**121`**

ITMD 455/555 *Intelligent Device Applications* Lab 4

#### SAX/XML App- 50 points

**Introduction**. For this lab create a Login screen and do some snazzy SAX xml parsing!

**STEP 1 Creating a New Android Project**

Create a new project called **Parser**.

**STEP 2 Creating your first screen. The first screen will contain a login script you will create where a successful login (3 attempts only!) will get the user into a MainActivity screen which will serve as your XML dump eventually.**

Create your first activity as an “Empty Activity” and call it **LoginActivity**. Use the default layout name given.

**STEP 3 Working your layout.**

Add in necessary elements into your layout to allow for a username and

password to be accepted. Password field must be protected. Include a button as well to allow the user to Submit their credentials, to be accepted or not!

**STEP 4 Coding your file.**

Allow the user up to three tries to input their correct credentials. Each incorrect response to either a bad username and/or password should display a long toast message stating a message such as “Wrong Credentials”. Include also how many attempts are left in your message.

If the user has successfully input their correct credentials show a long toast message as “Redirecting…” to the screen. Upon successfully logging in, finish your current activity and be able to start a **MainActivity** screen (to be set up next) so its visible to the user.

Be thematic with your app for this project so include a nice subtle color for all of your activity screens!

**STEP 5 Adding in a new Empty Activity to your src folder and call it MainActivity.**

Include as your layout name activity\_main. Do not enable the Launcher Activity checkbox as option you go thru your wizard. Just press finish when your complete. (Notice the new activity now has been added to your manifest automatically when you created your new Activity file).

**STEP 6 Run your app and try to pass the login screen.**

If successful, you should see a blank Activity screen.

**Snapshot your Login screen showing an incorrect attempt message via your Toast widget that will display “Wrong Credentials…” here and then for a**

**successful attempt widget will display the “Redirecting…” message). Paste your**

**screenshots into a Word file and *label* your screenshots Login 1- bad attempt,**

**Login-2 Successful attempt.**

[Now that you have your LoginActivity screen working, it may be a very *wise* time saver to copy your Intent code block and place it into your OnCreate method under perhaps the SetContentView() method call so you don’t have to login everytime when you run your app. This will come in handy when you run further tests over and over again below.

Alternatively you can always adjust your manifest file to spark up your main activity as the launcher activity versus the login activity. When you turn in your project files to BB you can reset your settings back to normal operations.]

**STEP 7 Creating the logic to parse XML data.**

You will need to add in some class files to assist you parsing your xml data. Let’s add in our first class file. Create a class file called XMLGettersSetters. Code your class as

follows (make sure to include the following import statements):

**import** java.util.ArrayList;

**import** android.util.Log;

/\*This class contains all getter and setter methods to set and retrieve data.\*/

**public** **class** XMLGettersSetters {

**private** ArrayList<String> title = **new** ArrayList<String>();

**private** ArrayList<String> artist = **new** ArrayList<String>();

**private** ArrayList<String> country = **new** ArrayList<String>();

**private** ArrayList<String> company = **new** ArrayList<String>();

**private** ArrayList<String> price = **new** ArrayList<String>();

**private** ArrayList<String> year = **new** ArrayList<String>();

**public** ArrayList<String> getCompany() {

**return** company;

}

**public** **void** setCompany(String company) {

**this**.company.add(company);

Log.*i*("This is the company:", company);

}

**public** ArrayList<String> getPrice() {

**return** price;

}

**public** **void** setPrice(String price) {

**this**.price.add(price);

Log.*i*("This is the price:", price);

}

**public** ArrayList<String> getYear() {

**return** year;

}

**public** **void** setYear(String year) {

**this**.year.add(year);

Log.*i*("This is the year:", year);

}

**public** ArrayList<String> getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title.add(title);

Log.*i*("This is the title:", title);

}

**public** ArrayList<String> getArtist() {

**return** artist;

}

**public** **void** setArtist(String artist) {

**this**.artist.add(artist);

Log.*i*("This is the artist:", artist);

}

**public** ArrayList<String> getCountry() {

**return** country;

}

**public** **void** setCountry(String country) {

**this**.country.add(country);

Log.*i*("This is the country:", country);

}

}

Notice the file uses “getter” and “setter” type functions to handle SAX/XML processing as you will see evident in the next class file and in your MainActivity file. You will tweak this file in a later step. Further notice the use of Log.*i* methods in your code. The i stands for the info mode that is displayed from your Logcat file onto your screen. Logcat will serve as a necessity many times in determining what is *running* in the background across your project and any possible errors that may occur as well!

**STEP 8 Add in a handler that will take XML data allow for data to be stored via your**

**“setter” type methods in the arraylists just created in your XMLGetterSetter class file via a XMLGetterSetter object (called data).**

Add in yet another class file to your java source folder called XMLHandler. Add in the

following code to your class file and make sure to include the following import

statements.

**import** org.xml.sax.Attributes;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.helpers.DefaultHandler;

**public** **class** XMLHandler **extends** DefaultHandler {

String elementValue = **null**;

Boolean elementOn = **false**;

**public** **static** XMLGettersSetters *data* = **null**;

**public** **static** XMLGettersSetters getXMLData() {

**return** *data*;

}

**public** **static** **void** setXMLData(XMLGettersSetters data) {

XMLHandler.*data* = data;

}

/\*\*

\* This will be called when the tags of the XML starts.

\*\*/

@Override

**public** **void** startElement(String uri, String localName, String qName,

Attributes attributes) **throws** SAXException {

elementOn = **true**;

**if** (localName.equals("CATALOG"))

{

*data* = **new** XMLGettersSetters();

}

}

/\*\*

\* This will be called when the tags of the XML end.

\*\*/

@Override

**public** **void** endElement(String uri, String localName, String qName)

**throws** SAXException {

elementOn = **false**;

/\*\*

\* Sets the values after retrieving the values from the XML tags

\* \*/

**if** (localName.equalsIgnoreCase("title"))

*data*.setTitle(elementValue);

**else** **if** (localName.equalsIgnoreCase("artist"))

*data*.setArtist(elementValue);

**else** **if** (localName.equalsIgnoreCase("country"))

*data*.setCountry(elementValue);

**else** **if** (localName.equalsIgnoreCase("company"))

*data*.setCompany(elementValue);

**else** **if** (localName.equalsIgnoreCase("price"))

*data*.setPrice(elementValue);

**else** **if** (localName.equalsIgnoreCase("year"))

*data*.setYear(elementValue);

}

/\*\*

\* This is called to get the tags value

\*\*/

@Override

**public** **void** characters(**char**[] ch, **int** start, **int** length)

**throws** SAXException {

**if** (elementOn) {

elementValue = **new** String(ch, start, length);

elementOn = **false**;

}

}

}

Notice the code has the processing of a start element in your XML file (you’ll see this reasoning more in the next step) and an end element. The main root tag will be “CATALOG” which will serve as the documents ‘root’ element so the processing will know that is where all sub roots will be located automatically from your XML file. Notice the ending element code block is necessary to determine each elements ending point where the inner text of the element will ultimately be read and stored into your respective arraylists from title up through year via your setter methods.

**STEP 9 To finish up, let’s concentrate on working your MainActivity file and associated layout file (activity\_main.xml).**

Open up your activity\_main.xml file and include the following simple markup code:

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*

android:orientation=*"vertical"*

android:id=*"@+id/layout"*>

<TextView

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:textSize=*"15sp"*

android:gravity=*"center\_horizontal"*

android:id=*"@+id/layout\_string"*/>

</LinearLayout>

Textview will serve the purpose for a series of ViewGroups which will serve as on screen display views for each of your arraylists.

Next open up your MainActivity.java file and code it as follows:

**public class** MainActivity **extends** AppCompatActivity {  
  
 XMLGettersSetters **data**;  
  
 */\*\* Called when the activity is first created. \*/* @Override  
 **public void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
  
 **new** BackgroundTask().execute();  
 }  
  
 **public class** BackgroundTask **extends** AsyncTask<Void, Integer, Void> {  
  
 @Override  
 **protected void** onPreExecute() {  
 **super**.onPreExecute();  
 }  
  
 @Override  
 **protected void** onProgressUpdate(Integer... values) {  
 **super**.onProgressUpdate(values);  
 }  
  
 @Override  
 **protected void** onPostExecute(Void aVoid) {  
 **super**.onPostExecute(aVoid);  
 viewSaxData();  
 }  
  
 @Override  
 **protected** Void doInBackground(Void... params) {  
  
 **try** {  
 **synchronized** (**this**) {  
 saxParser();  
 }  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return null**;  
 }  
 }  
 **private void** saxParser() {  
  
 **try** {  
 */\* Create a new instance of the SAX parser \*/* SAXParserFactory saxPF = SAXParserFactory.*newInstance*();  
 SAXParser saxP = saxPF.newSAXParser();  
 XMLReader xmlR = saxP.getXMLReader();  
  
 *// URL of the XML* URL url = **new** URL(**"http://www.papademas.net:81/cd\_catalog3.xml"**);  
  
 */\* Create the Handler to handle each of the XML tags \*/* XMLHandler myXMLHandler = **new** XMLHandler();  
 xmlR.setContentHandler(myXMLHandler);  
 xmlR.parse(**new** InputSource(url.openStream()));  
  
 } **catch** (Exception e) {  
 System.***out***.println(e);  
 }  
  
 }  
 **private void** viewSaxData()  
 {  
 */\* Get the view of the layout within the main layout, so that we can  
 \* add TextViews.  
 \*/* View layout = findViewById(R.id.***layout***);  
  
 */\*Create TextView Arrays to add the retrieved data to \*/* TextView title[];  
 TextView artist[];  
 TextView country[];  
 TextView company[];  
 TextView price[];  
 TextView year[];  
  
 **data** = XMLHandler.*data*;  
  
 */\* Make the TextView array length the size of the arraylist \*/* title = **new** TextView[**data**.getTitle().size()];  
 artist = **new** TextView[**data**.getArtist().size()];  
 country = **new** TextView[**data**.getCountry().size()];  
 company = **new** TextView[**data**.getCompany().size()];  
 price = **new** TextView[**data**.getPrice().size()];  
 year = **new** TextView[**data**.getYear().size()];  
  
  
 */\*\*  
 \* Run a for loop to set All the TextViews with text until  
 \* the given size of the arraylist is reached.  
 \*\*/* **for** (**int** i = 0; i < **data**.getTitle().size(); i++) {  
  
 title[i] = **new** TextView(**this**);  
 title[i].setText(**"Title = "**+**data**.getTitle().get(i));  
  
 artist[i] = **new** TextView(**this**);  
 artist[i].setText(**"Artist = "**+**data**.getArtist().get(i));  
  
 country[i] = **new** TextView(**this**);  
 country[i].setText(**"Country = "**+**data**.getCountry().get(i));  
  
 company[i] = **new** TextView(**this**);  
 company[i].setText(**"Company = "**+**data**.getCompany().get(i));  
  
 price[i] = **new** TextView(**this**);  
 price[i].setText(**"Price = "**+**data**.getPrice().get(i));  
  
 year[i] = **new** TextView(**this**);  
 year[i].setText(**"Year = "**+**data**.getYear().get(i));  
  
 ((ViewGroup) layout).addView(title[i]);  
 ((ViewGroup) layout).addView(artist[i]);  
 ((ViewGroup) layout).addView(country[i]);  
 ((ViewGroup) layout).addView(company[i]);  
 ((ViewGroup) layout).addView(price[i]);  
 ((ViewGroup) layout).addView(year[i]);  
 }  
  
 setContentView(layout);  
 }  
}

As a double check make sure you have the following import statements at the top ☺

**import** android.os.AsyncTask;  
**import** android.os.Bundle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.TextView;  
  
**import** org.xml.sax.InputSource;  
**import** org.xml.sax.XMLReader;  
  
**import** java.net.URL;  
  
**import** javax.xml.parsers.SAXParser;  
**import** javax.xml.parsers.SAXParserFactory;

Notice your code creates a XMLGetterSetter data object to call the getter methods of your app to obtain data for each tag and place that data into a dedicated TextView which ultimately will be attached to its own ViewGroup.

One more thing before you run the app, place the following markup into your manifest file located in your manifests folder, otherwise you won’t connect to the web to retrieve file data.

-You can place the code ***above*** your beginning application tag...

<**uses-permission android:name="android.permission.INTERNET"**/>

Also in your manifest file, as of Android M (api 28), we need to include a line within your application tag the following line => **android:usesCleartextTraffic="true"** to allow for clear text to be read from an “outside” source.

Example:

<**application  
 android:usesCleartextTraffic="true"**

::

**STEP 10 Run your app.**

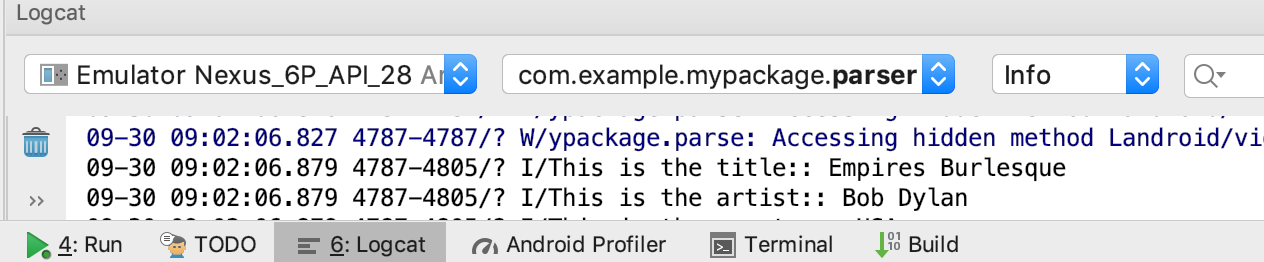
If successful, you should see a screen full of the XML file (cd\_catalog3.xml)

from the papademas.net server! Whoa!

**Snapshot your screen showing the XML data at this juncture and paste it**

**into Word. Label your snapshot XML data 1.**

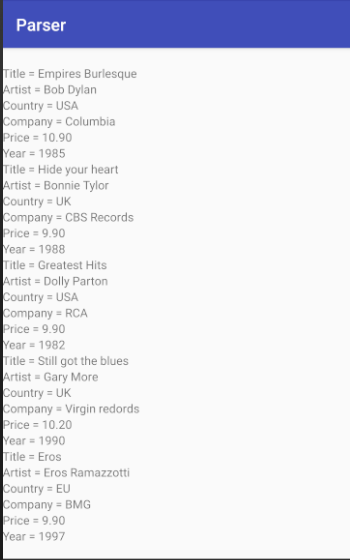
Notice a few things here. Locate your Logcat tab in your Android Monitor display area and select Log level as Info.



Now in your logcat view, scroll towards the bottom to view records pulled in from the XML

file as parsed. Do the records match the view on your MainActivity screen? The screen may be a little cropped at the bottom though depending on your emulator AVD display size. A ScrollView layout wrapped around your LinearLayout can help cure that in activity\_main!

Sample activity screen off a Nexus 6P API 26 emulator



**STEP 11. Now for the fun part. Modify any activity/class files you deem pertinent to show**

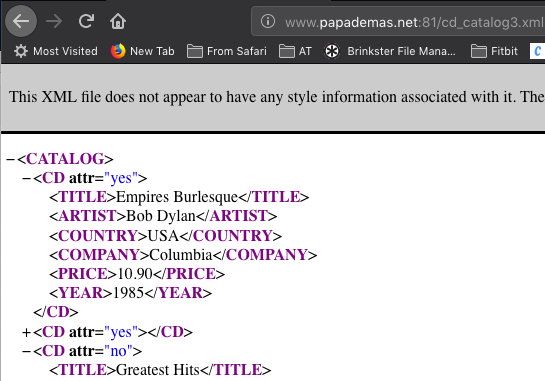
**EACH CD tag to the screen which actually has an *attribute* (named attr) attached**

**to it (please see the cd\_catalog3.xml file from the server url given). Each CD**

**attribute has either a value of yes or no to indicate whether a CD is sold out or**

**not.**

Snapshot showing attribute details from file follows.



Include appropriate getter and setter methods to allow for the CD attribute values to be

stored/displayed just like the other tags you have set up already with given getter and

setter methods.

An example of a setter method maybe something like the following that would be placed in the XMLGettersSetters file:

**public** **void** setAttribute(String attr) {

**this**.CD.add(attr);

Log.*i*("Sold out?: ", attr);

}

In your XMLHandler file you can then code (shown in bold below) the following

modification to read in CD data from the .xml file where there is an attribute exception

**if** (localName.equals("CATALOG")) {

*data* = **new** XMLGettersSetters();

}

**else if (localName.equals("CD")) {**

**try{**

**String attributeValue = attributes.getValue("attr");**

***data*.setAttribute(attributeValue);**

**}//end try**

**//catch error if thrown to avoid possible null pointer exception**

**catch (Exception e) {**

**Log.*i*("err on handler " , e.getMessage()); }**

**}**

**Oh, as a final note, make sure to include the words “Sold out” to the screen, with a corresponding yes or no value coming from the attribute for a given CD tag displayed with an appropriate response.**

**STEP 12 Run your app. If successful you should now see the CD tag added in as a**

**new TextView. If you have errors, check your LogCat file, that may help you out figuring some things out (run in verbose mode to see all information, warnings, errors, etc.).**

**Snapshot your screen showing the new XML data and paste it into Word.**

**Label your snapshot XML data 2.**

**STEP 13 Okay your not off the hook so fast! Modify your existing code to show**

**ONLY CD’s that are marked “Sold Out” meaning the attribute has a “yes”**

**value.**

**Snapshot your screen showing the even newer XML data dump result and**

**paste it into Word. Label your snapshot XML data 3.**

**Make sure to include your pdf file of your code including relevant xml files into your Word doc and a pdf of your snapshots for credit. Include both portrait and landscape view screenshots.**

**\*\* Grads include the following for credit.**

-Show the count of the amount of CD’s that were sold out.

You probably want to add in a scroll view if your count is at the end of the screen.

-Add any alternate colors to records within your view (ex. Background or foreground colors, one color for one record that will differ from another records color) and/or a dividing line to separate each record.

-Finally add in a progress dialog/bar with a wait message when your MainActivity loads for 3 seconds.

Show/label these additional screenshots accordingly for full possible credit.