if (deferCallbacks) {
                    mParentView.post(mSetIdleRunnable);
                } else {
                    setDragState(STATE_IDLE);
                }
            }
        }

        return mDragState == STATE_SETTLING;
    }

    /**
     * Like all callback events this must happen on the UI thread, but release
     * involves some extra semantics. During a release (mReleaseInProgress)
     * is the only time it is valid to call {@link #settleCapturedViewAt(int, int)}
     * or {@link #flingCapturedView(int, int, int, int)}.
     */
    private void dispatchViewReleased(float xvel, float yvel) {
        mReleaseInProgress = true;
        mCallback.onViewReleased(mCapturedView, xvel, yvel);
        mReleaseInProgress = false;

        if (mDragState == STATE_DRAGGING) {
            // onViewReleased didn't call a method that would have changed this. Go idle.
            setDragState(STATE_IDLE);
        }
    }

    private void clearMotionHistory() {
        if (mInitialMotionX == null) {
            return;
        }
        Arrays.fill(mInitialMotionX, 0);
        Arrays.fill(mInitialMotionY, 0);
        Arrays.fill(mLastMotionX, 0);
        Arrays.fill(mLastMotionY, 0);
        Arrays.fill(mInitialEdgesTouched, 0);
        Arrays.fill(mEdgeDragsInProgress, 0);
        Arrays.fill(mEdgeDragsLocked, 0);
        mPointersDown = 0;
    }

    private void clearMotionHistory(int pointerId) {
        if (mInitialMotionX == null) {
            return;
        }
        mInitialMotionX[pointerId] = 0;
        mInitialMotionY[pointerId] = 0;
        mLastMotionX[pointerId] = 0;
        mLastMotionY[pointerId] = 0;
        mInitialEdgesTouched[pointerId] = 0;
        mEdgeDragsInProgress[pointerId] = 0;
        mEdgeDragsLocked[pointerId] = 0;
        mPointersDown &= ~(1 << pointerId);
    }

    private void ensureMotionHistorySizeForId(int pointerId) {
        if (mInitialMotionX == null || mInitialMotionX.length <= pointerId) {
            float[] imx = new float[pointerId + 1];
            float[] imy = new float[pointerId + 1];
            float[] lmx = new float[pointerId + 1];
            float[] lmy = new float[pointerId + 1];
            int[] iit = new int[pointerId + 1];
            int[] edip = new int[pointerId + 1];
            int[] edl = new int[pointerId + 1];

            if (mInitialMotionX != null) {
                System.arraycopy(mInitialMotionX, 0, imx, 0, mInitialMotionX.length);
                System.arraycopy(mInitialMotionY, 0, imy, 0, mInitialMotionY.length);
                System.arraycopy(mLastMotionX, 0, lmx, 0, mLastMotionX.length);
                System.arraycopy(mLastMotionY, 0, lmy, 0, mLastMotionY.length);
                System.arraycopy(mInitialEdgesTouched, 0, iit, 0, mInitialEdgesTouched.length);
                System.arraycopy(mEdgeDragsInProgress, 0, edip, 0, mEdgeDragsInProgress.length);
                System.arraycopy(mEdgeDragsLocked, 0, edl, 0, mEdgeDragsLocked.length);
            }

            mInitialMotionX = imx;
            mInitialMotionY = imy;
            mLastMotionX = lmx;
            mLastMotionY = lmy;
            mInitialEdgesTouched = iit;
            mEdgeDragsInProgress = edip;
            mEdgeDragsLocked = edl;
        }
    }

    private void saveInitialMotion(float x, float y, int pointerId) {
        ensureMotionHistorySizeForId(pointerId);
        mInitialMotionX[pointerId] = mLastMotionX[pointerId] = x;
        mInitialMotionY[pointerId] = mLastMotionY[pointerId] = y;
        mInitialEdgesTouched[pointerId] = getEdgesTouched((int) x, (int) y);
        mPointersDown |= 1 << pointerId;
    }
```

```java
    private void saveLastMotion(MotionEvent ev) {
        final int pointerCount = MotionEventCompat.getPointerCount(ev);
        for (int i = 0; i < pointerCount; i++) {
            final int pointerId = MotionEventCompat.getPointerId(ev, i);
            final float x = MotionEventCompat.getX(ev, i);
            final float y = MotionEventCompat.getY(ev, i);
            mLastMotionX[pointerId] = x;
            mLastMotionY[pointerId] = y;
        }
    }

    /**
     * Check if the given pointer ID represents a pointer that is currently down (to the best
     * of the ViewDragHelper's knowledge).
     * <p/>
     * <p>The state used to report this information is populated by the methods
     * {@link #shouldInterceptTouchEvent(android.view.MotionEvent)} or
     * {@link #processTouchEvent(android.view.MotionEvent)}. If one of these methods has not
     * been called for all relevant MotionEvents to track, the information reported
     * by this method may be stale or incorrect.</p>
     *
     * @param pointerId pointer ID to check; corresponds to IDs provided by MotionEvent
     * @return true if the pointer with the given ID is still down
     */
    public boolean isPointerDown(int pointerId) {
        return (mPointersDown & 1 << pointerId) != 0;
    }

    void setDragState(int state) {
        if (mDragState != state) {
            mDragState = state;
            mCallback.onViewDragStateChanged(state);
            if (state == STATE_IDLE) {
                mCapturedView = null;
            }
        }
    }

    /**
     * Attempt to capture the view with the given pointer ID. The callback will be involved.
     * This will put us into the "dragging" state. If we've already captured this view with
     * this pointer this method will immediately return true without consulting the callback.
     *
     * @param toCapture View to capture
     * @param pointerId Pointer to capture with
     * @return true if capture was successful
     */
    boolean tryCaptureViewForDrag(View toCapture, int pointerId) {
        if (toCapture == mCapturedView && mActivePointerId == pointerId) {
            // Already done!
            return true;
        }
        if (toCapture != null && mCallback.tryCaptureView(toCapture, pointerId)) {
            mActivePointerId = pointerId;
            captureChildView(toCapture, pointerId);
            return true;
        }
        return false;
    }

    /**
     * Tests scrollability within child views of v given a delta of dx.
     *
     * @param v      View to test for horizontal scrollability
     * @param checkV Whether the view v passed should itself be checked for scrollability (true),
     *               or just its children (false).
     * @param dx     Delta scrolled in pixels along the X axis
     * @param dy     Delta scrolled in pixels along the Y axis
     * @param x      X coordinate of the active touch point
     * @param y      Y coordinate of the active touch point
     * @return true if child views of v can be scrolled by delta of dx.
     */
    protected boolean canScroll(View v, boolean checkV, int dx, int dy, int x, int y) {
        if (v instanceof ViewGroup) {
            final ViewGroup group = (ViewGroup) v;
            final int scrollX = v.getScrollX();
            final int scrollY = v.getScrollY();
            final int count = group.getChildCount();
            // Count backwards - let topmost views consume scroll distance first.
            for (int i = count - 1; i >= 0; i--) {
                // TODO: Add versioned support here for transformed views.
                // This will not work for transformed views in Honeycomb+
                final View child = group.getChildAt(i);
                if (x + scrollX >= child.getLeft() && x + scrollX < child.getRight() &&
                        y + scrollY >= child.getTop() && y + scrollY < child.getBottom() &&
                        canScroll(child, true, dx, dy, x + scrollX - child.getLeft(),
                                y + scrollY - child.getTop())) {
                    return true;
                }
            }
        }

        return checkV && (ViewCompat.canScrollHorizontally(v, -dx) ||
                ViewCompat.canScrollVertically(v, -dy));
    }

    /**
     * Check if this event as provided to the parent view's onInterceptTouchEvent should
     * cause the parent to intercept the touch event stream.
     *
     * @param ev MotionEvent provided to onInterceptTouchEvent
     * @return true if the parent view should return true from onInterceptTouchEvent
     */
    public boolean shouldInterceptTouchEvent(MotionEvent ev) {
        final int action = MotionEventCompat.getActionMasked(ev);
        final int actionIndex = MotionEventCompat.getActionIndex(ev);

        if (action == MotionEvent.ACTION_DOWN) {
            // Reset things for a new event stream, just in case we didn't get
            // the whole previous stream.
            cancel();
        }

        if (mVelocityTracker == null) {
            mVelocityTracker = VelocityTracker.obtain();
        }
        mVelocityTracker.addMovement(ev);

        switch (action) {
            case MotionEvent.ACTION_DOWN: {
                final float x = ev.getX();
                final float y = ev.getY();
                final int pointerId = MotionEventCompat.getPointerId(ev, 0);
                saveInitialMotion(x, y, pointerId);

                final View toCapture = findTopChildUnder((int) x, (int) y);

                // Catch a settling view if possible.
                if (toCapture == mCapturedView && mDragState == STATE_SETTLING) {
                    tryCaptureViewForDrag(toCapture, pointerId);
                }

                final int edgesTouched = mInitialEdgesTouched[pointerId];
                if ((edgesTouched & mTrackingEdges) != 0) {
                    mCallback.onEdgeTouched(edgesTouched & mTrackingEdges, pointerId);
                }
                break;
            }

            case MotionEventCompat.ACTION_POINTER_DOWN: {
                final int pointerId = MotionEventCompat.getPointerId(ev, actionIndex);
                final float x = MotionEventCompat.getX(ev, actionIndex);
                final float y = MotionEventCompat.getY(ev, actionIndex);

                saveInitialMotion(x, y, pointerId);

                // A ViewDragHelper can only manipulate one view at a time.
                if (mDragState == STATE_IDLE) {
                    final int edgesTouched = mInitialEdgesTouched[pointerId];
                    if ((edgesTouched & mTrackingEdges) != 0) {
                        mCallback.onEdgeTouched(edgesTouched & mTrackingEdges, pointerId);
                    }
                } else if (mDragState == STATE_SETTLING) {
                    // Catch a settling view if possible.
                    final View toCapture = findTopChildUnder((int) x, (int) y);
                    if (toCapture == mCapturedView) {
                        tryCaptureViewForDrag(toCapture, pointerId);
                    }
                }
                break;
            }

            case MotionEvent.ACTION_MOVE: {
                // First to cross a touch slop over a draggable view wins. Also report edge drags.
                final int pointerCount = MotionEventCompat.getPointerCount(ev);
                for (int i = 0; i < pointerCount && mInitialMotionX != null && mInitialMotionY != null; i++) {
                    final int pointerId = MotionEventCompat.getPointerId(ev, i);
                    final float x = MotionEventCompat.getX(ev, i);
                    final float y = MotionEventCompat.getY(ev, i);
                    final float dx = x - mInitialMotionX[pointerId];
                    final float dy = y - mInitialMotionY[pointerId];

                    reportNewEdgeDrags(dx, dy, pointerId);
                    if (mDragState == STATE_DRAGGING) {
                        // Callback might have started an edge drag
                        break;
                    }

                    final View toCapture = findTopChildUnder((int) mInitialMotionX[pointerId], (int) mInitialMotionY[pointer
                    if (toCapture != null && checkTouchSlop(toCapture, dx, dy) &&
                            tryCaptureViewForDrag(toCapture, pointerId)) {
                        break;
                    }
                }
                saveLastMotion(ev);
                break;
            }

            case MotionEventCompat.ACTION_POINTER_UP: {
                final int pointerId = MotionEventCompat.getPointerId(ev, actionIndex);
                clearMotionHistory(pointerId);
                break;
            }

            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL: {
                cancel();
                break;
            }
        }

        return mDragState == STATE_DRAGGING;
    }

    /**
     * Process a touch event received by the parent view. This method will dispatch callback events
     * as needed before returning. The parent view's onTouchEvent implementation should call this.
     *
     * @param ev The touch event received by the parent view
     */
    public void processTouchEvent(MotionEvent ev) {
        final int action = MotionEventCompat.getActionMasked(ev);
        final int actionIndex = MotionEventCompat.getActionIndex(ev);

        if (action == MotionEvent.ACTION_DOWN) {
            // Reset things for a new event stream, just in case we didn't get
            // the whole previous stream.
            cancel();
        }

        if (mVelocityTracker == null) {
            mVelocityTracker = VelocityTracker.obtain();
        }
        mVelocityTracker.addMovement(ev);

        switch (action) {
            case MotionEvent.ACTION_DOWN: {
                final float x = ev.getX();
                final float y = ev.getY();
                final int pointerId = MotionEventCompat.getPointerId(ev, 0);
                final View toCapture = findTopChildUnder((int) x, (int) y);

                saveInitialMotion(x, y, pointerId);

                // Since the parent is already directly processing this touch event,
                // there is no reason to delay for a slop before dragging.
                // Start immediately if possible.
                tryCaptureViewForDrag(toCapture, pointerId);

                final int edgesTouched = mInitialEdgesTouched[pointerId];
                if ((edgesTouched & mTrackingEdges) != 0) {
                    mCallback.onEdgeTouched(edgesTouched & mTrackingEdges, pointerId);
                }
                break;
            }

            case MotionEventCompat.ACTION_POINTER_DOWN: {
                final int pointerId = MotionEventCompat.getPointerId(ev, actionIndex);
                final float x = MotionEventCompat.getX(ev, actionIndex);
                final float y = MotionEventCompat.getY(ev, actionIndex);

                saveInitialMotion(x, y, pointerId);

                // A ViewDragHelper can only manipulate one view at a time.
                if (mDragState == STATE_IDLE) {
                    // If we're idle we can do anything! Treat it like a normal down event.

                    final View toCapture = findTopChildUnder((int) x, (int) y);
                    tryCaptureViewForDrag(toCapture, pointerId);

                    final int edgesTouched = mInitialEdgesTouched[pointerId];
                    if ((edgesTouched & mTrackingEdges) != 0) {
                        mCallback.onEdgeTouched(edgesTouched & mTrackingEdges, pointerId);
                    }
                } else if (isCapturedViewUnder((int) x, (int) y)) {
                    // We're still tracking a captured view. If the same view is under this
                    // point, we'll swap to controlling it with this pointer instead.
                    // (This will still work if we're "catching" a settling view.)

                    tryCaptureViewForDrag(mCapturedView, pointerId);
                }
                break;
            }

            case MotionEvent.ACTION_MOVE: {
                if (mDragState == STATE_DRAGGING) {
                    final int index = MotionEventCompat.findPointerIndex(ev, mActivePointerId);
                    final float x = MotionEventCompat.getX(ev, index);
                    final float y = MotionEventCompat.getY(ev, index);
                    final int idx = (int) (x - mLastMotionX[mActivePointerId]);
                    final int idy = (int) (y - mLastMotionY[mActivePointerId]);

                    dragTo(mCapturedView.getLeft() + idx, mCapturedView.getTop() + idy, idx, idy);

                    saveLastMotion(ev);
                } else {
                    // Check to see if any pointer is now over a draggable view.
                    final int pointerCount = MotionEventCompat.getPointerCount(ev);
                    for (int i = 0; i < pointerCount; i++) {
                        final int pointerId = MotionEventCompat.getPointerId(ev, i);
                        final float x = MotionEventCompat.getX(ev, i);
                        final float y = MotionEventCompat.getY(ev, i);
                        final float dx = x - mInitialMotionX[pointerId];
                        final float dy = y - mInitialMotionY[pointerId];

                        reportNewEdgeDrags(dx, dy, pointerId);
                        if (mDragState == STATE_DRAGGING) {
                            // Callback might have started an edge drag.
                            break;
                        }

                        final View toCapture = findTopChildUnder((int) x, (int) y);
                        if (checkTouchSlop(toCapture, dx, dy) &&
                                tryCaptureViewForDrag(toCapture, pointerId)) {
                            break;
                        }
                    }
                    saveLastMotion(ev);
                }
                break;
            }

            case MotionEventCompat.ACTION_POINTER_UP: {
                final int pointerId = MotionEventCompat.getPointerId(ev, actionIndex);
                if (mDragState == STATE_DRAGGING && pointerId == mActivePointerId) {
                    // Try to find another pointer that's still holding on to the captured view.
                    int newActivePointer = INVALID_POINTER;
                    final int pointerCount = MotionEventCompat.getPointerCount(ev);
                    for (int i = 0; i < pointerCount; i++) {
                        final int id = MotionEventCompat.getPointerId(ev, i);
                        if (id == mActivePointerId) {
                            // This one's going away, skip.
                            continue;
                        }

                        final float x = MotionEventCompat.getX(ev, i);
                        final float y = MotionEventCompat.getY(ev, i);
                        if (findTopChildUnder((int) x, (int) y) == mCapturedView &&
                                tryCaptureViewForDrag(mCapturedView, id)) {
                            newActivePointer = mActivePointerId;
                            break;
                        }
                    }

                    if (newActivePointer == INVALID_POINTER) {
                        // We didn't find another pointer still touching the view, release it.
                        releaseViewForPointerUp();
                    }
                }
                clearMotionHistory(pointerId);
                break;
            }

            case MotionEvent.ACTION_UP: {
                if (mDragState == STATE_DRAGGING) {
                    releaseViewForPointerUp();
                }
                cancel();
                break;
            }
```

```
            case MotionEvent.ACTION_CANCEL: {
                if (mDragState == STATE_DRAGGING) {
                    dispatchViewReleased(0, 0);
                }
                cancel();
                break;
            }
        }
    }

    private void reportNewEdgeDrags(float dx, float dy, int pointerId) {
        int dragsStarted = 0;
        if (checkNewEdgeDrag(dx, dy, pointerId, EDGE_LEFT)) {
            dragsStarted |= EDGE_LEFT;
        }
        if (checkNewEdgeDrag(dy, dx, pointerId, EDGE_TOP)) {
            dragsStarted |= EDGE_TOP;
        }
        if (checkNewEdgeDrag(dx, dy, pointerId, EDGE_RIGHT)) {
            dragsStarted |= EDGE_RIGHT;
        }
        if (checkNewEdgeDrag(dy, dx, pointerId, EDGE_BOTTOM)) {
            dragsStarted |= EDGE_BOTTOM;
        }

        if (dragsStarted != 0) {
            mEdgeDragsInProgress[pointerId] |= dragsStarted;
            mCallback.onEdgeDragStarted(dragsStarted, pointerId);
        }
    }

    private boolean checkNewEdgeDrag(float delta, float odelta, int pointerId, int edge) {
        final float absDelta = Math.abs(delta);
        final float absODelta = Math.abs(odelta);

        if ((mInitialEdgesTouched[pointerId] & edge) != edge || (mTrackingEdges & edge) == 0 ||
                (mEdgeDragsLocked[pointerId] & edge) == edge ||
                (mEdgeDragsInProgress[pointerId] & edge) == edge ||
                (absDelta <= mTouchSlop && absODelta <= mTouchSlop)) {
            return false;
        }
        if (absDelta < absODelta * 0.5f && mCallback.onEdgeLock(edge)) {
            mEdgeDragsLocked[pointerId] |= edge;
            return false;
        }
        return (mEdgeDragsInProgress[pointerId] & edge) == 0 && absDelta > mTouchSlop;
    }

    /**
     * Check if we've crossed a reasonable touch slop for the given child view.
     * If the child cannot be dragged along the horizontal or vertical axis, motion
     * along that axis will not count toward the slop check.
     *
     * @param child Child to check
     * @param dx    Motion since initial position along X axis
     * @param dy    Motion since initial position along Y axis
     * @return true if the touch slop has been crossed
     */
    private boolean checkTouchSlop(View child, float dx, float dy) {
        if (child == null) {
            return false;
        }
        final boolean checkHorizontal = mCallback.getViewHorizontalDragRange(child) > 0;
        final boolean checkVertical = mCallback.getViewVerticalDragRange(child) > 0;

        if (checkHorizontal && checkVertical) {
            return dx * dx + dy * dy > mTouchSlop * mTouchSlop;
        } else if (checkHorizontal) {
            return Math.abs(dx) > mTouchSlop;
        } else if (checkVertical) {
            return Math.abs(dy) > mTouchSlop;
        }
        return false;
    }

    /**
     * Check if any pointer tracked in the current gesture has crossed
     * the required slop threshold.
     * <p/>
     * <p>This depends on internal state populated by
     * {@link #shouldInterceptTouchEvent(android.view.MotionEvent)} or
     * {@link #processTouchEvent(android.view.MotionEvent)}. You should only rely on
     * the results of this method after all currently available touch data
     * has been provided to one of these two methods.</p>
     *
     * @param directions Combination of direction flags, see {@link #DIRECTION_HORIZONTAL},
     *                   {@link #DIRECTION_VERTICAL}, {@link #DIRECTION_ALL}
     * @return true if the slop threshold has been crossed, false otherwise
     */
    public boolean checkTouchSlop(int directions) {
        final int count = mInitialMotionX.length;
        for (int i = 0; i < count; i++) {
            if (checkTouchSlop(directions, i)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Check if the specified pointer tracked in the current gesture has crossed
     * the required slop threshold.
     * <p/>
     * <p>This depends on internal state populated by
     * {@link #shouldInterceptTouchEvent(android.view.MotionEvent)} or
     * {@link #processTouchEvent(android.view.MotionEvent)}. You should only rely on
     * the results of this method after all currently available touch data
     * has been provided to one of these two methods.</p>
     *
     * @param directions Combination of direction flags, see {@link #DIRECTION_HORIZONTAL},
     *                   {@link #DIRECTION_VERTICAL}, {@link #DIRECTION_ALL}
     * @param pointerId  ID of the pointer to slop check as specified by MotionEvent
     * @return true if the slop threshold has been crossed, false otherwise
     */
    public boolean checkTouchSlop(int directions, int pointerId) {
        if (!isPointerDown(pointerId)) {
            return false;
        }

        final boolean checkHorizontal = (directions & DIRECTION_HORIZONTAL) == DIRECTION_HORIZONTAL;
        final boolean checkVertical = (directions & DIRECTION_VERTICAL) == DIRECTION_VERTICAL;

        final float dx = mLastMotionX[pointerId] - mInitialMotionX[pointerId];
        final float dy = mLastMotionY[pointerId] - mInitialMotionY[pointerId];

        if (checkHorizontal && checkVertical) {
            return dx * dx + dy * dy > mTouchSlop * mTouchSlop;
        } else if (checkHorizontal) {
            return Math.abs(dx) > mTouchSlop;
        } else if (checkVertical) {
            return Math.abs(dy) > mTouchSlop;
        }
        return false;
    }

    /**
     * Check if any of the edges specified were initially touched in the currently active gesture.
     * If there is no currently active gesture this method will return false.
     *
     * @param edges Edges to check for an initial edge touch. See {@link #EDGE_LEFT},
     *              {@link #EDGE_TOP}, {@link #EDGE_RIGHT}, {@link #EDGE_BOTTOM} and
     *              {@link #EDGE_ALL}
     * @return true if any of the edges specified were initially touched in the current gesture
     */
    public boolean isEdgeTouched(int edges) {
        final int count = mInitialEdgesTouched.length;
        for (int i = 0; i < count; i++) {
            if (isEdgeTouched(edges, i)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Check if any of the edges specified were initially touched by the pointer with
     * the specified ID. If there is no currently active gesture or if there is no pointer with
     * the given ID currently down this method will return false.
     *
     * @param edges Edges to check for an initial edge touch. See {@link #EDGE_LEFT},
     *              {@link #EDGE_TOP}, {@link #EDGE_RIGHT}, {@link #EDGE_BOTTOM} and
     *              {@link #EDGE_ALL}
     * @return true if any of the edges specified were initially touched in the current gesture
     */
    public boolean isEdgeTouched(int edges, int pointerId) {
        return isPointerDown(pointerId) && (mInitialEdgesTouched[pointerId] & edges) != 0;
    }

    private void releaseViewForPointerUp() {
        mVelocityTracker.computeCurrentVelocity(1000, mMaxVelocity);
        final float xvel = clampMag(
                VelocityTrackerCompat.getXVelocity(mVelocityTracker, mActivePointerId),
                mMinVelocity, mMaxVelocity);
        final float yvel = clampMag(
                VelocityTrackerCompat.getYVelocity(mVelocityTracker, mActivePointerId),
                mMinVelocity, mMaxVelocity);
        dispatchViewReleased(xvel, yvel);
    }

    private void dragTo(int left, int top, int dx, int dy) {
        int clampedX = left;
        int clampedY = top;
        final int oldLeft = mCapturedView.getLeft();
        final int oldTop = mCapturedView.getTop();
        if (dx != 0) {
            clampedX = mCallback.clampViewPositionHorizontal(mCapturedView, left, dx);
            mCapturedView.offsetLeftAndRight(clampedX - oldLeft);
        }
        if (dy != 0) {
            clampedY = mCallback.clampViewPositionVertical(mCapturedView, top, dy);
            mCapturedView.offsetTopAndBottom(clampedY - oldTop);
        }

        if (dx != 0 || dy != 0) {
            final int clampedDx = clampedX - oldLeft;
            final int clampedDy = clampedY - oldTop;
            mCallback.onViewPositionChanged(mCapturedView, clampedX, clampedY,
                    clampedDx, clampedDy);
        }
    }

    /**
     * Determine if the currently captured view is under the given point in the
     * parent view's coordinate system. If there is no captured view this method
     * will return false.
     *
     * @param x X position to test in the parent's coordinate system
     * @param y Y position to test in the parent's coordinate system
     * @return true if the captured view is under the given point, false otherwise
     */
    public boolean isCapturedViewUnder(int x, int y) {
        return isViewUnder(mCapturedView, x, y);
    }

    /**
     * Determine if the supplied view is under the given point in the
     * parent view's coordinate system.
     *
     * @param view Child view of the parent to hit test
     * @param x    X position to test in the parent's coordinate system
     * @param y    Y position to test in the parent's coordinate system
     * @return true if the supplied view is under the given point, false otherwise
     */
    public boolean isViewUnder(View view, int x, int y) {
        if (view == null) {
            return false;
        }
        return x >= view.getLeft() &&
                x < view.getRight() &&
                y >= view.getTop() &&
                y < view.getBottom();
    }

    /**
     * Find the topmost child under the given point within the parent view's coordinate system.
     * The child order is determined using {@link Callback#getOrderedChildIndex(int)}.
     *
     * @param x X position to test in the parent's coordinate system
     * @param y Y position to test in the parent's coordinate system
     * @return The topmost child view under (x, y) or null if none found.
     */
    public View findTopChildUnder(int x, int y) {
        final int childCount = mParentView.getChildCount();
        for (int i = childCount - 1; i >= 0; i--) {
            final View child = mParentView.getChildAt(mCallback.getOrderedChildIndex(i));
            if (x >= child.getLeft() && x < child.getRight() &&
                    y >= child.getTop() && y < child.getBottom()) {
                return child;
            }
        }
        return null;
    }

    private int getEdgesTouched(int x, int y) {
        int result = 0;

        if (x < mParentView.getLeft() + mEdgeSize) result |= EDGE_LEFT;
        if (y < mParentView.getTop() + mEdgeSize) result |= EDGE_TOP;
        if (x > mParentView.getRight() - mEdgeSize) result |= EDGE_RIGHT;
        if (y > mParentView.getBottom() - mEdgeSize) result |= EDGE_BOTTOM;

        return result;
    }

    /**
     * A Callback is used as a communication channel with the ViewDragHelper back to the
     * parent view using it. <code>on*</code>methods are invoked on siginficant events and several
     * accessor methods are expected to provide the ViewDragHelper with more information
     * about the state of the parent view upon request. The callback also makes decisions
     * governing the range and draggability of child views.
     */
    public static abstract class Callback {
        /**
         * Called when the drag state changes. See the <code>STATE_*</code> constants
         * for more information.
         *
         * @param state The new drag state
         * @see #STATE_IDLE
         * @see #STATE_DRAGGING
         * @see #STATE_SETTLING
         */
        public void onViewDragStateChanged(int state) {
        }

        /**
         * Called when the captured view's position changes as the result of a drag or settle.
         *
         * @param changedView View whose position changed
         * @param left        New X coordinate of the left edge of the view
         * @param top         New Y coordinate of the top edge of the view
         * @param dx          Change in X position from the last call
         * @param dy          Change in Y position from the last call
         */
        public void onViewPositionChanged(View changedView, int left, int top, int dx, int dy) {
        }

        /**
         * Called when a child view is captured for dragging or settling. The ID of the pointer
         * currently dragging the captured view is supplied. If activePointerId is
         * identified as {@link #INVALID_POINTER} the capture is programmatic instead of
         * pointer-initiated.
         *
         * @param capturedChild   Child view that was captured
         * @param activePointerId Pointer id tracking the child capture
         */
        public void onViewCaptured(View capturedChild, int activePointerId) {
        }

        /**
         * Called when the child view is no longer being actively dragged.
         * The fling velocity is also supplied, if relevant. The velocity values may
         * be clamped to system minimums or maximums.
         * <p/>
         * <p>Calling code may decide to fling or otherwise release the view to let it
         * settle into place. It should do so using {@link #settleCapturedViewAt(int, int)}
         * or {@link #flingCapturedView(int, int, int, int)}. If the Callback invokes
         * one of these methods, the ViewDragHelper will enter {@link #STATE_SETTLING}
         * and the view capture will not fully end until it comes to a complete stop.
         * If neither of these methods is invoked before <code>onViewReleased</code> returns,
         * the view will stop in place and the ViewDragHelper will return to
         * {@link #STATE_IDLE}.</p>
         *
         * @param releasedChild The captured child view now being released
         * @param xvel          X velocity of the pointer as it left the screen in pixels per second.
         * @param yvel          Y velocity of the pointer as it left the screen in pixels per second.
         */
        public void onViewReleased(View releasedChild, float xvel, float yvel) {
        }

        /**
         * Called when one of the subscribed edges in the parent view has been touched
         * by the user while no child view is currently captured.
         *
         * @param edgeFlags A combination of edge flags describing the edge(s) currently touched
         * @param pointerId ID of the pointer touching the described edge(s)
         * @see #EDGE_LEFT
         * @see #EDGE_TOP
         * @see #EDGE_RIGHT
         * @see #EDGE_BOTTOM
         */
        public void onEdgeTouched(int edgeFlags, int pointerId) {
        }

        /**
         * Called when the given edge may become locked. This can happen if an edge drag
         * was preliminarily rejected before beginning, but after {@link #onEdgeTouched(int, int)}
         * was called. This method should return true to lock this edge or false to leave it
         * unlocked. The default behavior is to leave edges unlocked.
         *
         * @param edgeFlags A combination of edge flags describing the edge(s) locked
         * @return true to lock the edge, false to leave it unlocked
         */
        public boolean onEdgeLock(int edgeFlags) {
            return false;
        }

        /**
         * Called when the user has started a deliberate drag away from one
         * of the subscribed edges in the parent view while no child view is currently captured.
         *
         * @param edgeFlags A combination of edge flags describing the edge(s) dragged
         * @param pointerId ID of the pointer touching the described edge(s)
         * @see #EDGE_LEFT
         * @see #EDGE_TOP
         * @see #EDGE_RIGHT
         * @see #EDGE_BOTTOM
         */
        public void onEdgeDragStarted(int edgeFlags, int pointerId) {
        }

        /**
         * Called to determine the Z-order of child views.
         *
         * @param index the ordered position to query for
         * @return index of the view that should be ordered at position <code>index</code>
         */
        public int getOrderedChildIndex(int index) {
            return index;
        }

        /**
         * Return the magnitude of a draggable child view's horizontal range of motion in pixels.
         * This method should return 0 for views that cannot move horizontally.
         *
         * @param child Child view to check
         * @return range of horizontal motion in pixels
         */
        public int getViewHorizontalDragRange(View child) {
            return 0;
        }

        /**
         * Return the magnitude of a draggable child view's vertical range of motion in pixels.
         * This method should return 0 for views that cannot move vertically.
         *
         * @param child Child view to check
         * @return range of vertical motion in pixels
         */
        public int getViewVerticalDragRange(View child) {
            return 0;
        }

        /**
         * Called when the user's input indicates that they want to capture the given child view
         * with the pointer indicated by pointerId. The callback should return true if the user
         * is permitted to drag the given view with the indicated pointer.
         * <p/>
         * <p>ViewDragHelper may call this method multiple times for the same view even if
         * the view is already captured; this indicates that a new pointer is trying to take
         * control of the view.</p>
         * <p/>
         * <p>If this method returns true, a call to {@link #onViewCaptured(android.view.View, int)}
         * will follow if the capture is successful.</p>
         *
         * @param child     Child the user is attempting to capture
         * @param pointerId ID of the pointer attempting the capture
         * @return true if capture should be allowed, false otherwise
         */
        public abstract boolean tryCaptureView(View child, int pointerId);
        /**
         * Restrict the motion of the dragged child view along the horizontal axis.
         * The default implementation does not allow horizontal motion; the extending
         * class must override this method and provide the desired clamping.
         *
         * @param child Child view being dragged
         * @param left  Attempted motion along the X axis
         * @param dx    Proposed change in position for left
         * @return The new clamped position for left
         */
        public int clampViewPositionHorizontal(View child, int left, int dx) {
            return 0;
        }

        /**
         * Restrict the motion of the dragged child view along the vertical axis.
         * The default implementation does not allow vertical motion; the extending
         * class must override this method and provide the desired clamping.
         *
         * @param child Child view being dragged
         * @param top   Attempted motion along the Y axis
         * @param dy    Proposed change in position for top
         * @return The new clamped position for top
         */
        public int clampViewPositionVertical(View child, int top, int dy) {
            return 0;
        }
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.subfragments;

import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;

public class ArtistTagFragment extends Fragment {

    private static final String ARG_PAGE_NUMBER = "pageNumber";

    public static ArtistTagFragment newInstance(int pageNumber) {
        ArtistTagFragment fragment = new ArtistTagFragment();
        Bundle bundle = new Bundle();
        bundle.putInt(ARG_PAGE_NUMBER, pageNumber);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.layout_artist_tag, container, false);
        return rootView;
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\subfragments\LyricsFragment.jav

```
package com.naman14.timber.subfragments;

import android.content.CursorLoader;
import android.database.Cursor;
import android.graphics.Color;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.utils.LyricsExtractor;
import com.naman14.timber.utils.LyricsLoader;

import java.io.File;

import retrofit.Callback;
import retrofit.RetrofitError;
import retrofit.client.Response;

/**
 * Created by christoph on 10.12.16.
 */

public class LyricsFragment extends Fragment {

    private String lyrics = null;
    private Toolbar toolbar;
    private View rootView;

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
        rootView = inflater.inflate(R.layout.fragment_lyrics,container,false);

        toolbar = (Toolbar) rootView.findViewById(R.id.toolbar);
        setupToolbar();

        loadLyrics();

        return rootView;
    }

    private void loadLyrics() {

        final View lyricsView = rootView.findViewById(R.id.lyrics);
        final TextView poweredbyTextView = (TextView) lyricsView.findViewById(R.id.lyrics_makeitpersonal);
        poweredbyTextView.setVisibility(View.GONE);
        final TextView lyricsTextView = (TextView) lyricsView.findViewById(R.id.lyrics_text);
        lyricsTextView.setText(getString(R.string.lyrics_loading));
        String filename = getRealPathFromURI(Uri.parse(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI + "/" + MusicPlayer.getCu
        if (filename != null && lyrics == null) {
            lyrics = LyricsExtractor.getLyrics(new File(filename));
        }

        if (lyrics != null) {
            lyricsTextView.setText(lyrics);
        } else {
            String artist = MusicPlayer.getArtistName();
            if (artist != null) {
                int i = artist.lastIndexOf(" feat");
                if (i != -1) {
                    artist = artist.substring(0, i);
                }

                LyricsLoader.getInstance(this.getContext()).getLyrics(artist, MusicPlayer.getTrackName(), new Callback<Strin
                    @Override
                    public void success(String s, Response response) {
                        lyrics = s;
                        if (s.equals("Sorry, We don't have lyrics for this song yet.\n")) {
                            lyricsTextView.setText(R.string.no_lyrics);
                        } else {
                            lyricsTextView.setText(s);
                            poweredbyTextView.setVisibility(View.VISIBLE);
                        }
                    }

                    @Override
                    public void failure(RetrofitError error) {
                        lyricsTextView.setText(R.string.no_lyrics);
                    }
                });

            } else {
                lyricsTextView.setText(R.string.no_lyrics);
            }
        }
    }

    private void setupToolbar() {

        ((AppCompatActivity) getActivity()).setSupportActionBar(toolbar);

        final ActionBar ab = ((AppCompatActivity) getActivity()).getSupportActionBar();
        ab.setDisplayHomeAsUpEnabled(true);
        if (MusicPlayer.getTrackName() != null) {
            ab.setTitle(MusicPlayer.getTrackName());
        }
    }

    @Override
    public void onResume() {
        super.onResume();
        toolbar.setBackgroundColor(Color.TRANSPARENT);
    }

    private String getRealPathFromURI(Uri contentUri) {
        String[] proj = {MediaStore.Audio.Media.DATA};
        CursorLoader loader = new CursorLoader(this.getContext(), contentUri, proj, null, null, null);
        Cursor cursor = loader.loadInBackground();
        int column_index = cursor.getColumnIndexOrThrow(MediaStore.Audio.Media.DATA);
        cursor.moveToFirst();
        String result = cursor.getString(column_index);
        cursor.close();
        return result;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\subfragments\PlaylistPagerFragm

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.subfragments;

import android.content.Context;
import android.graphics.Bitmap;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.util.Pair;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import com.naman14.timber.R;
import com.naman14.timber.dataloaders.LastAddedLoader;
import com.naman14.timber.dataloaders.PlaylistLoader;
import com.naman14.timber.dataloaders.PlaylistSongLoader;
import com.naman14.timber.dataloaders.SongLoader;
import com.naman14.timber.dataloaders.TopTracksLoader;
import com.naman14.timber.models.Playlist;
import com.naman14.timber.models.Song;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class PlaylistPagerFragment extends Fragment {

    private static final String ARG_PAGE_NUMBER = "pageNumber";
    private int[] foregroundColors = {R.color.pink_transparent, R.color.green_transparent, R.color.blue_transparent, R.color
    private int pageNumber, songCountInt, totalRuntime;
    private int foregroundColor;
    private long firstAlbumID = -1;
    private Playlist playlist;
    private TextView playlistame, songcount, playlistnumber, playlisttype, runtime;
    private ImageView playlistImage;
    private View foreground;
    private Context mContext;
    private boolean showAuto;

    public static PlaylistPagerFragment newInstance(int pageNumber) {
        PlaylistPagerFragment fragment = new PlaylistPagerFragment();
        Bundle bundle = new Bundle();
        bundle.putInt(ARG_PAGE_NUMBER, pageNumber);
        fragment.setArguments(bundle);
        return fragment;
    }
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        showAuto = PreferencesUtility.getInstance(getActivity()).showAutoPlaylist();
        View rootView = inflater.inflate(R.layout.fragment_playlist_pager, container, false);

        final List<Playlist> playlists = PlaylistLoader.getPlaylists(getActivity(), showAuto);

        pageNumber = getArguments().getInt(ARG_PAGE_NUMBER);
        playlist = playlists.get(pageNumber);

        playlistame = (TextView) rootView.findViewById(R.id.name);
        playlistnumber = (TextView) rootView.findViewById(R.id.number);
        songcount = (TextView) rootView.findViewById(R.id.songcount);
        runtime = (TextView) rootView.findViewById(R.id.runtime);
        playlisttype = (TextView) rootView.findViewById(R.id.playlisttype);
        playlistImage = (ImageView) rootView.findViewById(R.id.playlist_image);
        foreground = rootView.findViewById(R.id.foreground);

        playlistImage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ArrayList<Pair> tranitionViews = new ArrayList<>();
                tranitionViews.add(0, Pair.create((View) playlistame, "transition_playlist_name"));
                tranitionViews.add(1, Pair.create((View) playlistImage, "transition_album_art"));
                tranitionViews.add(2, Pair.create(foreground, "transition_foreground"));
                NavigationUtils.navigateToPlaylistDetail(getActivity(), getPlaylistType(), firstAlbumID, String.valueOf(play
            }
        });

        mContext = this.getContext();
        setUpPlaylistDetails();
        return rootView;
    }


    @Override
    public void onViewCreated(View view, Bundle savedinstancestate) {
        new loadPlaylistImage().execute("");
    }

    private void setUpPlaylistDetails() {
        playlistame.setText(playlist.name);

        int number = getArguments().getInt(ARG_PAGE_NUMBER) + 1;
        String playlistnumberstring;

        if (number > 9) {
            playlistnumberstring = String.valueOf(number);
        } else {
            playlistnumberstring = "0" + String.valueOf(number);
        }
        playlistnumber.setText(playlistnumberstring);

        Random random = new Random();
        int rndInt = random.nextInt(foregroundColors.length);

        foregroundColor = foregroundColors[rndInt];
        foreground.setBackgroundColor(foregroundColor);

        if (showAuto) {
            if (pageNumber <= 2)
                playlisttype.setVisibility(View.VISIBLE);
        }

    }

    private String getPlaylistType() {
        if (showAuto) {
            switch (pageNumber) {
                case 0:
                    return Constants.NAVIGATE_PLAYLIST_LASTADDED;
                case 1:
                    return Constants.NAVIGATE_PLAYLIST_RECENT;
                case 2:
                    return Constants.NAVIGATE_PLAYLIST_TOPTRACKS;
                default:
                    return Constants.NAVIGATE_PLAYLIST_USERCREATED;
            }
        } else return Constants.NAVIGATE_PLAYLIST_USERCREATED;
    }


    private class loadPlaylistImage extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            if (getActivity() != null) {
                if (showAuto) {
                    switch (pageNumber) {
                        case 0:
                            List<Song> lastAddedSongs = LastAddedLoader.getLastAddedSongs(getActivity());
                            songCountInt = lastAddedSongs.size();
                            for(Song song : lastAddedSongs) {
                                totalRuntime += song.duration / 1000; //for some reason default playlists have songs with du
                            }
                            if (songCountInt != 0) {
                                firstAlbumID = lastAddedSongs.get(0).albumId;
                                return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                            } else return "nosongs";
                        case 1:
                            TopTracksLoader recentloader = new TopTracksLoader(getActivity(), TopTracksLoader.QueryType.Rece
                            List<Song> recentsongs = SongLoader.getSongsForCursor(TopTracksLoader.getCursor());
                            songCountInt = recentsongs.size();
                            for(Song song : recentsongs){
                                    totalRuntime += song.duration / 1000;
                            }

                            if (songCountInt != 0) {
                                firstAlbumID = recentsongs.get(0).albumId;
                                return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                            } else return "nosongs";
                        case 2:
                            TopTracksLoader topTracksLoader = new TopTracksLoader(getActivity(), TopTracksLoader.QueryType.T
                            List<Song> topsongs = SongLoader.getSongsForCursor(TopTracksLoader.getCursor());
                            songCountInt = topsongs.size();
                            for(Song song : topsongs){
                                    totalRuntime += song.duration / 1000;
                            }
                            if (songCountInt != 0) {
                                firstAlbumID = topsongs.get(0).albumId;
                                return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                            } else return "nosongs";
                        default:
                            List<Song> playlistsongs = PlaylistSongLoader.getSongsInPlaylist(getActivity(), playlist.id);
                            songCountInt = playlistsongs.size();
                            for(Song song : playlistsongs){
                                totalRuntime += song.duration;
                            }
                            if (songCountInt != 0) {
                                firstAlbumID = playlistsongs.get(0).albumId;
                                return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                            } else return "nosongs";

                    }
                } else {
                    List<Song> playlistsongs = PlaylistSongLoader.getSongsInPlaylist(getActivity(), playlist.id);
                    songCountInt = playlistsongs.size();
                    for(Song song : playlistsongs){
                        totalRuntime += song.duration;
                    }
                    if (songCountInt != 0) {
                        firstAlbumID = playlistsongs.get(0).albumId;
                        return TimberUtils.getAlbumArtUri(firstAlbumID).toString();
                    } else return "nosongs";
                }
            } else return "context is null";

        }

        @Override
        protected void onPostExecute(String uri) {
            ImageLoader.getInstance().displayImage(uri, playlistImage,
                    new DisplayImageOptions.Builder().cacheInMemory(true)
                            .showImageOnFail(R.drawable.ic_empty_music2)
                            .resetViewBeforeLoading(true)
                            .build(), new SimpleImageLoadingListener() {
                        @Override
                        public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                        }
                    });
            songcount.setText(" " + String.valueOf(songCountInt) + " " + mContext.getString(R.string.songs));
            runtime.setText(" " + TimberUtils.makeShortTimeString(mContext, totalRuntime));
        }

        @Override
        protected void onPreExecute() {
        }
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\subfragments\QuickControlsFragm

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.subfragments;

import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.TransitionDrawable;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.SeekBar;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.activities.BaseActivity;
import com.naman14.timber.listeners.MusicStateListener;
import com.naman14.timber.utils.Helpers;
import com.naman14.timber.utils.ImageUtils;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;
import com.naman14.timber.utils.SlideTrackSwitcher;
import com.naman14.timber.utils.TimberUtils;
import com.naman14.timber.widgets.PlayPauseButton;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.listener.ImageLoadingListener;

import net.steamcrafted.materialiconlib.MaterialIconView;

public class QuickControlsFragment extends Fragment implements MusicStateListener {


    public static View topContainer;
    private ProgressBar mProgress;
    private SeekBar mSeekBar;
    private int overflowcounter = 0;
    private PlayPauseButton mPlayPause, mPlayPauseExpanded;
    private TextView mTitle, mTitleExpanded;
    private TextView mArtist, mArtistExpanded;
    private ImageView mAlbumArt, mBlurredArt;
    private View rootView;
    private View playPauseWrapper, playPauseWrapperExpanded;
    private MaterialIconView previous, next;
    private boolean duetoplaypause = false;
    private boolean fragmentPaused = false;
    public Runnable mUpdateProgress = new Runnable() {

        @Override
        public void run() {

            long position = MusicPlayer.position();
            mProgress.setProgress((int) position);
            mSeekBar.setProgress((int) position);

            overflowcounter--;
            if (MusicPlayer.isPlaying()) {
                int delay = (int) (1500 - (position % 1000));
                if (overflowcounter < 0 && !fragmentPaused) {
                    overflowcounter++;
                    mProgress.postDelayed(mUpdateProgress, delay);
                }
            } else mProgress.removeCallbacks(this);

        }
    };

    private final View.OnClickListener mPlayPauseListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            duetoplaypause = true;
            if (!mPlayPause.isPlayed()) {
                mPlayPause.setPlayed(true);
                mPlayPause.startAnimation();
            } else {
                mPlayPause.setPlayed(false);
                mPlayPause.startAnimation();
            }
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.playOrPause();
                }
            }, 200);

        }
    };

    private final View.OnClickListener mPlayPauseExpandedListener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            duetoplaypause = true;
            if (!mPlayPauseExpanded.isPlayed()) {
                mPlayPauseExpanded.setPlayed(true);
                mPlayPauseExpanded.startAnimation();
            } else {
                mPlayPauseExpanded.setPlayed(false);
                mPlayPauseExpanded.startAnimation();
            }
            Handler handler = new Handler();
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    MusicPlayer.playOrPause();
                }
            }, 200);

        }
    };

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_playback_controls, container, false);
        this.rootView = rootView;
        mPlayPause = (PlayPauseButton) rootView.findViewById(R.id.play_pause);
        mPlayPauseExpanded = (PlayPauseButton) rootView.findViewById(R.id.playpause);
        playPauseWrapper = rootView.findViewById(R.id.play_pause_wrapper);
        playPauseWrapperExpanded = rootView.findViewById(R.id.playpausewrapper);
        playPauseWrapper.setOnClickListener(mPlayPauseListener);
        playPauseWrapperExpanded.setOnClickListener(mPlayPauseExpandedListener);
        mProgress = (ProgressBar) rootView.findViewById(R.id.song_progress_normal);
        mSeekBar = (SeekBar) rootView.findViewById(R.id.song_progress);
        mTitle = (TextView) rootView.findViewById(R.id.title);
        mArtist = (TextView) rootView.findViewById(R.id.artist);
        mTitleExpanded = (TextView) rootView.findViewById(R.id.song_title);
        mArtistExpanded = (TextView) rootView.findViewById(R.id.song_artist);
        mAlbumArt = (ImageView) rootView.findViewById(R.id.album_art_nowplayingcard);
        mBlurredArt = (ImageView) rootView.findViewById(R.id.blurredAlbumart);
        next = (MaterialIconView) rootView.findViewById(R.id.next);
        previous = (MaterialIconView) rootView.findViewById(R.id.previous);
        topContainer = rootView.findViewById(R.id.topContainer);

        LinearLayout.LayoutParams layoutParams = (LinearLayout.LayoutParams) mProgress.getLayoutParams();
        mProgress.measure(0, 0);
        layoutParams.setMargins(0, -(mProgress.getMeasuredHeight() / 2), 0, 0);
        mProgress.setLayoutParams(layoutParams);

        mPlayPause.setColor(Config.accentColor(getActivity(), Helpers.getATEKey(getActivity())));
        mPlayPauseExpanded.setColor(Color.WHITE);

        mSeekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
                if (b) {
                    MusicPlayer.seek((long) i);
                }
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
            }
        });

        next.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        MusicPlayer.next();
                    }
                }, 200);

            }
        });

        previous.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Handler handler = new Handler();
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        MusicPlayer.previous(getActivity(), false);
                    }
                }, 200);

            }
        });


        ((BaseActivity) getActivity()).setMusicStateListenerListener(this);

        if (PreferencesUtility.getInstance(getActivity()).isGesturesEnabled()) {
            new SlideTrackSwitcher() {
                @Override
                public void onClick() {
                    NavigationUtils.navigateToNowplaying(getActivity(), false);
                }
            }.attach(rootView.findViewById(R.id.root_view));
        }


        return rootView;
    }

    @Override
    public void onPause() {
        super.onPause();
        fragmentPaused = true;
    }

    public void updateNowplayingCard() {
        mTitle.setText(MusicPlayer.getTrackName());
        mArtist.setText(MusicPlayer.getArtistName());
        mTitleExpanded.setText(MusicPlayer.getTrackName());
        mArtistExpanded.setText(MusicPlayer.getArtistName());
        if (!duetoplaypause) {
            ImageLoader.getInstance().displayImage(TimberUtils.getAlbumArtUri(MusicPlayer.getCurrentAlbumId()).toString(), m
                    new DisplayImageOptions.Builder().cacheInMemory(true)
                            .showImageOnFail(R.drawable.ic_empty_music2)
                            .resetViewBeforeLoading(true)
                            .build(), new ImageLoadingListener() {
                        @Override
                        public void onLoadingStarted(String imageUri, View view) {

                        }

                        @Override
                        public void onLoadingFailed(String imageUri, View view, FailReason failReason) {
                            Bitmap failedBitmap = ImageLoader.getInstance().loadImageSync("drawable://" + R.drawable.ic_empt
                            if (getActivity() != null)
                                new setBlurredAlbumArt().execute(failedBitmap);
                        }

                        @Override
                        public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                            if (getActivity() != null)
                                new setBlurredAlbumArt().execute(loadedImage);

                        }

                        @Override
                        public void onLoadingCancelled(String imageUri, View view) {

                        }
                    });
        }
        duetoplaypause = false;
        mProgress.setMax((int) MusicPlayer.duration());
        mSeekBar.setMax((int) MusicPlayer.duration());
        mProgress.postDelayed(mUpdateProgress, 10);
    }

    @Override
    public void onStart() {
        super.onStart();
    }

    @Override
    public void onStop() {
        super.onStop();

    }

    @Override
    public void onResume() {
        super.onResume();
        topContainer = rootView.findViewById(R.id.topContainer);
        fragmentPaused = false;
        if (mProgress != null)
            mProgress.postDelayed(mUpdateProgress, 10);

    }

    public void updateState() {
        if (MusicPlayer.isPlaying()) {
            if (!mPlayPause.isPlayed()) {
                mPlayPause.setPlayed(true);
                mPlayPause.startAnimation();
            }
            if (!mPlayPauseExpanded.isPlayed()) {
                mPlayPauseExpanded.setPlayed(true);
                mPlayPauseExpanded.startAnimation();
            }
        } else {
            if (mPlayPause.isPlayed()) {
                mPlayPause.setPlayed(false);
                mPlayPause.startAnimation();
            }
            if (mPlayPauseExpanded.isPlayed()) {
                mPlayPauseExpanded.setPlayed(false);
                mPlayPauseExpanded.startAnimation();
            }
        }
    }

    public void restartLoader() {

    }

    public void onPlaylistChanged() {

    }

    public void onMetaChanged() {
        updateNowplayingCard();
        updateState();
    }

    private class setBlurredAlbumArt extends AsyncTask<Bitmap, Void, Drawable> {

        @Override
        protected Drawable doInBackground(Bitmap... loadedImage) {
            Drawable drawable = null;
            try {
                drawable = ImageUtils.createBlurredImageFromBitmap(loadedImage[0], getActivity(), 6);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return drawable;
        }

        @Override
        protected void onPostExecute(Drawable result) {
            if (result != null) {
                if (mBlurredArt.getDrawable() != null) {
                    final TransitionDrawable td =
                            new TransitionDrawable(new Drawable[]{
                                    mBlurredArt.getDrawable(),
                                    result
                            });
                    mBlurredArt.setImageDrawable(td);
                    td.startTransition(400);

                } else {
                    mBlurredArt.setImageDrawable(result);
                }
            }
        }

        @Override
        protected void onPreExecute() {
        }
    }


}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.subfragments;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentStatePagerAdapter;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.naman14.timber.R;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.widgets.MultiViewPager;

public class StyleSelectorFragment extends Fragment {

    public String ACTION = "action";
    private FragmentStatePagerAdapter adapter;
    private MultiViewPager pager;
    private SubStyleSelectorFragment selectorFragment;
    private SharedPreferences preferences;

    public static StyleSelectorFragment newInstance(String what) {
        StyleSelectorFragment fragment = new StyleSelectorFragment();
        Bundle bundle = new Bundle();
        bundle.putString(Constants.SETTINGS_STYLE_SELECTOR_WHAT, what);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            ACTION = getArguments().getString(Constants.SETTINGS_STYLE_SELECTOR_WHAT);
        }
        preferences = getActivity().getSharedPreferences(Constants.FRAGMENT_ID, Context.MODE_PRIVATE);
    }


    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_style_selector, container, false);

        if (ACTION.equals(Constants.SETTINGS_STYLE_SELECTOR_NOWPLAYING)) {

        }
        pager = (MultiViewPager) rootView.findViewById(R.id.pager);

        adapter = new FragmentStatePagerAdapter(getChildFragmentManager()) {

            @Override
            public int getCount() {
                return 6;
            }

            @Override
            public Fragment getItem(int position) {
                selectorFragment = SubStyleSelectorFragment.newInstance(position, ACTION);
                return selectorFragment;
            }

            @Override
            public int getItemPosition(Object object) {
                return POSITION_NONE;
            }
        };
        pager.setAdapter(adapter);
        scrollToCurrentStyle();

        return rootView;
    }

    public void updateCurrentStyle() {
        if (selectorFragment != null) {
            adapter.notifyDataSetChanged();
            scrollToCurrentStyle();
        }

    }

    public void scrollToCurrentStyle() {
        String fragmentID = preferences.getString(Constants.NOWPLAYING_FRAGMENT_ID, Constants.TIMBER3);
        pager.setCurrentItem(NavigationUtils.getIntForCurrentNowplaying(fragmentID));
    }

}
```

```
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.subfragments;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.afollestad.materialdialogs.DialogAction;
import com.afollestad.materialdialogs.MaterialDialog;
import com.naman14.timber.R;
import com.naman14.timber.activities.DonateActivity;
import com.naman14.timber.utils.Constants;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.PreferencesUtility;

public class SubStyleSelectorFragment extends Fragment {

    private static final String ARG_PAGE_NUMBER = "pageNumber";
    private static final String WHAT = "what";
    private SharedPreferences.Editor editor;
    private SharedPreferences preferences;
    private LinearLayout currentStyle;
    private View foreground;
    private ImageView styleImage, imgLock;

    public static SubStyleSelectorFragment newInstance(int pageNumber, String what) {
        SubStyleSelectorFragment fragment = new SubStyleSelectorFragment();
        Bundle bundle = new Bundle();
        bundle.putInt(ARG_PAGE_NUMBER, pageNumber);
        bundle.putString(WHAT, what);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container,
                             @Nullable Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_style_selector_pager, container, false);

        TextView styleName = (TextView) rootView.findViewById(R.id.style_name);
        styleName.setText(String.valueOf(getArguments().getInt(ARG_PAGE_NUMBER) + 1));
        preferences = getActivity().getSharedPreferences(Constants.FRAGMENT_ID, Context.MODE_PRIVATE);

        styleImage = (ImageView) rootView.findViewById(R.id.style_image);
        imgLock = (ImageView) rootView.findViewById(R.id.img_lock);

        styleImage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (getArguments().getInt(ARG_PAGE_NUMBER) >= 4) {
                    if (isUnlocked()) {
                        setPreferences();
                    } else {
                        showPurchaseDialog();
                    }
                } else
                    setPreferences();
            }
        });

        switch (getArguments().getInt(ARG_PAGE_NUMBER)) {
            case 0:
                styleImage.setImageResource(R.drawable.timber_1_nowplaying_x);
                break;
            case 1:
                styleImage.setImageResource(R.drawable.timber_2_nowplaying_x);
                break;
            case 2:
                styleImage.setImageResource(R.drawable.timber_3_nowplaying_x);
                break;
            case 3:
                styleImage.setImageResource(R.drawable.timber_4_nowplaying_x);
                break;
            case 4:
                styleImage.setImageResource(R.drawable.timber_5_nowplaying_x);
                break;
            case 5:
                styleImage.setImageResource(R.drawable.timber_6_nowplaying_x);
                break;
        }

        currentStyle = (LinearLayout) rootView.findViewById(R.id.currentStyle);
        foreground = rootView.findViewById(R.id.foreground);

        setCurrentStyle();

        return rootView;
    }

    private boolean isUnlocked() {
        return getActivity() != null && PreferencesUtility.getInstance(getActivity()).fullUnlocked();
    }

    @Override
    public void onResume() {
        super.onResume();
        updateLockedStatus();
    }

    private void updateLockedStatus() {
        if (getArguments().getInt(ARG_PAGE_NUMBER) >= 4 && !isUnlocked()) {
            imgLock.setVisibility(View.VISIBLE);
            foreground.setVisibility(View.VISIBLE);
        }
        else {
            imgLock.setVisibility(View.GONE);
            foreground.setVisibility(View.GONE);
        }
    }
    private void showPurchaseDialog() {
        MaterialDialog dialog = new MaterialDialog.Builder(getActivity())
                .title("Purchase")
                .content("This now playing style is available after a one time purchase of any amount. Support development a
                .positiveText("Support development")
                .neutralText("Restore purchases")
                .onPositive(new MaterialDialog.SingleButtonCallback() {
                    @Override
```

```java
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        startActivity(new Intent(getActivity(), DonateActivity.class));
                        dialog.dismiss();
                    }
                }).onNeutral(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        Intent intent = new Intent(getActivity(), DonateActivity.class);
                        intent.putExtra("title", "Restoring purchases..");
                        intent.setAction("restore");
                        startActivity(intent);
                        dialog.dismiss();
                    }
                })
                .show();
    }

    public void setCurrentStyle() {
        String fragmentID = preferences.getString(Constants.NOWPLAYING_FRAGMENT_ID, Constants.TIMBER3);

        if (getArguments().getInt(ARG_PAGE_NUMBER) == NavigationUtils.getIntForCurrentNowplaying(fragmentID)) {
            currentStyle.setVisibility(View.VISIBLE);
            foreground.setVisibility(View.VISIBLE);
        } else {
            currentStyle.setVisibility(View.GONE);
            foreground.setVisibility(View.GONE);
        }

    }

    private void setPreferences() {

        if (getArguments().getString(WHAT).equals(Constants.SETTINGS_STYLE_SELECTOR_NOWPLAYING)) {
            editor = getActivity().getSharedPreferences(Constants.FRAGMENT_ID, Context.MODE_PRIVATE).edit();
            editor.putString(Constants.NOWPLAYING_FRAGMENT_ID, getStyleForPageNumber());
            editor.apply();
            if (getActivity() != null)
                PreferencesUtility.getInstance(getActivity()).setNowPlayingThemeChanged(true);
            setCurrentStyle();
            ((StyleSelectorFragment) getParentFragment()).updateCurrentStyle();
        }
    }

    private String getStyleForPageNumber() {
        switch (getArguments().getInt(ARG_PAGE_NUMBER)) {
            case 0:
                return Constants.TIMBER1;
            case 1:
                return Constants.TIMBER2;
            case 2:
                return Constants.TIMBER3;
            case 3:
                return Constants.TIMBER4;
            case 4:
                return Constants.TIMBER5;
            case 5:
                return Constants.TIMBER6;
            default:
                return Constants.TIMBER3;
        }
    }


}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\TimelyView.java

```
/*
* Copyright 2014 Adnan A M.
* Copyright 2015 Naman Dwivedi.

* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at

*   http://www.apache.org/licenses/LICENSE-2.0

* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.timely;

import android.animation.ObjectAnimator;
import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.util.Property;
import android.view.View;

import com.naman14.timber.R;
import com.naman14.timber.timely.animation.TimelyEvaluator;
import com.naman14.timber.timely.model.NumberUtils;

public class TimelyView extends View {
    private static final float RATIO = 1f;
    private static final Property<TimelyView, float[][]> CONTROL_POINTS_PROPERTY = new Property<TimelyView, float[][]>(float
        @Override
        public float[][] get(TimelyView object) {
            return object.getControlPoints();
        }

        @Override
        public void set(TimelyView object, float[][] value) {
            object.setControlPoints(value);
        }
    };
    private Paint mPaint = null;
    private Path mPath = null;
    private float[][] controlPoints = null;

    private int textColor;

    public TimelyView(Context context) {
        super(context);
        init();
    }

    public TimelyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        TypedArray typedArray = context.obtainStyledAttributes(attrs, R.styleable.TimelyView);
        textColor = typedArray.getColor(R.styleable.TimelyView_text_color, Color.BLACK);
        init();
    }

    public TimelyView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        init();
    }
```

```java
    public float[][] getControlPoints() {
        return controlPoints;
    }

    public void setControlPoints(float[][] controlPoints) {
        this.controlPoints = controlPoints;
        invalidate();
    }

    public ObjectAnimator animate(int start, int end) {
        float[][] startPoints = NumberUtils.getControlPointsFor(start);
        float[][] endPoints = NumberUtils.getControlPointsFor(end);

        return ObjectAnimator.ofObject(this, CONTROL_POINTS_PROPERTY, new TimelyEvaluator(), startPoints, endPoints);
    }

    public ObjectAnimator animate(int end) {
        float[][] startPoints = NumberUtils.getControlPointsFor(-1);
        float[][] endPoints = NumberUtils.getControlPointsFor(end);

        return ObjectAnimator.ofObject(this, CONTROL_POINTS_PROPERTY, new TimelyEvaluator(), startPoints, endPoints);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        if (controlPoints == null) return;

        int length = controlPoints.length;

        int height = getMeasuredHeight();
        int width = getMeasuredWidth();

        float minDimen = height > width ? width : height;

        mPath.reset();
        mPath.moveTo(minDimen * controlPoints[0][0], minDimen * controlPoints[0][1]);
        for (int i = 1; i < length; i += 3) {
            mPath.cubicTo(minDimen * controlPoints[i][0], minDimen * controlPoints[i][1],
                    minDimen * controlPoints[i + 1][0], minDimen * controlPoints[i + 1][1],
                    minDimen * controlPoints[i + 2][0], minDimen * controlPoints[i + 2][1]);
        }
        canvas.drawPath(mPath, mPaint);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);

        int width = getMeasuredWidth();
        int height = getMeasuredHeight();
        int widthWithoutPadding = width - getPaddingLeft() - getPaddingRight();
        int heigthWithoutPadding = height - getPaddingTop() - getPaddingBottom();

        int maxWidth = (int) (heigthWithoutPadding * RATIO);
        int maxHeight = (int) (widthWithoutPadding / RATIO);

        if (widthWithoutPadding > maxWidth) {
            width = maxWidth + getPaddingLeft() + getPaddingRight();
        } else {
            height = maxHeight + getPaddingTop() + getPaddingBottom();
        }

        setMeasuredDimension(width, height);
    }

    private void init() {
        // A new paint with the style as stroke.
        mPaint = new Paint();
        mPaint.setAntiAlias(true);
        mPaint.setColor(textColor);
        mPaint.setStrokeWidth(5.0f);
        mPaint.setStyle(Paint.Style.STROKE);
        mPath = new Path();
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\animation\TimelyEvaluato

```java
/*
* Copyright 2014 Adnan A M.

* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at

*   http://www.apache.org/licenses/LICENSE-2.0

* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.timely.animation;

import android.animation.TypeEvaluator;

public class TimelyEvaluator implements TypeEvaluator<float[][]> {
    private float[][] _cachedPoints = null;

    @Override
    public float[][] evaluate(float fraction, float[][] startValue, float[][] endValue) {
        int pointsCount = startValue.length;
        initCache(pointsCount);

        for (int i = 0; i < pointsCount; i++) {
            _cachedPoints[i][0] = startValue[i][0] + fraction * (endValue[i][0] - startValue[i][0]);
            _cachedPoints[i][1] = startValue[i][1] + fraction * (endValue[i][1] - startValue[i][1]);
        }

        return _cachedPoints;
    }

    private void initCache(int pointsCount) {
        if (_cachedPoints == null || _cachedPoints.length != pointsCount) {
            _cachedPoints = new float[pointsCount][2];
        }
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\NumberUtils.java

```java
package com.naman14.timber.timely.model;

import com.naman14.timber.timely.model.number.*;

import java.security.InvalidParameterException;

public class NumberUtils {

    public static float[][] getControlPointsFor(int start) {
        switch (start) {
            case (-1):
                return Null.getInstance().getControlPoints();
            case 0:
                return Zero.getInstance().getControlPoints();
            case 1:
                return One.getInstance().getControlPoints();
            case 2:
                return Two.getInstance().getControlPoints();
            case 3:
                return Three.getInstance().getControlPoints();
            case 4:
                return Four.getInstance().getControlPoints();
            case 5:
                return Five.getInstance().getControlPoints();
            case 6:
                return Six.getInstance().getControlPoints();
            case 7:
                return Seven.getInstance().getControlPoints();
            case 8:
                return Eight.getInstance().getControlPoints();
            case 9:
                return Nine.getInstance().getControlPoints();
            default:
                throw new InvalidParameterException("Unsupported number requested");
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\core\Figure.java

```java
/*
* Copyright 2014 Adnan A M.

* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at

*   http://www.apache.org/licenses/LICENSE-2.0

* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.naman14.timber.timely.model.core;

/**
 * Model class for cubic bezier figure
 */
public abstract class Figure {
    public static final int NO_VALUE = -1;

    protected int pointsCount = NO_VALUE;

    //A chained sequence of points P0,P1,P2,P3/0,P1,P2,P3/0,...
    protected float[][] controlPoints = null;

    protected Figure(float[][] controlPoints) {
        this.controlPoints = controlPoints;
        this.pointsCount = (controlPoints.length + 2) / 3;
    }

    public int getPointsCount() {
        return pointsCount;
    }

    public float[][] getControlPoints() {
        return controlPoints;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Eight.java

```
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Eight extends Figure {
    private static final float[][] POINTS = {
            {0.558011049723757f, 0.530386740331492f}, {0.243093922651934f, 0.524861878453039f}, {0.243093922651934f, 0.10497
            {0.558011049723757f, 0.104972375690608f}, {0.850828729281768f, 0.104972375690608f}, {0.850828729281768f, 0.53038
            {0.558011049723757f, 0.530386740331492f}, {0.243093922651934f, 0.530386740331492f}, {0.198895027624309f, 0.98895
            {0.558011049723757f, 0.988950276243094f}, {0.850828729281768f, 0.988950276243094f}, {0.850828729281768f, 0.53038
            {0.558011049723757f, 0.530386740331492f}
    };

    private static Eight INSTANCE = new Eight();

    protected Eight() {
        super(POINTS);
    }

    public static Eight getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Five.java

```java
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Five extends Figure {
    private static final float[][] POINTS = {
            {0.806629834254144f, 0.110497237569061f}, {0.502762430939227f, 0.110497237569061f}, {0.502762430939227f, 0.11049
            {0.502762430939227f, 0.110497237569061f}, {0.397790055248619f, 0.430939226519337f}, {0.397790055248619f, 0.43093
            {0.397790055248619f, 0.430939226519337f}, {0.535911602209945f, 0.364640883977901f}, {0.801104972375691f, 0.46961
            {0.801104972375691f, 0.712707182320442f}, {0.773480662983425f, 1.01104972375691f}, {0.375690607734807f, 1.093922
            {0.248618784530387f, 0.850828729281768f}
    };

    private static Five INSTANCE = new Five();

    protected Five() {
        super(POINTS);
    }

    public static Five getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Four.java

```
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Four extends Figure {
    private static final float[][] POINTS = {
            {0.856353591160221f, 0.806629834254144f}, {0.856353591160221f, 0.806629834254144f}, {0.237569060773481f, 0.80662
            {0.237569060773481f, 0.806629834254144f}, {0.237569060773481f, 0.806629834254144f}, {0.712707182320442f, 0.13812
            {0.712707182320442f, 0.138121546961326f}, {0.712707182320442f, 0.138121546961326f}, {0.712707182320442f, 0.80662
            {0.712707182320442f, 0.806629834254144f}, {0.712707182320442f, 0.806629834254144f}, {0.712707182320442f, 0.98895
            {0.712707182320442f, 0.988950276243094f}

    };

    private static Four INSTANCE = new Four();

    protected Four() {
        super(POINTS);
    }

    public static Four getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Nine.java

```
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Nine extends Figure {
    private static final float[][] POINTS = {
            {0.80939226519337f, 0.552486187845304f}, {0.685082872928177f, 0.751381215469613f}, {0.298342541436464f, 0.740331
            {0.259668508287293f, 0.408839779005525f}, {0.232044198895028f, 0.0441988950276243f}, {0.81767955801105f, -0.0441
            {0.850828729281768f, 0.408839779005525f}, {0.839779005524862f, 0.596685082872928f}, {0.712707182320442f, 0.66850
            {0.497237569060773f, 0.994475138121547f}, {0.497237569060773f, 0.994475138121547f}, {0.497237569060773f, 0.99447
            {0.497237569060773f, 0.994475138121547f}
    };

    private static Nine INSTANCE = new Nine();

    protected Nine() {
        super(POINTS);
    }

    public static Nine getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Null.java

```java
package com.naman14.timber.timely.model.number;


import com.naman14.timber.timely.model.core.Figure;

public class Null extends Figure {
    private static final float[][] POINTS = {
            {0.5f, 0.5f}, {0.5f, 0.5f}, {0.5f, 0.5f},
            {0.5f, 0.5f}, {0.5f, 0.5f}, {0.5f, 0.5f},
            {0.5f, 0.5f}, {0.5f, 0.5f}, {0.5f, 0.5f},
            {0.5f, 0.5f}, {0.5f, 0.5f}, {0.5f, 0.5f},
            {0.5f, 0.5f}
    };

    private static final Null INSTANCE = new Null();

    protected Null() {
        super(POINTS);
    }

    public static Null getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\One.java

```
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class One extends Figure {
    private static final float[][] POINTS = {
            {0.425414364640884f, 0.113259668508287f}, {0.425414364640884f, 0.113259668508287f}, {0.577348066298343f, 0.11325
            {0.577348066298343f, 0.113259668508287f}, {0.577348066298343f, 0.113259668508287f}, {0.577348066298343f, 1f},
            {0.577348066298343f, 1f}, {0.577348066298343f, 1f}, {0.577348066298343f, 1f},
            {0.577348066298343f, 1f}, {0.577348066298343f, 1f}, {0.577348066298343f, 1f},
            {0.577348066298343f, 1f}
    };

    private static One INSTANCE = new One();

    protected One() {
        super(POINTS);
    }

    public static One getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Seven.java

```java
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Seven extends Figure {
    private static final float[][] POINTS = {
            {0.259668508287293f, 0.116022099447514f}, {0.259668508287293f, 0.116022099447514f}, {0.87292817679558f, 0.116022
            {0.87292817679558f, 0.116022099447514f}, {0.87292817679558f, 0.116022099447514f}, {0.7f, 0.422099447513812f},
            {0.7f, 0.422099447513812f}, {0.7f, 0.422099447513812f}, {0.477348066298343f, 0.733149171270718f},
            {0.477348066298343f, 0.733149171270718f}, {0.477348066298343f, 0.733149171270718f}, {0.25414364640884f, 1f},
            {0.25414364640884f, 1f}
    };

    private static Seven INSTANCE = new Seven();

    protected Seven() {
        super(POINTS);
    }

    public static Seven getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Six.java

```
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Six extends Figure {
    private static final float[][] POINTS = {
            {0.607734806629834f, 0.110497237569061f}, {0.607734806629834f, 0.110497237569061f}, {0.607734806629834f, 0.11049
            {0.607734806629834f, 0.110497237569061f}, {0.392265193370166f, 0.43646408839779f}, {0.265193370165746f, 0.508287
            {0.25414364640884f, 0.696132596685083f}, {0.287292817679558f, 1.13017127071823f}, {0.87292817679558f, 1.06077348
            {0.845303867403315f, 0.696132596685083f}, {0.806629834254144f, 0.364640883977901f}, {0.419889502762431f, 0.35359
            {0.295580110497238f, 0.552486187845304f}
    };

    private static Six INSTANCE = new Six();

    protected Six() {
        super(POINTS);
    }

    public static Six getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Three.java

```java
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Three extends Figure {
    private static final float[][] POINTS = {
            {0.361878453038674f, 0.298342541436464f}, {0.348066298342541f, 0.149171270718232f}, {0.475138121546961f, 0.09944
            {0.549723756906077f, 0.0994475138121547f}, {0.861878453038674f, 0.0994475138121547f}, {0.806629834254144f, 0.530
            {0.549723756906077f, 0.530386740331492f}, {0.87292817679558f, 0.530386740331492f}, {0.828729281767956f, 0.994475
            {0.552486187845304f, 0.994475138121547f}, {0.298342541436464f, 0.994475138121547f}, {0.30939226519337f, 0.828729
            {0.312154696132597f, 0.790055248618785f}
    };

    private static Three INSTANCE = new Three();

    protected Three() {
        super(POINTS);
    }

    public static Three getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Two.java

```java
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Two extends Figure {
    private static final float[][] POINTS = {
            {0.30939226519337f, 0.331491712707182f}, {0.325966850828729f, 0.0110497237569061f}, {0.790055248618785f, 0.02209
            {0.798342541436464f, 0.337016574585635f}, {0.798342541436464f, 0.430939226519337f}, {0.718232044198895f, 0.54143
            {0.596685082872928f, 0.674033149171271f}, {0.519337016574586f, 0.762430939226519f}, {0.408839779005525f, 0.85635
            {0.314917127071823f, 0.977900552486188f}, {0.314917127071823f, 0.977900552486188f}, {0.812154696132597f, 0.97790
            {0.812154696132597f, 0.977900552486188f}
    };

    private static Two INSTANCE = new Two();

    protected Two() {
        super(POINTS);
    }

    public static Two getInstance() {
        return INSTANCE;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\timely\model\number\Zero.java

```java
package com.naman14.timber.timely.model.number;

import com.naman14.timber.timely.model.core.Figure;

public class Zero extends Figure {
    private static final float[][] POINTS = {
            {0.24585635359116f, 0.552486187845304f}, {0.24585635359116f, 0.331491712707182f}, {0.370165745856354f, 0.0994475
            {0.552486187845304f, 0.0994475138121547f}, {0.734806629834254f, 0.0994475138121547f}, {0.861878453038674f, 0.331
            {0.861878453038674f, 0.552486187845304f}, {0.861878453038674f, 0.773480662983425f}, {0.734806629834254f, 0.99447
            {0.552486187845304f, 0.994475138121547f}, {0.370165745856354f, 0.994475138121547f}, {0.24585635359116f, 0.773480
            {0.24585635359116f, 0.552486187845304f}
    };

    private static Zero INSTANCE = new Zero();

    protected Zero() {
        super(POINTS);
    }

    public static Zero getInstance() {
        return INSTANCE;
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.transition;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.animation.TimeInterpolator;
import android.annotation.TargetApi;
import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Path;
import android.graphics.Rect;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.ColorDrawable;
import android.graphics.drawable.Drawable;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.OvalShape;
import android.transition.Transition;
import android.transition.TransitionValues;
import android.util.ArrayMap;
import android.util.AttributeSet;
import android.view.View;
import android.view.ViewAnimationUtils;
import android.view.ViewGroup;

import com.naman14.timber.R;

import java.util.ArrayList;

@TargetApi(21)
public class PlayTransition extends Transition {
    private static final String PROPERTY_BOUNDS = "circleTransition:bounds";
    private static final String PROPERTY_POSITION = "circleTransition:position";
    private static final String PROPERTY_IMAGE = "circleTransition:image";
    private static final String[] TRANSITION_PROPERTIES = {
            PROPERTY_BOUNDS,
            PROPERTY_POSITION,
    };

    private int mColor = Color.parseColor("#6c1622");

    public PlayTransition() {
    }

    public PlayTransition(Context context, AttributeSet attrs) {
        super(context, attrs);
        TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.PlayTransition);
        setColor(a.getColor(R.styleable.PlayTransition_colorCT, getColor()));
        a.recycle();
    }

    public void setColor(int color) {
        mColor = color;
    }

    public int getColor() {
        return mColor;
    }

    @Override
    public String[] getTransitionProperties() {
        return TRANSITION_PROPERTIES;
    }

    private void captureValues(TransitionValues transitionValues) {
        final View view = transitionValues.view;
        transitionValues.values.put(PROPERTY_BOUNDS, new Rect(
                view.getLeft(), view.getTop(), view.getRight(), view.getBottom()
        ));
        int[] position = new int[2];
        transitionValues.view.getLocationInWindow(position);
        transitionValues.values.put(PROPERTY_POSITION, position);
    }

    @Override
    public void captureEndValues(TransitionValues transitionValues) {
        final View view = transitionValues.view;
        if (view.getWidth() <= 0 || view.getHeight() <= 0) {
            return;
        }
        captureValues(transitionValues);
    }

    @Override
    public void captureStartValues(TransitionValues transitionValues) {
        final View view = transitionValues.view;
        if (view.getWidth() <= 0 || view.getHeight() <= 0) {
            return;
        }
        captureValues(transitionValues);
        Bitmap bitmap = Bitmap.createBitmap(view.getWidth(), view.getHeight(),
                Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(bitmap);
        view.draw(canvas);
        transitionValues.values.put(PROPERTY_IMAGE, bitmap);
    }

    @Override
    public Animator createAnimator(final ViewGroup sceneRoot, TransitionValues startValues,
                                   final TransitionValues endValues) {
        if (startValues == null || endValues == null) {
            return null;
        }
        Rect startBounds = (Rect) startValues.values.get(PROPERTY_BOUNDS);
        Rect endBounds = (Rect) endValues.values.get(PROPERTY_BOUNDS);
        if (startBounds == null || endBounds == null || startBounds.equals(endBounds)) {
            return null;
        }
        Bitmap startImage = (Bitmap) startValues.values.get(PROPERTY_IMAGE);
        Drawable startBackground = new BitmapDrawable(sceneRoot.getContext().getResources(), startImage);
        final View startView = addViewToOverlay(sceneRoot, startImage.getWidth(),
                startImage.getHeight(), startBackground);
        Drawable shrinkingBackground = new ColorDrawable(mColor);
        final View shrinkingView = addViewToOverlay(sceneRoot, startImage.getWidth(),
                startImage.getHeight(), shrinkingBackground);

        int[] sceneRootLoc = new int[2];
        sceneRoot.getLocationInWindow(sceneRootLoc);
        int[] startLoc = (int[]) startValues.values.get(PROPERTY_POSITION);
        int startTranslationX = startLoc[0] - sceneRootLoc[0];
        int startTranslationY = startLoc[1] - sceneRootLoc[1];

        startView.setTranslationX(startTranslationX);
        startView.setTranslationY(startTranslationY);
        shrinkingView.setTranslationX(startTranslationX);
        shrinkingView.setTranslationY(startTranslationY);

        final View endView = endValues.view;
        float startRadius = calculateMaxRadius(shrinkingView);
        int minRadius = Math.min(calculateMinRadius(shrinkingView), calculateMinRadius(endView));

        ShapeDrawable circleBackground = new ShapeDrawable(new OvalShape());
        circleBackground.getPaint().setColor(mColor);
        final View circleView = addViewToOverlay(sceneRoot, minRadius * 2, minRadius * 2,
                circleBackground);
        float circleStartX = startLoc[0] - sceneRootLoc[0] +
                ((startView.getWidth() - circleView.getWidth()) / 2);
        float circleStartY = startLoc[1] - sceneRootLoc[1] +
                ((startView.getHeight() - circleView.getHeight()) / 2);
        circleView.setTranslationX(circleStartX);
        circleView.setTranslationY(circleStartY);

        circleView.setVisibility(View.INVISIBLE);
        shrinkingView.setAlpha(0f);
        endView.setAlpha(0f);

        Animator shrinkingAnimator = createCircularReveal(shrinkingView, startRadius, minRadius);
        shrinkingAnimator.addListener(new AnimatorListenerAdapter() {

            @Override
            public void onAnimationEnd(Animator animation) {
                shrinkingView.setVisibility(View.INVISIBLE);
                startView.setVisibility(View.INVISIBLE);
                circleView.setVisibility(View.VISIBLE);
            }
        });

        Animator startAnimator = createCircularReveal(startView, startRadius, minRadius);
        Animator fadeInAnimator = ObjectAnimator.ofFloat(shrinkingView, View.ALPHA, 0, 1);

        AnimatorSet shrinkFadeSet = new AnimatorSet();
        shrinkFadeSet.playTogether(shrinkingAnimator, startAnimator,
                fadeInAnimator);

        int[] endLoc = (int[]) endValues.values.get(PROPERTY_POSITION);
        float circleEndX = endLoc[0] - sceneRootLoc[0] +
                ((endView.getWidth() - circleView.getWidth()) / 2);
        float circleEndY = endLoc[1] - sceneRootLoc[1] +
                ((endView.getHeight() - circleView.getHeight()) / 2);
        Path circlePath = getPathMotion().getPath(circleStartX, circleStartY, circleEndX,
                circleEndY);
        Animator circleAnimator = ObjectAnimator.ofFloat(circleView, View.TRANSLATION_X,
                View.TRANSLATION_Y, circlePath);

        final View growingView = addViewToOverlay(sceneRoot, endView.getWidth(),
                endView.getHeight(), shrinkingBackground);
        growingView.setVisibility(View.INVISIBLE);
        float endTranslationX = endLoc[0] - sceneRootLoc[0];
        float endTranslationY = endLoc[1] - sceneRootLoc[1];
        growingView.setTranslationX(endTranslationX);
        growingView.setTranslationY(endTranslationY);

        float endRadius = calculateMaxRadius(endView);

        circleAnimator.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                circleView.setVisibility(View.INVISIBLE);
                growingView.setVisibility(View.VISIBLE);
                endView.setAlpha(1f);
            }
        });
        Animator fadeOutAnimator = ObjectAnimator.ofFloat(growingView, View.ALPHA, 1, 0);
        Animator endAnimator = createCircularReveal(endView, minRadius, endRadius);
        Animator growingAnimator = createCircularReveal(growingView, minRadius, endRadius);

        growingAnimator.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                sceneRoot.getOverlay().remove(startView);
                sceneRoot.getOverlay().remove(shrinkingView);
                sceneRoot.getOverlay().remove(circleView);
                sceneRoot.getOverlay().remove(growingView);
            }
        });
        AnimatorSet growingFadeSet = new AnimatorSet();
        growingFadeSet.playTogether(fadeOutAnimator, endAnimator, growingAnimator);

        AnimatorSet animatorSet = new AnimatorSet();
        animatorSet.playSequentially(shrinkFadeSet, circleAnimator, growingFadeSet);
        return animatorSet;
    }

    private View addViewToOverlay(ViewGroup sceneRoot, int width, int height, Drawable background) {
        View view = new NoOverlapView(sceneRoot.getContext());
        view.setBackground(background);
        int widthSpec = View.MeasureSpec.makeMeasureSpec(width, View.MeasureSpec.EXACTLY);
        int heightSpec = View.MeasureSpec.makeMeasureSpec(height, View.MeasureSpec.EXACTLY);
        view.measure(widthSpec, heightSpec);
        view.layout(0, 0, width, height);
        sceneRoot.getOverlay().add(view);
        return view;
    }

    private Animator createCircularReveal(View view, float startRadius, float endRadius) {
        int centerX = view.getWidth() / 2;
        int centerY = view.getHeight() / 2;

        Animator reveal = ViewAnimationUtils.createCircularReveal(view, centerX, centerY,
                startRadius, endRadius);
        return new NoPauseAnimator(reveal);
    }

    static float calculateMaxRadius(View view) {
        float widthSquared = view.getWidth() * view.getWidth();
        float heightSquared = view.getHeight() * view.getHeight();
        float radius = (float) Math.sqrt(widthSquared + heightSquared) / 2;
        return radius;
    }

    static int calculateMinRadius(View view) {
        return Math.min(view.getWidth() / 2, view.getHeight() / 2);
    }

    private static class NoPauseAnimator extends Animator {
        private final Animator mAnimator;
        private final ArrayMap<AnimatorListener, AnimatorListener> mListeners =
                new ArrayMap<AnimatorListener, AnimatorListener>();

        public NoPauseAnimator(Animator animator) {
            mAnimator = animator;
        }

        @Override
        public void addListener(AnimatorListener listener) {
            AnimatorListener wrapper = new AnimatorListenerWrapper(this, listener);
            if (!mListeners.containsKey(listener)) {
                mListeners.put(listener, wrapper);
                mAnimator.addListener(wrapper);
            }
        }
        @Override
        public void cancel() {
            mAnimator.cancel();
        }

        @Override
        public void end() {
            mAnimator.end();
        }

        @Override
        public long getDuration() {
            return mAnimator.getDuration();
        }

        @Override
        public TimeInterpolator getInterpolator() {
            return mAnimator.getInterpolator();
        }

        @Override
        public ArrayList<AnimatorListener> getListeners() {
            return new ArrayList<AnimatorListener>(mListeners.keySet());
        }

        @Override
        public long getStartDelay() {
            return mAnimator.getStartDelay();
        }

        @Override
        public boolean isPaused() {
            return mAnimator.isPaused();
        }

        @Override
        public boolean isRunning() {
            return mAnimator.isRunning();
        }

        @Override
        public boolean isStarted() {
            return mAnimator.isStarted();
        }

        @Override
        public void removeAllListeners() {
            mListeners.clear();
            mAnimator.removeAllListeners();
        }

        @Override
        public void removeListener(AnimatorListener listener) {
            AnimatorListener wrapper = mListeners.get(listener);
            if (wrapper != null) {
                mListeners.remove(listener);
                mAnimator.removeListener(wrapper);
            }
        }

        @Override
        public Animator setDuration(long durationMS) {
            mAnimator.setDuration(durationMS);
            return this;
        }

        @Override
        public void setInterpolator(TimeInterpolator timeInterpolator) {
            mAnimator.setInterpolator(timeInterpolator);
        }
```

```java
        @Override
        public void setStartDelay(long delayMS) {
            mAnimator.setStartDelay(delayMS);
        }

        @Override
        public void setTarget(Object target) {
            mAnimator.setTarget(target);
        }

        @Override
        public void setupEndValues() {
            mAnimator.setupEndValues();
        }

        @Override
        public void setupStartValues() {
            mAnimator.setupStartValues();
        }

        @Override
        public void start() {
            mAnimator.start();
        }
    }

    private static class AnimatorListenerWrapper implements Animator.AnimatorListener {
        private final Animator mAnimator;
        private final Animator.AnimatorListener mListener;

        public AnimatorListenerWrapper(Animator animator, Animator.AnimatorListener listener) {
            mAnimator = animator;
            mListener = listener;
        }

        @Override
        public void onAnimationStart(Animator animator) {
            mListener.onAnimationStart(mAnimator);
        }

        @Override
        public void onAnimationEnd(Animator animator) {
            mListener.onAnimationEnd(mAnimator);
        }

        @Override
        public void onAnimationCancel(Animator animator) {
            mListener.onAnimationCancel(mAnimator);
        }

        @Override
        public void onAnimationRepeat(Animator animator) {
            mListener.onAnimationRepeat(mAnimator);
        }
    }

    private static class NoOverlapView extends View {
        public NoOverlapView(Context context) {
            super(context);
        }

        @Override
        public boolean hasOverlappingRendering() {
            return false;
        }
    }
}
```

```java
package com.naman14.timber.utils;

import android.annotation.TargetApi;
import android.app.Activity;
import android.app.ActivityManager;
import android.content.res.ColorStateList;
import android.graphics.Color;
import android.graphics.drawable.BitmapDrawable;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.widget.DrawerLayout;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;

import com.afollestad.appthemeengine.Config;
import com.afollestad.appthemeengine.util.Util;

/**
 * Created by naman on 02/01/16.
 */
public class ATEUtils {

    public static void setStatusBarColor(Activity activity, String key, int color) {
        try {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                final Window window = activity.getWindow();
                if (Config.coloredStatusBar(activity, key))
                    window.setStatusBarColor(getStatusBarColor(color));
                else window.setStatusBarColor(Color.BLACK);
                if (Config.coloredNavigationBar(activity, key))
                    window.setNavigationBarColor(color);
                else window.setNavigationBarColor(Color.BLACK);
                applyTaskDescription(activity, key, color);
            }
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
                final View decorView = activity.getWindow().getDecorView();
                final int lightStatusMode = Config.lightStatusBarMode(activity, key);
                boolean lightStatusEnabled = false;
                switch (lightStatusMode) {
                    case Config.LIGHT_STATUS_BAR_OFF:
                    default:
                        break;
                    case Config.LIGHT_STATUS_BAR_ON:
                        lightStatusEnabled = true;
                        break;
                    case Config.LIGHT_STATUS_BAR_AUTO:
                        lightStatusEnabled = Util.isColorLight(color);
                        break;
                }

                final int systemUiVisibility = decorView.getSystemUiVisibility();
                if (lightStatusEnabled) {
                    decorView.setSystemUiVisibility(systemUiVisibility | View.SYSTEM_UI_FLAG_LIGHT_STATUS_BAR);
                } else {
                    decorView.setSystemUiVisibility(systemUiVisibility & ~View.SYSTEM_UI_FLAG_LIGHT_STATUS_BAR);
                }
            }

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                final int color2 = Config.coloredStatusBar(activity, key) ?
                        Color.TRANSPARENT : Color.BLACK;
                activity.getWindow().setStatusBarColor(color2);
            }
            if (Config.coloredStatusBar(activity, key))
                ((DrawerLayout) ((ViewGroup) activity.findViewById(android.R.id.content)).getChildAt(0)).setStatusBarBackgro
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @TargetApi(Build.VERSION_CODES.LOLLIPOP)
    private static void applyTaskDescription(@NonNull Activity activity, @Nullable String key, int color) {
        // Sets color of entry in the system recents page
        try {
            ActivityManager.TaskDescription td = new ActivityManager.TaskDescription(
                    (String) activity.getTitle(),
                    ((BitmapDrawable) activity.getApplicationInfo().loadIcon(activity.getPackageManager())).getBitmap(),
                    color);
            activity.setTaskDescription(td);
        } catch (Exception ignored) {

        }
    }

    public static int getStatusBarColor(int primaryColor) {
        float[] arrayOfFloat = new float[3];
        Color.colorToHSV(primaryColor, arrayOfFloat);
        arrayOfFloat[2] *= 0.9F;
        return Color.HSVToColor(arrayOfFloat);
    }

    public static void setFabBackgroundTint(FloatingActionButton fab, int color) {
        ColorStateList fabColorStateList = new ColorStateList(
                new int[][]{
                        new int[]{}
                },
                new int[]{
                        color,
                }
        );
        fab.setBackgroundTintList(fabColorStateList);
    }
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

public class Constants {

    public static final String NAVIGATE_LIBRARY = "navigate_library";
    public static final String NAVIGATE_PLAYLIST = "navigate_playlist";
    public static final String NAVIGATE_QUEUE = "navigate_queue";
    public static final String NAVIGATE_ALBUM = "navigate_album";
    public static final String NAVIGATE_ARTIST = "navigate_artist";
    public static final String NAVIGATE_NOWPLAYING = "navigate_nowplaying";
    public static final String NAVIGATE_LYRICS = "navigate_lyrics";

    public static final String NAVIGATE_PLAYLIST_RECENT = "navigate_playlist_recent";
    public static final String NAVIGATE_PLAYLIST_LASTADDED = "navigate_playlist_lastadded";
    public static final String NAVIGATE_PLAYLIST_TOPTRACKS = "navigate_playlist_toptracks";
    public static final String NAVIGATE_PLAYLIST_USERCREATED = "navigate_playlist";
    public static final String PLAYLIST_FOREGROUND_COLOR = "foreground_color";
    public static final String PLAYLIST_NAME = "playlist_name";

    public static final String ALBUM_ID = "album_id";
    public static final String ARTIST_ID = "artist_id";
    public static final String PLAYLIST_ID = "playlist_id";

    public static final String FRAGMENT_ID = "fragment_id";
    public static final String NOWPLAYING_FRAGMENT_ID = "nowplaying_fragment_id";

    public static final String WITH_ANIMATIONS = "with_animations";

    public static final String TIMBER1 = "timber1";
    public static final String TIMBER2 = "timber2";
    public static final String TIMBER3 = "timber3";
    public static final String TIMBER4 = "timber4";
    public static final String TIMBER5 = "timber5";
    public static final String TIMBER6 = "timber6";

    public static final String NAVIGATE_SETTINGS = "navigate_settings";
    public static final String NAVIGATE_SEARCH = "navigate_search";

    public static final String SETTINGS_STYLE_SELECTOR_NOWPLAYING = "style_selector_nowplaying";
    public static final String SETTINGS_STYLE_SELECTOR_ARTIST = "style_selector_artist";
    public static final String SETTINGS_STYLE_SELECTOR_ALBUM = "style_selector_album";
    public static final String SETTINGS_STYLE_SELECTOR_WHAT = "style_selector_what";

    public static final String SETTINGS_STYLE_SELECTOR = "settings_style_selector";

    public static final int PLAYLIST_VIEW_DEFAULT = 0;
    public static final int PLAYLIST_VIEW_LIST = 1;
    public static final int PLAYLIST_VIEW_GRID = 2;

    public static final int PLAYLIST_ALBUM_ART_TAG = 888;
    public static final int ACTION_DELETE_PLAYLIST = 111;


    public static final String ACTIVITY_TRANSITION = "activity_transition";

    public static final int CAST_SERVER_PORT = 8080;
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

import android.os.Build;
import android.support.v4.view.ViewCompat;
import android.support.v4.view.ViewPropertyAnimatorListener;
import android.support.v4.view.animation.FastOutSlowInInterpolator;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.Interpolator;

import com.naman14.timber.R;

public class FabAnimationUtils {

    private static final long DEFAULT_DURATION = 200L;
    private static final Interpolator FAST_OUT_SLOW_IN_INTERPOLATOR = new FastOutSlowInInterpolator();

    public static void scaleIn(final View fab) {
        scaleIn(fab, DEFAULT_DURATION, null);
    }

    public static void scaleIn(final View fab, long duration, final ScaleCallback callback) {
        fab.setVisibility(View.VISIBLE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH) {
            ViewCompat.animate(fab)
                    .scaleX(1.0F)
                    .scaleY(1.0F)
                    .alpha(1.0F)
                    .setDuration(duration)
                    .setInterpolator(FAST_OUT_SLOW_IN_INTERPOLATOR)
                    .withLayer()
                    .setListener(new ViewPropertyAnimatorListener() {
                        public void onAnimationStart(View view) {
                            if (callback != null) callback.onAnimationStart();
                        }

                        public void onAnimationCancel(View view) {
                        }

                        public void onAnimationEnd(View view) {
                            view.setVisibility(View.VISIBLE);
                            if (callback != null) callback.onAnimationEnd();
                        }
                    }).start();
        } else {
            Animation anim = AnimationUtils.loadAnimation(fab.getContext(), R.anim.design_fab_out);
            anim.setDuration(duration);
            anim.setInterpolator(FAST_OUT_SLOW_IN_INTERPOLATOR);
            anim.setAnimationListener(new Animation.AnimationListener() {
                public void onAnimationStart(Animation animation) {
                    if (callback != null) callback.onAnimationStart();
                }

                public void onAnimationEnd(Animation animation) {
                    fab.setVisibility(View.VISIBLE);
                    if (callback != null) callback.onAnimationEnd();
                }

                @Override
                public void onAnimationRepeat(Animation animation) {
                    //
                }
            });
            fab.startAnimation(anim);
        }
    }

    public static void scaleOut(final View fab) {
        scaleOut(fab, DEFAULT_DURATION, null);
    }

    public static void scaleOut(final View fab, final ScaleCallback callback) {
        scaleOut(fab, DEFAULT_DURATION, callback);
    }

    public static void scaleOut(final View fab, long duration, final ScaleCallback callback) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH) {
            ViewCompat.animate(fab)
                    .scaleX(0.0F)
                    .scaleY(0.0F).alpha(0.0F)
                    .setInterpolator(FAST_OUT_SLOW_IN_INTERPOLATOR)
                    .setDuration(duration)
                    .withLayer()
                    .setListener(new ViewPropertyAnimatorListener() {
                        public void onAnimationStart(View view) {
                            if (callback != null) callback.onAnimationStart();
                        }

                        public void onAnimationCancel(View view) {
                        }

                        public void onAnimationEnd(View view) {
                            view.setVisibility(View.INVISIBLE);
                            if (callback != null) callback.onAnimationEnd();
                        }
                    }).start();
        } else {
            Animation anim = AnimationUtils.loadAnimation(fab.getContext(), R.anim.design_fab_out);
            anim.setInterpolator(FAST_OUT_SLOW_IN_INTERPOLATOR);
            anim.setDuration(duration);
            anim.setAnimationListener(new Animation.AnimationListener() {
                public void onAnimationStart(Animation animation) {
                    if (callback != null) callback.onAnimationStart();
                }

                public void onAnimationEnd(Animation animation) {
                    fab.setVisibility(View.INVISIBLE);
                    if (callback != null) callback.onAnimationEnd();
                }

                @Override
                public void onAnimationRepeat(Animation animation) {
                    //
                }
            });
            fab.startAnimation(anim);
        }
    }

    public interface ScaleCallback {
        void onAnimationStart();

        void onAnimationEnd();
    }
```

```
}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.graphics.Paint;
import android.net.Uri;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v4.app.DialogFragment;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.naman14.timber.R;

public class Helpers {

    public static void showAbout(AppCompatActivity activity) {
        FragmentManager fm = activity.getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();
        Fragment prev = fm.findFragmentByTag("dialog_about");
        if (prev != null) {
            ft.remove(prev);
        }
        ft.addToBackStack(null);

        new AboutDialog().show(ft, "dialog_about");
    }

    public static String getATEKey(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context).getBoolean("dark_theme", false) ?
                "dark_theme" : "light_theme";
    }

    public static class AboutDialog extends DialogFragment {

        String urlgooglelus = "https://plus.google.com/u/0/+NamanDwivedi14";
        String urlcommunity = "https://plus.google.com/communities/111029425713454201429";
        String urltwitter = "https://twitter.com/naman1405";
        String urlgithub = "https://github.com/naman14";
        String urlsource = "https://github.com/naman14/Timber/issues";

        public AboutDialog() {
        }

        @Override
        public Dialog onCreateDialog(Bundle savedInstanceState) {
```

```
        LayoutInflater layoutInflater = (LayoutInflater) getActivity().getSystemService(
                Context.LAYOUT_INFLATER_SERVICE);
        LinearLayout aboutBodyView = (LinearLayout) layoutInflater.inflate(R.layout.layout_about_dialog, null);

        TextView appversion = (TextView) aboutBodyView.findViewById(R.id.app_version_name);

        TextView googleplus = (TextView) aboutBodyView.findViewById(R.id.googleplus);
        TextView twitter = (TextView) aboutBodyView.findViewById(R.id.twitter);
        TextView github = (TextView) aboutBodyView.findViewById(R.id.github);
        TextView source = (TextView) aboutBodyView.findViewById(R.id.source);
        TextView community = (TextView) aboutBodyView.findViewById(R.id.feature_request);

        TextView dismiss = (TextView) aboutBodyView.findViewById(R.id.dismiss_dialog);
        dismiss.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                dismiss();
            }
        });
        googleplus.setPaintFlags(googleplus.getPaintFlags() | Paint.UNDERLINE_TEXT_FLAG);
        twitter.setPaintFlags(twitter.getPaintFlags() | Paint.UNDERLINE_TEXT_FLAG);
        github.setPaintFlags(github.getPaintFlags() | Paint.UNDERLINE_TEXT_FLAG);

        googleplus.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_VIEW);
                i.setData(Uri.parse(urlgooglelus));
                startActivity(i);
            }

        });
        twitter.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_VIEW);
                i.setData(Uri.parse(urltwitter));
                startActivity(i);
            }

        });
        github.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_VIEW);
                i.setData(Uri.parse(urlgithub));
                startActivity(i);
            }

        });
        source.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_VIEW);
                i.setData(Uri.parse(urlsource));
                startActivity(i);
            }
        });
        community.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(Intent.ACTION_VIEW);
                i.setData(Uri.parse(urlcommunity));
                startActivity(i);
            }
        });
        try {
            PackageInfo pInfo = getActivity().getPackageManager().getPackageInfo(getActivity().getPackageName(), 0);
            String version = pInfo.versionName;
```

```
                int versionCode = pInfo.versionCode;
                appversion.setText("Timber " + version);
            } catch (PackageManager.NameNotFoundException e) {
                e.printStackTrace();
            }

            return new AlertDialog.Builder(getActivity())
                    .setView(aboutBodyView)
                    .create();
        }

    }

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.support.v8.renderscript.RenderScript;
import android.view.View;
import android.widget.ImageView;

import com.naman14.timber.R;
import com.naman14.timber.dataloaders.AlbumLoader;
import com.naman14.timber.lastfmapi.LastFmClient;
import com.naman14.timber.lastfmapi.callbacks.AlbumInfoListener;
import com.naman14.timber.lastfmapi.models.AlbumQuery;
import com.naman14.timber.lastfmapi.models.LastfmAlbum;
import com.naman14.timber.models.Album;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.listener.ImageLoadingListener;
import com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

public class ImageUtils {
    private static final DisplayImageOptions lastfmDisplayImageOptions =
                                                new DisplayImageOptions.Builder()
                                                        .cacheInMemory(true)
                                                        .cacheOnDisk(true)
                                                        .showImageOnFail(R.drawable.ic_empty_music2)
                                                        .build();

    private static final DisplayImageOptions diskDisplayImageOptions =
                                                new DisplayImageOptions.Builder()
                                                        .cacheInMemory(true)
                                                        .build();

    public static void loadAlbumArtIntoView(final long albumId, final ImageView view) {
        loadAlbumArtIntoView(albumId, view, new SimpleImageLoadingListener());
    }

    public static void loadAlbumArtIntoView(final long albumId, final ImageView view,
                                            final ImageLoadingListener listener) {
        if (PreferencesUtility.getInstance(view.getContext()).alwaysLoadAlbumImagesFromLastfm()) {
            loadAlbumArtFromLastfm(albumId, view, listener);
        } else {
            loadAlbumArtFromDiskWithLastfmFallback(albumId, view, listener);
        }
    }

    private static void loadAlbumArtFromDiskWithLastfmFallback(final long albumId, ImageView view,
                                                               final ImageLoadingListener listener) {
        ImageLoader.getInstance()
```

```java
                .displayImage(TimberUtils.getAlbumArtUri(albumId).toString(),
                              view,
                              diskDisplayImageOptions,
                              new SimpleImageLoadingListener() {
                                  @Override
                                  public void onLoadingFailed(String imageUri, View view,
                                                              FailReason failReason) {
                                      loadAlbumArtFromLastfm(albumId, (ImageView) view, listener);
                                      listener.onLoadingFailed(imageUri, view, failReason);
                                  }

                                  @Override
                                  public void onLoadingComplete(String imageUri, View view, Bitmap loadedImage) {
                                      listener.onLoadingComplete(imageUri, view, loadedImage);
                                  }
                              });
    }

    private static void loadAlbumArtFromLastfm(long albumId, final ImageView albumArt, final ImageLoadingListener listener)
        Album album = AlbumLoader.getAlbum(albumArt.getContext(), albumId);
        LastFmClient.getInstance(albumArt.getContext())
                .getAlbumInfo(new AlbumQuery(album.title, album.artistName),
                              new AlbumInfoListener() {
                                  @Override
                                  public void albumInfoSuccess(final LastfmAlbum album) {
                                      if (album != null) {
                                          ImageLoader.getInstance()
                                                  .displayImage(album.mArtwork.get(4).mUrl,
                                                                albumArt,
                                                                lastfmDisplayImageOptions, new SimpleImageLoadingListener(){
                                                              @Override
                                                              public void onLoadingComplete(String imageUri, View view, Bitm
                                                                  listener.onLoadingComplete(imageUri, view, loadedImage);
                                                              }

                                                              @Override
                                                              public void onLoadingFailed(String imageUri, View view, FailRe
                                                                  listener.onLoadingFailed(imageUri, view, failReason);
                                                              }
                                                          });
                                      }
                                  }

                                  @Override
                                  public void albumInfoFailed() { }
                              });
    }

    public static Drawable createBlurredImageFromBitmap(Bitmap bitmap, Context context, int inSampleSize) {

        RenderScript rs = RenderScript.create(context);
        final BitmapFactory.Options options = new BitmapFactory.Options();
        options.inSampleSize = inSampleSize;

        ByteArrayOutputStream stream = new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.JPEG, 100, stream);
        byte[] imageInByte = stream.toByteArray();
        ByteArrayInputStream bis = new ByteArrayInputStream(imageInByte);
        Bitmap blurTemplate = BitmapFactory.decodeStream(bis, null, options);

        final android.support.v8.renderscript.Allocation input = android.support.v8.renderscript.Allocation.createFromBitmap
        final android.support.v8.renderscript.Allocation output = android.support.v8.renderscript.Allocation.createTyped(rs,
        final android.support.v8.renderscript.ScriptIntrinsicBlur script = android.support.v8.renderscript.ScriptIntrinsicBl
        script.setRadius(8f);
        script.setInput(input);
        script.forEach(output);
        output.copyTo(blurTemplate);

        return new BitmapDrawable(context.getResources(), blurTemplate);
    }
```

```
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\utils\LyricsExtractor.java

```java
package com.naman14.timber.utils;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.Charset;
import java.util.Arrays;

/**
 * Created by Christoph Walcher on 03.12.16.
 */

public class LyricsExtractor {
    public static String getLyrics(File file){
        String filename = file.getName();
        String fileending = filename.substring(filename.lastIndexOf('.')+1,filename.length()).toLowerCase();
        try{
            switch(fileending){
                case "mp3":
                    return getLyricsID3(file);
                case "mp4":
                case "m4a":
                case "aac":
                    return getLyricsMP4(file);
                case "ogg":
                case "oga":
                    return getLyricsVorbis(file);
            }
        }catch(Exception e){}
        return null;
    }

    private static int readOgg(byte[] buf, InputStream in, int bytesinpage, int skip) throws IOException {
        int toread = skip!=-1?skip:buf.length;
        int offset = 0;
        while(toread>0){
            if(bytesinpage==0){
                byte magic[] = new byte[4];
                in.read(magic);
                if(!Arrays.equals(magic,new byte[]{'O','g','g','S'})){
                    in.close();
                    throw new IOException();
                }
                byte header[] = new byte[23];
                in.read(header);
                int count = header[22]& 0xFF;
                while(count-->0){
                    bytesinpage += in.read();
                }
            }
            int read = toread;
            if(bytesinpage-toread<0)read = bytesinpage;
            if(skip != -1)
                in.skip(read);
            else
                in.read(buf, offset, read);
            offset += read;
            toread -= read;
            bytesinpage -= read;
        }
        return bytesinpage;
    }

    private static String getLyricsVorbis(File file) throws Exception{
        FileInputStream in = new FileInputStream(file);
        int bytesinpage = 0;
        byte buffer[] = new byte[7];
        bytesinpage = readOgg(buffer,in,bytesinpage,-1);
        if(!Arrays.equals(buffer, new byte[]{1,'v','o','r','b','i','s'})){
            in.close();
            return null;
        }
        bytesinpage = readOgg(null,in,bytesinpage, 23);
        bytesinpage = readOgg(buffer,in,bytesinpage,-1);
        if(!Arrays.equals(buffer, new byte[]{3,'v','o','r','b','i','s'})){
            in.close();
            return null;
        }
        byte length[] = new byte[4];
        bytesinpage = readOgg(length, in, bytesinpage,-1);
        bytesinpage = readOgg(null, in, bytesinpage, byteArrayToInt(length));
        bytesinpage = readOgg(length, in, bytesinpage,-1);
        int count = byteArrayToIntLE(length);
        while(count-->0){
            bytesinpage = readOgg(length, in, bytesinpage,-1);
            int comment_len = byteArrayToIntLE(length);
            byte lyrics_tag[] = new byte[]{'L','Y','R','I','C','S','='};
            if(comment_len<=lyrics_tag.length){
                bytesinpage = readOgg(null, in, bytesinpage, comment_len);
                continue;
            }
            byte comment_probe[] = new byte[lyrics_tag.length];
            bytesinpage = readOgg(comment_probe, in, bytesinpage,-1);
            if(Arrays.equals(comment_probe,lyrics_tag)){
                byte  lyrics[] = new byte[comment_len - lyrics_tag.length];
                readOgg(lyrics, in, bytesinpage,-1);
                in.close();
                return new String(lyrics);
            }else{
                bytesinpage = readOgg(null, in, bytesinpage, comment_len - lyrics_tag.length);
            }
        }
        in.close();
        return null;

    }


    private static String getLyricsMP4(File file) throws Exception{
        FileInputStream in = new FileInputStream(file);

        byte head[] = new byte[4];
        in.read(head);
        int len = byteArrayToInt(head);
        in.read(head);
        if (!Arrays.equals(head, new byte[]{'f','t','y','p'})){
            in.close();
            return null;
        }
        in.skip(len - 8);
        final byte path[][] = new byte[][]{{'m','o','o','v'},{'u','d','t','a'},{'m','e','t','a'},{'i','l','s','t'},{(byte) '
        int atom_size = Integer.MAX_VALUE;
        outter:
        for(byte[] atom: path){
            while(in.available()>0){
                byte buffer[] = new byte[4];
                in.read(buffer);
                len = byteArrayToInt(buffer);
                if(len==0)continue;
                in.read(buffer);
                if(len>atom_size){
                    in.close();
                    return null;
                }
                if (Arrays.equals(buffer, atom)){
                    atom_size = len - 8;
                    //Found Atom search next atom
                    continue outter;
                }else{
                    //Skip Atom
                    in.skip(len - 8);
                    atom_size-=len;
                }
            }
            in.close();
            return null;
        }
        in.skip(8);
        byte buffer[] = new byte[atom_size-8];
        in.read(buffer);
        in.close();
        return new String(buffer);
    }


    private static String getLyricsID3(File file) throws Exception{
        FileInputStream in = new FileInputStream(file);
        byte buffer[] = new byte[4];
        in.read(buffer, 0, 3);
        if (!Arrays.equals(buffer, new byte[] { 'I', 'D', '3', 0 })){
            in.close();
            return null;
        }

        in.read(buffer, 0, 3);
        boolean ext = (buffer[2] & (byte) 0b0100000) != 0;
        in.read(buffer);
        int len = buffer[3] & 0x7F | (buffer[2] & 0x7F) << 7 | (buffer[1] & 0x7F) << 14 | (buffer[0] & 0x7F) << 21;
        if (ext) {
            in.read(buffer); len-=4;
            int ext_len = byteArrayToInt(buffer);
            in.skip(ext_len); len -= ext_len;

        }
        while (len > 0) {
            byte tag_name[] = new byte[4];
            in.read(tag_name); len-=4;
            if(tag_name[0]==0)break;
            in.read(buffer); len -=4;
            int tag_len = byteArrayToInt(buffer);
            in.read(buffer,0,2); len-=2;
            if(Arrays.equals(tag_name, new byte[] { 'U', 'S', 'L', 'T' })){
                byte head[] = new byte[4];
                in.read(head); len -= 4; tag_len -= 4;
                while(in.read()!=0){
                    len--;
                    tag_len--;
                }
                if(head[0]==1)in.read();
                byte tag_value[] = new byte[tag_len];
                in.read(tag_value); len -= tag_len;
                in.close();
                Charset charset = null;
                switch (head[0]){
                    case 0: charset = Charset.forName("ISO-8859-1");
                        break;
                    case 1: charset = Charset.forName("UTF-16");
                        break;
                    case 2: charset = Charset.forName("UTF-16BE");
                        break;
                    case 3: charset = Charset.forName("UTF-8");
                        break;
                    default:
                        return null;
                }
                return new String(tag_value,charset);

            }else{
                in.skip(tag_len); len -= tag_len;
            }

        }
        in.close();
        return null;

    }

    private static int byteArrayToInt(byte[] b) {
        return b[3] & 0xFF | (b[2] & 0xFF) << 8 | (b[1] & 0xFF) << 16 | (b[0] & 0xFF) << 24;
    }

    private static int byteArrayToIntLE(byte[] b) {
        return b[0] & 0xFF | (b[1] & 0xFF) << 8 | (b[2] & 0xFF) << 16 | (b[3] & 0xFF) << 24;
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\utils\LyricsLoader.java

```java
package com.naman14.timber.utils;

import android.content.Context;

import com.squareup.okhttp.Cache;
import com.squareup.okhttp.OkHttpClient;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.lang.reflect.Type;
import java.util.concurrent.TimeUnit;

import retrofit.Callback;
import retrofit.RequestInterceptor;
import retrofit.RestAdapter;
import retrofit.client.OkClient;
import retrofit.converter.ConversionException;
import retrofit.converter.Converter;
import retrofit.http.GET;
import retrofit.http.Headers;
import retrofit.http.Query;
import retrofit.mime.TypedInput;
import retrofit.mime.TypedOutput;

/**
 * Created by Christoph Walcher on 03.12.16.
 */

public class LyricsLoader {
    private static LyricsLoader instance = null;
    private static final String BASE_API_URL = "https://makeitpersonal.co";
    private static final long CACHE_SIZE = 1024 * 1024;
    private LyricsRestService service;

    public static LyricsLoader getInstance(Context con) {
        if(instance==null)instance = new LyricsLoader(con);
        return instance;
    }

    private LyricsLoader(Context con){
        final OkHttpClient okHttpClient = new OkHttpClient();

        okHttpClient.setCache(new Cache(con.getApplicationContext().getCacheDir(),
                CACHE_SIZE));
        okHttpClient.setConnectTimeout(20, TimeUnit.SECONDS);

        RequestInterceptor interceptor = new RequestInterceptor() {
            @Override
            public void intercept(RequestFacade request) {
                //7-days cache
                request.addHeader("Cache-Control", String.format("max-age=%d,max-stale=%d", Integer.valueOf(60 * 60 * 24 * 7
            }
        };

        RestAdapter.Builder builder = new RestAdapter.Builder()
                .setEndpoint(BASE_API_URL)
                .setRequestInterceptor(interceptor)
                .setConverter(new Converter() {
                    @Override
                    public Object fromBody(TypedInput arg0, Type arg1)
                            throws ConversionException {

                        try {
                            BufferedReader br = null;
                            StringBuilder sb = new StringBuilder();

                            String line;

                            br = new BufferedReader(new InputStreamReader(arg0.in()));
                            while ((line = br.readLine()) != null) {
                                sb.append(line);
                                sb.append('\n');
                            }
                            return sb.toString();

                        } catch (IOException e) {
                            e.printStackTrace();
                            return null;
                        }
                    }

                    @Override
                    public TypedOutput toBody(Object arg0) {
                        return null;
                    }
                })
                .setClient(new OkClient(okHttpClient));

         service = builder
                .build()
                .create(LyricsRestService.class);
    }

    public void getLyrics(String artist, String title, Callback<String> callback){
        service.getLyrics(artist,title,callback);
    }

    private interface LyricsRestService {
        @Headers("Cache-Control: public")
        @GET("/lyrics")
        void getLyrics(@Query("artist") String artist, @Query("title") String title, Callback<String> callback);
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\utils\NavigationUtils.java

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

import android.annotation.TargetApi;
import android.app.Activity;
import android.app.ActivityOptions;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.transition.Transition;
import android.transition.TransitionInflater;
import android.util.Pair;
import android.view.View;
import android.widget.Toast;

import com.naman14.timber.R;
import com.naman14.timber.activities.MainActivity;
import com.naman14.timber.activities.NowPlayingActivity;
import com.naman14.timber.activities.PlaylistDetailActivity;
import com.naman14.timber.activities.SearchActivity;
import com.naman14.timber.activities.SettingsActivity;
import com.naman14.timber.fragments.AlbumDetailFragment;
import com.naman14.timber.fragments.ArtistDetailFragment;
import com.naman14.timber.nowplaying.Timber1;
import com.naman14.timber.nowplaying.Timber2;
import com.naman14.timber.nowplaying.Timber3;
import com.naman14.timber.nowplaying.Timber4;
import com.naman14.timber.nowplaying.Timber5;
import com.naman14.timber.nowplaying.Timber6;

import java.util.ArrayList;

public class NavigationUtils {

    @TargetApi(21)
    public static void navigateToAlbum(Activity context, long albumID, Pair<View, String> transitionViews) {

        FragmentTransaction transaction = ((AppCompatActivity) context).getSupportFragmentManager().beginTransaction();
        Fragment fragment;

        transaction.setCustomAnimations(R.anim.activity_fade_in,
                R.anim.activity_fade_out, R.anim.activity_fade_in, R.anim.activity_fade_out);
        fragment = AlbumDetailFragment.newInstance(albumID, false, null);

        transaction.hide(((AppCompatActivity) context).getSupportFragmentManager().findFragmentById(R.id.fragment_container)
        transaction.add(R.id.fragment_container, fragment);
        transaction.addToBackStack(null).commit();

    }

    @TargetApi(21)
    public static void navigateToArtist(Activity context, long artistID, Pair<View, String> transitionViews) {

        FragmentTransaction transaction = ((AppCompatActivity) context).getSupportFragmentManager().beginTransaction();
        Fragment fragment;

        transaction.setCustomAnimations(R.anim.activity_fade_in,
                R.anim.activity_fade_out, R.anim.activity_fade_in, R.anim.activity_fade_out);
        fragment = ArtistDetailFragment.newInstance(artistID, false, null);

        transaction.hide(((AppCompatActivity) context).getSupportFragmentManager().findFragmentById(R.id.fragment_container)
        transaction.add(R.id.fragment_container, fragment);
        transaction.addToBackStack(null).commit();

    }

    public static void goToArtist(Context context, long artistId) {
        Intent intent = new Intent(context, MainActivity.class);
        intent.setAction(Constants.NAVIGATE_ARTIST);
        intent.putExtra(Constants.ARTIST_ID, artistId);
        context.startActivity(intent);
    }

    public static void goToAlbum(Context context, long albumId) {
        Intent intent = new Intent(context, MainActivity.class);
        intent.setAction(Constants.NAVIGATE_ALBUM);
        intent.putExtra(Constants.ALBUM_ID, albumId);
        context.startActivity(intent);
    }

    public static void goToLyrics(Context context) {
        Intent intent = new Intent(context, MainActivity.class);
        intent.setAction(Constants.NAVIGATE_LYRICS);
        context.startActivity(intent);
    }

    public static void navigateToNowplaying(Activity context, boolean withAnimations) {

        final Intent intent = new Intent(context, NowPlayingActivity.class);
        context.startActivity(intent);
    }

    public static Intent getNowPlayingIntent(Context context) {

        final Intent intent = new Intent(context, MainActivity.class);
        intent.setAction(Constants.NAVIGATE_NOWPLAYING);
        return intent;
    }

    public static void navigateToSettings(Activity context) {
        final Intent intent = new Intent(context, SettingsActivity.class);
        intent.setAction(Constants.NAVIGATE_SETTINGS);
        context.startActivity(intent);
    }

    public static void navigateToSearch(Activity context) {
        final Intent intent = new Intent(context, SearchActivity.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_NO_ANIMATION);
        intent.setAction(Constants.NAVIGATE_SEARCH);
        context.startActivity(intent);
    }


    @TargetApi(21)
    public static void navigateToPlaylistDetail(Activity context, String action, long firstAlbumID, String playlistName, int
        final Intent intent = new Intent(context, PlaylistDetailActivity.class);
        intent.setAction(action);
        intent.putExtra(Constants.PLAYLIST_ID, playlistID);
        intent.putExtra(Constants.PLAYLIST_FOREGROUND_COLOR, foregroundcolor);
        intent.putExtra(Constants.ALBUM_ID, firstAlbumID);
        intent.putExtra(Constants.PLAYLIST_NAME, playlistName);
        intent.putExtra(Constants.ACTIVITY_TRANSITION, transitionViews != null);

        if (transitionViews != null && TimberUtils.isLollipop()) {
```

```java
            ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(context, transitionViews.get(0), transiti
            context.startActivityForResult(intent, Constants.ACTION_DELETE_PLAYLIST, options.toBundle());
        } else {
            context.startActivityForResult(intent, Constants.ACTION_DELETE_PLAYLIST);
        }
    }

    public static void navigateToEqualizer(Activity context) {
        try {
            // The google MusicFX apps need to be started using startActivityForResult
            context.startActivityForResult(TimberUtils.createEffectsIntent(), 666);
        } catch (final ActivityNotFoundException notFound) {
            Toast.makeText(context, "Equalizer not found", Toast.LENGTH_SHORT).show();
        }
    }

    public static Intent getNavigateToStyleSelectorIntent(Activity context, String what) {
        final Intent intent = new Intent(context, SettingsActivity.class);
        intent.setAction(Constants.SETTINGS_STYLE_SELECTOR);
        intent.putExtra(Constants.SETTINGS_STYLE_SELECTOR_WHAT, what);
        return intent;
    }

    public static Fragment getFragmentForNowplayingID(String fragmentID) {
        switch (fragmentID) {
            case Constants.TIMBER1:
                return new Timber1();
            case Constants.TIMBER2:
                return new Timber2();
            case Constants.TIMBER3:
                return new Timber3();
            case Constants.TIMBER4:
                return new Timber4();
            case Constants.TIMBER5:
                return new Timber5();
            case Constants.TIMBER6:
                return new Timber6();
            default:
                return new Timber1();
        }

    }

    public static int getIntForCurrentNowplaying(String nowPlaying) {
        switch (nowPlaying) {
            case Constants.TIMBER1:
                return 0;
            case Constants.TIMBER2:
                return 1;
            case Constants.TIMBER3:
                return 2;
            case Constants.TIMBER4:
                return 3;
            case Constants.TIMBER5:
                return 4;
            case Constants.TIMBER6:
                return 5;
            default:
                return 2;
        }

    }

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.os.Environment;
import android.preference.PreferenceManager;

import com.naman14.timber.MusicPlayer;
import com.naman14.timber.MusicService;

public final class PreferencesUtility {

    public static final String ARTIST_SORT_ORDER = "artist_sort_order";
    public static final String ARTIST_SONG_SORT_ORDER = "artist_song_sort_order";
    public static final String ARTIST_ALBUM_SORT_ORDER = "artist_album_sort_order";
    public static final String ALBUM_SORT_ORDER = "album_sort_order";
    public static final String ALBUM_SONG_SORT_ORDER = "album_song_sort_order";
    public static final String SONG_SORT_ORDER = "song_sort_order";
    private static final String NOW_PLAYING_SELECTOR = "now_paying_selector";
    private static final String TOGGLE_ANIMATIONS = "toggle_animations";
    private static final String TOGGLE_SYSTEM_ANIMATIONS = "toggle_system_animations";
    private static final String TOGGLE_ARTIST_GRID = "toggle_artist_grid";
    private static final String TOGGLE_ALBUM_GRID = "toggle_album_grid";
    private static final String TOGGLE_PLAYLIST_VIEW = "toggle_playlist_view";
    private static final String TOGGLE_SHOW_AUTO_PLAYLIST = "toggle_show_auto_playlist";
    private static final String LAST_FOLDER = "last_folder";

    private static final String TOGGLE_HEADPHONE_PAUSE = "toggle_headphone_pause";
    private static final String THEME_PREFERNCE = "theme_preference";
    private static final String START_PAGE_INDEX = "start_page_index";
    private static final String START_PAGE_PREFERENCE_LASTOPENED = "start_page_preference_latopened";
    private static final String NOW_PLAYNG_THEME_VALUE = "now_playing_theme_value";
    private static final String TOGGLE_XPOSED_TRACKSELECTOR = "toggle_xposed_trackselector";
    public static final String LAST_ADDED_CUTOFF = "last_added_cutoff";
    public static final String GESTURES = "gestures";

    public static final String FULL_UNLOCKED = "full_version_unlocked";

    private static final String SHOW_LOCKSCREEN_ALBUMART = "show_albumart_lockscreen";
    private static final String ARTIST_ALBUM_IMAGE = "artist_album_image";
    private static final String ARTIST_ALBUM_IMAGE_MOBILE = "artist_album_image_mobile";
    private static final String ALWAYS_LOAD_ALBUM_IMAGES_LASTFM = "always_load_album_images_lastfm";

    private static PreferencesUtility sInstance;

    private static SharedPreferences mPreferences;
    private static Context context;
    private ConnectivityManager connManager = null;

    public PreferencesUtility(final Context context) {
        this.context = context;
        mPreferences = PreferenceManager.getDefaultSharedPreferences(context);
    }

    public static final PreferencesUtility getInstance(final Context context) {
        if (sInstance == null) {
            sInstance = new PreferencesUtility(context.getApplicationContext());
        }
        return sInstance;
    }


    public void setOnSharedPreferenceChangeListener(OnSharedPreferenceChangeListener listener) {
        mPreferences.registerOnSharedPreferenceChangeListener(listener);
    }

    public boolean isArtistsInGrid() {
        return mPreferences.getBoolean(TOGGLE_ARTIST_GRID, true);
    }

    public void setArtistsInGrid(final boolean b) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putBoolean(TOGGLE_ARTIST_GRID, b);
        editor.apply();
    }

    public boolean isAlbumsInGrid() {
        return mPreferences.getBoolean(TOGGLE_ALBUM_GRID, true);
    }

    public void setAlbumsInGrid(final boolean b) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putBoolean(TOGGLE_ALBUM_GRID, b);
        editor.apply();
    }

    public boolean pauseEnabledOnDetach() {
        return mPreferences.getBoolean(TOGGLE_HEADPHONE_PAUSE, true);
    }

    public String getTheme() {
        return mPreferences.getString(THEME_PREFERNCE, "light");
    }

    public int getStartPageIndex() {
        return mPreferences.getInt(START_PAGE_INDEX, 0);
    }

    public void setStartPageIndex(final int index) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putInt(START_PAGE_INDEX, index);
        editor.apply();
    }

    public void setLastOpenedAsStartPagePreference(boolean preference) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putBoolean(START_PAGE_PREFERENCE_LASTOPENED, preference);
        editor.apply();
    }

    public boolean lastOpenedIsStartPagePreference() {
        return mPreferences.getBoolean(START_PAGE_PREFERENCE_LASTOPENED, true);
    }

    private void setSortOrder(final String key, final String value) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putString(key, value);
        editor.apply();
    }

    public final String getArtistSortOrder() {
        return mPreferences.getString(ARTIST_SORT_ORDER, SortOrder.ArtistSortOrder.ARTIST_A_Z);
    }

    public void setArtistSortOrder(final String value) {
        setSortOrder(ARTIST_SORT_ORDER, value);
    }

    public final String getArtistSongSortOrder() {
        return mPreferences.getString(ARTIST_SONG_SORT_ORDER,
                SortOrder.ArtistSongSortOrder.SONG_A_Z);
    }

    public void setArtistSongSortOrder(final String value) {
        setSortOrder(ARTIST_SONG_SORT_ORDER, value);
    }

    public final String getArtistAlbumSortOrder() {
        return mPreferences.getString(ARTIST_ALBUM_SORT_ORDER,
                SortOrder.ArtistAlbumSortOrder.ALBUM_A_Z);
    }

    public void setArtistAlbumSortOrder(final String value) {
        setSortOrder(ARTIST_ALBUM_SORT_ORDER, value);
    }

    public final String getAlbumSortOrder() {
        return mPreferences.getString(ALBUM_SORT_ORDER, SortOrder.AlbumSortOrder.ALBUM_A_Z);
    }

    public void setAlbumSortOrder(final String value) {
        setSortOrder(ALBUM_SORT_ORDER, value);
    }

    public final String getAlbumSongSortOrder() {
        return mPreferences.getString(ALBUM_SONG_SORT_ORDER,
                SortOrder.AlbumSongSortOrder.SONG_TRACK_LIST);
    }

    public void setAlbumSongSortOrder(final String value) {
        setSortOrder(ALBUM_SONG_SORT_ORDER, value);
    }

    public final String getSongSortOrder() {
        return mPreferences.getString(SONG_SORT_ORDER, SortOrder.SongSortOrder.SONG_A_Z);
    }

    public void setSongSortOrder(final String value) {
        setSortOrder(SONG_SORT_ORDER, value);
    }

    public final boolean didNowplayingThemeChanged() {
        return mPreferences.getBoolean(NOW_PLAYNG_THEME_VALUE, false);
    }

    public void setNowPlayingThemeChanged(final boolean value) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putBoolean(NOW_PLAYNG_THEME_VALUE, value);
        editor.apply();
    }

    public boolean getXPosedTrackselectorEnabled() {
        return mPreferences.getBoolean(TOGGLE_XPOSED_TRACKSELECTOR, false);
    }

    public int getPlaylistView() {
        return mPreferences.getInt(TOGGLE_PLAYLIST_VIEW ,0);
    }

    public void setPlaylistView(final int i) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putInt(TOGGLE_PLAYLIST_VIEW, i);
        editor.apply();
    }

    public boolean showAutoPlaylist() {
        return mPreferences.getBoolean(TOGGLE_SHOW_AUTO_PLAYLIST, true);
    }

    public void setToggleShowAutoPlaylist(final boolean b) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putBoolean(TOGGLE_SHOW_AUTO_PLAYLIST, b);
        editor.apply();
    }

    /** @parm lastAddedMillis timestamp in millis used as a cutoff for last added playlist */
    public void setLastAddedCutoff(long lastAddedMillis) {
        mPreferences.edit().putLong(LAST_ADDED_CUTOFF, lastAddedMillis).apply();
    }

    public long getLastAddedCutoff() {
        return mPreferences.getLong(LAST_ADDED_CUTOFF, 0L);
    }

    public boolean isGesturesEnabled() {
        return mPreferences.getBoolean(GESTURES, true);
    }

    public void storeLastFolder(String path) {
        SharedPreferences.Editor editor = mPreferences.edit();
        editor.putString(LAST_FOLDER, path);
        editor.apply();
    }

    public String getLastFolder() {
        return mPreferences.getString(LAST_FOLDER, Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_MUSIC
    }

    public boolean fullUnlocked() {
        return mPreferences.getBoolean(FULL_UNLOCKED, false);
    }

    public void setFullUnlocked(final boolean b) {
        final SharedPreferences.Editor editor = mPreferences.edit();
        editor.putBoolean(FULL_UNLOCKED, b);
        editor.apply();
    }

    public boolean getSetAlbumartLockscreen() {
        return mPreferences.getBoolean(SHOW_LOCKSCREEN_ALBUMART, true);
    }

    public void updateService(Bundle extras) {
        if(!MusicPlayer.isPlaybackServiceConnected())return;
        final Intent intent = new Intent(context, MusicService.class);
        intent.setAction(MusicService.UPDATE_PREFERENCES);
        intent.putExtras(extras);
        context.startService(intent);
    }

    public boolean loadArtistAndAlbumImages() {
        if (mPreferences.getBoolean(ARTIST_ALBUM_IMAGE, true)) {
            if (!mPreferences.getBoolean(ARTIST_ALBUM_IMAGE_MOBILE, true)) {
                if (connManager == null) connManager = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_S
                NetworkInfo ni = connManager.getActiveNetworkInfo();
                return ni != null && ni.getType() == ConnectivityManager.TYPE_WIFI;
            }
            return true;
        }
        return false;
    }
    public boolean alwaysLoadAlbumImagesFromLastfm() {
        return mPreferences.getBoolean(ALWAYS_LOAD_ALBUM_IMAGES_LASTFM, false);
    }
}
```

```java
package com.naman14.timber.utils;

import android.support.annotation.NonNull;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;

import com.naman14.timber.MusicPlayer;

/**
 * Created by nv95 on 02.11.16.
 */

public class SlideTrackSwitcher implements View.OnTouchListener {

    private static final int SWIPE_THRESHOLD = 200;
    private static final int SWIPE_VELOCITY_THRESHOLD = 100;

    private GestureDetector mDetector;
    private View mView;


    public SlideTrackSwitcher() {
    }

    public void attach(@NonNull View v) {
        mView = v;
        mDetector = new GestureDetector(v.getContext(), new SwipeListener());
        v.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        return mDetector.onTouchEvent(event);
    }

    private class SwipeListener extends GestureDetector.SimpleOnGestureListener {

        @Override
        public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
            boolean result = false;
            try {
                float diffY = e2.getY() - e1.getY();
                float diffX = e2.getX() - e1.getX();
                if (Math.abs(diffX) > Math.abs(diffY)) {
                    if (Math.abs(diffX) > SWIPE_THRESHOLD && Math.abs(velocityX) > SWIPE_VELOCITY_THRESHOLD) {
                        if (diffX > 0) {
                            onSwipeRight();
                        } else {
                            onSwipeLeft();
                        }
                    }
                    result = true;
                }
                else if (Math.abs(diffY) > SWIPE_THRESHOLD && Math.abs(velocityY) > SWIPE_VELOCITY_THRESHOLD) {
                    if (diffY > 0) {
                        onSwipeBottom();
                    } else {
                        onSwipeTop();
                    }
                }
                result = true;

            } catch (Exception exception) {
                exception.printStackTrace();
            }
            return result;
        }

        @Override
        public boolean onDown(MotionEvent e) {
            return true;
        }

        @Override
        public boolean onDoubleTap(MotionEvent e) {
            MusicPlayer.playOrPause();
            return true;
        }

        @Override
        public boolean onSingleTapConfirmed(MotionEvent e) {
            onClick();
            return super.onSingleTapConfirmed(e);
        }
    }

    public void onSwipeRight() {
        MusicPlayer.previous(mView.getContext(), true);
    }

    public void onSwipeLeft() {
        MusicPlayer.next();
    }

    public void onSwipeTop() {
    }

    public void onSwipeBottom() {
    }

    public void onClick() {

    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\utils\SortOrder.java

```java
/*
 * Copyright (C) 2012 Andrew Neal
 * Copyright (C) 2014 The CyanogenMod Project
 * Licensed under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with the
 * License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
 * or agreed to in writing, software distributed under the License is
 * distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied. See the License for the specific language
 * governing permissions and limitations under the License.
 */

package com.naman14.timber.utils;

import android.provider.MediaStore;

/**
 * Holds all of the sort orders for each list type.
 *
 * @author Andrew Neal (andrewdneal@gmail.com)
 */
public final class SortOrder {

    /**
     * This class is never instantiated
     */
    public SortOrder() {
    }

    /**
     * Artist sort order entries.
     */
    public interface ArtistSortOrder {
        /* Artist sort order A-Z */
        String ARTIST_A_Z = MediaStore.Audio.Artists.DEFAULT_SORT_ORDER;

        /* Artist sort order Z-A */
        String ARTIST_Z_A = ARTIST_A_Z + " DESC";

        /* Artist sort order number of songs */
        String ARTIST_NUMBER_OF_SONGS = MediaStore.Audio.Artists.NUMBER_OF_TRACKS
                + " DESC";

        /* Artist sort order number of albums */
        String ARTIST_NUMBER_OF_ALBUMS = MediaStore.Audio.Artists.NUMBER_OF_ALBUMS
                + " DESC";
    }

    /**
     * Album sort order entries.
     */
    public interface AlbumSortOrder {
        /* Album sort order A-Z */
        String ALBUM_A_Z = MediaStore.Audio.Albums.DEFAULT_SORT_ORDER;

        /* Album sort order Z-A */
        String ALBUM_Z_A = ALBUM_A_Z + " DESC";

        /* Album sort order songs */
        String ALBUM_NUMBER_OF_SONGS = MediaStore.Audio.Albums.NUMBER_OF_SONGS
                + " DESC";

        /* Album sort order artist */
        String ALBUM_ARTIST = MediaStore.Audio.Albums.ARTIST;

        /* Album sort order year */
        String ALBUM_YEAR = MediaStore.Audio.Albums.FIRST_YEAR + " DESC";

    }
    /**
     * Song sort order entries.
     */
    public interface SongSortOrder {
        /* Song sort order A-Z */
        String SONG_A_Z = MediaStore.Audio.Media.DEFAULT_SORT_ORDER;

        /* Song sort order Z-A */
        String SONG_Z_A = SONG_A_Z + " DESC";

        /* Song sort order artist */
        String SONG_ARTIST = MediaStore.Audio.Media.ARTIST;

        /* Song sort order album */
        String SONG_ALBUM = MediaStore.Audio.Media.ALBUM;

        /* Song sort order year */
        String SONG_YEAR = MediaStore.Audio.Media.YEAR + " DESC";

        /* Song sort order duration */
        String SONG_DURATION = MediaStore.Audio.Media.DURATION + " DESC";

        /* Song sort order date */
        String SONG_DATE = MediaStore.Audio.Media.DATE_ADDED + " DESC";

        /* Song sort order filename */
        String SONG_FILENAME = MediaStore.Audio.Media.DATA;
    }

    /**
     * Album song sort order entries.
     */
    public interface AlbumSongSortOrder {
        /* Album song sort order A-Z */
        String SONG_A_Z = MediaStore.Audio.Media.DEFAULT_SORT_ORDER;

        /* Album song sort order Z-A */
        String SONG_Z_A = SONG_A_Z + " DESC";

        /* Album song sort order track list */
        String SONG_TRACK_LIST = MediaStore.Audio.Media.TRACK + ", "
                + MediaStore.Audio.Media.DEFAULT_SORT_ORDER;

        /* Album song sort order duration */
        String SONG_DURATION = SongSortOrder.SONG_DURATION;

        /* Album Song sort order year */
        String SONG_YEAR = MediaStore.Audio.Media.YEAR + " DESC";

        /* Album song sort order filename */
        String SONG_FILENAME = SongSortOrder.SONG_FILENAME;
    }

    /**
     * Artist song sort order entries.
     */
    public interface ArtistSongSortOrder {
        /* Artist song sort order A-Z */
        String SONG_A_Z = MediaStore.Audio.Media.DEFAULT_SORT_ORDER;

        /* Artist song sort order Z-A */
        String SONG_Z_A = SONG_A_Z + " DESC";

        /* Artist song sort order album */
        String SONG_ALBUM = MediaStore.Audio.Media.ALBUM;

        /* Artist song sort order year */
        String SONG_YEAR = MediaStore.Audio.Media.YEAR + " DESC";
        /* Artist song sort order duration */
        String SONG_DURATION = MediaStore.Audio.Media.DURATION + " DESC";

        /* Artist song sort order date */
        String SONG_DATE = MediaStore.Audio.Media.DATE_ADDED + " DESC";

        /* Artist song sort order filename */
        String SONG_FILENAME = SongSortOrder.SONG_FILENAME;
    }

    /**
     * Artist album sort order entries.
     */
    public interface ArtistAlbumSortOrder {
        /* Artist album sort order A-Z */
        String ALBUM_A_Z = MediaStore.Audio.Albums.DEFAULT_SORT_ORDER;

        /* Artist album sort order Z-A */
        String ALBUM_Z_A = ALBUM_A_Z + " DESC";

        /* Artist album sort order songs */
        String ALBUM_NUMBER_OF_SONGS = MediaStore.Audio.Artists.Albums.NUMBER_OF_SONGS
                + " DESC";

        /* Artist album sort order year */
        String ALBUM_YEAR = MediaStore.Audio.Artists.Albums.FIRST_YEAR
                + " DESC";
    }

}
```

```java
/*
 * Copyright (C) 2015 Naman Dwivedi
 *
 * Licensed under the GNU General Public License v3
 *
 * This is free software: you can redistribute it and/or modify it
 * under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
 * without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 * See the GNU General Public License for more details.
 */

package com.naman14.timber.utils;

import android.app.Activity;
import android.content.ContentResolver;
import android.content.ContentUris;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.graphics.Color;
import android.media.MediaMetadataRetriever;
import android.media.audiofx.AudioEffect;
import android.net.Uri;
import android.os.Build;
import android.provider.BaseColumns;
import android.provider.MediaStore;
import android.support.annotation.NonNull;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.util.TypedValue;
import android.widget.Toast;

import com.afollestad.materialdialogs.DialogAction;
import com.afollestad.materialdialogs.MaterialDialog;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.R;
import com.naman14.timber.adapters.BaseQueueAdapter;
import com.naman14.timber.adapters.BaseSongAdapter;
import com.naman14.timber.provider.RecentStore;
import com.naman14.timber.provider.SongPlayCount;

import java.io.File;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.util.Collections;
import java.util.List;

public class TimberUtils {

    public static final String MUSIC_ONLY_SELECTION = MediaStore.Audio.AudioColumns.IS_MUSIC + "=1"
            + " AND " + MediaStore.Audio.AudioColumns.TITLE + " != ''";

    public static boolean isOreo() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.O;
    }

    public static boolean isMarshmallow() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.M;
    }

    public static boolean isLollipop() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP;
    }


    public static boolean isJellyBeanMR2() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2;
    }

    public static boolean isJellyBean() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN;
    }

    public static boolean isJellyBeanMR1() {
        return Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1;
    }

    public static Uri getAlbumArtUri(long albumId) {
        return ContentUris.withAppendedId(Uri.parse("content://media/external/audio/albumart"), albumId);
    }
    public static String getAlbumArtForFile(String filePath) {
        MediaMetadataRetriever mmr = new MediaMetadataRetriever();
        mmr.setDataSource(filePath);

        return mmr.extractMetadata(MediaMetadataRetriever.METADATA_KEY_ALBUM);
    }

    public static final String makeCombinedString(final Context context, final String first,
                                                  final String second) {
        final String formatter = context.getResources().getString(R.string.combine_two_strings);
        return String.format(formatter, first, second);
    }

    public static final String makeLabel(final Context context, final int pluralInt,
                                         final int number) {
        return context.getResources().getQuantityString(pluralInt, number, number);
    }

    public static final String makeShortTimeString(final Context context, long secs) {
        long hours, mins;

        hours = secs / 3600;
        secs %= 3600;
        mins = secs / 60;
        secs %= 60;

        final String durationFormat = context.getResources().getString(
                hours == 0 ? R.string.durationformatshort : R.string.durationformatlong);
        return String.format(durationFormat, hours, mins, secs);
    }

    public static int getActionBarHeight(Context context) {
        int mActionBarHeight;
        TypedValue mTypedValue = new TypedValue();

        context.getTheme().resolveAttribute(R.attr.actionBarSize, mTypedValue, true);

        mActionBarHeight = TypedValue.complexToDimensionPixelSize(mTypedValue.data, context.getResources().getDisplayMetrics

        return mActionBarHeight;
    }

    public static final int getSongCountForPlaylist(final Context context, final long playlistId) {
        Cursor c = context.getContentResolver().query(
                MediaStore.Audio.Playlists.Members.getContentUri("external", playlistId),
                new String[]{BaseColumns._ID}, MUSIC_ONLY_SELECTION, null, null);

        if (c != null) {
            int count = 0;
            if (c.moveToFirst()) {
                count = c.getCount();
            }
            c.close();
            c = null;
            return count;
        }
        return 0;
    }

    public static boolean hasEffectsPanel(final Activity activity) {
        final PackageManager packageManager = activity.getPackageManager();
        return packageManager.resolveActivity(createEffectsIntent(),
                PackageManager.MATCH_DEFAULT_ONLY) != null;
    }

    public static Intent createEffectsIntent() {
        final Intent effects = new Intent(AudioEffect.ACTION_DISPLAY_AUDIO_EFFECT_CONTROL_PANEL);
        effects.putExtra(AudioEffect.EXTRA_AUDIO_SESSION, MusicPlayer.getAudioSessionId());
        return effects;
    }

    public static int getBlackWhiteColor(int color) {
        double darkness = 1 - (0.299 * Color.red(color) + 0.587 * Color.green(color) + 0.114 * Color.blue(color)) / 255;
        if (darkness >= 0.5) {
            return Color.WHITE;
        } else return Color.BLACK;
    }

    public enum IdType {
        NA(0),
        Artist(1),
        Album(2),
        Playlist(3);

        public final int mId;

        IdType(final int id) {
            mId = id;
        }

        public static IdType getTypeById(int id) {
            for (IdType type : values()) {
                if (type.mId == id) {
                    return type;
                }
            }

            throw new IllegalArgumentException("Unrecognized id: " + id);
        }
    }

    public enum PlaylistType {
        LastAdded(-1, R.string.playlist_last_added),
        RecentlyPlayed(-2, R.string.playlist_recently_played),
        TopTracks(-3, R.string.playlist_top_tracks);

        public long mId;
        public int mTitleId;

        PlaylistType(long id, int titleId) {
            mId = id;
            mTitleId = titleId;
        }

        public static PlaylistType getTypeById(long id) {
            for (PlaylistType type : PlaylistType.values()) {
                if (type.mId == id) {
                    return type;
                }
            }

            return null;
        }
    }
    public static void removeFromPlaylist(final Context context, final long id,
                                          final long playlistId) {
        final Uri uri = MediaStore.Audio.Playlists.Members.getContentUri("external", playlistId);
        final ContentResolver resolver = context.getContentResolver();
        resolver.delete(uri, MediaStore.Audio.Playlists.Members.AUDIO_ID + " = ? ", new String[]{
                Long.toString(id)
        });
    }

    public static void clearTopTracks(Context context) {
        SongPlayCount.getInstance(context).deleteAll();
    }

    public static void clearRecent(Context context) {
        RecentStore.getInstance(context).deleteAll();
    }

    public static void clearLastAdded(Context context) {
        PreferencesUtility.getInstance(context)
                .setLastAddedCutoff(System.currentTimeMillis());
    }

    public static void showDeleteDialog(final Context context, final String name, final long[] list, final BaseSongAdapter a

        new MaterialDialog.Builder(context)
                .title("Delete song?")
                .content("Are you sure you want to delete " + name + " ?")
                .positiveText("Delete")
                .negativeText("Cancel")
                .onPositive(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        TimberUtils.deleteTracks(context, list);
                        adapter.removeSongAt(pos);
                        adapter.notifyItemRemoved(pos);
                        adapter.notifyItemRangeChanged(pos, adapter.getItemCount());
                    }
                })
                .onNegative(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        dialog.dismiss();
                    }
                })
                .show();
    }

    public static void showDeleteDialog(final Context context, final String name, final long[] list, final BaseQueueAdapter

        new MaterialDialog.Builder(context)
                .title("Delete song?")
                .content("Are you sure you want to delete " + name + " ?")
                .positiveText("Delete")
                .negativeText("Cancel")
                .onPositive(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        TimberUtils.deleteTracks(context, list);
                        qAdapter.removeSongAt(pos);
                        qAdapter.notifyItemRemoved(pos);
                        qAdapter.notifyItemRangeChanged(pos, qAdapter.getItemCount());
                    }
                })
                .onNegative(new MaterialDialog.SingleButtonCallback() {
                    @Override
                    public void onClick(@NonNull MaterialDialog dialog, @NonNull DialogAction which) {
                        dialog.dismiss();
                    }
                })
                .show();
```

```java
    }


    public static void deleteTracks(final Context context, final long[] list) {
        final String[] projection = new String[]{
                BaseColumns._ID, MediaStore.MediaColumns.DATA, MediaStore.Audio.AudioColumns.ALBUM_ID
        };
        final StringBuilder selection = new StringBuilder();
        selection.append(BaseColumns._ID + " IN (");
        for (int i = 0; i < list.length; i++) {
            selection.append(list[i]);
            if (i < list.length - 1) {
                selection.append(",");
            }
        }
        selection.append(")");
        final Cursor c = context.getContentResolver().query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection, selection.toString(),
                null, null);
        if (c != null) {
            // Step 1: Remove selected tracks from the current playlist, as well
            // as from the album art cache
            c.moveToFirst();
            while (!c.isAfterLast()) {
                // Remove from current playlist
                final long id = c.getLong(0);
                MusicPlayer.removeTrack(id);
                // Remove the track from the play count
                SongPlayCount.getInstance(context).removeItem(id);
                // Remove any items in the recents database
                RecentStore.getInstance(context).removeItem(id);
                c.moveToNext();
            }

            // Step 2: Remove selected tracks from the database
            context.getContentResolver().delete(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
                    selection.toString(), null);

            // Step 3: Remove files from card
            c.moveToFirst();
            while (!c.isAfterLast()) {
                final String name = c.getString(1);
                final File f = new File(name);
                try { // File.delete can throw a security exception
                    if (!f.delete()) {
                        // I'm not sure if we'd ever get here (deletion would
                        // have to fail, but no exception thrown)
                        Log.e("MusicUtils", "Failed to delete file " + name);
                    }
                    c.moveToNext();
                } catch (final SecurityException ex) {
                    c.moveToNext();
                }
            }
            c.close();
        }

        final String message = makeLabel(context, R.plurals.NNNtracksdeleted, list.length);

        Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
        context.getContentResolver().notifyChange(Uri.parse("content://media"), null);
        MusicPlayer.refresh();
    }

    public static void shareTrack(final Context context, long id) {

        try {
            Intent share = new Intent(Intent.ACTION_SEND);
            share.setType("audio/*");
            share.putExtra(Intent.EXTRA_STREAM, getSongUri(context, id));
            context.startActivity(Intent.createChooser(share, "Share"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Uri getSongUri(Context context, long id) {
        final String[] projection = new String[]{
                BaseColumns._ID, MediaStore.MediaColumns.DATA, MediaStore.Audio.AudioColumns.ALBUM_ID
        };
        final StringBuilder selection = new StringBuilder();
        selection.append(BaseColumns._ID + " IN (");
        selection.append(id);
        selection.append(")");
        final Cursor c = context.getContentResolver().query(
                MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, projection, selection.toString(),
                null, null);

        if (c == null) {
            return null;
        }
        c.moveToFirst();


        try {

            Uri uri = Uri.parse(c.getString(1));
            c.close();

            return uri;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public static String getIPAddress(boolean useIPv4) {
        try {
            List<NetworkInterface> interfaces = Collections.list(NetworkInterface.getNetworkInterfaces());
            for (NetworkInterface intf : interfaces) {
                List<InetAddress> addrs = Collections.list(intf.getInetAddresses());
                for (InetAddress addr : addrs) {
                    if (!addr.isLoopbackAddress()) {
                        String sAddr = addr.getHostAddress();
                        //boolean isIPv4 = InetAddressUtils.isIPv4Address(sAddr);
                        boolean isIPv4 = sAddr.indexOf(':')<0;

                        if (useIPv4) {
                            if (isIPv4)
                                return sAddr;
                        } else {
                            if (!isIPv4) {
                                int delim = sAddr.indexOf('%'); // drop ip6 zone suffix
                                return delim<0 ? sAddr.toUpperCase() : sAddr.substring(0, delim).toUpperCase();
                            }
                        }
                    }
                }
            }
        } catch (Exception ex) { }
        return "";
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\BaseRecyclerView.java

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.preference.PreferenceManager;
import android.support.v7.widget.RecyclerView;
import android.util.AttributeSet;
import android.view.View;
import android.widget.TextView;

import com.afollestad.appthemeengine.ATE;
import com.afollestad.appthemeengine.Config;
import com.naman14.timber.MusicPlayer;
import com.naman14.timber.utils.Helpers;

import net.steamcrafted.materialiconlib.MaterialDrawableBuilder;

public class BaseRecyclerView extends RecyclerView {

    private View emptyView;

    private AdapterDataObserver emptyObserver = new AdapterDataObserver() {

        @Override
        public void onChanged() {
            Adapter<?> adapter =  getAdapter();
            if(adapter != null && emptyView != null) {
                if(adapter.getItemCount() == 0) {
                    emptyView.setVisibility(View.VISIBLE);
                    BaseRecyclerView.this.setVisibility(View.GONE);
                }
                else {
                    emptyView.setVisibility(View.GONE);
                    BaseRecyclerView.this.setVisibility(View.VISIBLE);
                }
            }

        }
    };

    public BaseRecyclerView(Context context) {
        super(context);
    }

    public BaseRecyclerView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public BaseRecyclerView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    @Override
    public void setAdapter(Adapter adapter) {
        super.setAdapter(adapter);

        if(adapter != null) {
            adapter.registerAdapterDataObserver(emptyObserver);
        }

        emptyObserver.onChanged();
    }

    public void setEmptyView(Context context, View emptyView, String text) {
        this.emptyView = emptyView;
        ((TextView) emptyView).setText(text);

        MaterialDrawableBuilder builder = MaterialDrawableBuilder.with(context)
                .setIcon(MaterialDrawableBuilder.IconValue.MUSIC_NOTE)
                .setColor(Config.textColorPrimary(context, Helpers.getATEKey(context)))
                .setSizeDp(30);
        ((TextView) emptyView).setCompoundDrawables(null, builder.build(), null, null);
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\BubbleTextGetter.java

```java
package com.naman14.timber.widgets;

public interface BubbleTextGetter {
    String getTextToShowInBubble(int pos);
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\CircleImageView.java

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Bitmap;
import android.graphics.BitmapShader;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ColorFilter;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.RectF;
import android.graphics.Shader;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.ColorDrawable;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.support.annotation.ColorRes;
import android.support.annotation.DrawableRes;
import android.support.v4.content.ContextCompat;
import android.util.AttributeSet;
import android.widget.ImageView;

import com.naman14.timber.R;

public class CircleImageView extends ImageView {

    private static final ScaleType SCALE_TYPE = ScaleType.CENTER_CROP;

    private static final Bitmap.Config BITMAP_CONFIG = Bitmap.Config.ARGB_8888;
    private static final int COLORDRAWABLE_DIMENSION = 2;

    private static final int DEFAULT_BORDER_WIDTH = 0;
    private static final int DEFAULT_BORDER_COLOR = Color.BLACK;
    private static final boolean DEFAULT_BORDER_OVERLAY = false;

    private final RectF mDrawableRect = new RectF();
    private final RectF mBorderRect = new RectF();

    private final Matrix mShaderMatrix = new Matrix();
    private final Paint mBitmapPaint = new Paint();
    private final Paint mBorderPaint = new Paint();

    private int mBorderColor = DEFAULT_BORDER_COLOR;
    private int mBorderWidth = DEFAULT_BORDER_WIDTH;

    private Bitmap mBitmap;
    private BitmapShader mBitmapShader;
    private int mBitmapWidth;
    private int mBitmapHeight;

    private float mDrawableRadius;
    private float mBorderRadius;

    private ColorFilter mColorFilter;

    private boolean mReady;
    private boolean mSetupPending;
    private boolean mBorderOverlay;

    public CircleImageView(Context context) {
        super(context);

        init();
    }

    public CircleImageView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }
    public CircleImageView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);

        TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.CircleImageView, defStyle, 0);

        mBorderWidth = a.getDimensionPixelSize(R.styleable.CircleImageView_border_width, DEFAULT_BORDER_WIDTH);
        mBorderColor = a.getColor(R.styleable.CircleImageView_border_color, DEFAULT_BORDER_COLOR);
        mBorderOverlay = a.getBoolean(R.styleable.CircleImageView_border_overlay, DEFAULT_BORDER_OVERLAY);

        a.recycle();

        init();
    }

    private void init() {
        super.setScaleType(SCALE_TYPE);
        mReady = true;

        if (mSetupPending) {
            setup();
            mSetupPending = false;
        }
    }

    @Override
    public ScaleType getScaleType() {
        return SCALE_TYPE;
    }

    @Override
    public void setScaleType(ScaleType scaleType) {
        if (scaleType != SCALE_TYPE) {
            throw new IllegalArgumentException(String.format("ScaleType %s not supported.", scaleType));
        }
    }

    @Override
    public void setAdjustViewBounds(boolean adjustViewBounds) {
        if (adjustViewBounds) {
            throw new IllegalArgumentException("adjustViewBounds not supported.");
        }
    }

    @Override
    protected void onDraw(Canvas canvas) {
        if (getDrawable() == null) {
            return;
        }

        canvas.drawCircle(getWidth() / 2, getHeight() / 2, mDrawableRadius, mBitmapPaint);
        if (mBorderWidth != 0) {
            canvas.drawCircle(getWidth() / 2, getHeight() / 2, mBorderRadius, mBorderPaint);
        }
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        super.onSizeChanged(w, h, oldw, oldh);
        setup();
    }

    public int getBorderColor() {
        return mBorderColor;
    }

    public void setBorderColor(int borderColor) {
        if (borderColor == mBorderColor) {
            return;
        }
        mBorderColor = borderColor;
        mBorderPaint.setColor(mBorderColor);
        invalidate();
    }

    public void setBorderColorResource(@ColorRes int borderColorRes) {
        setBorderColor(ContextCompat.getColor(getContext(), borderColorRes));
    }

    public int getBorderWidth() {
        return mBorderWidth;
    }

    public void setBorderWidth(int borderWidth) {
        if (borderWidth == mBorderWidth) {
            return;
        }

        mBorderWidth = borderWidth;
        setup();
    }

    public boolean isBorderOverlay() {
        return mBorderOverlay;
    }

    public void setBorderOverlay(boolean borderOverlay) {
        if (borderOverlay == mBorderOverlay) {
            return;
        }

        mBorderOverlay = borderOverlay;
        setup();
    }

    @Override
    public void setImageBitmap(Bitmap bm) {
        super.setImageBitmap(bm);
        mBitmap = bm;
        setup();
    }

    @Override
    public void setImageDrawable(Drawable drawable) {
        super.setImageDrawable(drawable);
        mBitmap = getBitmapFromDrawable(drawable);
        setup();
    }

    @Override
    public void setImageResource(@DrawableRes int resId) {
        super.setImageResource(resId);
        mBitmap = getBitmapFromDrawable(getDrawable());
        setup();
    }

    @Override
    public void setImageURI(Uri uri) {
        super.setImageURI(uri);
        mBitmap = getBitmapFromDrawable(getDrawable());
        setup();
    }

    @Override
    public void setColorFilter(ColorFilter cf) {
        if (cf == mColorFilter) {
            return;
        }

        mColorFilter = cf;
        mBitmapPaint.setColorFilter(mColorFilter);
        invalidate();
    }

    private Bitmap getBitmapFromDrawable(Drawable drawable) {
        if (drawable == null) {
            return null;
        }

        if (drawable instanceof BitmapDrawable) {
            return ((BitmapDrawable) drawable).getBitmap();
        }

        try {
            Bitmap bitmap;

            if (drawable instanceof ColorDrawable) {
                bitmap = Bitmap.createBitmap(COLORDRAWABLE_DIMENSION, COLORDRAWABLE_DIMENSION, BITMAP_CONFIG);
            } else {
                bitmap = Bitmap.createBitmap(drawable.getIntrinsicWidth(), drawable.getIntrinsicHeight(), BITMAP_CONFIG);
            }

            Canvas canvas = new Canvas(bitmap);
            drawable.setBounds(0, 0, canvas.getWidth(), canvas.getHeight());
            drawable.draw(canvas);
            return bitmap;
        } catch (OutOfMemoryError e) {
            return null;
        }
    }

    private void setup() {
        if (!mReady) {
            mSetupPending = true;
            return;
        }

        if (mBitmap == null) {
            return;
        }

        mBitmapShader = new BitmapShader(mBitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);

        mBitmapPaint.setAntiAlias(true);
        mBitmapPaint.setShader(mBitmapShader);

        mBorderPaint.setStyle(Paint.Style.STROKE);
        mBorderPaint.setAntiAlias(true);
        mBorderPaint.setColor(mBorderColor);
        mBorderPaint.setStrokeWidth(mBorderWidth);

        mBitmapHeight = mBitmap.getHeight();
        mBitmapWidth = mBitmap.getWidth();

        mBorderRect.set(0, 0, getWidth(), getHeight());
        mBorderRadius = Math.min((mBorderRect.height() - mBorderWidth) / 2, (mBorderRect.width() - mBorderWidth) / 2);

        mDrawableRect.set(mBorderRect);
        if (!mBorderOverlay) {
            mDrawableRect.inset(mBorderWidth, mBorderWidth);
        }
        mDrawableRadius = Math.min(mDrawableRect.height() / 2, mDrawableRect.width() / 2);

        updateShaderMatrix();
        invalidate();
    }

    private void updateShaderMatrix() {
        float scale;
        float dx = 0;
        float dy = 0;

        mShaderMatrix.set(null);

        if (mBitmapWidth * mDrawableRect.height() > mDrawableRect.width() * mBitmapHeight) {
            scale = mDrawableRect.height() / (float) mBitmapHeight;
            dx = (mDrawableRect.width() - mBitmapWidth * scale) * 0.5f;
        } else {
            scale = mDrawableRect.width() / (float) mBitmapWidth;
            dy = (mDrawableRect.height() - mBitmapHeight * scale) * 0.5f;
        }

        mShaderMatrix.setScale(scale, scale);
        mShaderMatrix.postTranslate((int) (dx + 0.5f) + mDrawableRect.left, (int) (dy + 0.5f) + mDrawableRect.top);

        mBitmapShader.setLocalMatrix(mShaderMatrix);
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\CircularSeekBar.java

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.BlurMaskFilter;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PathMeasure;
import android.graphics.RectF;
import android.os.Bundle;
import android.os.Parcelable;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

import com.naman14.timber.R;

public class CircularSeekBar extends View {

    // Default values
    private static final float DEFAULT_CIRCLE_X_RADIUS = 30f;
    private static final float DEFAULT_CIRCLE_Y_RADIUS = 30f;
    private static final float DEFAULT_POINTER_RADIUS = 7f;
    private static final float DEFAULT_POINTER_HALO_WIDTH = 6f;
    private static final float DEFAULT_POINTER_HALO_BORDER_WIDTH = 2f;
    private static final float DEFAULT_CIRCLE_STROKE_WIDTH = 5f;
    private static final float DEFAULT_START_ANGLE = 270f; // Geometric (clockwise, relative to 3 o'clock)
    private static final float DEFAULT_END_ANGLE = 270f; // Geometric (clockwise, relative to 3 o'clock)
    private static final int DEFAULT_MAX = 100;
    private static final int DEFAULT_PROGRESS = 0;
    private static final int DEFAULT_CIRCLE_COLOR = Color.DKGRAY;
    private static final int DEFAULT_CIRCLE_PROGRESS_COLOR = Color.argb(235, 74, 138, 255);
    private static final int DEFAULT_POINTER_COLOR = Color.argb(235, 74, 138, 255);
    private static final int DEFAULT_POINTER_HALO_COLOR = Color.argb(135, 74, 138, 255);
    private static final int DEFAULT_POINTER_HALO_COLOR_ONTOUCH = Color.argb(135, 74, 138, 255);
    private static final int DEFAULT_CIRCLE_FILL_COLOR = Color.TRANSPARENT;
    private static final int DEFAULT_POINTER_ALPHA = 135;
    private static final int DEFAULT_POINTER_ALPHA_ONTOUCH = 100;
    private static final boolean DEFAULT_USE_CUSTOM_RADII = false;
    private static final boolean DEFAULT_MAINTAIN_EQUAL_CIRCLE = true;
    private static final boolean DEFAULT_MOVE_OUTSIDE_CIRCLE = false;
    private static final boolean DEFAULT_LOCK_ENABLED = true;
    /**
     * Used to scale the dp units to pixels
     */
    private final float DPTOPX_SCALE = getResources().getDisplayMetrics().density;
    /**
     * Minimum touch target size in DP. 48dp is the Android design recommendation
     */
    private final float MIN_TOUCH_TARGET_DP = 48;
    /**
     * {@code Paint} instance used to draw the inactive circle.
     */
    private Paint mCirclePaint;

    /**
     * {@code Paint} instance used to draw the circle fill.
     */
    private Paint mCircleFillPaint;

    /**
     * {@code Paint} instance used to draw the active circle (represents progress).
     */
    private Paint mCircleProgressPaint;

    /**
     * {@code Paint} instance used to draw the glow from the active circle.
     */
    private Paint mCircleProgressGlowPaint;

    /**
     * {@code Paint} instance used to draw the center of the pointer.
     * Note: This is broken on 4.0+, as BlurMasks do not work with hardware acceleration.
     */
    private Paint mPointerPaint;

    /**
     * {@code Paint} instance used to draw the halo of the pointer.
     * Note: The halo is the part that changes transparency.
     */
    private Paint mPointerHaloPaint;

    /**
     * {@code Paint} instance used to draw the border of the pointer, outside of the halo.
     */
    private Paint mPointerHaloBorderPaint;

    /**
     * The width of the circle (in pixels).
     */
    private float mCircleStrokeWidth;

    /**
     * The X radius of the circle (in pixels).
     */
    private float mCircleXRadius;

    /**
     * The Y radius of the circle (in pixels).
     */
    private float mCircleYRadius;

    /**
     * The radius of the pointer (in pixels).
     */
    private float mPointerRadius;

    /**
     * The width of the pointer halo (in pixels).
     */
    private float mPointerHaloWidth;

    /**
     * The width of the pointer halo border (in pixels).
     */
    private float mPointerHaloBorderWidth;

    /**
     * Start angle of the CircularSeekBar.
     * Note: If mStartAngle and mEndAngle are set to the same angle, 0.1 is subtracted
     * from the mEndAngle to make the circle function properly.
     */
    private float mStartAngle;

    /**
     * End angle of the CircularSeekBar.
     * Note: If mStartAngle and mEndAngle are set to the same angle, 0.1 is subtracted
     * from the mEndAngle to make the circle function properly.
     */
    private float mEndAngle;

    /**
     * {@code RectF} that represents the circle (or ellipse) of the seekbar.
     */
    private RectF mCircleRectF = new RectF();

    /**
     * Holds the color value for {@code mPointerPaint} before the {@code Paint} instance is created.
     */
    private int mPointerColor = DEFAULT_POINTER_COLOR;

    /**
     * Holds the color value for {@code mPointerHaloPaint} before the {@code Paint} instance is created.
     */
    private int mPointerHaloColor = DEFAULT_POINTER_HALO_COLOR;

    /**
     * Holds the color value for {@code mPointerHaloPaint} before the {@code Paint} instance is created.
     */
    private int mPointerHaloColorOnTouch = DEFAULT_POINTER_HALO_COLOR_ONTOUCH;

    /**
     * Holds the color value for {@code mCirclePaint} before the {@code Paint} instance is created.
     */
    private int mCircleColor = DEFAULT_CIRCLE_COLOR;

    /**
     * Holds the color value for {@code mCircleFillPaint} before the {@code Paint} instance is created.
     */
    private int mCircleFillColor = DEFAULT_CIRCLE_FILL_COLOR;

    /**
     * Holds the color value for {@code mCircleProgressPaint} before the {@code Paint} instance is created.
     */
    private int mCircleProgressColor = DEFAULT_CIRCLE_PROGRESS_COLOR;

    /**
     * Holds the alpha value for {@code mPointerHaloPaint}.
     */
    private int mPointerAlpha = DEFAULT_POINTER_ALPHA;

    /**
     * Holds the OnTouch alpha value for {@code mPointerHaloPaint}.
     */
    private int mPointerAlphaOnTouch = DEFAULT_POINTER_ALPHA_ONTOUCH;

    /**
     * Distance (in degrees) that the the circle/semi-circle makes up.
     * This amount represents the max of the circle in degrees.
     */
    private float mTotalCircleDegrees;

    /**
     * Distance (in degrees) that the current progress makes up in the circle.
     */
    private float mProgressDegrees;

    /**
     * {@code Path} used to draw the circle/semi-circle.
     */
    private Path mCirclePath;

    /**
     * {@code Path} used to draw the progress on the circle.
     */
    private Path mCircleProgressPath;

    /**
     * Max value that this CircularSeekBar is representing.
     */
    private int mMax;

    /**
     * Progress value that this CircularSeekBar is representing.
     */
    private int mProgress;

    /**
```

```
     * If true, then the user can specify the X and Y radii.
     * If false, then the View itself determines the size of the CircularSeekBar.
     */
    private boolean mCustomRadii;

    /**
     * Maintain a perfect circle (equal x and y radius), regardless of view or custom attributes.
     * The smaller of the two radii will always be used in this case.
     * The default is to be a circle and not an ellipse, due to the behavior of the ellipse.
     */
    private boolean mMaintainEqualCircle;

    /**
     * Once a user has touched the circle, this determines if moving outside the circle is able
     * to change the position of the pointer (and in turn, the progress).
     */
    private boolean mMoveOutsideCircle;

    /**
     * Used for enabling/disabling the lock option for easier hitting of the 0 progress mark.
     */
    private boolean lockEnabled = true;

    /**
     * Used for when the user moves beyond the start of the circle when moving counter clockwise.
     * Makes it easier to hit the 0 progress mark.
     */
    private boolean lockAtStart = true;

    /**
     * Used for when the user moves beyond the end of the circle when moving clockwise.
     * Makes it easier to hit the 100% (max) progress mark.
     */
    private boolean lockAtEnd = false;

    /**
     * When the user is touching the circle on ACTION_DOWN, this is set to true.
     * Used when touching the CircularSeekBar.
     */
    private boolean mUserIsMovingPointer = false;

    /**
     * Represents the clockwise distance from {@code mStartAngle} to the touch angle.
     * Used when touching the CircularSeekBar.
     */
    private float cwDistanceFromStart;

    /**
     * Represents the counter-clockwise distance from {@code mStartAngle} to the touch angle.
     * Used when touching the CircularSeekBar.
     */
    private float ccwDistanceFromStart;

    /**
     * Represents the clockwise distance from {@code mEndAngle} to the touch angle.
     * Used when touching the CircularSeekBar.
     */
    private float cwDistanceFromEnd;

    /**
     * Represents the counter-clockwise distance from {@code mEndAngle} to the touch angle.
     * Used when touching the CircularSeekBar.
     * Currently unused, but kept just in case.
     */
    @SuppressWarnings("unused")
    private float ccwDistanceFromEnd;

    /**
     * The previous touch action value for {@code cwDistanceFromStart}.
     * Used when touching the CircularSeekBar.
```

```java
     */
    private float lastCWDistanceFromStart;

    /**
     * Represents the clockwise distance from {@code mPointerPosition} to the touch angle.
     * Used when touching the CircularSeekBar.
     */
    private float cwDistanceFromPointer;

    /**
     * Represents the counter-clockwise distance from {@code mPointerPosition} to the touch angle.
     * Used when touching the CircularSeekBar.
     */
    private float ccwDistanceFromPointer;

    /**
     * True if the user is moving clockwise around the circle, false if moving counter-clockwise.
     * Used when touching the CircularSeekBar.
     */
    private boolean mIsMovingCW;

    /**
     * The width of the circle used in the {@code RectF} that is used to draw it.
     * Based on either the View width or the custom X radius.
     */
    private float mCircleWidth;

    /**
     * The height of the circle used in the {@code RectF} that is used to draw it.
     * Based on either the View width or the custom Y radius.
     */
    private float mCircleHeight;

    /**
     * Represents the progress mark on the circle, in geometric degrees.
     * This is not provided by the user; it is calculated;
     */
    private float mPointerPosition;

    /**
     * Pointer position in terms of X and Y coordinates.
     */
    private float[] mPointerPositionXY = new float[2];

    /**
     * Listener.
     */
    private OnCircularSeekBarChangeListener mOnCircularSeekBarChangeListener;

    public CircularSeekBar(Context context) {
        super(context);
        init(null, 0);
    }

    public CircularSeekBar(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(attrs, 0);
    }

    public CircularSeekBar(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        init(attrs, defStyle);
    }

    /**
     * Initialize the CircularSeekBar with the attributes from the XML style.
     * Uses the defaults defined at the top of this file when an attribute is not specified by the user.
     *
     * @param attrArray TypedArray containing the attributes.
     */
    private void initAttributes(TypedArray attrArray) {
        mCircleXRadius = attrArray.getFloat(R.styleable.CircularSeekBar_circle_x_radius, DEFAULT_CIRCLE_X_RADIUS) * DPTOPX_S
        mCircleYRadius = attrArray.getFloat(R.styleable.CircularSeekBar_circle_y_radius, DEFAULT_CIRCLE_Y_RADIUS) * DPTOPX_S
        mPointerRadius = attrArray.getFloat(R.styleable.CircularSeekBar_pointer_radius, DEFAULT_POINTER_RADIUS) * DPTOPX_SCA
        mPointerHaloWidth = attrArray.getFloat(R.styleable.CircularSeekBar_pointer_halo_width, DEFAULT_POINTER_HALO_WIDTH) *
        mPointerHaloBorderWidth = attrArray.getFloat(R.styleable.CircularSeekBar_pointer_halo_border_width, DEFAULT_POINTER_
        mCircleStrokeWidth = attrArray.getFloat(R.styleable.CircularSeekBar_circle_stroke_width, DEFAULT_CIRCLE_STROKE_WIDTH

        String tempColor = attrArray.getString(R.styleable.CircularSeekBar_pointer_color);
        if (tempColor != null) {
            try {
                mPointerColor = Color.parseColor(tempColor);
            } catch (IllegalArgumentException e) {
                mPointerColor = DEFAULT_POINTER_COLOR;
            }
        }

        tempColor = attrArray.getString(R.styleable.CircularSeekBar_pointer_halo_color);
        if (tempColor != null) {
            try {
                mPointerHaloColor = Color.parseColor(tempColor);
            } catch (IllegalArgumentException e) {
                mPointerHaloColor = DEFAULT_POINTER_HALO_COLOR;
            }
        }

        tempColor = attrArray.getString(R.styleable.CircularSeekBar_pointer_halo_color_ontouch);
        if (tempColor != null) {
            try {
                mPointerHaloColorOnTouch = Color.parseColor(tempColor);
            } catch (IllegalArgumentException e) {
                mPointerHaloColorOnTouch = DEFAULT_POINTER_HALO_COLOR_ONTOUCH;
            }
        }

        tempColor = attrArray.getString(R.styleable.CircularSeekBar_circle_color);
        if (tempColor != null) {
            try {
                mCircleColor = Color.parseColor(tempColor);
            } catch (IllegalArgumentException e) {
                mCircleColor = DEFAULT_CIRCLE_COLOR;
            }
        }

        tempColor = attrArray.getString(R.styleable.CircularSeekBar_circle_progress_color);
        if (tempColor != null) {
            try {
                mCircleProgressColor = Color.parseColor(tempColor);
            } catch (IllegalArgumentException e) {
                mCircleProgressColor = DEFAULT_CIRCLE_PROGRESS_COLOR;
            }
        }

        tempColor = attrArray.getString(R.styleable.CircularSeekBar_circle_fill);
        if (tempColor != null) {
            try {
                mCircleFillColor = Color.parseColor(tempColor);
            } catch (IllegalArgumentException e) {
                mCircleFillColor = DEFAULT_CIRCLE_FILL_COLOR;
            }
        }

        mPointerAlpha = Color.alpha(mPointerHaloColor);

        mPointerAlphaOnTouch = attrArray.getInt(R.styleable.CircularSeekBar_pointer_alpha_ontouch, DEFAULT_POINTER_ALPHA_ONT
        if (mPointerAlphaOnTouch > 255 || mPointerAlphaOnTouch < 0) {
            mPointerAlphaOnTouch = DEFAULT_POINTER_ALPHA_ONTOUCH;
        }

        mMax = attrArray.getInt(R.styleable.CircularSeekBar_max, DEFAULT_MAX);
        mProgress = attrArray.getInt(R.styleable.CircularSeekBar_progress, DEFAULT_PROGRESS);
        mCustomRadii = attrArray.getBoolean(R.styleable.CircularSeekBar_use_custom_radii, DEFAULT_USE_CUSTOM_RADII);
        mMaintainEqualCircle = attrArray.getBoolean(R.styleable.CircularSeekBar_maintain_equal_circle, DEFAULT_MAINTAIN_EQUA
        mMoveOutsideCircle = attrArray.getBoolean(R.styleable.CircularSeekBar_move_outside_circle, DEFAULT_MOVE_OUTSIDE_CIRC
        lockEnabled = attrArray.getBoolean(R.styleable.CircularSeekBar_lock_enabled, DEFAULT_LOCK_ENABLED);

        // Modulo 360 right now to avoid constant conversion
        mStartAngle = ((360f + (attrArray.getFloat((R.styleable.CircularSeekBar_start_angle), DEFAULT_START_ANGLE) % 360f))
        mEndAngle = ((360f + (attrArray.getFloat((R.styleable.CircularSeekBar_end_angle), DEFAULT_END_ANGLE) % 360f)) % 360f

        if (mStartAngle == mEndAngle) {
            //mStartAngle = mStartAngle + 1f;
            mEndAngle = mEndAngle - .1f;
        }


    }

    /**
     * Initializes the {@code Paint} objects with the appropriate styles.
     */
    private void initPaints() {
        mCirclePaint = new Paint();
        mCirclePaint.setAntiAlias(true);
        mCirclePaint.setDither(true);
        mCirclePaint.setColor(mCircleColor);
        mCirclePaint.setStrokeWidth(mCircleStrokeWidth);
        mCirclePaint.setStyle(Paint.Style.STROKE);
        mCirclePaint.setStrokeJoin(Paint.Join.ROUND);
        mCirclePaint.setStrokeCap(Paint.Cap.ROUND);

        mCircleFillPaint = new Paint();
        mCircleFillPaint.setAntiAlias(true);
        mCircleFillPaint.setDither(true);
        mCircleFillPaint.setColor(mCircleFillColor);
        mCircleFillPaint.setStyle(Paint.Style.FILL);

        mCircleProgressPaint = new Paint();
        mCircleProgressPaint.setAntiAlias(true);
        mCircleProgressPaint.setDither(true);
        mCircleProgressPaint.setColor(mCircleProgressColor);
        mCircleProgressPaint.setStrokeWidth(mCircleStrokeWidth);
        mCircleProgressPaint.setStyle(Paint.Style.STROKE);
        mCircleProgressPaint.setStrokeJoin(Paint.Join.ROUND);
        mCircleProgressPaint.setStrokeCap(Paint.Cap.ROUND);

        mCircleProgressGlowPaint = new Paint();
        mCircleProgressGlowPaint.set(mCircleProgressPaint);
        mCircleProgressGlowPaint.setMaskFilter(new BlurMaskFilter((5f * DPTOPX_SCALE), BlurMaskFilter.Blur.NORMAL));

        mPointerPaint = new Paint();
        mPointerPaint.setAntiAlias(true);
        mPointerPaint.setDither(true);
        mPointerPaint.setStyle(Paint.Style.FILL);
        mPointerPaint.setColor(mPointerColor);
        mPointerPaint.setStrokeWidth(mPointerRadius);

        mPointerHaloPaint = new Paint();
        mPointerHaloPaint.set(mPointerPaint);
        mPointerHaloPaint.setColor(mPointerHaloColor);
        mPointerHaloPaint.setAlpha(mPointerAlpha);
        mPointerHaloPaint.setStrokeWidth(mPointerRadius + mPointerHaloWidth);

        mPointerHaloBorderPaint = new Paint();
        mPointerHaloBorderPaint.set(mPointerPaint);
        mPointerHaloBorderPaint.setStrokeWidth(mPointerHaloBorderWidth);
        mPointerHaloBorderPaint.setStyle(Paint.Style.STROKE);

    }
    /**
     * Calculates the total degrees between mStartAngle and mEndAngle, and sets mTotalCircleDegrees
     * to this value.
     */
    private void calculateTotalDegrees() {
        mTotalCircleDegrees = (360f - (mStartAngle - mEndAngle)) % 360f; // Length of the entire circle/arc
        if (mTotalCircleDegrees <= 0f) {
            mTotalCircleDegrees = 360f;
        }
    }

    /**
     * Calculate the degrees that the progress represents. Also called the sweep angle.
     * Sets mProgressDegrees to that value.
     */
    private void calculateProgressDegrees() {
        mProgressDegrees = mPointerPosition - mStartAngle; // Verified
        mProgressDegrees = (mProgressDegrees < 0 ? 360f + mProgressDegrees : mProgressDegrees); // Verified
    }

    /**
     * Calculate the pointer position (and the end of the progress arc) in degrees.
     * Sets mPointerPosition to that value.
     */
    private void calculatePointerAngle() {
        float progressPercent = ((float) mProgress / (float) mMax);
        mPointerPosition = (progressPercent * mTotalCircleDegrees) + mStartAngle;
        mPointerPosition = mPointerPosition % 360f;
    }

    private void calculatePointerXYPosition() {
        PathMeasure pm = new PathMeasure(mCircleProgressPath, false);
        boolean returnValue = pm.getPosTan(pm.getLength(), mPointerPositionXY, null);
        if (!returnValue) {
            pm = new PathMeasure(mCirclePath, false);
            returnValue = pm.getPosTan(0, mPointerPositionXY, null);
        }
    }

    /**
     * Initialize the {@code Path} objects with the appropriate values.
     */
    private void initPaths() {
        mCirclePath = new Path();
        mCirclePath.addArc(mCircleRectF, mStartAngle, mTotalCircleDegrees);

        mCircleProgressPath = new Path();
        mCircleProgressPath.addArc(mCircleRectF, mStartAngle, mProgressDegrees);
    }

    /**
     * Initialize the {@code RectF} objects with the appropriate values.
     */
    private void initRects() {
        mCircleRectF.set(-mCircleWidth, -mCircleHeight, mCircleWidth, mCircleHeight);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        canvas.translate(this.getWidth() / 2, this.getHeight() / 2);

        canvas.drawPath(mCirclePath, mCirclePaint);
        canvas.drawPath(mCircleProgressPath, mCircleProgressGlowPaint);
        canvas.drawPath(mCircleProgressPath, mCircleProgressPaint);

        canvas.drawPath(mCirclePath, mCircleFillPaint);

        canvas.drawCircle(mPointerPositionXY[0], mPointerPositionXY[1], mPointerRadius + mPointerHaloWidth, mPointerHaloPain
        canvas.drawCircle(mPointerPositionXY[0], mPointerPositionXY[1], mPointerRadius, mPointerPaint);
        if (mUserIsMovingPointer) {
            canvas.drawCircle(mPointerPositionXY[0], mPointerPositionXY[1], mPointerRadius + mPointerHaloWidth + (mPointerHa
        }
    }

    /**
     * Get the progress of the CircularSeekBar.
     *
     * @return The progress of the CircularSeekBar.
     */
    public int getProgress() {
        int progress = Math.round((float) mMax * mProgressDegrees / mTotalCircleDegrees);
        return progress;
    }

    /**
     * Set the progress of the CircularSeekBar.
     * If the progress is the same, then any listener will not receive a onProgressChanged event.
     *
     * @param progress The progress to set the CircularSeekBar to.
     */
    public void setProgress(int progress) {
        if (mProgress != progress) {
            mProgress = progress;
            if (mOnCircularSeekBarChangeListener != null) {
                mOnCircularSeekBarChangeListener.onProgressChanged(this, progress, false);
            }

            recalculateAll();
            invalidate();
        }
    }

    private void setProgressBasedOnAngle(float angle) {
        mPointerPosition = angle;
        calculateProgressDegrees();
        mProgress = Math.round((float) mMax * mProgressDegrees / mTotalCircleDegrees);
    }

    private void recalculateAll() {
        calculateTotalDegrees();
        calculatePointerAngle();
        calculateProgressDegrees();

        initRects();

        initPaths();

        calculatePointerXYPosition();
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int height = getDefaultSize(getSuggestedMinimumHeight(), heightMeasureSpec);
        int width = getDefaultSize(getSuggestedMinimumWidth(), widthMeasureSpec);
        if (mMaintainEqualCircle) {
            int min = Math.min(width, height);
            setMeasuredDimension(min, min);
        } else {
            setMeasuredDimension(width, height);
        }

        // Set the circle width and height based on the view for the moment
        mCircleHeight = (float) height / 2f - mCircleStrokeWidth - mPointerRadius - (mPointerHaloBorderWidth * 1.5f);
        mCircleWidth = (float) width / 2f - mCircleStrokeWidth - mPointerRadius - (mPointerHaloBorderWidth * 1.5f);

        // If it is not set to use custom
        if (mCustomRadii) {
            // Check to make sure the custom radii are not out of the view. If they are, just use the view values
```

```java
            if ((mCircleYRadius - mCircleStrokeWidth - mPointerRadius - mPointerHaloBorderWidth) < mCircleHeight) {
                mCircleHeight = mCircleYRadius - mCircleStrokeWidth - mPointerRadius - (mPointerHaloBorderWidth * 1.5f);
            }

            if ((mCircleXRadius - mCircleStrokeWidth - mPointerRadius - mPointerHaloBorderWidth) < mCircleWidth) {
                mCircleWidth = mCircleXRadius - mCircleStrokeWidth - mPointerRadius - (mPointerHaloBorderWidth * 1.5f);
            }
        }

        if (mMaintainEqualCircle) { // Applies regardless of how the values were determined
            float min = Math.min(mCircleHeight, mCircleWidth);
            mCircleHeight = min;
            mCircleWidth = min;
        }

        recalculateAll();
    }

    public boolean isLockEnabled() {
        return lockEnabled;
    }

    public void setLockEnabled(boolean lockEnabled) {
        this.lockEnabled = lockEnabled;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        // Convert coordinates to our internal coordinate system
        float x = event.getX() - getWidth() / 2;
        float y = event.getY() - getHeight() / 2;

        // Get the distance from the center of the circle in terms of x and y
        float distanceX = mCircleRectF.centerX() - x;
        float distanceY = mCircleRectF.centerY() - y;

        // Get the distance from the center of the circle in terms of a radius
        float touchEventRadius = (float) Math.sqrt((Math.pow(distanceX, 2) + Math.pow(distanceY, 2)));

        float minimumTouchTarget = MIN_TOUCH_TARGET_DP * DPTOPX_SCALE; // Convert minimum touch target into px
        float additionalRadius; // Either uses the minimumTouchTarget size or larger if the ring/pointer is larger

        if (mCircleStrokeWidth < minimumTouchTarget) { // If the width is less than the minimumTouchTarget, use the minimumT
            additionalRadius = minimumTouchTarget / 2;
        } else {
            additionalRadius = mCircleStrokeWidth / 2; // Otherwise use the width
        }
        float outerRadius = Math.max(mCircleHeight, mCircleWidth) + additionalRadius; // Max outer radius of the circle, inc
        float innerRadius = Math.min(mCircleHeight, mCircleWidth) - additionalRadius; // Min inner radius of the circle, inc

        if (mPointerRadius < (minimumTouchTarget / 2)) { // If the pointer radius is less than the minimumTouchTarget, use t
            additionalRadius = minimumTouchTarget / 2;
        } else {
            additionalRadius = mPointerRadius; // Otherwise use the radius
        }

        float touchAngle;
        touchAngle = (float) ((java.lang.Math.atan2(y, x) / Math.PI * 180) % 360); // Verified
        touchAngle = (touchAngle < 0 ? 360 + touchAngle : touchAngle); // Verified

        cwDistanceFromStart = touchAngle - mStartAngle; // Verified
        cwDistanceFromStart = (cwDistanceFromStart < 0 ? 360f + cwDistanceFromStart : cwDistanceFromStart); // Verified
        ccwDistanceFromStart = 360f - cwDistanceFromStart; // Verified

        cwDistanceFromEnd = touchAngle - mEndAngle; // Verified
        cwDistanceFromEnd = (cwDistanceFromEnd < 0 ? 360f + cwDistanceFromEnd : cwDistanceFromEnd); // Verified
        ccwDistanceFromEnd = 360f - cwDistanceFromEnd; // Verified

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                // These are only used for ACTION_DOWN for handling if the pointer was the part that was touched
                float pointerRadiusDegrees = (float) ((mPointerRadius * 180) / (Math.PI * Math.max(mCircleHeight, mCircleWid
                cwDistanceFromPointer = touchAngle - mPointerPosition;
                cwDistanceFromPointer = (cwDistanceFromPointer < 0 ? 360f + cwDistanceFromPointer : cwDistanceFromPointer);
                ccwDistanceFromPointer = 360f - cwDistanceFromPointer;
                // This is for if the first touch is on the actual pointer.
                if (((touchEventRadius >= innerRadius) && (touchEventRadius <= outerRadius)) && ((cwDistanceFromPointer <= p
                    setProgressBasedOnAngle(mPointerPosition);
                    lastCWDistanceFromStart = cwDistanceFromStart;
                    mIsMovingCW = true;
                    mPointerHaloPaint.setAlpha(mPointerAlphaOnTouch);
                    mPointerHaloPaint.setColor(mPointerHaloColorOnTouch);
                    recalculateAll();
                    invalidate();
                    if (mOnCircularSeekBarChangeListener != null) {
                        mOnCircularSeekBarChangeListener.onStartTrackingTouch(this);
                    }
                    mUserIsMovingPointer = true;
                    lockAtEnd = false;
                    lockAtStart = false;
                } else if (cwDistanceFromStart > mTotalCircleDegrees) { // If the user is touching outside of the start AND
                    mUserIsMovingPointer = false;
                    return false;
                } else if ((touchEventRadius >= innerRadius) && (touchEventRadius <= outerRadius)) { // If the user is touch
                    setProgressBasedOnAngle(touchAngle);
                    lastCWDistanceFromStart = cwDistanceFromStart;
                    mIsMovingCW = true;
                    mPointerHaloPaint.setAlpha(mPointerAlphaOnTouch);
                    mPointerHaloPaint.setColor(mPointerHaloColorOnTouch);
                    recalculateAll();
                    invalidate();
                    if (mOnCircularSeekBarChangeListener != null) {
                        mOnCircularSeekBarChangeListener.onStartTrackingTouch(this);
                        mOnCircularSeekBarChangeListener.onProgressChanged(this, mProgress, true);
                    }
                    mUserIsMovingPointer = true;
                    lockAtEnd = false;
                    lockAtStart = false;
                } else { // If the user is not touching near the circle
                    mUserIsMovingPointer = false;
                    return false;
                }
                break;
            case MotionEvent.ACTION_MOVE:
                if (mUserIsMovingPointer) {
                    if (lastCWDistanceFromStart < cwDistanceFromStart) {
                        if ((cwDistanceFromStart - lastCWDistanceFromStart) > 180f && !mIsMovingCW) {
                            lockAtStart = true;
                            lockAtEnd = false;
                        } else {
                            mIsMovingCW = true;
                        }
                    } else {
                        if ((lastCWDistanceFromStart - cwDistanceFromStart) > 180f && mIsMovingCW) {
                            lockAtEnd = true;
                            lockAtStart = false;
                        } else {
                            mIsMovingCW = false;
                        }
                    }

                    if (lockAtStart && mIsMovingCW) {
                        lockAtStart = false;
                    }
                    if (lockAtEnd && !mIsMovingCW) {
                        lockAtEnd = false;
                    }
                    if (lockAtStart && !mIsMovingCW && (ccwDistanceFromStart > 90)) {
                        lockAtStart = false;
                    }
                if (lockAtEnd && mIsMovingCW && (cwDistanceFromEnd > 90)) {
                    lockAtEnd = false;
                }
                // Fix for passing the end of a semi-circle quickly
                if (!lockAtEnd && cwDistanceFromStart > mTotalCircleDegrees && mIsMovingCW && lastCWDistanceFromStart <
                    lockAtEnd = true;
                }

                if (lockAtStart && lockEnabled) {
                    // TODO: Add a check if mProgress is already 0, in which case don't call the listener
                    mProgress = 0;
                    recalculateAll();
                    invalidate();
                    if (mOnCircularSeekBarChangeListener != null) {
                        mOnCircularSeekBarChangeListener.onProgressChanged(this, mProgress, true);
                    }

                } else if (lockAtEnd && lockEnabled) {
                    mProgress = mMax;
                    recalculateAll();
                    invalidate();
                    if (mOnCircularSeekBarChangeListener != null) {
                        mOnCircularSeekBarChangeListener.onProgressChanged(this, mProgress, true);
                    }
                } else if ((mMoveOutsideCircle) || (touchEventRadius <= outerRadius)) {
                    if (!(cwDistanceFromStart > mTotalCircleDegrees)) {
                        setProgressBasedOnAngle(touchAngle);
                    }
                    recalculateAll();
                    invalidate();
                    if (mOnCircularSeekBarChangeListener != null) {
                        mOnCircularSeekBarChangeListener.onProgressChanged(this, mProgress, true);
                    }
                } else {
                    break;
                }

                lastCWDistanceFromStart = cwDistanceFromStart;
            } else {
                return false;
            }
            break;
        case MotionEvent.ACTION_UP:
            mPointerHaloPaint.setAlpha(mPointerAlpha);
            mPointerHaloPaint.setColor(mPointerHaloColor);
            if (mUserIsMovingPointer) {
                mUserIsMovingPointer = false;
                invalidate();
                if (mOnCircularSeekBarChangeListener != null) {
                    mOnCircularSeekBarChangeListener.onStopTrackingTouch(this);
                }
            } else {
                return false;
            }
            break;
        case MotionEvent.ACTION_CANCEL: // Used when the parent view intercepts touches for things like scrolling
            mPointerHaloPaint.setAlpha(mPointerAlpha);
            mPointerHaloPaint.setColor(mPointerHaloColor);
            mUserIsMovingPointer = false;
            invalidate();
            break;
    }

    if (event.getAction() == MotionEvent.ACTION_MOVE && getParent() != null) {
        getParent().requestDisallowInterceptTouchEvent(true);
    }

    return true;
}
```

```java
    private void init(AttributeSet attrs, int defStyle) {
        final TypedArray attrArray = getContext().obtainStyledAttributes(attrs, R.styleable.CircularSeekBar, defStyle, 0);

        initAttributes(attrArray);

        attrArray.recycle();

        initPaints();
    }

    @Override
    protected Parcelable onSaveInstanceState() {
        Parcelable superState = super.onSaveInstanceState();

        Bundle state = new Bundle();
        state.putParcelable("PARENT", superState);
        state.putInt("MAX", mMax);
        state.putInt("PROGRESS", mProgress);
        state.putInt("mCircleColor", mCircleColor);
        state.putInt("mCircleProgressColor", mCircleProgressColor);
        state.putInt("mPointerColor", mPointerColor);
        state.putInt("mPointerHaloColor", mPointerHaloColor);
        state.putInt("mPointerHaloColorOnTouch", mPointerHaloColorOnTouch);
        state.putInt("mPointerAlpha", mPointerAlpha);
        state.putInt("mPointerAlphaOnTouch", mPointerAlphaOnTouch);
        state.putBoolean("lockEnabled", lockEnabled);

        return state;
    }

    @Override
    protected void onRestoreInstanceState(Parcelable state) {
        Bundle savedState = (Bundle) state;

        Parcelable superState = savedState.getParcelable("PARENT");
        super.onRestoreInstanceState(superState);

        mMax = savedState.getInt("MAX");
        mProgress = savedState.getInt("PROGRESS");
        mCircleColor = savedState.getInt("mCircleColor");
        mCircleProgressColor = savedState.getInt("mCircleProgressColor");
        mPointerColor = savedState.getInt("mPointerColor");
        mPointerHaloColor = savedState.getInt("mPointerHaloColor");
        mPointerHaloColorOnTouch = savedState.getInt("mPointerHaloColorOnTouch");
        mPointerAlpha = savedState.getInt("mPointerAlpha");
        mPointerAlphaOnTouch = savedState.getInt("mPointerAlphaOnTouch");
        lockEnabled = savedState.getBoolean("lockEnabled");

        initPaints();

        recalculateAll();
    }


    public void setOnSeekBarChangeListener(OnCircularSeekBarChangeListener l) {
        mOnCircularSeekBarChangeListener = l;
    }

    /**
     * Gets the circle color.
     *
     * @return An integer color value for the circle
     */
    public int getCircleColor() {
        return mCircleColor;
    }

    /**
     * Sets the circle color.
     *
     * @param color the color of the circle
     */
    public void setCircleColor(int color) {
        mCircleColor = color;
        mCirclePaint.setColor(mCircleColor);
        invalidate();
    }

    /**
     * Gets the circle progress color.
     *
     * @return An integer color value for the circle progress
     */
    public int getCircleProgressColor() {
        return mCircleProgressColor;
    }

    /**
     * Sets the circle progress color.
     *
     * @param color the color of the circle progress
     */
    public void setCircleProgressColor(int color) {
        mCircleProgressColor = color;
        mCircleProgressPaint.setColor(mCircleProgressColor);
        invalidate();
    }

    /**
     * Gets the pointer color.
     *
     * @return An integer color value for the pointer
     */
    public int getPointerColor() {
        return mPointerColor;
    }

    /**
     * Sets the pointer color.
     *
     * @param color the color of the pointer
     */
    public void setPointerColor(int color) {
        mPointerColor = color;
        mPointerPaint.setColor(mPointerColor);
        invalidate();
    }

    /**
     * Gets the pointer halo color.
     *
     * @return An integer color value for the pointer halo
     */
    public int getPointerHaloColor() {
        return mPointerHaloColor;
    }

    /**
     * Sets the pointer halo color.
     *
     * @param color the color of the pointer halo
     */
    public void setPointerHaloColor(int color) {
        mPointerHaloColor = color;
        mPointerHaloPaint.setColor(mPointerHaloColor);
        invalidate();
    }

    /**
     * Gets the pointer alpha value.
     *
     * @return An integer alpha value for the pointer (0..255)
     */
    public int getPointerAlpha() {
        return mPointerAlpha;
    }

    /**
     * Sets the pointer alpha.
     *
     * @param alpha the alpha of the pointer
     */
    public void setPointerAlpha(int alpha) {
        if (alpha >= 0 && alpha <= 255) {
            mPointerAlpha = alpha;
            mPointerHaloPaint.setAlpha(mPointerAlpha);
            invalidate();
        }
    }

    /**
     * Gets the pointer alpha value when touched.
     *
     * @return An integer alpha value for the pointer (0..255) when touched
     */
    public int getPointerAlphaOnTouch() {
        return mPointerAlphaOnTouch;
    }

    /**
     * Sets the pointer alpha when touched.
     *
     * @param alpha the alpha of the pointer (0..255) when touched
     */
    public void setPointerAlphaOnTouch(int alpha) {
        if (alpha >= 0 && alpha <= 255) {
            mPointerAlphaOnTouch = alpha;
        }
    }

    /**
     * Gets the circle fill color.
     *
     * @return An integer color value for the circle fill
     */
    public int getCircleFillColor() {
        return mCircleFillColor;
    }

    /**
     * Sets the circle fill color.
     *
     * @param color the color of the circle fill
     */
    public void setCircleFillColor(int color) {
        mCircleFillColor = color;
        mCircleFillPaint.setColor(mCircleFillColor);
        invalidate();
    }

    /**
     * Get the current max of the CircularSeekBar.
     *
     * @return Synchronized integer value of the max.
     */
    public synchronized int getMax() {
        return mMax;
    }

    /**
```

```
     * Set the max of the CircularSeekBar.
     * If the new max is less than the current progress, then the progress will be set to zero.
     * If the progress is changed as a result, then any listener will receive a onProgressChanged event.
     *
     * @param max The new max for the CircularSeekBar.
     */
    public void setMax(int max) {
        if (!(max <= 0)) { // Check to make sure it's greater than zero
            if (max <= mProgress) {
                mProgress = 0; // If the new max is less than current progress, set progress to zero
                if (mOnCircularSeekBarChangeListener != null) {
                    mOnCircularSeekBarChangeListener.onProgressChanged(this, mProgress, false);
                }
            }
            mMax = max;

            recalculateAll();
            invalidate();
        }
    }

    /**
     * Listener for the CircularSeekBar. Implements the same methods as the normal OnSeekBarChangeListener.
     */
    public interface OnCircularSeekBarChangeListener {

        void onProgressChanged(CircularSeekBar circularSeekBar, int progress, boolean fromUser);

        void onStopTrackingTouch(CircularSeekBar seekBar);

        void onStartTrackingTouch(CircularSeekBar seekBar);
    }

}
```

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Rect;
import android.graphics.drawable.Drawable;
import android.support.v4.content.ContextCompat;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.view.View;

import com.naman14.timber.R;
import com.naman14.timber.utils.PreferencesUtility;

public class DividerItemDecoration extends RecyclerView.ItemDecoration {

    public static final int HORIZONTAL_LIST = LinearLayoutManager.HORIZONTAL;
    public static final int VERTICAL_LIST = LinearLayoutManager.VERTICAL;
    private static final int[] ATTRS = new int[]{
            android.R.attr.listDivider
    };
    private Drawable mDivider;

    private int mOrientation;

    public DividerItemDecoration(Context context, int orientation) {
        final TypedArray a = context.obtainStyledAttributes(ATTRS);
        if (PreferencesUtility.getInstance(context).getTheme().equals("light"))
            mDivider = ContextCompat.getDrawable(context, R.drawable.item_divider_black);
        else mDivider = ContextCompat.getDrawable(context, R.drawable.item_divider_white);
//        mDivider = a.getDrawable(0);
        a.recycle();
        setOrientation(orientation);
    }

    public DividerItemDecoration(Context context, int orientation, int resId) {
        mDivider = ContextCompat.getDrawable(context, resId);
        setOrientation(orientation);
    }

    public void setOrientation(int orientation) {
        if (orientation != HORIZONTAL_LIST && orientation != VERTICAL_LIST) {
            throw new IllegalArgumentException("invalid orientation");
        }
        mOrientation = orientation;
    }

    @Override
    public void onDraw(Canvas c, RecyclerView parent) {
        if (mOrientation == VERTICAL_LIST) {
            drawVertical(c, parent);
        } else {
            drawHorizontal(c, parent);
        }
    }

    public void drawVertical(Canvas c, RecyclerView parent) {
        final int left = parent.getPaddingLeft();
        final int right = parent.getWidth() - parent.getPaddingRight();

        final int childCount = parent.getChildCount();
        for (int i = 0; i < childCount; i++) {
            final View child = parent.getChildAt(i);
            final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child
                    .getLayoutParams();
            final int top = child.getBottom() + params.bottomMargin;
            final int bottom = top + mDivider.getIntrinsicHeight();
            mDivider.setBounds(left, top, right, bottom);
            mDivider.draw(c);
        }
    }

    public void drawHorizontal(Canvas c, RecyclerView parent) {
        final int top = parent.getPaddingTop();
        final int bottom = parent.getHeight() - parent.getPaddingBottom();

        final int childCount = parent.getChildCount();
        for (int i = 0; i < childCount; i++) {
            final View child = parent.getChildAt(i);
            final RecyclerView.LayoutParams params = (RecyclerView.LayoutParams) child
                    .getLayoutParams();
            final int left = child.getRight() + params.rightMargin;
            final int right = left + mDivider.getIntrinsicHeight();
            mDivider.setBounds(left, top, right, bottom);
            mDivider.draw(c);
        }
    }

    @Override
    public void getItemOffsets(Rect outRect, int itemPosition, RecyclerView parent) {
        if (mOrientation == VERTICAL_LIST) {
            outRect.set(0, 0, 0, mDivider.getIntrinsicHeight());
        } else {
            outRect.set(0, 0, mDivider.getIntrinsicWidth(), 0);
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\DragSortRecycler.java

```java
package com.naman14.timber.widgets;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.graphics.drawable.BitmapDrawable;
import android.support.annotation.Nullable;
import android.support.v7.widget.RecyclerView;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;


public class DragSortRecycler extends RecyclerView.ItemDecoration implements RecyclerView.OnItemTouchListener {

    final String TAG = "DragSortRecycler";

    final boolean DEBUG = false;
    OnItemMovedListener moveInterface;
    @Nullable
    OnDragStateChangedListener dragStateChangedListener;
    Paint bgColor = new Paint();
    private int dragHandleWidth = 0;
    private int selectedDragItemPos = -1;
    private int fingerAnchorY;
    RecyclerView.OnScrollListener scrollListener = new RecyclerView.OnScrollListener() {
        @Override
        public void onScrollStateChanged(RecyclerView recyclerView, int newState) {
            super.onScrollStateChanged(recyclerView, newState);
        }

        @Override
        public void onScrolled(RecyclerView recyclerView, int dx, int dy) {
            super.onScrolled(recyclerView, dx, dy);
            debugLog("Scrolled: " + dx + " " + dy);
            fingerAnchorY -= dy;
        }
    };
    private int fingerY;
    private int fingerOffsetInViewY;
    private float autoScrollWindow = 0.1f;
    private float autoScrollSpeed = 0.5f;
    private BitmapDrawable floatingItem;
    private Rect floatingItemStatingBounds;
    private Rect floatingItemBounds;
    private float floatingItemAlpha = 0.5f;
    private int floatingItemBgColor = 0;
    private int viewHandleId = -1;
    private boolean isDragging;

    private void debugLog(String log) {
        if (DEBUG)
            Log.d(TAG, log);
    }

    public RecyclerView.OnScrollListener getScrollListener() {
        return scrollListener;
    }

    /*
     * Set the item move interface
     */
    public void setOnItemMovedListener(OnItemMovedListener swif) {
        moveInterface = swif;
    }

    public void setViewHandleId(int id) {
        viewHandleId = id;
    }
    public void setLeftDragArea(int w) {
        dragHandleWidth = w;
    }

    public void setFloatingAlpha(float a) {
        floatingItemAlpha = a;
    }

    public void setFloatingBgColor(int c) {
        floatingItemBgColor = c;
    }

    /*
     Set the window at top and bottom of list, must be between 0 and 0.5
     For example 0.1 uses the top and bottom 10% of the lists for scrolling
     */
    public void setAutoScrollWindow(float w) {
        autoScrollWindow = w;
    }

    /*
    Set the autoscroll speed, default is 0.5
     */
    public void setAutoScrollSpeed(float speed) {
        autoScrollSpeed = speed;
    }

    @Override
    public void getItemOffsets(Rect outRect, View view, RecyclerView rv, RecyclerView.State state) {
        super.getItemOffsets(outRect, view, rv, state);

        debugLog("getItemOffsets");

        debugLog("View top = " + view.getTop());
        if (selectedDragItemPos != -1) {
            int itemPos = rv.getChildLayoutPosition(view);
            debugLog("itemPos =" + itemPos);

            if (!canDragOver(itemPos)) {
                return;
            }

            //Movement of finger
            float totalMovement = fingerY - fingerAnchorY;

            if (itemPos == selectedDragItemPos) {
                view.setVisibility(View.INVISIBLE);
            } else {
                //Make view visible incase invisible
                view.setVisibility(View.VISIBLE);

                //Find middle of the floatingItem
                float floatMiddleY = floatingItemBounds.top + floatingItemBounds.height() / 2;

                //Moving down the list
                //These will auto-animate if the device continually sends touch motion events
                // if (totalMovment>0)
                {
                    if ((itemPos > selectedDragItemPos) && (view.getTop() < floatMiddleY)) {
                        float amountUp = (floatMiddleY - view.getTop()) / (float) view.getHeight();
                        //  amountUp *= 0.5f;
                        if (amountUp > 1)
                            amountUp = 1;

                        outRect.top = -(int) (floatingItemBounds.height() * amountUp);
                        outRect.bottom = (int) (floatingItemBounds.height() * amountUp);
                    }

                }//Moving up the list
                // else if (totalMovment < 0)
                {
                    if ((itemPos < selectedDragItemPos) && (view.getBottom() > floatMiddleY)) {
                        float amountDown = ((float) view.getBottom() - floatMiddleY) / (float) view.getHeight();
                        //  amountDown *= 0.5f;
                        if (amountDown > 1)
                            amountDown = 1;

                        outRect.top = (int) (floatingItemBounds.height() * amountDown);
                        outRect.bottom = -(int) (floatingItemBounds.height() * amountDown);
                    }
                }
            }
        } else {
            outRect.top = 0;
            outRect.bottom = 0;
            //Make view visible incase invisible
            view.setVisibility(View.VISIBLE);
        }
    }

    /**
     * Find the new position by scanning through the items on
     * screen and finding the positional relationship.
     * This *seems* to work, another method would be to use
     * getItemOffsets, but I think that could miss items?..
     */
    private int getNewPostion(RecyclerView rv) {
        int itemsOnScreen = rv.getLayoutManager().getChildCount();

        float floatMiddleY = floatingItemBounds.top + floatingItemBounds.height() / 2;

        int above = 0;
        int below = Integer.MAX_VALUE;
        for (int n = 0; n < itemsOnScreen; n++) //Scan though items on screen, however they may not
        {                                    // be in order!

            View view = rv.getLayoutManager().getChildAt(n);

            if (view.getVisibility() != View.VISIBLE)
                continue;

            int itemPos = rv.getChildLayoutPosition(view);

            if (itemPos == selectedDragItemPos) //Don't check against itself!
                continue;

            float viewMiddleY = view.getTop() + view.getHeight() / 2;
            if (floatMiddleY > viewMiddleY) //Is above this item
            {
                if (itemPos > above)
                    above = itemPos;
            } else if (floatMiddleY <= viewMiddleY) //Is below this item
            {
                if (itemPos < below)
                    below = itemPos;
            }
        }
        debugLog("above = " + above + " below = " + below);

        if (below != Integer.MAX_VALUE) {
            if (below < selectedDragItemPos) //Need to count itself
                below++;
            return below - 1;
        } else {
            if (above < selectedDragItemPos)
                above++;

            return above;
        }
```

```java
    }


    @Override
    public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {
        debugLog("onInterceptTouchEvent");

        //if (e.getAction() == MotionEvent.ACTION_DOWN)
        {
            View itemView = rv.findChildViewUnder(e.getX(), e.getY());

            if (itemView == null)
                return false;

            boolean dragging = false;

            if ((dragHandleWidth > 0) && (e.getX() < dragHandleWidth)) {
                dragging = true;
            } else if (viewHandleId != -1) {
                //Find the handle in the list item
                View handleView = itemView.findViewById(viewHandleId);

                if (handleView == null) {
                    Log.e(TAG, "The view ID " + viewHandleId + " was not found in the RecycleView item");
                    return false;
                }

                //View should be visible to drag
                if (handleView.getVisibility() != View.VISIBLE) {
                    return false;
                }

                //We need to find the relative position of the handle to the parent view
                //Then we can work out if the touch is within the handle
                int[] parentItemPos = new int[2];
                itemView.getLocationInWindow(parentItemPos);

                int[] handlePos = new int[2];
                handleView.getLocationInWindow(handlePos);

                int xRel = handlePos[0] - parentItemPos[0];
                int yRel = handlePos[1] - parentItemPos[1];

                Rect touchBounds = new Rect(itemView.getLeft() + xRel, itemView.getTop() + yRel,
                        itemView.getLeft() + xRel + handleView.getWidth(),
                        itemView.getTop() + yRel + handleView.getHeight()
                );

                if (touchBounds.contains((int) e.getX(), (int) e.getY()))
                    dragging = true;

                debugLog("parentItemPos = " + parentItemPos[0] + " " + parentItemPos[1]);
                debugLog("handlePos = " + handlePos[0] + " " + handlePos[1]);
            }


            if (dragging) {
                debugLog("Started Drag");

                setIsDragging(true);

                floatingItem = createFloatingBitmap(itemView);

                fingerAnchorY = (int) e.getY();
                fingerOffsetInViewY = fingerAnchorY - itemView.getTop();
                fingerY = fingerAnchorY;

                selectedDragItemPos = rv.getChildLayoutPosition(itemView);
                debugLog("selectedDragItemPos = " + selectedDragItemPos);
                return true;
            }
        }
        return false;
    }

    @Override
    public void onRequestDisallowInterceptTouchEvent(boolean b) {

    }

    @Override
    public void onTouchEvent(RecyclerView rv, MotionEvent e) {
        debugLog("onTouchEvent");

        if ((e.getAction() == MotionEvent.ACTION_UP) ||
                (e.getAction() == MotionEvent.ACTION_CANCEL)) {
            if ((e.getAction() == MotionEvent.ACTION_UP) && selectedDragItemPos != -1) {
                int newPos = getNewPostion(rv);
                if (moveInterface != null)
                    moveInterface.onItemMoved(selectedDragItemPos, newPos);
            }

            setIsDragging(false);
            selectedDragItemPos = -1;
            floatingItem = null;
            rv.invalidateItemDecorations();
            return;
        }


        fingerY = (int) e.getY();

        if (floatingItem != null) {
            floatingItemBounds.top = fingerY - fingerOffsetInViewY;

            if (floatingItemBounds.top < -floatingItemStatingBounds.height() / 2) //Allow half the view out the top
                floatingItemBounds.top = -floatingItemStatingBounds.height() / 2;

            floatingItemBounds.bottom = floatingItemBounds.top + floatingItemStatingBounds.height();

            floatingItem.setBounds(floatingItemBounds);
        }

        //Do auto scrolling at end of list
        float scrollAmount = 0;
        if (fingerY > (rv.getHeight() * (1 - autoScrollWindow))) {
            scrollAmount = (fingerY - (rv.getHeight() * (1 - autoScrollWindow)));
        } else if (fingerY < (rv.getHeight() * autoScrollWindow)) {
            scrollAmount = (fingerY - (rv.getHeight() * autoScrollWindow));
        }
        debugLog("Scroll: " + scrollAmount);

        scrollAmount *= autoScrollSpeed;
        rv.scrollBy(0, (int) scrollAmount);

        rv.invalidateItemDecorations();// Redraw
    }

    private void setIsDragging(final boolean dragging) {
        if (dragging != isDragging) {
            isDragging = dragging;
            if (dragStateChangedListener != null) {
                if (isDragging) {
                    dragStateChangedListener.onDragStart();
                } else {
                    dragStateChangedListener.onDragStop();
                }
            }
        }
```

```java
    }

    public void setOnDragStateChangedListener(final OnDragStateChangedListener dragStateChangedListener) {
        this.dragStateChangedListener = dragStateChangedListener;
    }

    @Override
    public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {
        if (floatingItem != null) {
            floatingItem.setAlpha((int) (255 * floatingItemAlpha));
            bgColor.setColor(floatingItemBgColor);
            c.drawRect(floatingItemBounds, bgColor);
            floatingItem.draw(c);
        }
    }

    /**
     * @param position
     * @return True if we can drag the item over this position, False if not.
     */
    protected boolean canDragOver(int position) {
        return true;
    }

    private BitmapDrawable createFloatingBitmap(View v) {
        floatingItemStatingBounds = new Rect(v.getLeft(), v.getTop(), v.getRight(), v.getBottom());
        floatingItemBounds = new Rect(floatingItemStatingBounds);

        Bitmap bitmap = Bitmap.createBitmap(floatingItemStatingBounds.width(),
                floatingItemStatingBounds.height(), Bitmap.Config.ARGB_8888);
        Canvas canvas = new Canvas(bitmap);
        v.draw(canvas);

        BitmapDrawable retDrawable = new BitmapDrawable(v.getResources(), bitmap);
        retDrawable.setBounds(floatingItemBounds);

        return retDrawable;
    }

    public interface OnItemMovedListener {
        void onItemMoved(int from, int to);
    }


    public interface OnDragStateChangedListener {
        void onDragStart();

        void onDragStop();
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\FastScroller.java

```java
package com.naman14.timber.widgets;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.content.Context;
import android.support.annotation.NonNull;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.naman14.timber.R;

import static android.support.v7.widget.RecyclerView.OnScrollListener;

public class FastScroller extends LinearLayout {
    private static final int BUBBLE_ANIMATION_DURATION = 100;
    private static final int TRACK_SNAP_RANGE = 5;
    private final ScrollListener scrollListener = new ScrollListener();
    private TextView bubble;
    private View handle;
    private RecyclerView recyclerView;
    private int height;

    private ObjectAnimator currentAnimator = null;

    public FastScroller(final Context context, final AttributeSet attrs, final int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        initialise(context);
    }

    public FastScroller(final Context context) {
        super(context);
        initialise(context);
    }

    public FastScroller(final Context context, final AttributeSet attrs) {
        super(context, attrs);
        initialise(context);
    }

    private void initialise(Context context) {
        setOrientation(HORIZONTAL);
        setClipChildren(false);
        LayoutInflater inflater = LayoutInflater.from(context);
        inflater.inflate(R.layout.recyclerview_fastscroller, this, true);
        bubble = (TextView) findViewById(R.id.fastscroller_bubble);
        handle = findViewById(R.id.fastscroller_handle);
        bubble.setVisibility(INVISIBLE);
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        super.onSizeChanged(w, h, oldw, oldh);
        height = h;
    }

    @Override
    public boolean onTouchEvent(@NonNull MotionEvent event) {
        final int action = event.getAction();
        switch (action) {
            case MotionEvent.ACTION_DOWN:
                if (event.getX() < handle.getX())
                    return false;
                if (currentAnimator != null)
                    currentAnimator.cancel();
                if (bubble.getVisibility() == INVISIBLE)
                    showBubble();
                handle.setSelected(true);
            case MotionEvent.ACTION_MOVE:
                final float y = event.getY();
                setBubbleAndHandlePosition(y);
                setRecyclerViewPosition(y);
                return true;
            case MotionEvent.ACTION_UP:
            case MotionEvent.ACTION_CANCEL:
                handle.setSelected(false);
                hideBubble();
                return true;
        }
        return super.onTouchEvent(event);
    }

    public void setRecyclerView(RecyclerView recyclerView) {
        this.recyclerView = recyclerView;
        recyclerView.addOnScrollListener(scrollListener);
    }

    private void setRecyclerViewPosition(float y) {
        if (recyclerView != null) {
            int itemCount = recyclerView.getAdapter().getItemCount();
            float proportion;
            if (handle.getY() == 0)
                proportion = 0f;
            else if (handle.getY() + handle.getHeight() >= height - TRACK_SNAP_RANGE)
                proportion = 1f;
            else
                proportion = y / (float) height;
            int targetPos = getValueInRange(0, itemCount - 1, (int) (proportion * (float) itemCount));
            ((LinearLayoutManager) recyclerView.getLayoutManager()).scrollToPositionWithOffset(targetPos, 0);
//      recyclerView.oPositionWithOffset(targetPos);
            String bubbleText = ((BubbleTextGetter) recyclerView.getAdapter()).getTextToShowInBubble(targetPos);
            bubble.setText(bubbleText);
        }
    }

    private int getValueInRange(int min, int max, int value) {
        int minimum = Math.max(min, value);
        return Math.min(minimum, max);
    }

    private void setBubbleAndHandlePosition(float y) {
        int bubbleHeight = bubble.getHeight();
        int handleHeight = handle.getHeight();
        handle.setY(getValueInRange(0, height - handleHeight, (int) (y - handleHeight / 2)));
        bubble.setY(getValueInRange(0, height - bubbleHeight - handleHeight / 2, (int) (y - bubbleHeight)));
    }

    private void showBubble() {
        AnimatorSet animatorSet = new AnimatorSet();
        bubble.setVisibility(VISIBLE);
        if (currentAnimator != null)
            currentAnimator.cancel();
        currentAnimator = ObjectAnimator.ofFloat(bubble, "alpha", 0f, 1f).setDuration(BUBBLE_ANIMATION_DURATION);
        currentAnimator.start();
    }

    private void hideBubble() {
        if (currentAnimator != null)
            currentAnimator.cancel();
        currentAnimator = ObjectAnimator.ofFloat(bubble, "alpha", 1f, 0f).setDuration(BUBBLE_ANIMATION_DURATION);
        currentAnimator.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                bubble.setVisibility(INVISIBLE);
                currentAnimator = null;
            }

            @Override
            public void onAnimationCancel(Animator animation) {
                super.onAnimationCancel(animation);
                bubble.setVisibility(INVISIBLE);
                currentAnimator = null;
            }
        });
        currentAnimator.start();
    }

    private class ScrollListener extends OnScrollListener {
        @Override
        public void onScrolled(RecyclerView rv, int dx, int dy) {

            if (handle.isSelected()) {
                return;
            }
            View firstVisibleView = recyclerView.getChildAt(0);
            int firstVisiblePosition = recyclerView.getChildLayoutPosition(firstVisibleView);
            int visibleRange = recyclerView.getChildCount();
            int lastVisiblePosition = firstVisiblePosition + visibleRange;
            int itemCount = recyclerView.getAdapter().getItemCount();
            int position;
            if (firstVisiblePosition == 0)
                position = 0;
            else if (lastVisiblePosition == itemCount)
                position = itemCount;
            else
                position = (int) (((float) firstVisiblePosition / (((float) itemCount - (float) visibleRange))) * (float) it
            float proportion = (float) position / (float) itemCount;
            setBubbleAndHandlePosition(height * proportion);
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\MultiViewPager.java

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Point;
import android.support.v4.view.ViewPager;
import android.util.AttributeSet;
import android.view.View;

import com.naman14.timber.R;

public class MultiViewPager extends ViewPager {

    private final Point size;
    private final Point maxSize;
    /**
     * Maximum size.
     */
    private int mMaxWidth = -1;
    /**
     * Maximum size.
     */
    private int mMaxHeight = -1;
    /**
     * Child view inside a page to match the page size against.
     */
    private int mMatchWidthChildResId;
    /**
     * Internal state to schedule a new measurement pass.
     */
    private boolean mNeedsMeasurePage;

    public MultiViewPager(Context context) {
        super(context);
        size = new Point();
        maxSize = new Point();
    }

    public MultiViewPager(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context, attrs);
        size = new Point();
        maxSize = new Point();
    }

    private static void constrainTo(Point size, Point maxSize) {
        if (maxSize.x >= 0) {
            if (size.x > maxSize.x) {
                size.x = maxSize.x;
            }
        }
        if (maxSize.y >= 0) {
            if (size.y > maxSize.y) {
                size.y = maxSize.y;
            }
        }
    }

    private void init(Context context, AttributeSet attrs) {
        setClipChildren(false);
        TypedArray ta = context.obtainStyledAttributes(attrs, R.styleable.MultiViewPager);
        setMaxWidth(ta.getDimensionPixelSize(R.styleable.MultiViewPager_android_maxWidth, -1));
        setMaxHeight(ta.getDimensionPixelSize(R.styleable.MultiViewPager_android_maxHeight, -1));
        setMatchChildWidth(ta.getResourceId(R.styleable.MultiViewPager_matchChildWidth, 0));
        ta.recycle();
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        size.set(MeasureSpec.getSize(widthMeasureSpec),
                MeasureSpec.getSize(heightMeasureSpec));
        if (mMaxWidth >= 0 || mMaxHeight >= 0) {
            maxSize.set(mMaxWidth, mMaxHeight);
            constrainTo(size, maxSize);
            widthMeasureSpec = MeasureSpec.makeMeasureSpec(
                    size.x,
                    MeasureSpec.EXACTLY);
            heightMeasureSpec = MeasureSpec.makeMeasureSpec(
                    size.y,
                    MeasureSpec.EXACTLY);
        }
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
        onMeasurePage(widthMeasureSpec, heightMeasureSpec);
    }

    protected void onMeasurePage(int widthMeasureSpec, int heightMeasureSpec) {
        // Only measure if a measurement pass was scheduled
        if (!mNeedsMeasurePage) {
            return;
        }
        if (mMatchWidthChildResId == 0) {
            mNeedsMeasurePage = false;
        } else if (getChildCount() > 0) {
            View child = getChildAt(0);
            child.measure(widthMeasureSpec, heightMeasureSpec);
            int pageWidth = child.getMeasuredWidth();
            View match = child.findViewById(mMatchWidthChildResId);
            if (match == null) {
                throw new NullPointerException(
                        "MatchWithChildResId did not find that ID in the first fragment of the ViewPager; "
                                + "is that view defined in the child view's layout? Note that MultiViewPager "
                                + "only measures the child for index 0.");
            }
            int childWidth = match.getMeasuredWidth();
            // Check that the measurement was successful
            if (childWidth > 0) {
                mNeedsMeasurePage = false;
                int difference = pageWidth - childWidth;
                setPageMargin(-difference);
                int offscreen = (int) Math.ceil((float) pageWidth / (float) childWidth) + 1;
                setOffscreenPageLimit(offscreen);
                requestLayout();
            }
        }
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        super.onSizeChanged(w, h, oldw, oldh);
        // Schedule a new measurement pass as the dimensions have changed
        mNeedsMeasurePage = true;
    }

    /**
     * Sets the child view inside a page to match the page size against.
     *
     * @param matchChildWidthResId the child id
     */
    public void setMatchChildWidth(int matchChildWidthResId) {
        if (mMatchWidthChildResId != matchChildWidthResId) {
            mMatchWidthChildResId = matchChildWidthResId;
            mNeedsMeasurePage = true;
        }
    }

    /**
     * Sets the maximum size.
     *
     * @param width in pixels
     */
    public void setMaxWidth(int width) {
        mMaxWidth = width;
    }

    /**
     * Sets the maximum size.
     *
     * @param height in pixels
     */
    public void setMaxHeight(int height) {
        mMaxHeight = height;
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\MusicVisualizer.java

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.util.TypedValue;
import android.view.View;

import java.util.Random;

/**
 * a music visualizer sort of animation (with random data)
 */
public class MusicVisualizer extends View {

    Random random = new Random();

    Paint paint = new Paint();
    private Runnable animateView = new Runnable() {
        @Override
        public void run() {

            //run every 100 ms
            postDelayed(this, 120);

            invalidate();
        }
    };

    public MusicVisualizer(Context context) {
        super(context);
        new MusicVisualizer(context, null);
    }

    public MusicVisualizer(Context context, AttributeSet attrs) {
        super(context, attrs);

        //start runnable
        removeCallbacks(animateView);
        post(animateView);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);

        //set paint style, Style.FILL will fill the color, Style.STROKE will stroke the color
        paint.setStyle(Paint.Style.FILL);

        canvas.drawRect(getDimensionInPixel(0), getHeight() - (40 + random.nextInt((int) (getHeight() / 1.5f) - 25)), getDim
        canvas.drawRect(getDimensionInPixel(10), getHeight() - (40 + random.nextInt((int) (getHeight() / 1.5f) - 25)), getDi
        canvas.drawRect(getDimensionInPixel(20), getHeight() - (40 + random.nextInt((int) (getHeight() / 1.5f) - 25)), getDi
    }

    public void setColor(int color) {
        paint.setColor(color);
        invalidate();
    }

    //get all dimensions in dp so that views behaves properly on different screen resolutions
    private int getDimensionInPixel(int dp) {
        return (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, dp, getResources().getDisplayMetrics());
    }

    @Override
    protected void onWindowVisibilityChanged(int visibility) {
        super.onWindowVisibilityChanged(visibility);
        if (visibility == VISIBLE) {
            removeCallbacks(animateView);
            post(animateView);
        } else if (visibility == GONE) {
            removeCallbacks(animateView);
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\PlayPauseButton.java

```java
package com.naman14.timber.widgets;

import android.animation.ValueAnimator;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.drawable.Drawable;
import android.os.Parcel;
import android.os.Parcelable;
import android.support.annotation.NonNull;
import android.util.AttributeSet;
import android.view.View;

public class PlayPauseButton extends View {

    /**
     * √3
     */
    private final static double SQRT_3 = Math.sqrt(3);

    /**
     * Animation⊠⊠⊠⊠⊠⊠
     */
    private final static int SPEED = 1;
    /**
     * ⊠⊠⊠⊠⊠⊠
     * ⊠⊠⊠⊠(0,0)
     * ⊠⊠⊠(1,1) ⊠⊠(1,⊠1) ⊠⊠(⊠1,1) ⊠⊠(⊠1,⊠1)
     * ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠Class
     */
    private final Point mPoint;
    /**
     * ⊠⊠⊠⊠Paint
     */
    private Paint mPaint;
    /**
     * ⊠⊠⊠⊠Path
     */
    private Path mLeftPath;
    /**
     * ⊠⊠⊠⊠Path
     */
    private Path mRightPath;
    /**
     * ⊠⊠⊠⊠Animator
     */
    private ValueAnimator mCenterEdgeAnimator;
    /**
     * ⊠⊠⊠⊠⊠Animator
     */
    private ValueAnimator mLeftEdgeAnimator;
    /**
     * ⊠⊠⊠⊠⊠Animator
     */
    private ValueAnimator mRightEdgeAnimator;
    /**
     * ⊠⊠⊠⊠⊠⊠
     */
    private boolean mPlayed;

    /**
     * ⊠⊠⊠⊠⊠
     */
    private int mBackgroundColor = Color.BLACK;

    /**
     * Animator⊠UpdateListener
     */
    private ValueAnimator.AnimatorUpdateListener mAnimatorUpdateListener =
            new ValueAnimator.AnimatorUpdateListener() {
                @Override
                public void onAnimationUpdate(ValueAnimator valueAnimator) {
                    invalidate();
                }
            };

    private OnControlStatusChangeListener mListener;

    /**
     * ⊠⊠⊠⊠⊠⊠⊠
     * {@inheritDoc}
     */
    public PlayPauseButton(Context context) {
        this(context, null, 0);
    }

    /**
     * ⊠⊠⊠⊠⊠⊠⊠
     * {@inheritDoc}
     */
    public PlayPauseButton(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    /**
     * ⊠⊠⊠⊠⊠⊠⊠
     * android:background⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     * {@inheritDoc}
     */
    public PlayPauseButton(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);

        mPoint = new Point();
        initView();
    }

    /**
     * View⊠⊠⊠⊠
     */
    private void initView() {
        setUpPaint();
        setUpPath();
        setUpAnimator();
    }

    /**
     * Animator⊠⊠⊠⊠
     * ⊠⊠⊠⊠⊠Animator⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠start⊠⊠⊠⊠⊠
     */
    private void setUpAnimator() {
        if (mPlayed) {
            mCenterEdgeAnimator = ValueAnimator.ofFloat(1.f, 1.f);
            mLeftEdgeAnimator = ValueAnimator.ofFloat((float) (-0.2f * SQRT_3), (float) (-0.2f * SQRT_3));
            mRightEdgeAnimator = ValueAnimator.ofFloat(1.f, 1.f);
        } else {
            mCenterEdgeAnimator = ValueAnimator.ofFloat(0.5f, 0.5f);
            mLeftEdgeAnimator = ValueAnimator.ofFloat(0.f, 0.f);
            mRightEdgeAnimator = ValueAnimator.ofFloat(0.f, 0.f);
        }

        mCenterEdgeAnimator.start();
        mLeftEdgeAnimator.start();
        mRightEdgeAnimator.start();
    }

    /**
     * Paint⊠⊠⊠⊠
     */
    private void setUpPaint() {
        mPaint = new Paint();
        mPaint.setColor(mBackgroundColor);
        mPaint.setAntiAlias(true);
        mPaint.setStyle(Paint.Style.FILL);
    }

    /**
     * Path⊠⊠⊠⊠
     */
    private void setUpPath() {
        mLeftPath = new Path();
        mRightPath = new Path();
    }

    /**
     * ⊠⊠⊠⊠
     * {@inheritDoc}
     */
    @Override
    protected void onDraw(Canvas canvas) {
        mPoint.setHeight(canvas.getHeight());
        mPoint.setWidth(canvas.getWidth());

        mLeftPath.reset();
        mRightPath.reset();

        //⊠⊠⊠Path⊠⊠⊠
        mLeftPath.moveTo(mPoint.getX(-0.5 * SQRT_3), mPoint.getY(1.f));
        mLeftPath.lineTo(mPoint.getY((Float) mLeftEdgeAnimator.getAnimatedValue()) + 0.7f,
                mPoint.getY((Float) mCenterEdgeAnimator.getAnimatedValue()));
        mLeftPath.lineTo(mPoint.getY((Float) mLeftEdgeAnimator.getAnimatedValue()) + 0.7f,
                mPoint.getY(-1 * (Float) mCenterEdgeAnimator.getAnimatedValue()));
        mLeftPath.lineTo(mPoint.getX(-0.5 * SQRT_3), mPoint.getY(-1.f));

        //⊠⊠⊠Path⊠⊠⊠
        mRightPath.moveTo(mPoint.getY(-1 * (Float) mLeftEdgeAnimator.getAnimatedValue()),
                mPoint.getY((Float) mCenterEdgeAnimator.getAnimatedValue()));
        mRightPath.lineTo(mPoint.getX(0.5 * SQRT_3),
                mPoint.getY((Float) mRightEdgeAnimator.getAnimatedValue()));
        mRightPath.lineTo(mPoint.getX(0.5 * SQRT_3),
                mPoint.getY(-1 * (Float) mRightEdgeAnimator.getAnimatedValue()));
        mRightPath.lineTo(mPoint.getY(-1 * (Float) mLeftEdgeAnimator.getAnimatedValue()),
                mPoint.getY(-1 * (Float) mCenterEdgeAnimator.getAnimatedValue()));

        canvas.drawPath(mLeftPath, mPaint);
        canvas.drawPath(mRightPath, mPaint);
    }

    /**
     * ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     */
    public void startAnimation() {
        mCenterEdgeAnimator = ValueAnimator.ofFloat(1.f, 0.5f);
        mCenterEdgeAnimator.setDuration(100 * SPEED);
        mCenterEdgeAnimator.addUpdateListener(mAnimatorUpdateListener);

        mLeftEdgeAnimator = ValueAnimator.ofFloat((float) (-0.2 * SQRT_3), 0.f);
        mLeftEdgeAnimator.setDuration(100 * SPEED);
        mLeftEdgeAnimator.addUpdateListener(mAnimatorUpdateListener);

        mRightEdgeAnimator = ValueAnimator.ofFloat(1.f, 0.f);
        mRightEdgeAnimator.setDuration(150 * SPEED);
        mRightEdgeAnimator.addUpdateListener(mAnimatorUpdateListener);

        if (!mPlayed) {
            mCenterEdgeAnimator.start();
            mLeftEdgeAnimator.start();
            mRightEdgeAnimator.start();
        } else {
            mCenterEdgeAnimator.reverse();
            mLeftEdgeAnimator.reverse();
            mRightEdgeAnimator.reverse();
        }
    }

    public void setOnControlStatusChangeListener(OnControlStatusChangeListener listener) {
        mListener = listener;
    }

//  /**
//   * ACTION_DOWN⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
//   * {@inheritDoc}
//   */
//  @Override public boolean onTouchEvent(@NonNull MotionEvent event) {
//    switch (event.getAction()) {
//      case MotionEvent.ACTION_DOWN:
//        setPlayed(!mPlayed);
//        startAnimation();
//        if (mListener != null) {
//          mListener.onStatusChange(this, mPlayed);
//        }
//        break;
//    }
//    return false;
//  }

    /**
     * ⊠⊠⊠⊠⊠ {@link PlayPauseButton.SavedState} ⊠⊠⊠⊠⊠
     * {@inheritDoc}
     */
    @Override
    public Parcelable onSaveInstanceState() {
        Parcelable superState = super.onSaveInstanceState();
        SavedState savedState = new SavedState(superState);
        savedState.played = isPlayed();
        return savedState;
    }

    /**
     * Restore⊠⊠⊠⊠⊠⊠⊠⊠ {@link PlayPauseButton.SavedState}
     * ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     * {@inheritDoc}
     */
    @Override
    public void onRestoreInstanceState(Parcelable state) {
        SavedState savedState = (SavedState) state;
        super.onRestoreInstanceState(savedState.getSuperState());
        setPlayed(savedState.played);
        setUpAnimator();
        invalidate();
    }

    /**
     * android:background ⊠⊠⊠ {@link View} ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     * {@inheritDoc}
     */
    @Override
    public void setBackground(Drawable background) {
    }

    /**
     * {@link PlayPauseButton#mPlayed} ⊠⊠⊠
     *
     * @return {@link PlayPauseButton#mPlayed}
     */
    public boolean isPlayed() {
        return mPlayed;
    }
    /**
     * {@link PlayPauseButton#mPlayed} ⊠⊠⊠⊠⊠⊠
     *
     * @param played ⊠⊠
     */
    public void setPlayed(boolean played) {
        if (mPlayed != played) {
            mPlayed = played;
            invalidate();
        }
    }

    /**
     * {@link PlayPauseButton#mBackgroundColor} ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     *
     * @param color ⊠⊠⊠⊠⊠⊠
     */
    public void setColor(int color) {
        mBackgroundColor = color;
        mPaint.setColor(mBackgroundColor);
        invalidate();
    }

    public interface OnControlStatusChangeListener {
        void onStatusChange(View view, boolean state);
    }

    /**
     * ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
     */
    static class SavedState extends BaseSavedState {

        public static final Parcelable.Creator<SavedState> CREATOR =
                new Parcelable.Creator<SavedState>() {
                    public SavedState createFromParcel(Parcel in) {
                        return new SavedState(in);
                    }

                    public SavedState[] newArray(int size) {
                        return new SavedState[size];
                    }
                };
        boolean played;

        SavedState(Parcelable superState) {
            super(superState);
        }

        private SavedState(Parcel in) {
            super(in);
            played = (Boolean) in.readValue(null);
        }

        @Override
        public void writeToParcel(@NonNull Parcel out, int flags) {
            super.writeToParcel(out, flags);
            out.writeValue(played);
        }
    }

    /**
     * ⊠⊠⊠⊠⊠⊠
     * ⊠⊠⊠⊠(0,0)
     * ⊠⊠⊠(1,1) ⊠⊠(1,⊠1) ⊠⊠(⊠1,1) ⊠⊠(⊠1,⊠1)
     * ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠Class
     */
    static class Point {

        private int width;
        private int height;

        public void setWidth(int width) {
            this.width = width;
        }

        public void setHeight(int height) {
            this.height = height;
        }

        public float getX(float x) {
            return (width / 2) * (x + 1);
        }

        public float getY(float y) {
            return (height / 2) * (y + 1);
        }

        public float getX(double x) {
            return getX((float) x);
        }

        public float getY(double y) {
            return getY((float) y);
        }
    }
}
```

```java
/**
 * This code was modified by me, Paul Woitaschek. All these changes are licensed under GPLv3. The
 * original source can be found here: {@link https://github.com/alexjlockwood/material-pause-play-
 * animation/blob/master/app/src/main/java/com/alexjlockwood/example/playpauseanimation/
 * PlayPauseView.java}
 * <p/>
 * The original licensing is as follows:
 * <p/>
 * <p/>
 * The MIT License (MIT)
 * <p/>
 * Copyright (c) 2015 Alex Lockwood
 * <p/>
 * Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
 * associated documentation files (the "Software"), to deal in the Software without restriction,
 * including without limitation the rights to use, copy, modify, merge, publish, distribute,
 * sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 * <p/>
 * The above copyright notice and this permission notice shall be included in all copies or
 * substantial portions of the Software.
 * <p/>
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING
 * BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
 * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
 * DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package com.naman14.timber.widgets;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.animation.ObjectAnimator;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PixelFormat;
import android.graphics.drawable.Drawable;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Property;
import android.view.animation.DecelerateInterpolator;


public class PlayPauseDrawable extends Drawable {


    private static final String TAG = PlayPauseDrawable.class.getSimpleName();
    private final Path leftPauseBar = new Path();
    private final Path rightPauseBar = new Path();
    private final Paint paint = new Paint();
    private float progress;
    private static final Property<PlayPauseDrawable, Float> PROGRESS =
            new Property<PlayPauseDrawable, Float>(Float.class, "progress") {
                @Override
                public Float get(PlayPauseDrawable d) {
                    return d.getProgress();
                }

                @Override
                public void set(PlayPauseDrawable d, Float value) {
                    d.setProgress(value);
                }
            };
    private boolean isPlay;
    @Nullable
    private Animator animator;
    public PlayPauseDrawable() {
        paint.setAntiAlias(true);
        paint.setStyle(Paint.Style.FILL);
        paint.setColor(Color.WHITE);
    }

    /**
     * Linear interpolate between a and b with parameter t.
     */
    private static float interpolate(float a, float b, float t) {
        return a + (b - a) * t;
    }

    @Override
    public void draw(Canvas canvas) {
        long startDraw = System.currentTimeMillis();

        leftPauseBar.rewind();
        rightPauseBar.rewind();

        // move to center of canvas
        canvas.translate(getBounds().left, getBounds().top);

        float pauseBarHeight = 7.0F / 12.0F * ((float) getBounds().height());
        float pauseBarWidth = pauseBarHeight / 3.0F;
        float pauseBarDistance = pauseBarHeight / 3.6F;

        // The current distance between the two pause bars.
        final float barDist = interpolate(pauseBarDistance, 0.0F, progress);
        // The current width of each pause bar.
        final float barWidth = interpolate(pauseBarWidth, pauseBarHeight / 1.75F, progress);
        // The current position of the left pause bar's top left coordinate.
        final float firstBarTopLeft = interpolate(0.0F, barWidth, progress);
        // The current position of the right pause bar's top right coordinate.
        final float secondBarTopRight = interpolate(2.0F * barWidth + barDist, barWidth + barDist, progress);

        // Draw the left pause bar. The left pause bar transforms into the
        // top half of the play button triangle by animating the position of the
        // rectangle's top left coordinate and expanding its bottom width.
        leftPauseBar.moveTo(0.0F, 0.0F);
        leftPauseBar.lineTo(firstBarTopLeft, -pauseBarHeight);
        leftPauseBar.lineTo(barWidth, -pauseBarHeight);
        leftPauseBar.lineTo(barWidth, 0.0F);
        leftPauseBar.close();

        // Draw the right pause bar. The right pause bar transforms into the
        // bottom half of the play button triangle by animating the position of the
        // rectangle's top right coordinate and expanding its bottom width.
        rightPauseBar.moveTo(barWidth + barDist, 0.0F);
        rightPauseBar.lineTo(barWidth + barDist, -pauseBarHeight);
        rightPauseBar.lineTo(secondBarTopRight, -pauseBarHeight);
        rightPauseBar.lineTo(2.0F * barWidth + barDist, 0.0F);
        rightPauseBar.close();

        canvas.save();

        // Translate the play button a tiny bit to the right so it looks more centered.
        canvas.translate(interpolate(0.0F, pauseBarHeight / 8.0F, progress), 0.0F);

        // (1) Pause --> Play: rotate 0 to 90 degrees clockwise.
        // (2) Play --> Pause: rotate 90 to 180 degrees clockwise.
        final float rotationProgress = isPlay ? 1.0F - progress : progress;
        final float startingRotation = isPlay ? 90.0F : 0.0F;
        canvas.rotate(interpolate(startingRotation, startingRotation + 90.0F, rotationProgress), getBounds().width() / 2.0F,

        // Position the pause/play button in the center of the drawable's bounds.
        canvas.translate(getBounds().width() / 2.0F - ((2.0F * barWidth + barDist) / 2.0F), getBounds().height() / 2.0F + (p

        // Draw the two bars that form the animated pause/play button.
        canvas.drawPath(leftPauseBar, paint);
        canvas.drawPath(rightPauseBar, paint);

        canvas.restore();

        long timeElapsed = System.currentTimeMillis() - startDraw;
        if (timeElapsed > 16) {
            Log.e(TAG, "Drawing took too long=" + timeElapsed);
        }
    }

    public void transformToPause(boolean animated) {
        if (isPlay) {
            if (animated) {
                toggle();
            } else {
                isPlay = false;
                setProgress(0.0F);
            }
        }
    }

    @Override
    public void jumpToCurrentState() {
        Log.v(TAG, "jumpToCurrentState()");
        if (animator != null) {
            animator.cancel();
        }
        setProgress(isPlay ? 1.0F : 0.0F);
    }

    public void transformToPlay(boolean animated) {
        if (!isPlay) {
            if (animated) {
                toggle();
            } else {
                isPlay = true;
                setProgress(1.0F);
            }
        }
    }

    private void toggle() {
        if (animator != null) {
            animator.cancel();
        }

        animator = ObjectAnimator.ofFloat(this, PROGRESS, isPlay ? 1.0F : 0.0F, isPlay ? 0.0F : 1.0F);
        animator.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                isPlay = !isPlay;
            }
        });

        animator.setInterpolator(new DecelerateInterpolator());
        animator.setDuration(200);
        animator.start();
    }

    private float getProgress() {
        return progress;
    }

    private void setProgress(float progress) {
        this.progress = progress;
        invalidateSelf();
    }

    @Override
    public void setAlpha(int alpha) {
        paint.setAlpha(alpha);
        invalidateSelf();
    }

    @Override
    public void setColorFilter(ColorFilter cf) {
        paint.setColorFilter(cf);
        invalidateSelf();
    }


    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\PopupImageView.java

```java
package com.naman14.timber.widgets;

import android.annotation.TargetApi;
import android.content.Context;
import android.graphics.Color;
import android.preference.PreferenceManager;
import android.util.AttributeSet;
import android.widget.ImageView;

import com.afollestad.appthemeengine.util.TintHelper;

/**
 * Created by naman on 29/10/16.
 */
public class PopupImageView extends ImageView {

    public PopupImageView(Context context) {
        super(context);
        tint();
    }

    public PopupImageView(Context context, AttributeSet attrs) {
        super(context, attrs);
        tint();
    }

    public PopupImageView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        tint();
    }

    @TargetApi(21)
    public PopupImageView(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {
        super(context, attrs, defStyleAttr, defStyleRes);
        tint();
    }

    private void tint() {
        if (PreferenceManager.getDefaultSharedPreferences(getContext()).getBoolean("dark_theme", false)) {
            TintHelper.setTint(this, Color.parseColor("#eeeeee"));
        } else  TintHelper.setTint(this, Color.parseColor("#434343"));
    }

}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\SquareImageView.java

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.util.AttributeSet;
import android.widget.ImageView;

public class SquareImageView extends ImageView {


    public SquareImageView(Context context) {
        super(context);
    }

    public SquareImageView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public SquareImageView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec);
        setMeasuredDimension(getMeasuredWidth(), getMeasuredWidth());
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\TextDrawable.java

```java
package com.naman14.timber.widgets;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ColorFilter;
import android.graphics.Paint;
import android.graphics.PixelFormat;
import android.graphics.Rect;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.OvalShape;
import android.graphics.drawable.shapes.RectShape;
import android.graphics.drawable.shapes.RoundRectShape;


public class TextDrawable extends ShapeDrawable {

    private static final float SHADE_FACTOR = 0.9f;
    private final Paint textPaint;
    private final Paint borderPaint;
    private final String text;
    private final int color;
    private final RectShape shape;
    private final int height;
    private final int width;
    private final int fontSize;
    private final float radius;
    private final int borderThickness;

    private TextDrawable(Builder builder) {
        super(builder.shape);

        // shape properties
        shape = builder.shape;
        height = builder.height;
        width = builder.width;
        radius = builder.radius;

        // text and color
        text = builder.toUpperCase ? builder.text.toUpperCase() : builder.text;
        color = builder.color;

        // text paint settings
        fontSize = builder.fontSize;
        textPaint = new Paint();
        textPaint.setColor(builder.textColor);
        textPaint.setAntiAlias(true);
        textPaint.setFakeBoldText(builder.isBold);
        textPaint.setStyle(Paint.Style.FILL);
        textPaint.setTypeface(builder.font);
        textPaint.setTextAlign(Paint.Align.CENTER);
        textPaint.setStrokeWidth(builder.borderThickness);

        // border paint settings
        borderThickness = builder.borderThickness;
        borderPaint = new Paint();
        borderPaint.setColor(getDarkerShade(color));
        borderPaint.setStyle(Paint.Style.STROKE);
        borderPaint.setStrokeWidth(borderThickness);

        // drawable paint color
        Paint paint = getPaint();
        paint.setColor(color);

    }

    public static IShapeBuilder builder() {
        return new Builder();
    }
    private int getDarkerShade(int color) {
        return Color.rgb((int) (SHADE_FACTOR * Color.red(color)),
                (int) (SHADE_FACTOR * Color.green(color)),
                (int) (SHADE_FACTOR * Color.blue(color)));
    }

    @Override
    public void draw(Canvas canvas) {
        super.draw(canvas);
        Rect r = getBounds();


        // draw border
        if (borderThickness > 0) {
            drawBorder(canvas);
        }

        int count = canvas.save();
        canvas.translate(r.left, r.top);

        // draw text
        int width = this.width < 0 ? r.width() : this.width;
        int height = this.height < 0 ? r.height() : this.height;
        int fontSize = this.fontSize < 0 ? (Math.min(width, height) / 2) : this.fontSize;
        textPaint.setTextSize(fontSize);
        canvas.drawText(text, width / 2, height / 2 - ((textPaint.descent() + textPaint.ascent()) / 2), textPaint);

        canvas.restoreToCount(count);

    }

    private void drawBorder(Canvas canvas) {
        RectF rect = new RectF(getBounds());
        rect.inset(borderThickness / 2, borderThickness / 2);

        if (shape instanceof OvalShape) {
            canvas.drawOval(rect, borderPaint);
        } else if (shape instanceof RoundRectShape) {
            canvas.drawRoundRect(rect, radius, radius, borderPaint);
        } else {
            canvas.drawRect(rect, borderPaint);
        }
    }

    @Override
    public void setAlpha(int alpha) {
        textPaint.setAlpha(alpha);
    }

    @Override
    public void setColorFilter(ColorFilter cf) {
        textPaint.setColorFilter(cf);
    }

    @Override
    public int getOpacity() {
        return PixelFormat.TRANSLUCENT;
    }

    @Override
    public int getIntrinsicWidth() {
        return width;
    }

    @Override
    public int getIntrinsicHeight() {
        return height;
    }
    public interface IConfigBuilder {
        IConfigBuilder width(int width);

        IConfigBuilder height(int height);

        IConfigBuilder textColor(int color);

        IConfigBuilder withBorder(int thickness);

        IConfigBuilder useFont(Typeface font);

        IConfigBuilder fontSize(int size);

        IConfigBuilder bold();

        IConfigBuilder toUpperCase();

        IShapeBuilder endConfig();
    }

    public interface IBuilder {

        TextDrawable build(String text, int color);
    }

    public interface IShapeBuilder {

        IConfigBuilder beginConfig();

        IBuilder rect();

        IBuilder round();

        IBuilder roundRect(int radius);

        TextDrawable buildRect(String text, int color);

        TextDrawable buildRoundRect(String text, int color, int radius);

        TextDrawable buildRound(String text, int color);
    }

    public static class Builder implements IConfigBuilder, IShapeBuilder, IBuilder {

        public int textColor;
        public float radius;
        private String text;
        private int color;
        private int borderThickness;
        private int width;
        private int height;
        private Typeface font;
        private RectShape shape;
        private int fontSize;
        private boolean isBold;
        private boolean toUpperCase;

        private Builder() {
            text = "";
            color = Color.GRAY;
            textColor = Color.WHITE;
            borderThickness = 0;
            width = -1;
            height = -1;
            shape = new RectShape();
            font = Typeface.create("sans-serif-light", Typeface.NORMAL);
            fontSize = -1;
            isBold = false;
            toUpperCase = false;
        }
```

```java
        public IConfigBuilder width(int width) {
            this.width = width;
            return this;
        }

        public IConfigBuilder height(int height) {
            this.height = height;
            return this;
        }

        public IConfigBuilder textColor(int color) {
            this.textColor = color;
            return this;
        }

        public IConfigBuilder withBorder(int thickness) {
            this.borderThickness = thickness;
            return this;
        }

        public IConfigBuilder useFont(Typeface font) {
            this.font = font;
            return this;
        }

        public IConfigBuilder fontSize(int size) {
            this.fontSize = size;
            return this;
        }

        public IConfigBuilder bold() {
            this.isBold = true;
            return this;
        }

        public IConfigBuilder toUpperCase() {
            this.toUpperCase = true;
            return this;
        }

        @Override
        public IConfigBuilder beginConfig() {
            return this;
        }

        @Override
        public IShapeBuilder endConfig() {
            return this;
        }

        @Override
        public IBuilder rect() {
            this.shape = new RectShape();
            return this;
        }

        @Override
        public IBuilder round() {
            this.shape = new OvalShape();
            return this;
        }

        @Override
        public IBuilder roundRect(int radius) {
            this.radius = radius;
            float[] radii = {radius, radius, radius, radius, radius, radius, radius, radius};
            this.shape = new RoundRectShape(radii, null, null);
            return this;
        }
        @Override
        public TextDrawable buildRect(String text, int color) {
            rect();
            return build(text, color);
        }

        @Override
        public TextDrawable buildRoundRect(String text, int color, int radius) {
            roundRect(radius);
            return build(text, color);
        }

        @Override
        public TextDrawable buildRound(String text, int color) {
            round();
            return build(text, color);
        }

        @Override
        public TextDrawable build(String text, int color) {
            this.color = color;
            this.text = text;
            return new TextDrawable(this);
        }
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\ThemedPreferenceCategor

```java
package com.naman14.timber.widgets;

import android.content.Context;
import android.preference.PreferenceCategory;
import android.util.AttributeSet;
import android.view.View;
import android.widget.TextView;

import com.afollestad.appthemeengine.Config;
import com.naman14.timber.utils.Helpers;

/**
 * Created by naman on 31/12/15.
 */
public class ThemedPreferenceCategory extends PreferenceCategory {

    private Context context;

    public ThemedPreferenceCategory(Context context) {
        super(context);
        this.context = context;
    }

    public ThemedPreferenceCategory(Context context, AttributeSet attrs) {
        super(context, attrs);
        this.context = context;
    }

    public ThemedPreferenceCategory(Context context, AttributeSet attrs,
                                    int defStyle) {
        super(context, attrs, defStyle);
        this.context = context;
    }

    @Override
    protected void onBindView(View view) {
        super.onBindView(view);
        TextView titleView = (TextView) view.findViewById(android.R.id.title);
        titleView.setTextColor(Config.accentColor(context, Helpers.getATEKey(context)));
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\desktop\BaseWidget.java

```java
package com.naman14.timber.widgets.desktop;

import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.LayoutRes;
import android.widget.RemoteViews;

import com.naman14.timber.MusicService;

/**
 * Created by nv95 on 02.11.16.
 */

public abstract class BaseWidget extends AppWidgetProvider {

    protected static final int REQUEST_NEXT = 1;
    protected static final int REQUEST_PREV = 2;
    protected static final int REQUEST_PLAYPAUSE = 3;

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        onUpdate(context, appWidgetManager, appWidgetIds, null);
    }

    private void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds,Bundle extras){
        ComponentName serviceName = new ComponentName(context, MusicService.class);
        RemoteViews remoteViews = new RemoteViews(context.getPackageName(), getLayoutRes());
        try {
            onViewsUpdate(context, remoteViews, serviceName, extras);
            appWidgetManager.updateAppWidget(appWidgetIds, remoteViews);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (action != null && action.startsWith("com.naman14.timber.")) {
            AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
            ComponentName thisAppWidget = new ComponentName(context.getPackageName(), this.getClass().getName());
            int[] appWidgetIds = appWidgetManager.getAppWidgetIds(thisAppWidget);
            onUpdate(context, appWidgetManager, appWidgetIds, intent.getExtras());
        } else {
            super.onReceive(context, intent);
        }
    }

    abstract void onViewsUpdate(Context context, RemoteViews remoteViews, ComponentName serviceName, Bundle extras);

    abstract @LayoutRes int getLayoutRes();
}
```

```java
package com.naman14.timber.widgets.desktop;

import android.app.PendingIntent;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.widget.RemoteViews;

import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.ImageLoader;

/**
 * Created by nv95 on 02.11.16.
 */

public class SmallWidget extends BaseWidget {

    @Override
    int getLayoutRes() {
        return R.layout.widget_small;
    }

    @Override
    void onViewsUpdate(Context context, RemoteViews remoteViews, ComponentName serviceName, Bundle extras) {
        remoteViews.setOnClickPendingIntent(R.id.image_next, PendingIntent.getService(
                context,
                REQUEST_NEXT,
                new Intent(context, MusicService.class)
                        .setAction(MusicService.NEXT_ACTION)
                        .setComponent(serviceName),
                0
        ));
        remoteViews.setOnClickPendingIntent(R.id.image_playpause, PendingIntent.getService(
                context,
                REQUEST_PLAYPAUSE,
                new Intent(context, MusicService.class)
                        .setAction(MusicService.TOGGLEPAUSE_ACTION)
                        .setComponent(serviceName),
                0
        ));
        if (extras != null) {
            String t = extras.getString("track");
            if (t != null) {
                remoteViews.setTextViewText(R.id.textView_title, t);
            }
            t = extras.getString("artist");
            if (t != null) {
                remoteViews.setTextViewText(R.id.textView_subtitle, t);
            }
            remoteViews.setImageViewResource(R.id.image_playpause,
                    extras.getBoolean("playing") ? R.drawable.ic_pause_white_36dp : R.drawable.ic_play_white_36dp);
            long albumId = extras.getLong("albumid");
            if (albumId != -1) {
                Bitmap artwork = ImageLoader.getInstance().loadImageSync(TimberUtils.getAlbumArtUri(albumId).toString());
                if (artwork != null) {
                    remoteViews.setImageViewBitmap(R.id.imageView_cover, artwork);
                } else {
                    remoteViews.setImageViewResource(R.id.imageView_cover, R.drawable.ic_empty_music2);
                }
            }
        }
        remoteViews.setOnClickPendingIntent(R.id.textView_title, PendingIntent.getActivity(
                context,
                0,
                NavigationUtils.getNowPlayingIntent(context),
                PendingIntent.FLAG_UPDATE_CURRENT
        ));
    }
}
```

```java
package com.naman14.timber.widgets.desktop;

import android.app.PendingIntent;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.text.TextUtils;
import android.widget.RemoteViews;

import com.naman14.timber.MusicService;
import com.naman14.timber.R;
import com.naman14.timber.utils.NavigationUtils;
import com.naman14.timber.utils.TimberUtils;
import com.nostra13.universalimageloader.core.ImageLoader;

/**
 * Created by nv95 on 08.07.16.
 */

public class StandardWidget extends BaseWidget {

    @Override
    int getLayoutRes() {
        return R.layout.widget_standard;
    }

    @Override
    void onViewsUpdate(Context context, RemoteViews remoteViews, ComponentName serviceName, Bundle extras) {
        remoteViews.setOnClickPendingIntent(R.id.image_next, PendingIntent.getService(
                context,
                REQUEST_NEXT,
                new Intent(context, MusicService.class)
                        .setAction(MusicService.NEXT_ACTION)
                        .setComponent(serviceName),
                0
        ));
        remoteViews.setOnClickPendingIntent(R.id.image_prev, PendingIntent.getService(
                context,
                REQUEST_PREV,
                new Intent(context, MusicService.class)
                        .setAction(MusicService.PREVIOUS_ACTION)
                        .setComponent(serviceName),
                0
        ));
        remoteViews.setOnClickPendingIntent(R.id.image_playpause, PendingIntent.getService(
                context,
                REQUEST_PLAYPAUSE,
                new Intent(context, MusicService.class)
                        .setAction(MusicService.TOGGLEPAUSE_ACTION)
                        .setComponent(serviceName),
                0
        ));

        if (extras != null) {
            String t = extras.getString("track");
            if (t != null) {
                remoteViews.setTextViewText(R.id.textView_title, t);
            }
            t = extras.getString("artist");
            ;
            if (t != null) {
                String album = extras.getString("album");
                ;
                if (!TextUtils.isEmpty(album)) {
                    t += " - " + album;
                }
                remoteViews.setTextViewText(R.id.textView_subtitle, t);
            }
            remoteViews.setImageViewResource(R.id.image_playpause,
                    extras.getBoolean("playing") ? R.drawable.ic_pause_white_36dp : R.drawable.ic_play_white_36dp);
            long albumId = extras.getLong("albumid");
            if (albumId != -1) {
                Bitmap artwork = ImageLoader.getInstance().loadImageSync(TimberUtils.getAlbumArtUri(albumId).toString());
                if (artwork != null) {
                    remoteViews.setImageViewBitmap(R.id.imageView_cover, artwork);
                } else {
                    remoteViews.setImageViewResource(R.id.imageView_cover, R.drawable.ic_empty_music2);
                }
            }
        }
        remoteViews.setOnClickPendingIntent(R.id.imageView_cover, PendingIntent.getActivity(
                context,
                0,
                NavigationUtils.getNowPlayingIntent(context),
                PendingIntent.FLAG_UPDATE_CURRENT
        ));
    }
}
```

D:\dwonloads\project\open source projects\Timber-master\app\src\main\java\com\naman14\timber\widgets\desktop\WhiteWidget.jav

```java
package com.naman14.timber.widgets.desktop;

import com.naman14.timber.R;

/**
 * Created by nv95 on 11.11.16.
 */

public class WhiteWidget extends StandardWidget {

    @Override
    int getLayoutRes() {
        return R.layout.widget_white;
    }
}
```