CS 213 Spring 2016
Lecture 23: April 12

Song Library Android App

Persistence, Action Bar,

Deleting in AddSong Screen

Part 6:
Implementing Persistence

Using Internal Storage

There are two methods in the current version of MySongList that are not implemented: load and store. These are the methods that need to be filled in to implement persistence.

They require access to the application's context, which is to be set by the application's call to the setContext method.

Context: `MySongList` Class

1.

`setContext`: Define a `android.content.Context` field in the class, and in the method, set the field to the parameter.

Store: `MySongList` Class

2.

`store`: Define a file name (such as "songs.dat") with a constant, e.g. `SONGS_FILE`, and store all songs in that file.

See Develop -> API Guides -> Data Storage -> Storage Options -> Using the Internal Storage

```
FileOutputStream fos =
      context.openFileOutput(SONGS_FILE,Context.MODE_PRIVATE);
PrintWriter pw = new PrintWriter(fos);
for (Song song: songs) {
    pw.println(song.getString());
}
pw.close();
```

The `getString` method in the Song class serves as the file output format

Load: `MySongList` Class

3.

`load`: When loading the songs from storage, id's have to be dealt out in sequence, `Song` objects created, and added to the song list instance. Also, the very first time the app is run, there will not be a file, in which case the songs should be read from the song array in `strings.xml`.
(Alternatively, you could start with a blank slate.)

Load: MySongList Class

3.

```
int maxId=-1;
try {
    FileInputStream fis =
            context.openFileInput(SONGS_FILE);
    BufferedReader br =
            new BufferedReader(
                    new InputStreamReader(fis));
    String songInfo;
    while ((songInfo = br.readLine()) != null) {
        Song song = Song.parseSong(songInfo);
        maxId++;
        song.id = maxId;
        songs.add(song);
    }
    this.maxId = maxId;
    fis.close();

} catch (FileNotFoundException e) {  // default to initial set
    …
}
```

Load: MySongList Class

3.

```
...
} catch (FileNotFoundException e) {  // default to initial set
   // load initial set of songs
   String[] initSongs = context.getResources()
                                .getStringArray(R.array.song_array);
   // break songs into name and artist, add to list
   for (String song: initSongs) {
       int pos = song.indexOf('|');
       add(song.substring(0,pos),song.substring(pos+1),
              null,null);
   }
}
```

Parsing Song from File: Song Class

Add this method in Song:

```
public static Song parseSong(String songInfo) {
    String[] tokens = songInfo.split(":");
    Song song = new Song();
    song.id = -1;
    song.name = tokens[0];
    song.artist = tokens[1];
    song.album = "";
    song.year = "";
    if (tokens.length > 2) {
       song.album = tokens[2];
    }
    if (tokens.length > 3) {
       song.year = tokens[3];
    }
    return song;
}
```

And a private constructor: `Song() { }`

Add/Update: `MySongList` Class

4.

Everywhere that a song is added or updated, call the `store()` method to immediately write the list of songs to file. Catch `IOException`, and show a Toast with an error message.

In method `add`:

```
if (songs.size() == 0) {
    songs.add(song);
    try {
       store();  // write through
    } catch (IOException e) {
       Toast.makeText(context, "Could not store songs to file",
                      Toast.LENGTH_LONG).show();
    }

    return song;
}
```

Add/Update: `MySongList` Class

4.

In method `remove`:

```
songs.remove(pos);
try {
    store();
} catch (IOException e) {
    Toast.makeText(context,"Could not store songs to file",
                    Toast.LENGTH_LONG).show();
}
return songs;
```

In method `update`:

```
songs.set(i, song);
try {
    store();
} catch (IOException e) {
    Toast.makeText(context,"Could not store songs to file",
                    Toast.LENGTH_LONG).show();
}
return songs;
```

Loading: SongLib Class

Replace the song loading from the strings array in strings.xml with loading from the song list instance.

```
/*
// load initial set of songs
String[] initSongs = getResources()
                        .getStringArray(R.array.song_array);
// break songs into name and artist, add to list
...
*/

try {
    myList.setContext(this);
    myList.load();
} catch (IOException e) {
    Toast.makeText(this, "Error loading songs", Toast.LENGTH_LONG)
       .show();
}
```

An Activity itself is a Context (subclass of Context)

Try it out!

When you add/delete and save the list, it goes into the `songs.dat` file

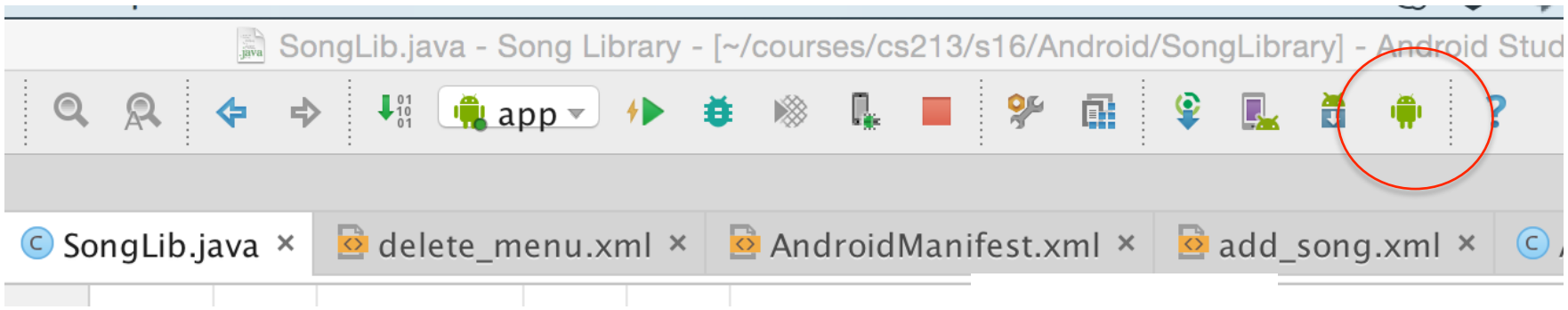
But you won't find the file in the project directories, because it is in the device storage outside the application package.

To see it you need to use the Android Device Monitor tool

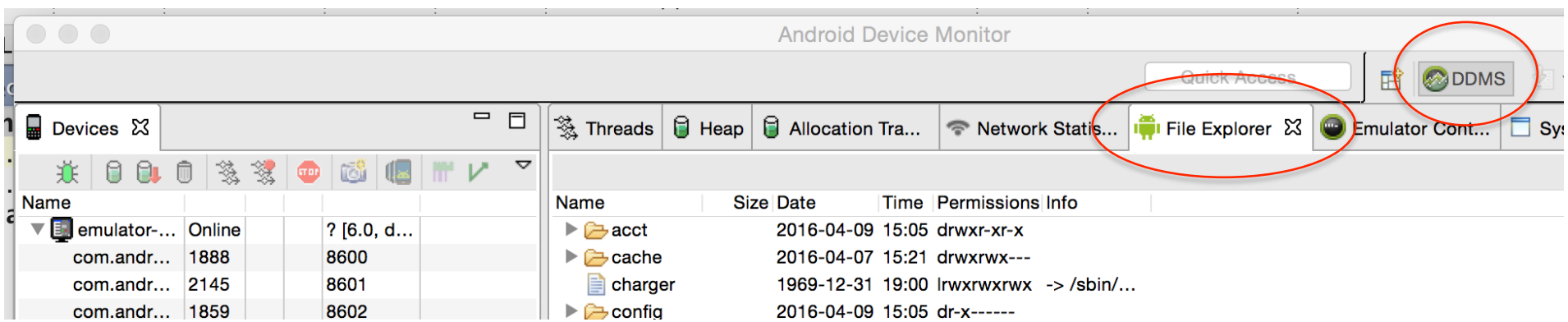
Android Device Monitor

See Develop -> Tools -> Tools Help -> Device Monitor

http://developer.android.com/tools/help/monitor.html



With the emulator running, you should see this:

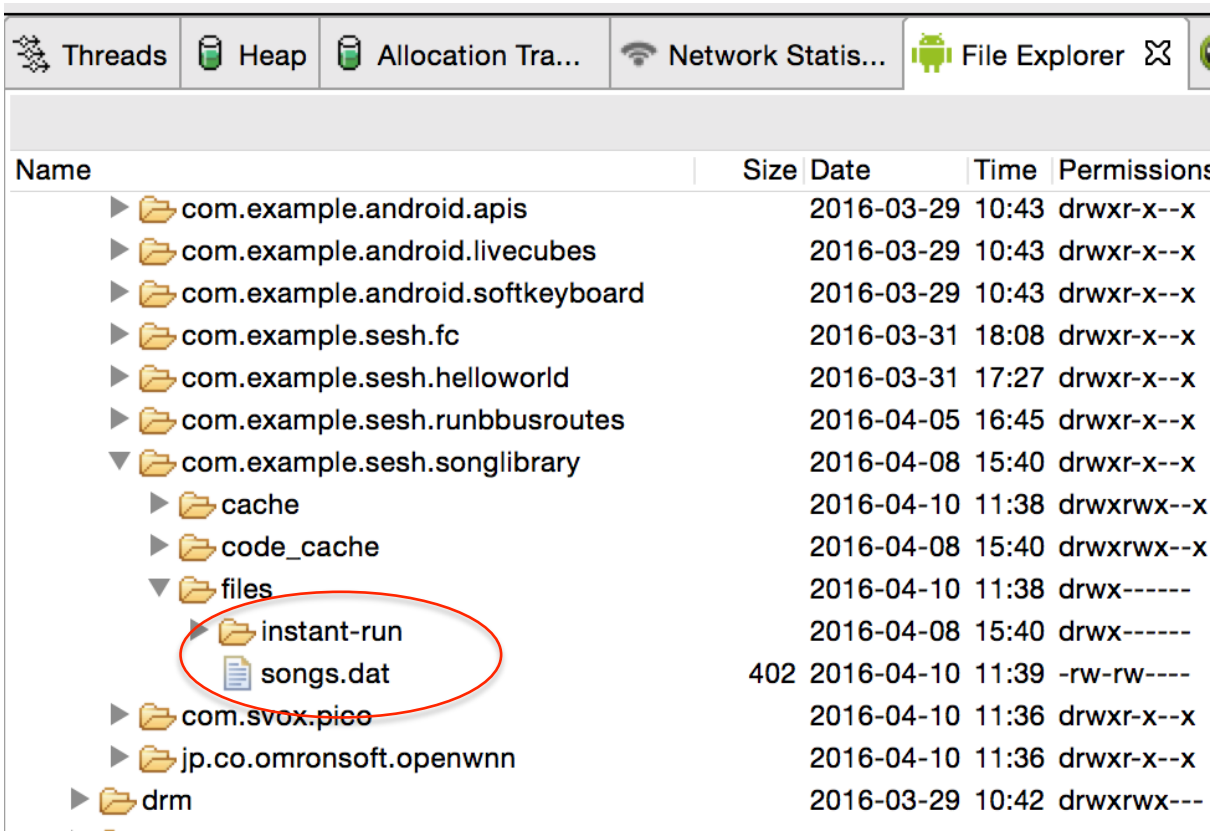


Working with Device Filesystem

Under Develop -> Tools -> Debugging Tools -> DDMS

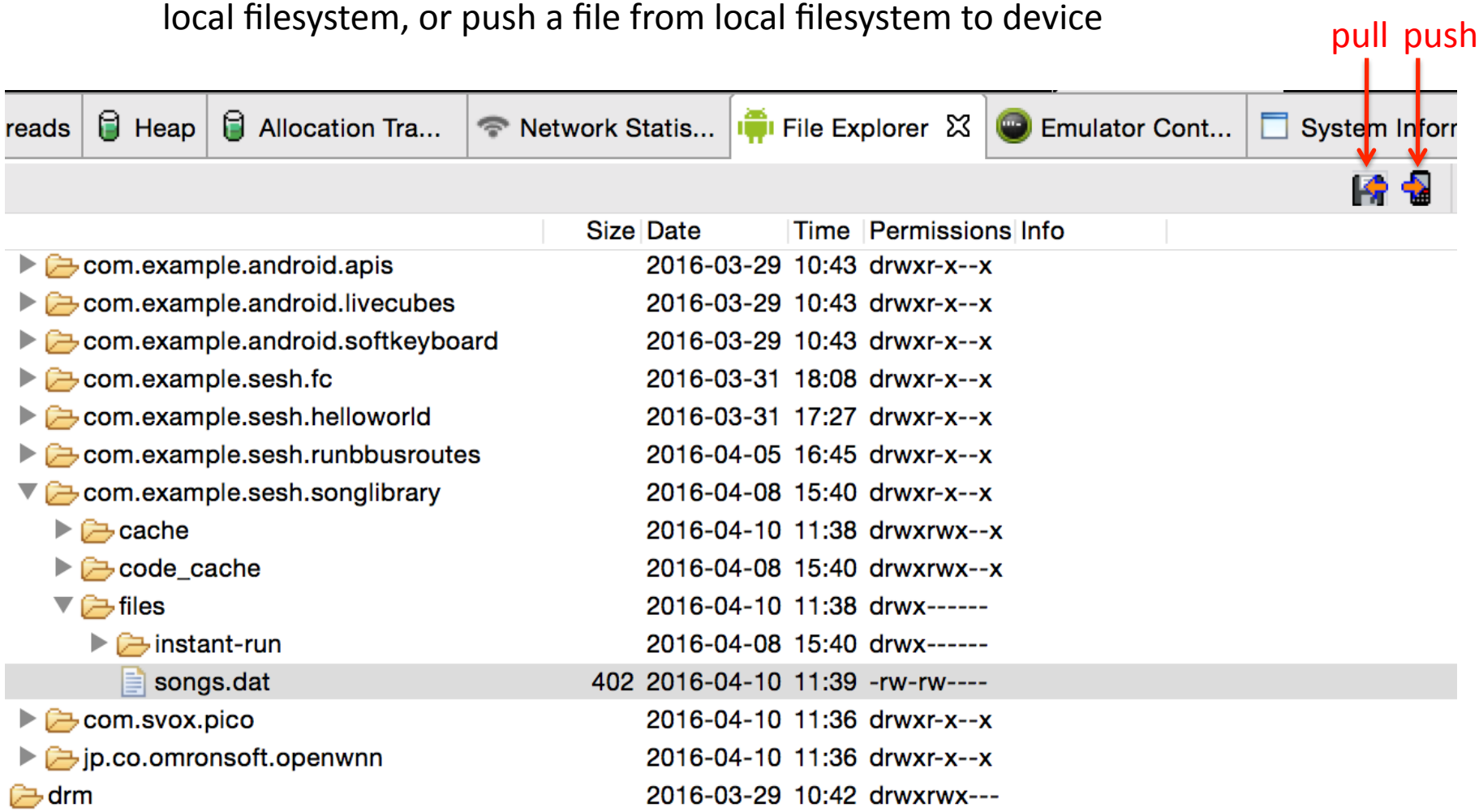
See Working with an emulator or device's filesystem

In File Explorer, you should see the file under data->data->com.example.sesh.songlibrary->files:



Working with Device Filesystem

You can use the “pull” and “push” buttons to pull a file from device to local filesystem, or push a file from local filesystem to device



After an add or delete, close the app in the emulator and reopen it

You will see the updated list loaded in from songs.dat,
but following this list, you will see the entire list repeated!

Why?

MySongList static instance

```
public class MySongList implements SongList {

    // single instance
    private static MySongList songList=null;
  …
}
```

The MySongList instance is static, so it persists beyond the run time of the SongLib and AddSong activties.

When SongLib is launched again after closing, the songs are loaded again, and added to the original ArrayList of songs in MySongList

```
try {
    myList.setContext(this);
    myList.load();
} catch (IOException e) {
    ...
}
```

→

```
while ((songInfo =
br.readLine()) != null) {
    ...
    songs.add(song);
}
```

MySongList static instance

To fix this, have the MySongList getInstance() method accept a Context parameter, and load the songs if the instance is null:

```
public static MySongList getInstance(Context ctx)
    throws IOException {
        if (songList == null) {
            songList = new MySongList();
            songList.context = ctx;
            songList.load();
        }
        return songList;
    }
```

And the setContext method is no longer needed

Now, when SongLib is launched again after closing, the MySongList instance that was created earlier is still around, and songs will not be reloaded. Make the appropriate change in SongLib:

```
try {
    myList = MySongList.getInstance(this);
    myList.setContext(this);
    myList.load();
} ...
```

Instead of the earlier call to getInstance()

Uninstall the app, then load it again from Android Studio and run it

Do an add, delete, or update, close the app, then relaunch it.

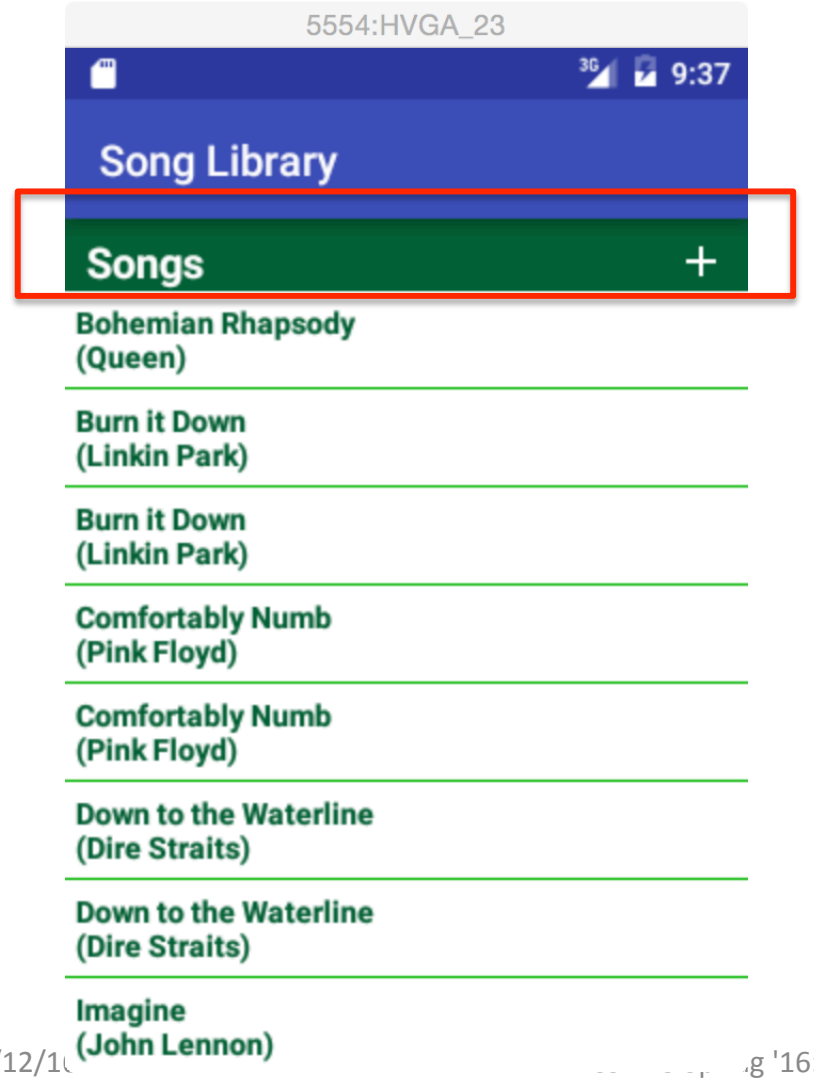
The songs should not reload, you should see a correct single list with the updates you made.

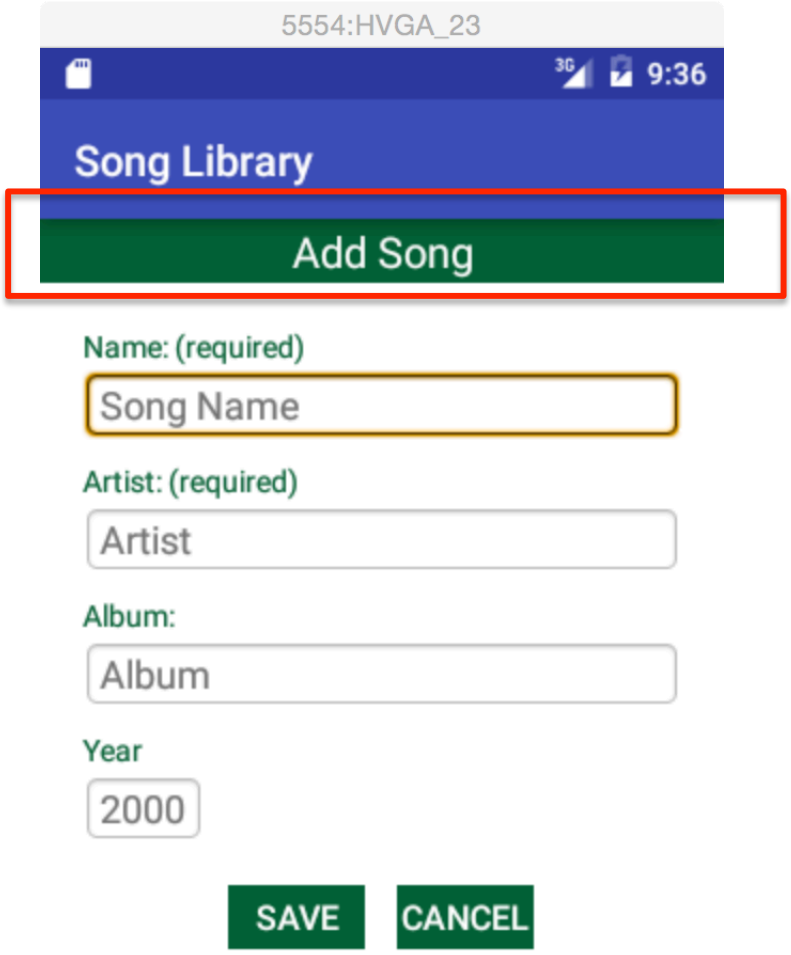
Part 7:
Switching to Action Bar

(Because the Action Bar theme has a light gray background, we will use the black versions of the plus and delete icons)

Action Bar

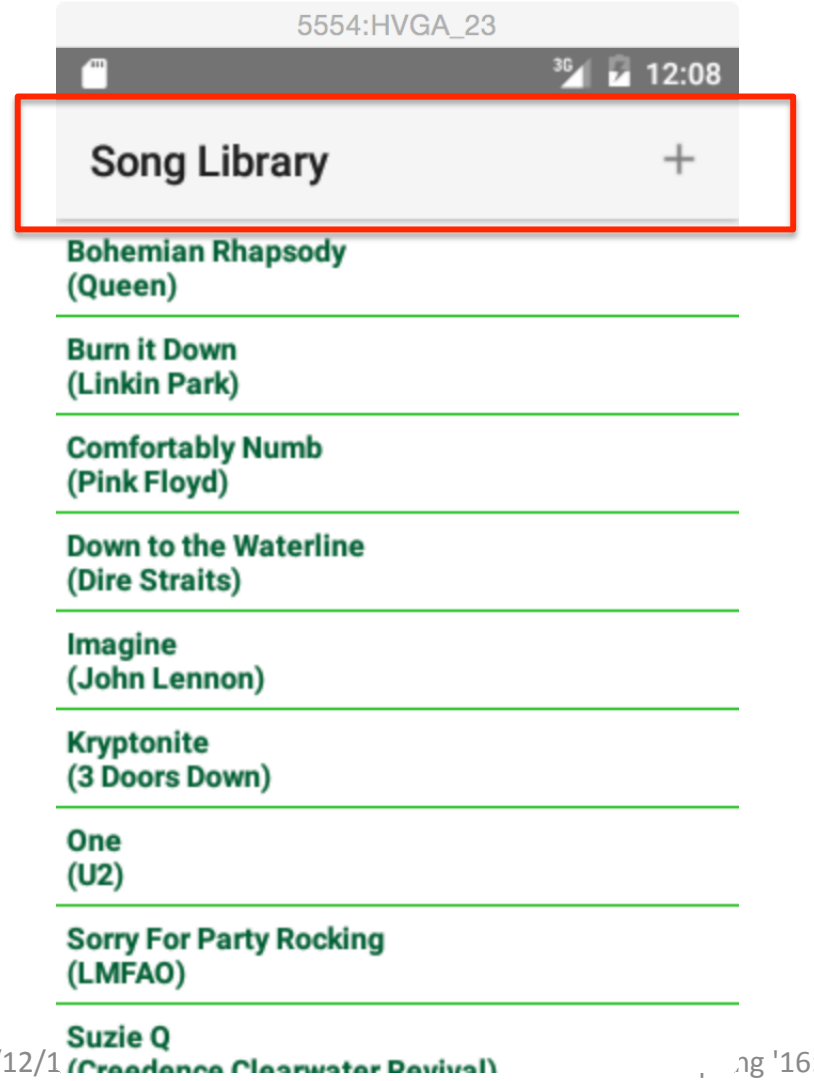
There is a separate title bar in `SongLib` and `AddSong`:

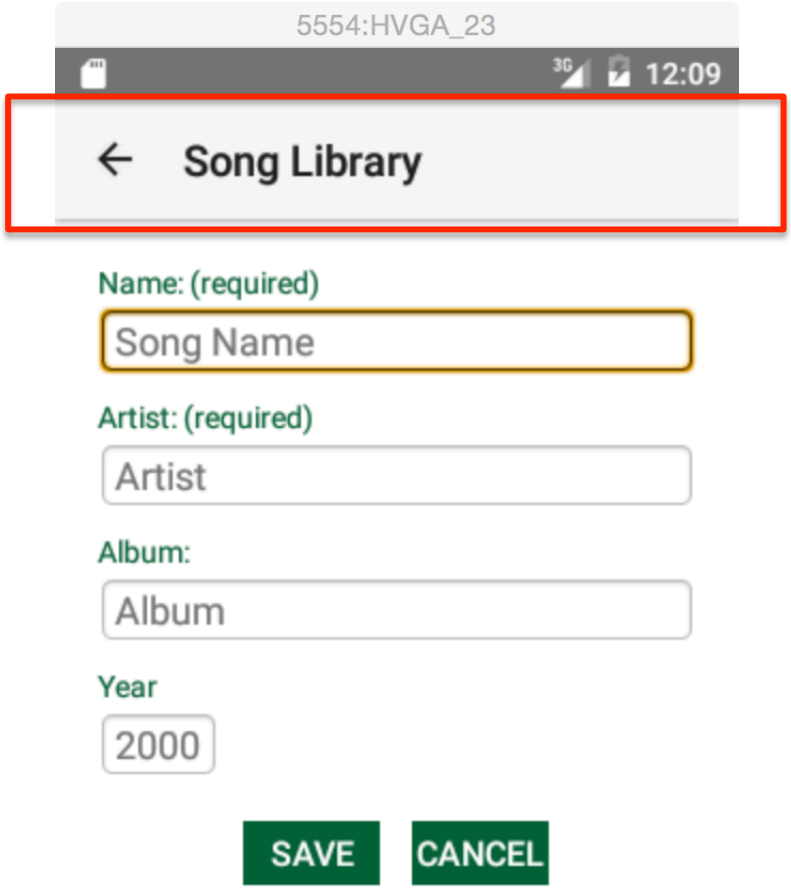




Action Bar

Want to replace the title bar with Action Bar:





Adding Action Bar

- In the application manifest file, turn off the ActionBar:

```
<application
  …
  android:theme="@style/Theme.AppCompat.Light.NoActionBar”/>
```

- Add a `ToolBar` widget to the top of the `song_list` and `add_song` layouts:

```
<LinearLayout
  ...>
  <android.support.v7.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
  ...

</LinearLayout>
```

Adding Action Bar

- Turn off the title bar `TextView` element in `song_list` and `add_song` layouts:

- In the `ShowRoute` code's `onCreate` method, get the `Toolbar`, and pass it in to the method to set the support action bar:

```
protected void onCreate() {
  ...
  Toolbar toolbar = (Toolbar)findViewById(R.id.my_toolbar);
  setSupportActionBar(toolbar);
  ...
}
```

Create a Menu Resource for SongLib Action Bar

- In the res/menu folder, create a menu resource file called add_menu.xml, with the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/android"
      xmlns:appcompat="http://schemas.android.com/apk/res-auto">
    <item android:id="@+id/action_add"
        android:icon="@drawable/ic_action_plus_black"
        android:title="@string/add”
        appcompat:showAsAction="always”/>
</menu>
```

Text version, in strings.xml

- Also, update delete_menu.xml to use the black version of the delete icon

Fitting the Action Bar with the Menu in `SongLib`

- Override the `onCreateOptionsMenu(Menu)` method to inflate the menu resource – this method will be called when the app is launched

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.add_menu,menu);
    return super.onCreateOptionsMenu(menu);
}
```

Set up a callback for add event

- In `SongLib`, override `onOptionsItemSelected` method (which is called whenever an item is clicked in the Action Bar):

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_add:
            addSong();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Remove parameter View from the addSong method definition, since it is no longer called from clicking on the + icon in the title bar

Adding the "Up" Action to AddSong

- In the app's manifest, add a parentActivityName attribute to the AddSong activity tag, AND add a meta-data tag for the Up navigation to work on older APIs:

```
<activity
    android:name=".AddSong">
    android:parentActivityName=".SongLib"
    <meta-data
          android:name="android.support.PARENT_ACTIVITY"
          android:value=".SongLib" />
>
</activity>
```

- In the AddSong onCreate method, enable the Up button:

```
ActionBar ab = getSupportActionBar();
ab.setDisplayHomeAsUpEnabled(true);
```

Setting up a callback for Up event

- In the `AddSong` activity, override `onOptionsItemSelected` method:

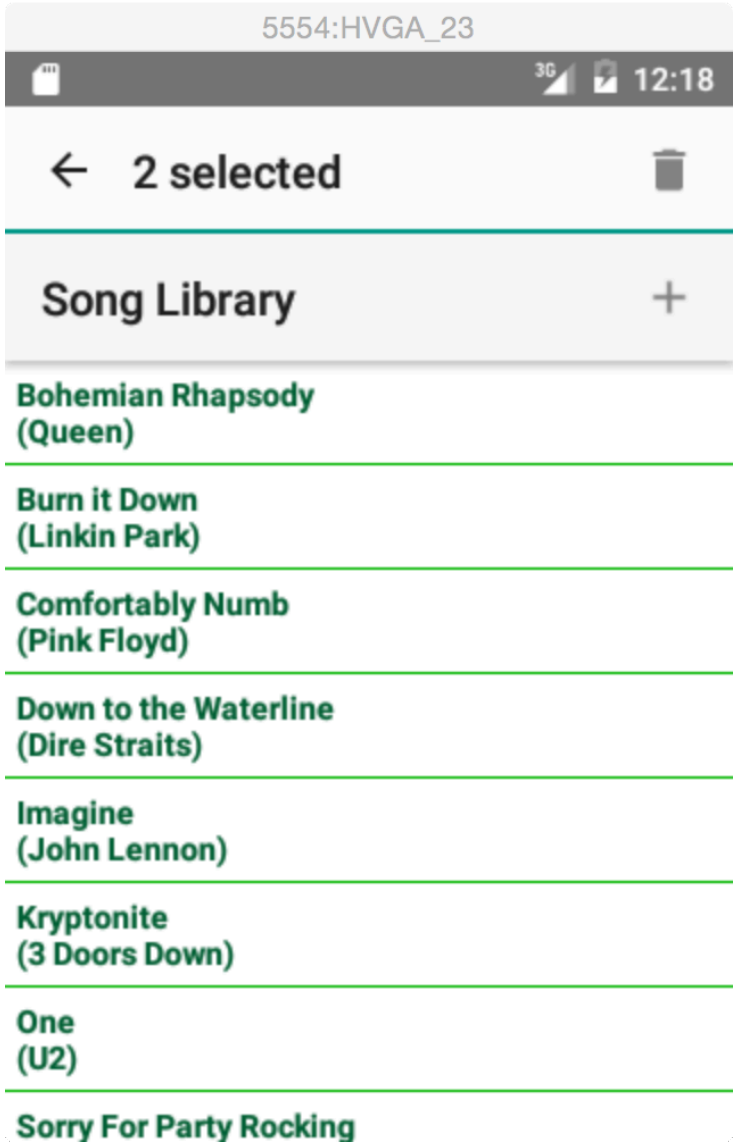
```
public boolean onOptionsItemSelected(MenuItem item) {
    return super.onOptionsItemSelected(item);

}
```

- This method is called back whenever an item is clicked on in the App Bar. For the special case of the Up navigation, the event is handled by the superclass

Try it Out!

Contextual Action Bar Above Action Bar



The Contextual Action Bar pushes the Action Bar down.

We want the CAB to overlay the Action Bar

CAB Overlay on Action Bar

- Modify `res/values/styles.xml` like this:

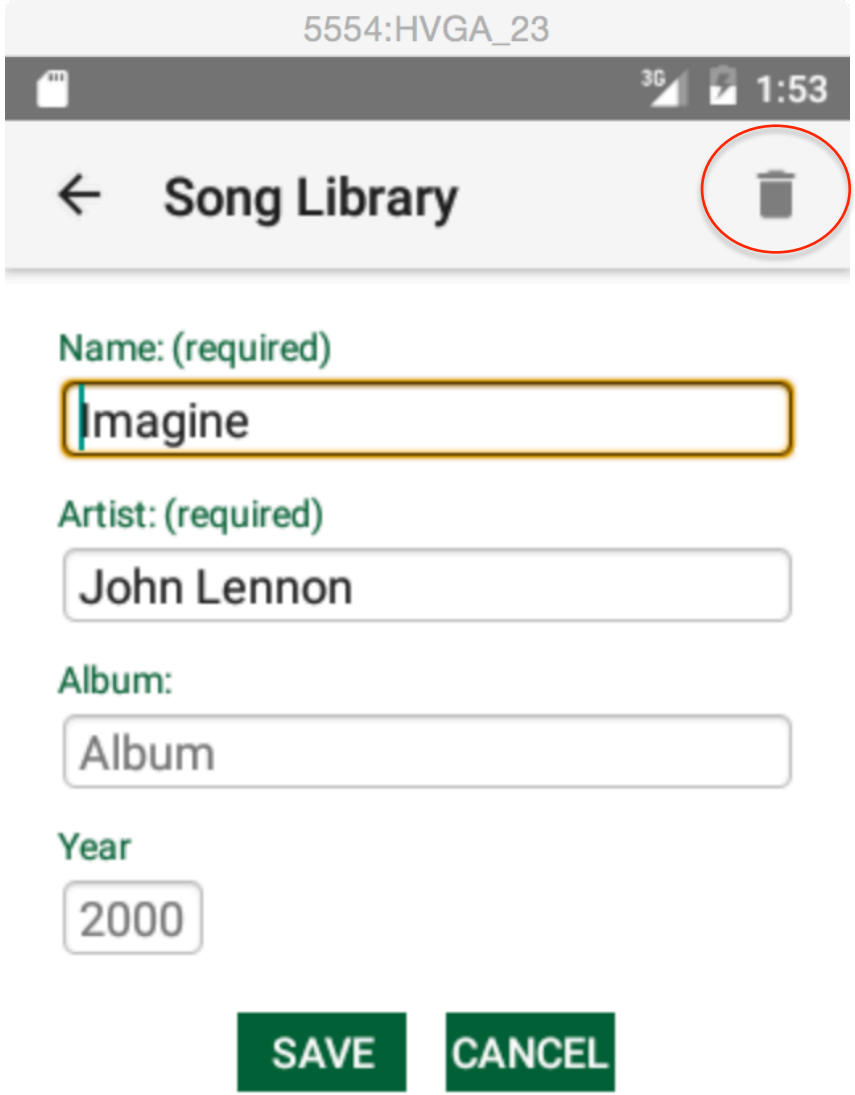
```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <!--
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        -->
        <item name="windowActionModeOverlay">true</item>
    </style>
</resources>
```

- In the Manifest file, change the application theme to AppTheme:

```
<application
      ...
      android:theme="@style/AppTheme”>
```

Contextual Action Bar Overlays Action Bar

Part 8:
Enabling Delete in AddSong Screen
(if started for Edit, not for Add)


5554:HVGA_23
1:53
Song Library
Name: (required)
Imagine
Artist: (required)
John Lennon
Album:
Album
Year
2000
SAVE
CANCEL

Adding Delete Icon to AddSong Action Bar

- We can reuse the menu resource delete_menu.xml that we are using for the CAB, with the addition of showAsAction:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
      xmlns:appcompat="http://schemas.android.com/apk/res-auto"
<item android:id="@+id/action_delete"
        android:icon="@drawable/ic_action_delete_black"
        android:title="@string/delete"
        appcompat:showAsAction="always"
        />
</menu>
```

Was menu_delete (update the event handler for CAB in SongLib)

Fitting the Action Bar with the Menu in AddSong

- Override the `onCreateOptionsMenu(Menu)` method to inflate the menu resource

Because delete should only be available in Edit mode (not add) – this id will be set to non-negative if song was passed in

```
public boolean onCreateOptionsMenu(Menu menu) {
    if (songID != -1) {
        getMenuInflater().inflate(R.menu.delete_menu,menu);
    }
    return super.onCreateOptionsMenu(menu);
}
```

- Initialize `songID` to -1 before incoming intent is checked for bundle

Handling the Delete Event

- Modify `onOptionsItemSelected(MenuItem)` like this:

```
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.action_delete:
         deleteSong();
         return true;
      default:
         return super.onOptionsItemSelected(item);
   }
}
```

To be implemented

Will be executed for the Up action

Implement `deleteSong`

- The actual deletion should be handled by `SongLib`, just as it is done for batch deletes via the CAB
- So the `deleteSong` method in `AddSong` should send a message back to `SongLib` that a delete was requested, which can be done by passing another key `SONG_DELETE` in a bundle, with a boolean value set to `true`

Implement deleteSong

```
public void deleteSong() {
      Bundle bundle = new Bundle();

      bundle.putString(SONG_NAME,songName.getText().toString());
      bundle.putString(SONG_ARTIST,songArtist.getText().toString());
      bundle.putString(SONG_ALBUM,songAlbum.getText().toString());
      bundle.putString(SONG_YEAR,songYear.getText().toString());
      bundle.putInt(SONG_ID, songID);
      bundle.putBoolean(SONG_DELETE,true);


      Intent intent = new Intent();
      intent.putExtras(bundle);

      setResult(RESULT_OK,intent);
      finish();
}
```

Callback in SongLib

- In SongLib, add code in onActivityResult to check if a delete was requested, and take appropriate action if it was

```
public void onActivityResult(...) {
   ...
    if (requestCode == ADD_SONG_CODE) {
        myList.add(name, artist, album, year);
    } else if (requestCode == EDIT_SONG_CODE) {
        Song song = new Song(songID, name, artist, album, year);
        if (bundle.getBoolean(AddSong.SONG_DELETE)) {
            myList.remove(song);
        } else {
            myList.update(song);
        }
    }
     ...
}
```

Returns value if key is in bundle, false otherwise

Take it for a spin!