ES3 Lab 6

Android Services

This lab

- Services
  - Starting services
  - Toasts
  - Defining AIDL interface
  - Binding to services
- AppWidget
  - Creating a homescreen AppWidget
  - Binding to a service

ASSESSED!

- **THIS IS THE ASSESSED EXERCISE FOR THE Android SECTION OF THE COURSE!**

Hand in two weeks from today

email me your source code

Individual work -- don't copy someone elses!

Feel free to use any online tutorials or references to learn more BUT don't just copy and paste code in

You will need to refer to the API docs and/or other resources *extensively*

Exercise

- Extend your music player
  - Add a progress bar
  - Make it play in the background using a Service
  - Make it controllable from the UI still by binding to it and sending messages
- Add notification messages when the track changes
- Create an AppWidget on the home screen
  - shows the current playing track, track length and current progress
  - make it update using information from the Service

Guide

- You will get at least a passing mark if you manage to implement basic background music playing with simple controls working and notifications for tracks changing

- You will get full credit if:
  - background music playing works
  - forward/previous/pause/stop/play all work
  - current playing track is highlighted in the UI
  - a notification appears every time the track changes
  - there is an AppWidget which shows current track and progress

Finishing the music player

- Your music player from the last lab should have all the following functionality working (if not, finish it!):
  - Play/pause/prev/next/stop
  - Auto-track advance at the end of a track
  - Shuffle toggle
  - Current track highlight

- Also, add a progress bar to the UI
  - Link it's value to the current progress through the track
  - Use android.os.Handler to implement a timer to update the track
    - hint: you need to use removeCallbacks() and postDelayed() methods
    - you will need to use postDelayed() at the end of each callback to keep the timer repeating...

Creating a blank Service

- To create a Service:
  - register it in the **AndroidManifest.xml**
    - entry should look like **<service android:name=".musicservice.MusicService"/>**
  - Create a new class file called **MusicService**
    - make sure it's package definition at the top is com.es3.labs.lab1.musicservice; or whatever matches the package definition in your main Activity
  - This class must inherit from **Service**
    - You must override at least **onCreate**() and **onDestroy**()
    - Add a simple **Toast** notification to **onCreate**() to show the **Service** has been started

Starting the Service

- In your main **Activity**
  - start the Service in **onCreate**
    - get an **Intent** with

```
Intent serviceIntent = new Intent(this, MusicService.class);
```

    - send it with **startService(serviceIntent)**
      - this method is available in **Activity** itself

- **Build this, check that the notification appears!**

- Add an **onDestroy** method to your main Activity
  - generate an Intent in the same way, and use **stopService** to stop it

Moving playback to a Service

- Add some code to the **Service** to start playing back a fixed track
  - make it start as soon as the **Service** is created
  - (remember to call **super.onCreate**() in **onCreate**()!)
- Make the track stop in **onDestroy**()
- Build it, test that the music starts and stops correctly when the Service is started and stopped

Showing track change notifications

- Now move the playlist loading code to the Service and make the tracks autoadvance when they finish playing
  - Don't worry about pause/stop/prev/next or anything else yet
  - Just get it to play tracks in sequence in the **Service**
- When a track ends, use **Toast** to show a notification giving the name of the next track playing
- **Build it, and you should have a working background music player!**

Next Steps

- To finish the project:
  - Construct an AIDL interface for track controls and to return currently playing track and progress
  - Implement the methods for the AIDL interface as the stub
  - Return the interface in the **onBind**() method of your **Service**
  - Bind to the **Service** in the activity
  - Make UI requests call the appropriate methods of the Service *interface*
- Create an **AppWidget**
  - This will show the currently playing track, length and progress, along with a pause button
  - Make this also connect to the Service
  - Add it to the home screen

Define an AIDL file

- Read developer.android.com/guide/developing/tools/aidl.html
- Use Eclipse to add an AIDL file called IMusicPlayer to the project (in src/)
  - The AIDL file must specify the methods that will be called
  - You will need play/pause, stop, prev and next methods
  - A method to get the current track name
  - A method to get the current track progress
  - **You will only need to use primitive types (String, double)**
- **When you compile, a matching Java interface file should be generated automatically**

Implement the stub

- You must now create a subclass of **IMusicPlayer.stub** which actually implements the methods
  - make it a public inner class of the Service, so that it can access the methods and variables you need
  - call it something like **MusicPlayerServiceStub**
- In your **Service** implementation, create an instance of **MusicPlayerServiceStub** in **onCreate()**
- Override **onBind**() and make it return the stub

Bind to the Service

- In your **Activity**, after starting the Service, bind to it with **bindService**
- **You will need to create a subclass of ServiceConnection to deal with the callbacks**
  - **see the documentation for how to do this**
- When **onServiceConnected()** is called, you must store the reference to the IBinder object (this is the remote interface)
  - **MusicPlayer.stub.asInterface(service)** will change it to an instance of **MusicServicePlayerStub**
- **Now, make the UI controls call methods on the MusicPlayerServiceStub object**
  - **i.e. transport controls, highlight, progress bar**
- **Build and test!**

AppWidgets

- AppWidgets allow certain limited UI components to be included in other applications
  - most importantly, the home screen

- Read *developer.android.com/guide/topics/appwidgets/index.html*
- Read *developer.com/ws/article.php/3833306/Creating-a-Home-Screen-App-Widget-on-Android.htm*
- Read *android-developers.blogspot.com/2009/04/introducing-home-screen-widgets-and.htm*

- **Important things to note**
  - Only a few UI components are available
  - You will have to implement a RemoteView
  - You will need an XML layout for the AppWidget
  - You will need to have the AppWidget update relatively frequently