ES3 Lab 5

Android development

This Lab

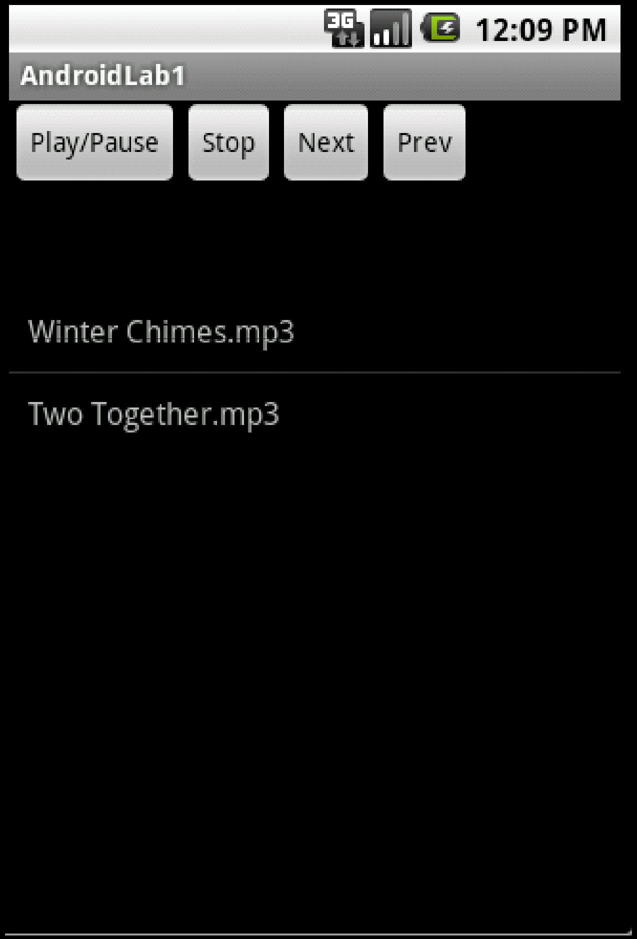
- Create a simple Android interface
- Use XML interface layouts
- Access the filesystem
- Play media files

- Info about Android development can be found at http://*developer.android.com/index.html*

- The Javadoc SDK can be found at http://*developer.android.com/reference/packages.html*

Assignment

- Create a basic MP3 media player



Installing Eclipse and ADT

- Eclipse and ADT aren't installed on the lab machines
  - Installers are provided
    - Install Eclipse JEE x64 for OS X (eclipse-j...tar.gz)
      - expand it to **eclipse**/ your home directory
    - Install the Android SDK
      - expand to **android**/ in your home directory (android-sdk...zip)
- Open eclipse, go to Help/Install new software...
  - Choose Add.. and enter **Android** for the name and the **adt-0.9.5.zip** file for the Archive
  - Install...
- Check the box by **Developer Tools**
  - Click next, and use the default options (DDMS and development tools)
  - Restart Eclipse

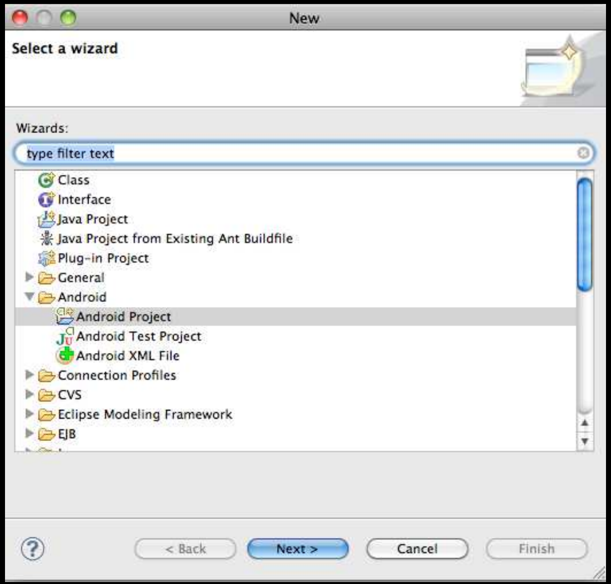
Create an emulator image

- Go to **android/tools** and run **android**
  - In the Virtual Devices tab, click New,
  - Call it **DefaultAVD**
  - **Use platform 2.1 (API level 7)**
  - Use a 1024Mb SD card
  - Use the default (HVGA) skin
  - Click **Create AVD**

- **Start up Eclipse**

Getting Started

- Create a new project **New/Other/Android/Android Project**
    - Call it **AndroidLab1**
    - Make it target **Android 2.1**
    - Fill in application name: **AndroidLab1**
    - Package name: com.es3.labs.**AndroidLab1**
    - Make sure **Create activity** is ticked, and call it **StartActivity**



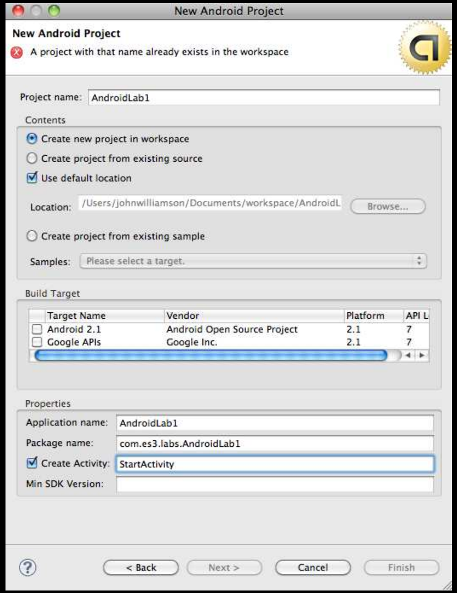
New

Select a wizard

Wizards:

type filter text

- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- Android
    - Android Project
    - Android Test Project
    - Android XML File
- Connection Profiles
- CVS
- Eclipse Modeling Framework
- EJB



New Android Project

New Android Project

A project with that name already exists in the workspace

Project name: AndroidLab1

Contents

- [✓ Create new project in workspace]
- Create project from existing source
- [✓] Use default location

Location: /Users/johnwilliamson/Documents/workspace/AndroidL

- Create project from existing sample

Samples: Please select a target.

Build Target

Target Name	Vendor	Platform	API L
☐ Android 2.1	Android Open Source Project	2.1	7
☐ Google APIs	Google Inc.	2.1	7

Properties

Application name: AndroidLab1

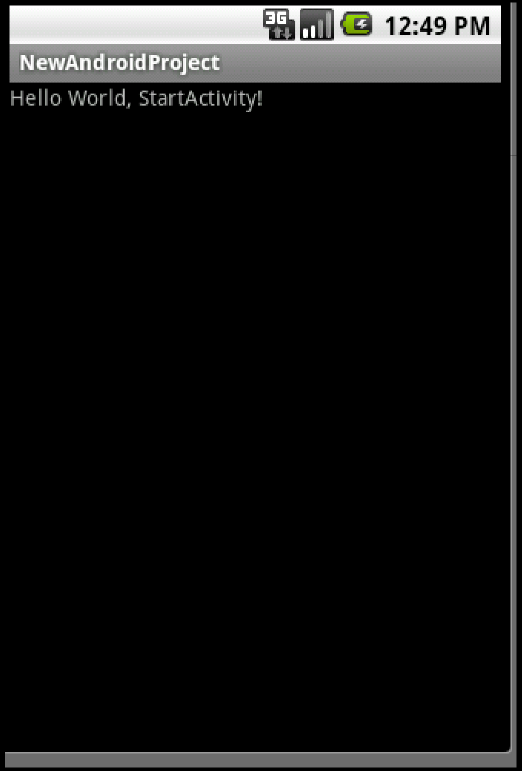
Package name: com.es3.labs.AndroidLab1

[✓] Create Activity: StartActivity

Min SDK Version:

Check it works

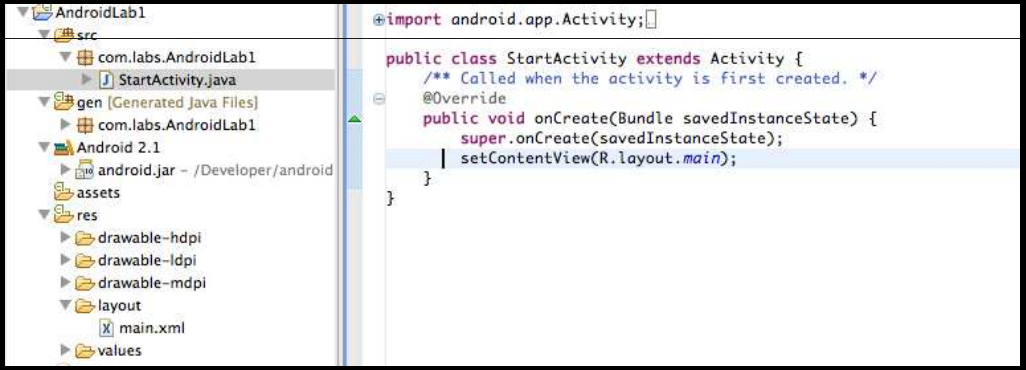
- Go to **Run/Run...** and choose run as **Android application**
  - After some time, this should appear:



Hello World, StartActivity!

The Layout

- Expand **src** and then **com.es3.labs.AndroidLab1**
  - Look at **StartActivity.java**
  - This is where the entry point for the application will be
  - Note that **onCreate** calls **setContentView** on a R.layout.main

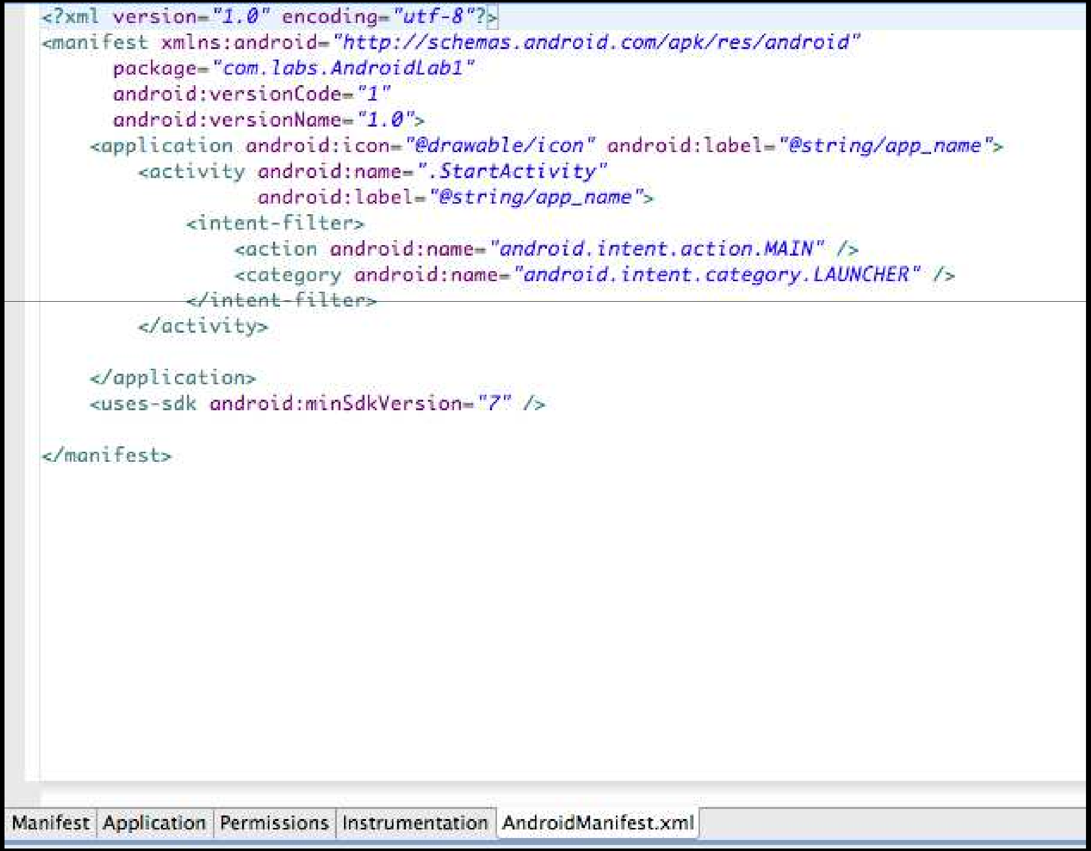


```
import android.app.Activity;

public class StartActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

- You can find the definition for this layout by expanding **res/layout** then opening **main.xml (**choose the **main.xml** tab)

main.xml



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.labs.AndroidLab1"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".StartActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="7" />

</manifest>
```

Default Manifest

- Have a look at **res/AndroidManifest.xml**
  - Choose the AndroidManifest.xml tab at the bottom to actually see the XML
  - Note the <**activity>** element and the <**intent-filter**> element within it
  - This filter marks that the **StartActivity** activity will receive the **MAIN** action intent and has category **LAUNCHER**
    - i.e. makes it the **entry point**

Adding an XML layout

- Go to the Layout tab of main.xml and right click the **TextView**
  - choose remove and remove it and the **LinearLayout** containing it

- Drag a new **LinearLayout** onto the blank canvas
  - Warning: the Eclipse UI preview is very buggy...

Adding some buttons

- Drag on four new **Button** instances
  - Go to the main.xml tab and manually edit the text attribute so they are **Play/Pause**, **Stop**, **Next** and **Prev**
  - Change the **id** attribute so the buttons are **@+id/PlayButton** etc.
- Click on **StartActivity.java** (important!) then do **Run/Run...**

Responding to button pushes

- Make **StartActivity** implement **OnClickListener**
  - add **import android.view.View.OnClickListener** to the top of **StartActivity.java**
  - and **import android.view.***

- You need to implement the method **onClick**

```
public void onClick(View v)
{

}
```

  - This gets passed the view that was clicked
  - You can get the id of a view with **getId**()

- Test each button to see if the id matches the view's id
  - Don't do anything in the blocks yet!

```
if(v.getId() == R.id.StopButton) { }
if(v.getId() == R.id.PlayButton) { }
// etc...
```

Adding the listeners

- For each button
  - Look up the Button instance using **findViewById()**
    - e.g. **findViewById(R.id.PlayButton**)
  - Add the listener to it using **setOnClickListener**

- **Now the listener will be called when the button is pushed**

Adding audio playback

- We need audio playback support
  - this is in **android.media.***
  - import this

- Create an instance variable in **StartActivity** of type **MediaPlayer**
  - Instantiate it in **onCreate**()

```
player = new MediaPlayer();
```

- in **onCreate**(), we need to load all the available MP3 files
  - first we list all available files
  - then we identify MP3 files
  - we add these to a list

Listing available files

- Create an instance method of **StartActivity** called **listAvailableMP3s()**

- **File** objects are used to access file system info (imported from **java.io.File**)
  - In **listAvailableMP3s**, create a new **File** object with "/media" as the path
    - **This is the Android path for media files like videos and music**
  - the **listFiles** method lists files in a directory (returns an array of **File[]**)
    - note: you only want MP3 files, so you'll probably want to create FIlenameFilter class to filter the files read (see the API docs)
  - check each element is an actual file (not a directory) with **isFile**()
  - Return the files and store them in a class instance
  - Copy the file *names* into a second array (this will be shown onscreen)

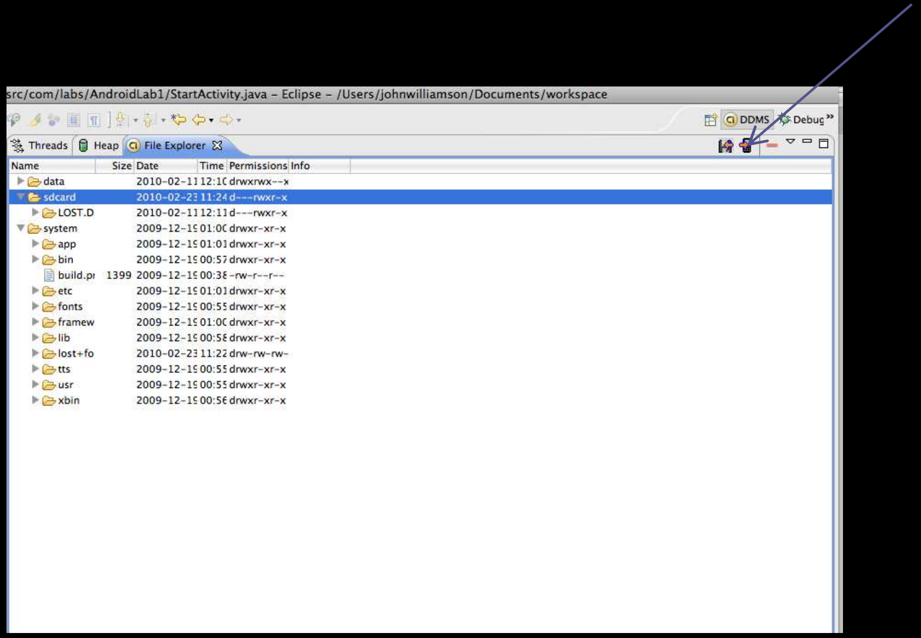
- This is the track list for the player

Copying files onto the device

- Obviously the device doesn't actually *have* any mp3 files yet
  - Get some (use your own or find some royalty-free music)

- Copy files over by going to **Window/Open perspective**...
  - choose **Other.../DDMS**
  - use the to phone button (tiny button at top-right, with an arrow pointing onto the phone)

- Copy the files to **/sdcard**


src/com/labs/AndroidLab1/StartActivity.java – Eclipse – /Users/johnwilliamson/Documents/workspace

Name	Size	Date	Time	Permissions	Info
data		2010-02-11	12:10	drwxrwx--x	
sdcard		2010-02-23	11:24	d---rwxr-x	
LOST.D		2010-02-11	12:11	d---rwxr-x	
system		2009-12-19	01:00	drwxr-xr-x	
app		2009-12-19	01:01	drwxr-xr-x	
bin		2009-12-19	00:57	drwxr-xr-x	
build.pr	1399	2009-12-19	00:38	-rw-r--r--	
etc		2009-12-19	01:01	drwxr-xr-x	
fonts		2009-12-19	00:55	drwxr-xr-x	
framew		2009-12-19	01:00	drwxr-xr-x	
lib		2009-12-19	00:58	drwxr-xr-x	
lost+fo		2010-02-23	11:22	drw-rw-rw-	
tts		2009-12-19	00:55	drwxr-xr-x	
usr		2009-12-19	00:55	drwxr-xr-x	
xbin		2009-12-19	00:56	drwxr-xr-x	

Media player usage

- Set the data source, prepare the media player, and begin playback

```
player.setDataSource(path);
player.prepare();
player.start();
```

- Pause with **player.pause**() and stop with **player.stop**()

Pause, Stop, Previous, Next

- Add a boolean variable to represent the play/pause state
  - e.g. isPlaying
  - it should initially **false**
  - Make it toggle when the play/pause button is pushed
    - If it's False, start playing (as above), make it **false**
    - If it's True, then call **player.pause**(), make it **true**

- Similarly, if the stop button is pressed, call **player.stop**()

- Create a variable to represent the current track index
  - For previous, decrement the track index, stop the current file, play the next file
    - if track index<0 make track index the last file
  - And similarly for next

- Test it!
  - You should have a fully functioning (if limited) media player!

Track view

- Go to the main layout (**res/layout/main.xml**)
  - Edit the XML directly and add a new **LinearLayout** around the whole thing
    - You can copy and paste the existing **LinearLayout**, but remember to change the **ID**!
  - Set the layout's **orientation** attribute to **vertical**

```
<LinearLayout android:orientation="vertical" android:id="...
```

- After the first **LinearLayout** is closed add a <ListView> element
  - set its **id** attribute to @+**id/TrackView**
  - Set the layout_height of the inner LinearLayout to "**100px**" instead of "**fill_parent**"
- **The rough XML structure should look like this**

```
<LinearLayout vertical>
            <LinearLayout horizontal>
                        <Button play>
                        <Button stop>
                        <Button prev>
                        <Button next>
            </LinearLayout>
            <ListView>
</LinearLayout>
```

Track View

- Each element of the **ListView** must be a View
  - Conventionally a **TextView**

- Configure the appearance of each of the rows by creating an XML file to represent the layout of *one row*

- Go to **res/layout** and right-click, **New**..
  - Choose other, Android XML
  - Set the file to track.xml
  - Make the root element a **TextView** (drop down at the bottom)
  - Click **Finish**

- Edit the generated **XML** file and change the **layout_width** attribute to "**fill_parent**" so that the list elements extend across the whole screen

Accessing the Track View

- Import **android.widget.***

- In **onCreate()** get hold of the **ListView** reference using **findViewById** on **R.id.TrackView**
  - You'll need to cast the result to **ListView**

- Link the **ListView** to the track array
  - **ListViews** use **ListAdapaters** to connect data to the list
  - We want to use an **ArrayAdapter**
    - Takes as arguments a context (**this**), the text object to use for each row (**R.layout.track**), and the array to use

```
trackList.setAdapter(new ArrayAdapter<String>(this, R.layout.track, tracks.toArray());
```

- Add a method **updateTrackView** which uses **setSelection** to match the **ListView's** selection to the current track index
  - Call it in **onCreate**(), and after the track index is updated when the buttons are pressed

Highlighting the current track

- We want to highlight the track that is currently playing
  - Do this by setting the **TextView's** color
    - You can get the child **TextView's** in the **ListView** by using **getChildAt(index**)
  - Get all **TextView** instances from the **ListView**
    - set their colors to gray
    - use **setTextColor**()
  - Get the current **TextView** instance from the **ListView**
    - Set the color of this to white

Extra functionality

- Make the media player automatically go to the next file when it finishes (look at the MediaPlayer methods to see how to do this)

- Add a shuffle mode, with a button to toggle it
  - This means that next and the auto-advance when a track finishes should go to a random track
  - But prev should go to the previous track
    - keep a list of previous tracks!

- Allow the user to tap on the list to select a track
  - add a listener to the list, make it set the track index and start playing the new track