CS 213 Spring 2016
Lecture 24: April 14

Song Library Android App

Search Functionality

Part 9:

Implementing Search Using the Android Search Framework

Android Search Framework

See Develop -> API Guides -> User Interface -> Search

There are two alternatives to providing a search function UI:

- A search dialog at the top of the screen
- OR, a search widget that is embedded in the action bar

We'll go with the search dialog at the top of the screen

Implementing the Search Function

1. Implement the search logic

The first thing to do is to add functionality to the app that will enable searching on songs.

The user can search for all songs that start with a prefix string.

Since the `MySongList` class maintains the song list, it would be appropriate to code the search functionality in that class: a method that would return a range of indices in the list of songs whose names start with the query prefix

Implementing the Search Function

1.
```
public int[] search(String query) {
   int lo=0, hi=songs.size()-1;
   int[] extent;
   String song = query.toLowerCase();
   while (lo <= hi) {
      int mid=(lo+hi)/2;
      if (songs.get(mid).name.toLowerCase().startsWith(song)) {
         // need to scan left and right of mid for all matches
         return getExtent(mid, song);
      }
      // mid does not start with the given name, go left or right
      int c = query.compareToIgnoreCase(songs.get(mid).name);
      if (c < 0) {
         hi = mid-1;
      } else {
         lo = mid+1;
      }
   }

   return null;
}
```

Implementing the Search Function

1.

```
private int[] getExtent(int mid, String song) {
   int[] extent = new int[2];
   extent[0] = mid;
   // scan left
   while (extent[0] > 0) {
      if (songs.get(extent[0]-1).name.toLowerCase().startsWith(
            song)) {
         extent[0]--;
      } else { break; }
   }
   // scan right
   extent[1] = mid;
   while (extent[1] < songs.size()-1) {
      if (songs.get(extent[1]+1).name.toLowerCase().startsWith(
            song)) {
         extent[1]++;
      } else { break; }
   }
   return extent;
}
```

Implementing the Search Function

2. Create a searchable configuration

To plug the search function into the Android search framework, we have to make what's called a searchable configuration, which is basically an xml file that should be placed in a directory called `xml` under `res`. (This is not a standard directory that's created with the project) Here's a sample file, called `searchable.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name"
    android:hint="@string/search_hint" >
</searchable>
```

In `strings.xml`:

```
<string name="search_hint">Search Song</string>
```

Implementing the Search Function

3. Creating/declaring a searchable activity

Next, one activity in the app must be declared to be searchable, which will have code to process the search query.
In our app, we will make `SongLib` the searchable activity – this is done by defining an intent-filter tag for `SongLib` in the manifest:

```
<intent-filter>
     <action android:name="android.intent.action.SEARCH" />
</intent-filter>
<meta-data android:name="android.app.searchable"
           android:resource="@xml/searchable"/>
```

Implementing the Search Function

4. Using a Search Widget as Action View in Action Bar

The search icon in the Action Bar is used as the "search widget", with the following modification to the `add_menu.xml` file:

```
<item android:id="@+id/action_search"
    android:icon="@drawable/ic_action_search_black"
    android:title="@string/action_search"
    appcompat:showAsAction="collapseActionView|always"
    appcompat:actionViewClass="android.support.v7.widget.SearchView" />
```

See
Develop->Training->Best Practices for User Interface->Adding the App Bar->
Action View and Action Providers

http://developer.android.com/training/appbar/action-views.html

Implementing the Search Function

5. Associating the Searchable Configuration (XML) with the Search View

   This is done by overriding the `onCreateOptionsMenu` method in `SongLib`:

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

    // Get the SearchView and set the searchable configuration
    SearchManager searchManager =
       (SearchManager) getSystemService(Context.SEARCH_SERVICE);
    SearchView searchView =
       (SearchView) menu.findItem(R.id.action_search).getActionView();
    // Assumes current activity is the searchable activity
    searchView.setSearchableInfo(searchManager
              .getSearchableInfo(getComponentName()));
    // Do not iconify the widget; expand it by default
    searchView.setIconifiedByDefault(false);

    return super.onCreateOptionsMenu(menu);
}
```

Implementing the Search Function

6. Running `SongLib` in `singleTop` mode

When search is activated, a new instance of `SongLib` would be created and launched (since is declared in the manifest as the activity to call when a search is done.

But it's a waste to have a new instance of an activity for every search request. Setting the activity to run in `singleTop` mode makes sure the same instance of SongLib that is on the top of the activity stack is used for the `search` as well:

```
<activity android:name=".SongLib"
          android:launchMode="singleTop">
```

Implementing the Search Function

7. Coding `SongLib` to work with regular and search intents

Since `SongLib` will also act as the target of a search request, there needs to be a way to distinguish between two intents.

First, override the `onNewIntent` method:

```
protected void onNewIntent(Intent intent) {
    setIntent(intent);
    handleIntent(intent);
}
```

Then, implement the `handleIntent` method:

```
private void handleIntent(Intent intent) {
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      String query =
           intent.getStringExtra(SearchManager.QUERY);
      showSearchResults(query);
    } else {
      showSongList();
    }
}
```

Implementing the Search Function

8. Refactoring the original intent (show song list) code

From `onCreate`, move the block of code that sets list adapter and list item selection listener, into the `showSongList` method:

```
private void showSongList() {
    listView.setAdapter(
       new ArrayAdapter<Song>(this, R.layout.song,myList.getSongs()));
    listView.setSelection(lastPickedIndex);
    listView.setOnItemClickListener(
         new AdapterView.OnItemClickListener() {
             @Override
             public void onItemClick(AdapterView<?> adapterView,
                        View view, int i, long l) {
                        showSong(i);
             }
         }
     );
}
```

index of song selected on search, instance field defaults to zero

Call `handleIntent(getIntent())` in place of the moved code block

Implementing the Search Function

9. Implementing `showSearchResults`

```
private void showSearchResults(String query) {
    int[] extent = myList.search(query);
    if (extent == null || extent.length == 0) {  // no matches
        String msg = getString(R.string.search_empty, new Object[] {query});
        Toast.makeText(this,msg,Toast.LENGTH_SHORT).show();
        return;
    }

    makeResultList(extent);

    searchListStartPos = extent[0];
    listView.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                public void onItemClick(AdapterView<?> parent,
                                        View view, int position,
                                        long id) {
                    lastPickedIndex =
                            SongLib.this.searchListStartPos+position;
                    showSongList();
                }
            });
}
```

No match found for \"%s\"

Define instance field

listener for search results list

Implementing the Search Function

9. Implementing `showSearchResults` - `makeResultList`

```
private void makeResultList(int[] extent) {
    int count = extent[1] - extent[0] + 1;
    String[] matches = new String[count];
    ArrayList<Song> songs = myList.getSongs();
    for (int i=0; i < matches.length; i++) {
        matches[i] = songs.get(extent[0]+i).name;
    }
    listView.setAdapter(
            new ArrayAdapter<String>(this, R.layout.song, matches));
}
```