CS 213 Spring 2016
Lecture 22: April 7

Song Library Android App

Create Project

- Make a project called Song Library
  - Use in SDK level default API 15
  - Empty Activity
- Call the main activity `SongLib`
- Call the (generated) main activity layout file `song_list`

Part 1:
Showing a List of Songs

Using an Icon for Adding Songs

- We will use a '+' icon to add songs. This icon will show up as an action in a "compound drawable" area (text + icon) at the top of `SongLib` activity that shows a list of songs
- There are prefab icons supplied by the Android guys for a whole lot of standard tasks, including one to add content (such as songs in our app)

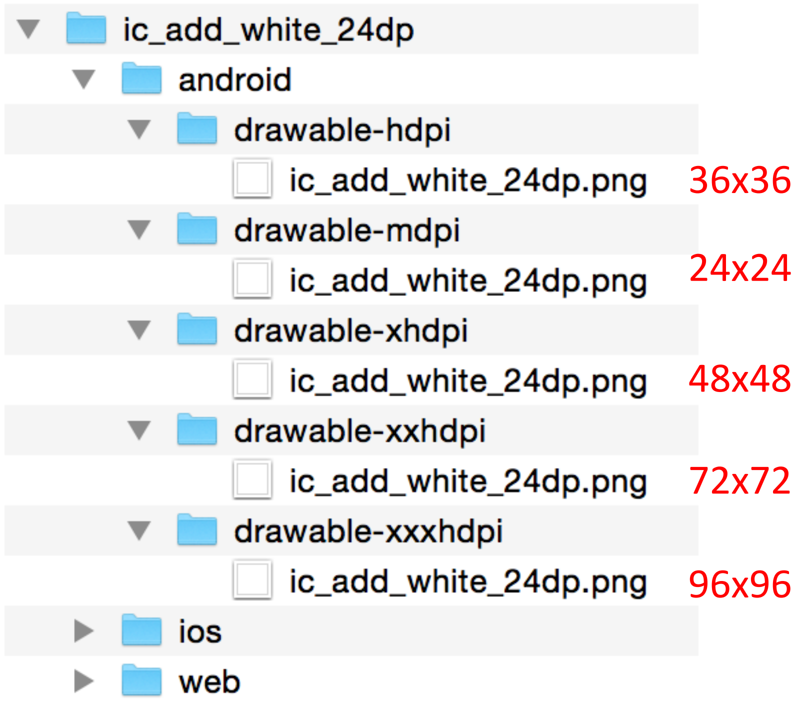
  Go to Design: Material Icon Collection (bottom of Design page) (https://design.google.com/icons/index.html)

  On the Material Icon Collection page, search for "Content" – this will show a collection of Content icons. The first one of them is the '+'

  Click on the '+' icon: this brings up a tool bar at the bottom of the browser page. Select the white version and download the PNGs. This will download a zip file which unzips to a folder named ic_add_white_24dp

+ icon for various screen densities

- The downloaded collection of '+' icons is distributed over several folders, one per screen density, with different sizes

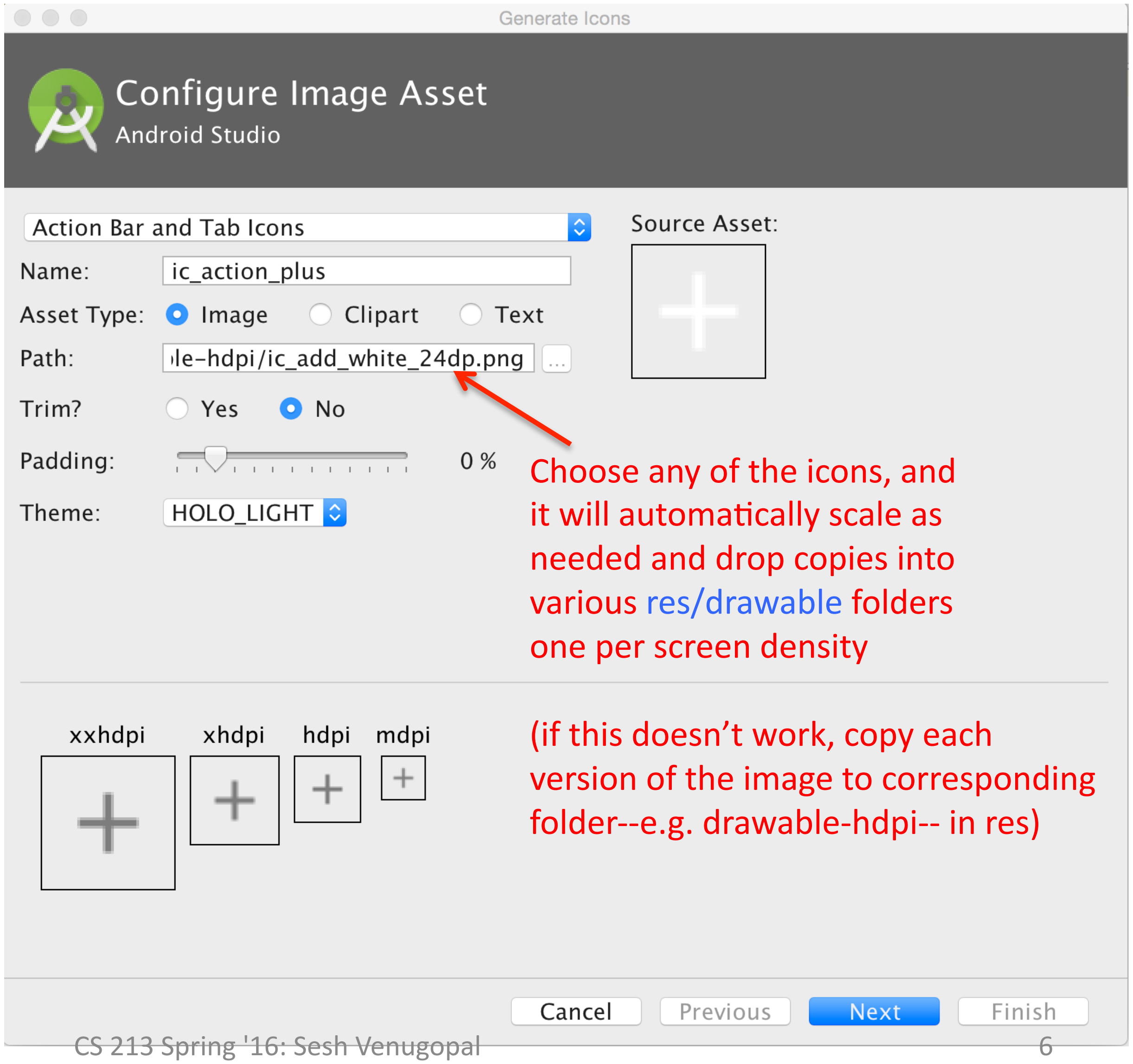


```
▼ ic_add_white_24dp
    ▼ android
        ▼ drawable-hdpi
            ic_add_white_24dp.png    36x36
        ▼ drawable-mdpi
            ic_add_white_24dp.png    24x24
        ▼ drawable-xhdpi
            ic_add_white_24dp.png    48x48
        ▼ drawable-xxhdpi
            ic_add_white_24dp.png    72x72
        ▼ drawable-xxxhdpi
            ic_add_white_24dp.png    96x96
    ▶ ios
    ▶ web
```

(See Develop -> API Guides -> Best Practices -> Supporting Multiple Screens)

Adding icons to project

Right click on res,
then choose
New -> Image Asset,
then configure like this



Action Bar and Tab Icons

Name: ic_action_plus

Asset Type: [✓ Image] Clipart Text

Path: le-hdpi/ic_add_white_24dp.png

Trim? Yes [✓ No]

Padding: 0 %

Theme: HOLO_LIGHT

Source Asset:

xxhdpi xhdpi hdpi mdpi

Choose any of the icons, and it will automatically scale as needed and drop copies into various res/drawable folders one per screen density

(if this doesn't work, copy each version of the image to corresponding folder--e.g. drawable-hdpi-- in res)

Bootstrap list of songs in `strings.xml`

- The `res/values/strings.xml` file will have a bootstrap list of song names (in alphabetical order) to show when app launches (copy from Sakai -> Resources -> Apr 7):

```
<resources>
    <string name="app_name">Song Library</string>

    <!-- Initial list of songs -->
    <string-array name="song_array">
        <item>Bohemian Rhapsody|Queen</item>
        <item>Burn it Down|Linkin Park</item>
        <item>Comfortably Numb|Pink Floyd</item>
        <item>Down to the Waterline|Dire Straits</item>
        <item>Imagine|John Lennon</item>
        <item>Kryptonite|3 Doors Down</item>
        <item>One|U2</item>
        <item>Sorry For Party Rocking|LMFAO</item>
        <item>Suzie Q|Creedence Clearwater Revival</item>
        <item>Uptown Funk|Mark Ronson ft. Bruno Mars</item>
    </string-array>
</resources>
```

Colors names in `colors.xml`

- The `res/values/colors.xml` file will have a list of named colors that may be used by reference, instead of hard coding (copy from Resources -> Apr 7):

```
<resources>
    <color name="White">#FFF</color>
    <color name="Black">#000</color>
    <color name="MyGreen">#04B404</color>
    <color name="DeepPurple">#5c2a7e</color>
    <color name="MyIndigo">#38279a</color>
    <color name="Yellow">#ffff00</color>
    <color name="MyLightGreen">#F8FBEF</color>
    <color name="MyDarkGreen">#0B6138</color>
</resources>
```

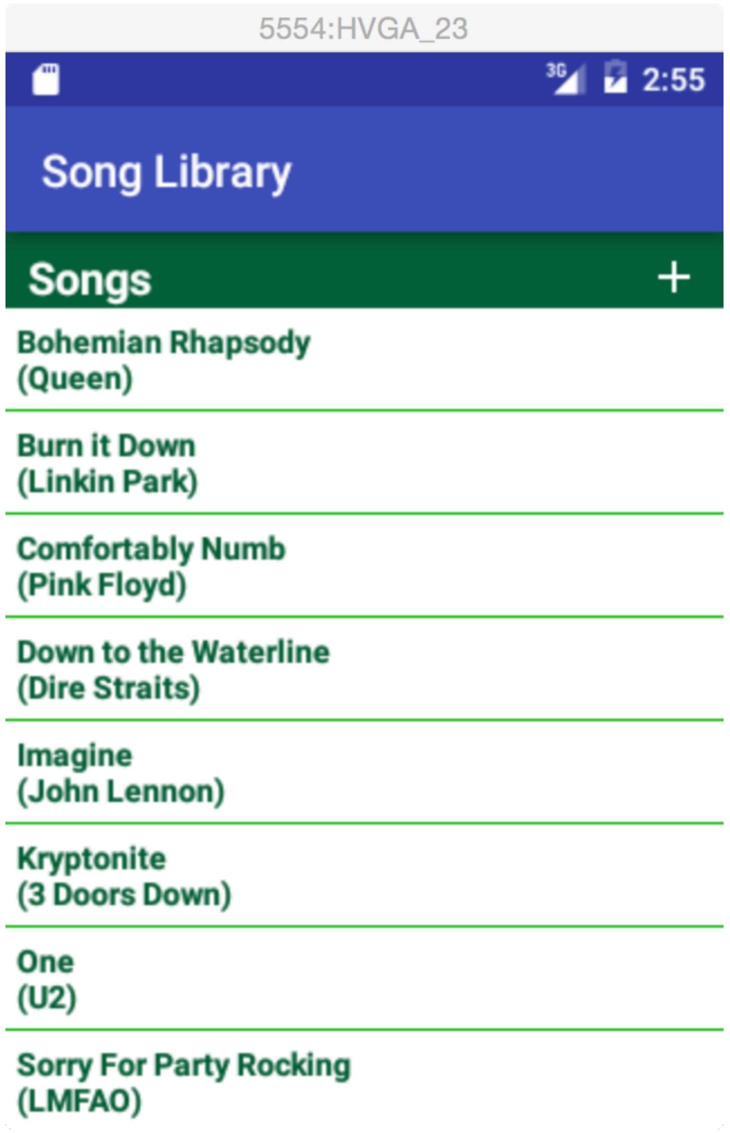
(Resources -> Apr 5) Layout `song_list.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width=”match_parent"
    android:layout_height=”match_parent"
    android:orientation="vertical"
    tools:context="com.example.sesh.songlibrary.SongLib">

<!-- Compound drawable (text + icon) layout for add song trigger -->
<TextView
     android:id="@+id/add_song"
     android:layout_width="match_parent”
     android:layout_height="wrap_content"
     android:background="@color/MyDarkGreen”
     android:drawableEnd="@drawable/ic_action_plus"
     android:drawableRight="@drawable/ic_action_plus"
     android:drawablePadding="5dp"
     android:paddingTop="6dp"
     android:paddingRight="10dp"
     android:paddingLeft="10dp"
     android:text="@string/song_list"
     android:textColor="@color/White"
     android:textSize="20sp"
     android:textStyle="bold"
     android:onClick="addSong"/>
...
```

Layout song_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>

<!-- Compound drawable (text + icon) layout for add song trigger -->
<TextView
     .../>

<!-- Layout for song list -->
<ListView
    android:id="@+id/song_list"
    android:layout_width=”match_parent”
    android:layout_height=”match_parent"
    android:divider="@color/MyGreen"
    android:dividerHeight="1dp" />

</LinearLayout>
```

Preview of List


5554:HVGA_23

2:55

Song Library

Songs +

Bohemian Rhapsody
(Queen)

Burn it Down
(Linkin Park)

Comfortably Numb
(Pink Floyd)

Down to the Waterline
(Dire Straits)

Imagine
(John Lennon)

Kryptonite
(3 Doors Down)

One
(U2)

Sorry For Party Rocking
(LMFAO)

Song Name Layout

- In `res/layout/song.xml` file: for instance, each song name is rendered in white lettering on a dark green background (Resources -> Apr 7)

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width=”match_parent"
    android:layout_height="wrap_content"
    android:textSize="14sp"
    android:background="@color/White"
    android:textColor="@color/MyDarkGreen"
    android:textStyle="bold"
    android:typeface="sans"
    android:padding="5dp” />
```

`Song` Class

- Copy `Song.java` into the project, alongside `SongLib.java`
- The `toString` method has been tailored for feeding in to the `ListView` Adapter that will be fitted to the `ListView` that will show the song list: it returns the name of the song. (The `ListView` Adapter will call `toString` on the `Song` objects in the adapter.)
- The `getString` method has been tailored for writing into a data file that will hold all songs

`SongList` Interface

- Copy `SongList.java` into the project, alongside `SongLib.java`

`SongList` is an interface that can be implemented by any class that wants to maintain a list of songs, with methods to:

  - load songs from (and store to) offline storage,
  - get the list of songs,
  - add/update/remove songs, and
  - get the index of a song in the list.

`MySongList` Class

- Copy `MySongList.java` into the project, alongside `SongLib.java`

`MySongList` implements the SongList interface (`load`, `store`, `setContext` methods to be filled in later):

- It implements the Singleton design pattern
- `add` method generates and assigns unique integer ids to songs
- `getPos` method binary searches on song name, then matches id
- `update` method is forced to sequential search on id since song name itself might change in the update

Code in SongLib.java to show list

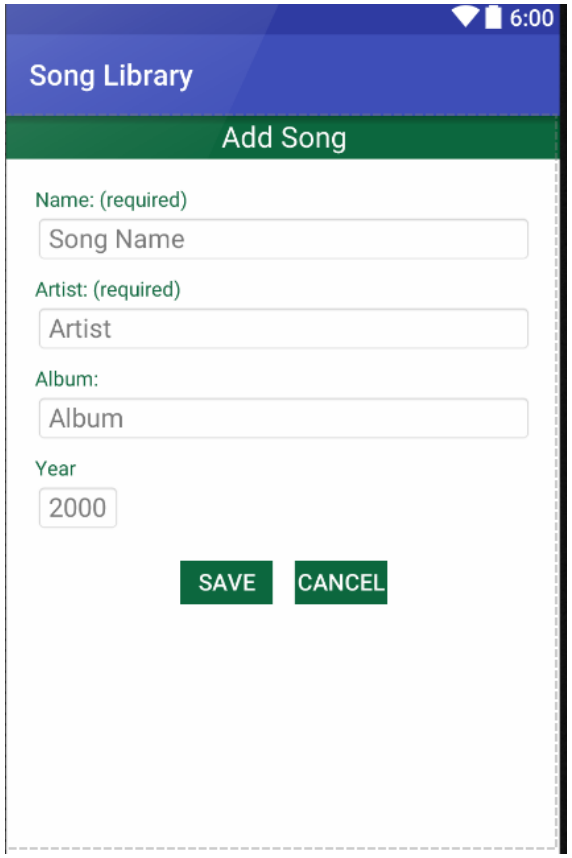
```
private MySongList myList;
private ListView listView;

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

    // get instance of MySongList
    myList = MySongList.getInstance();

    // load initial set of songs
    String[] initSongs = getResources().getStringArray(R.array.song_array);

    // break songs into name and artist, add to list
    for (String song: initSongs) {
        int pos = song.indexOf('|');
        myList.add(song.substring(0,pos),song.substring(pos+1),
                   null,null);
    }
    // get ListView object
    listView = (ListView)findViewById(R.id.song_list);

    // fit listView with adapter off of myList, and song layout
    listView.setAdapter(
            new ArrayAdapter<Song>(this,R.layout.song,myList.getSongs()));
}
```

Run the app and see the song list!

Part 2:
Adding a new Song

Edit Song Layout

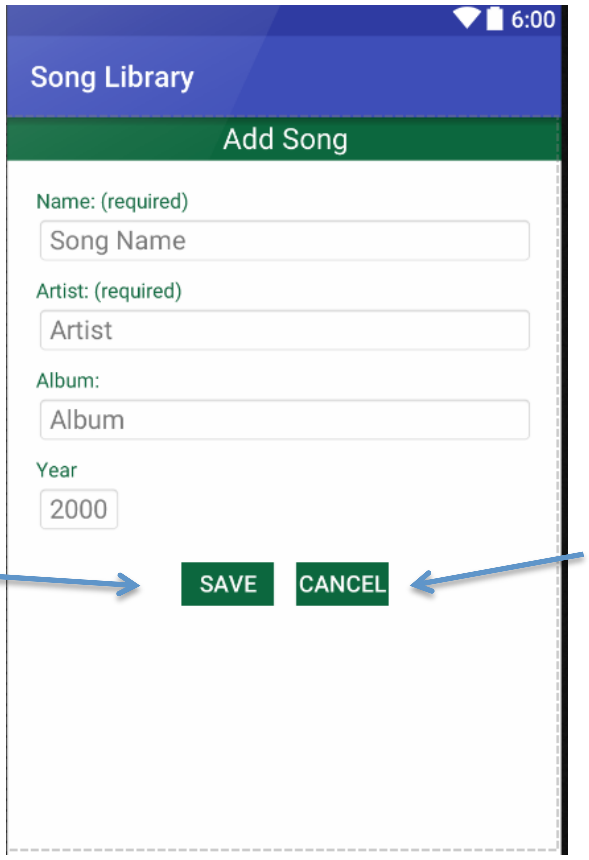
- Copy `add_song.xml` into `res/layout`: the design view is this:



Add Song Layout

- Copy `add_song.xml` into `res/layout`: the design view is this:

Callback
`android:onclick="save"`


6:00
Song Library
Add Song
Name: (required)
Song Name
Artist: (required)
Artist
Album:
Album
Year
2000
SAVE
CANCEL

Callback
`android:onclick="cancel"`

AddSong Class

- From Resources -> Apr 7, copy AddSong.java into the project

Note the following in AddSong:

- Keys to use for bundling song data BACK to parent (SongLib)
- save and cancel methods to handle events, called back by the Save and Cancel buttons, respectively
- The code to send song data back to parent, and release control:

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

- The result (RESULT_OK is a constant defined in the android.app.Activity class) is checked by the parent activity
- Errors in input are shown by an easy dialoging mechanism:

```
Toast.makeText(this, "Name and Artist are required",
               Toast.LENGTH_SHORT)
     .show();
```

SongLib.java : Launch AddSong

```
public static final int ADD_SONG_CODE=1;

public void addSong(View view) {
    Intent intent = new Intent(this, AddSong.class);
    startActivityForResult(intent, ADD_SONG_CODE);
}
```

When a child activity returns, this code will be used to determine which of potentially several children activities it is

The addSong method is the callback that is defined in the TextView in song_list.xml for when the + icon is clicked:

```
<!-- Compound drawable (text + icon) layout for add song trigger -->
<TextView
  ...
 android:onClick="addSong"
/>
```

SongLib.java : Return from AddSong

The onActivityResult method (overridden here) is called when an activity (AddSong) that was launched for result finishes up

```
protected void onActivityResult(int requestCode, int resultCode,
                                     Intent intent) {

    if (resultCode != RESULT_OK) { return; }

    Bundle bundle = intent.getExtras();
    if (bundle == null) { return; }

    String name = bundle.getString(AddSong.SONG_NAME);
    String album = bundle.getString(AddSong.SONG_ALBUM);
    String artist = bundle.getString(AddSong.SONG_ARTIST);
    String year = bundle.getString(AddSong.SONG_YEAR);

    if (requestCode == ADD_SONG_CODE) {
       myList.add(name, artist, album, year);
       listView.setAdapter(
           new ArrayAdapter<Song>(this,
                    R.layout.song, myList.getSongs()));

       }
}
```

This check is needed to determine which of potentially several children Actitivies is returning

Adapter has to be redone since the source content has changed

Try it Out!

Part 3:
Replacing Toast with Dialog

Building an “alert” Dialog

- If the user does not enter song name or artist, we showed a Toast, which is essentially a no-frills pop-up that shows up for a short of long moment:

```
Toast.makeText(this, "Name and Artist are required",
               Toast.LENGTH_SHORT)
     .show();
```

- Let’s replace it with a proper Dialog that gives the user control on when to pull it down
- This is going to be done using a `DialogFragment`

Using a DialogFragment

(See Develop -> API Guides -> User Interface -> Dialogs)

- Copy the code under the "Creating a Dialog Fragment" section, but in a `SongInfoDialogFragment` class (instead of `FireMissilesDialogFragment`) – the `DialogFragment` class is in the `android.app` package

- You should have the following imports (just to clarify, because some of these classes have versions in other packages):

```
import android.app.Dialog;
import android.content.DialogInterface;
import android.app.DialogFragment;
import android.support.v7.app.AlertDialog;
```

Modifying the code template: Setting OK and Cancel buttons

- We will have a single "OK" button in the dialog, which is the "positive" button. So use "OK" in place of `R.string.fire`

- We will not have a "negative" button for Cancel, since our dialog is a simple information dialog, so remove the `setNegativeButton` part of the code

Modifying the code template: Setting the Message

- The message to display in the dialog should come from the `AddSong` activity
- A fragment can be sent arguments via a bundle that can be retrieved via the `getArguments` method.
- Use this in place of the `R.string.dialog_fire_missiles` string in the `setMessage` method – define an appropriate key (e.g. `MESSAGE_KEY`) in the fragment class for use with the bundle

Replacing Toast in AddSong with Dialog

(See Develop -> API Guides -> User Interface -> Dialogs)

- Copy the code under the “Showing a Dialog” section
- Make a bundle for the message (such as “Name and artist are required”) and send it via the fragment’s `setArguments(Bundle)` method
- Use `getFragmentManager` instead of `getSupportFragmentManager` in the call to the show method of the `DialogFragment`

Try out the dialog!

Part 4:
Showing/Editing a Song with `AddSong` activity

Modifying `AddSong` to show a song/ accept updates

- If the user clicks on a song in the song list, the `AddSong` activity is launched, populating the fields with the song info - This allows the user to edit the song if they wish

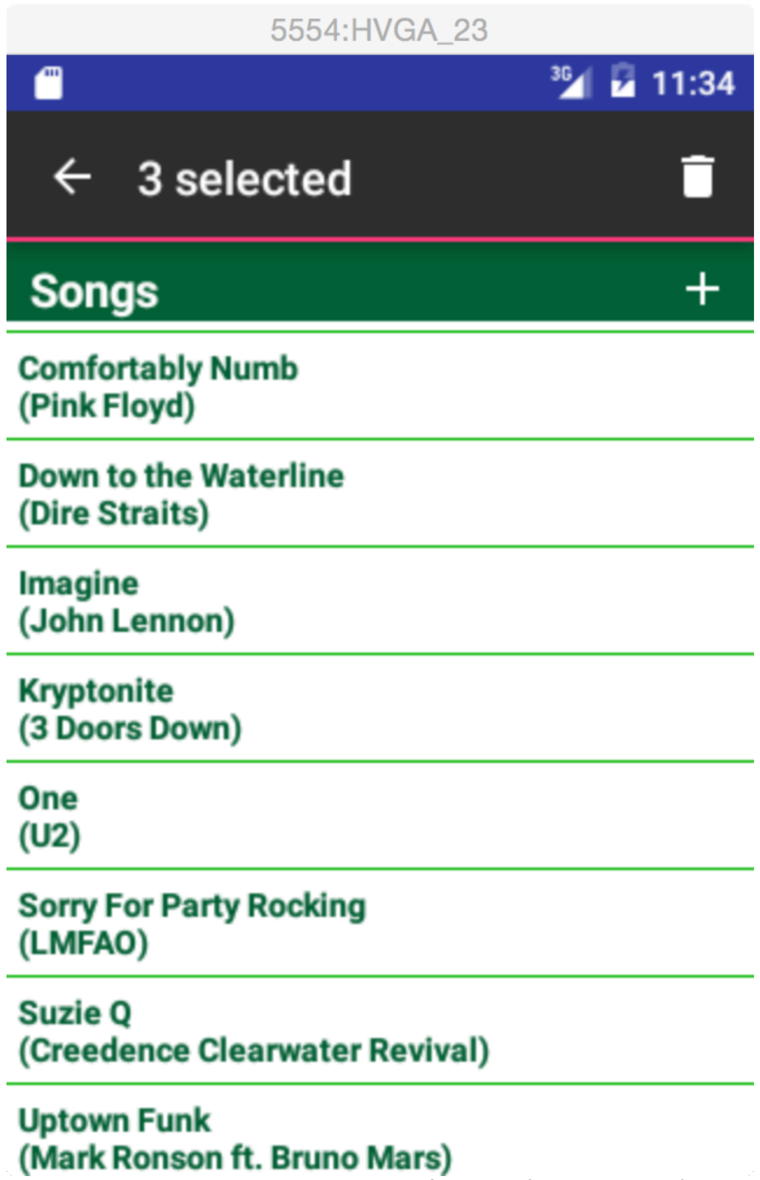
- Since `AddSong` now also permits editing, it needs to accept song info to populate the text fields:
    - Define key for song ID: ID is needed because user might change song name itself)
    - In `onCreate`, check if a `Bundle` is present (i.e. `getIntent().getExtras`() is not null) – if so, get song info and populate the text fields (`Bundle` will be sent by `SongLib` to show a song)
    - In the `save` method, add ID to returned bundle

Modifying `SongLib` to launch `AddSong` for showing/updating

- Define a method called `showSong` that is called when a user clicks on a song (in listener for list view items)
- In `showSong`, launch `AddSong` with a code for editing, sending in all the song info in a `Bundle`
- In the `onActivityResult` callback method, if the request code is found to be the edit code, save the updated song info in the song list data structure, and redo the `ListView` adapter

Part 5:
Allowing multiple deletes of songs in list with Contextual Action Mode

Preview of Contextual Action Mode



5554:HVGA_23

3 selected

Songs

Comfortably Numb
(Pink Floyd)

Down to the Waterline
(Dire Straits)

Imagine
(John Lennon)

Kryptonite
(3 Doors Down)

One
(U2)

Sorry For Party Rocking
(LMFAO)

Suzie Q
(Creedence Clearwater Revival)

Uptown Funk
(Mark Ronson ft. Bruno Mars)

Contextual Action Bar (CAB)

Get and install delete (trash can) icon

Get the delete icon (trash can, white version) – it's under the "Action" set of icons at Design: Material Icon Collection (https://design.google.com/icons/index.html)

Install into the various `res/drawable-<density>` folders, name the icon `ic_action_delete`

Create a Menu Resource

(See API Guides -> User Interface -> Menus -> Defining a menu in XML)

- Create a folder call `menu` under `res`
- In this folder, create a menu resource file called delete_menu.xml, with the following code:

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

Text version, in strings.xml

Delete icon

Set up the contextual action mode

(See API Guides -> User Interface -> Menus -> Using the contextual action mode)

Look under “Enabling batch contextual actions in a ListView or GridView”

Copy the code that is listed there
(except `listView = getListView()`)
into `SongLib.java` at the bottom of the `onCreate` method

`AbsListView.MultiChoiceModeListener` is an abstract class that is being subclassed here. All abstract methods must be Implemented, which is why they are all listed in the class – make modifications as follows for our objectives.

onCreateActionMode

This method is called when a user long-presses on a list item.
It uses a provided menu layout and "inflates" it (makes an object for it), for use as the contextual action bar

```
MenuInflater inflater = mode.getMenuInflater();
inflater.inflate(R.menu.delete_menu, menu);
return true;
```

onItemCheckStateChanged

This method is called when any item is selected/deselected.
We don't really need to do anything here for the app to work correctly, but just for fun, we are going to count the number of selected
items

```
mode.setTitle(listView.getCheckedItemCount() +
                    " selected");
```

onActionItemClicked

This method is called when the user takes an action on the selected items – in our case, clicking on the delete icon

```
switch (menuItem.getItemId()) {
    case R.id.menu_delete:

        deleteSelectedItems();

        actionMode.finish();

        return true;
    default:
        return false;
}
```

program id of the delete menu item

Will implement next

All done, take down the contextual action bar

deleteSelectedItems

This is our method to do all the deletions

```
SparseBooleanArray arr = listView.getCheckedItemPositions();

// gather songs in a to-delete list
ArrayList<Song> deleteSongs = new ArrayList<Song>();
for (int i=0; i < arr.size(); i++) {
    if (arr.valueAt(i)) {
        Song song = (Song)listView.getItemAtPosition(arr.keyAt(i));
        deleteSongs.add(song);
    }
}
for (Song song: deleteSongs) {
    myList.remove(song);
}
listView.setAdapter(new ArrayAdapter<Song>(
               this, R.layout.song, myList.getSongs()));
```

Tutorial on Using Contextual Action Mode

http://tinyurl.com/jcvzusj