**Advanced Mobile Application Development**
**Week 12: Fragments Data**

## Dialogs

https://developer.android.com/guide/topics/ui/dialogs.html
A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.
The Dialog class is the base class for dialogs, but should be subclasses and not used directly. Instead, use one of the following subclasses:

- AlertDialog: can show a title, up to three buttons, a list of selectable items, or a custom layout.
    - AlertDialog.Builder is used to create an alert dialog
    - Title (optional)
    - Content area
        - Message
        - Add selectable widgets, edittext, list
    - Three different action buttons
        - Positive to accept and continue with the action (the "OK" action).
        - Negative to cancel the action
        - Neutral for when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."
- DatePickerDialog or TimePickerDialog: allows the user to select a date or time.
- You can also create custom dialogs with custom layouts

Material Design components – dialogs https://material.io/guidelines/components/dialogs.html

## Menus

We'll use a contextual menu for deleting an item in the list.
When the view receives a long-click event, onCreateContextMenu() is called. This is where you define the menu items by inflating a menu resource.
https://developer.android.com/guide/topics/ui/ui-events.html

When the user selects a menu item onContextItemSelected() is called. This is where you respond based on which menu item was chosen. getItemId() returns the ID for the selected menu item.
Creating contextual menus https://developer.android.com/guide/topics/ui/menus.html#context-menu

## XML Data

Parsing XML Data https://developer.android.com/training/basics/network-ops/xml.html
Android/Java has multiple ways of dealing with XML, using XMLPullParser is recommended.
If your XML file is in your project, you create an xml directory in resources and then get access to it using getResources().getXml(R.xml.*filename*); which returns a XMLResourceParser. This is more efficient than adding it as an asset and opening it as an input stream.
If you're downloading it from the Internet you should do so in an asynchronous thread so as to not hold up the UI. We will do this when we deal with JSON.
XMLResourceParser is a subclass of XMLPullParser
https://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html
The current event state of the parser can be determined by calling the getEventType() method.

Event types:

- START_DOCUMENT: the beginning of the file
- START_TAG: an XML start tag was read
  - the tag name can be retrieved using the getName() method
- TEXT: text content was read
  - the text content can be retrieved using the getText() method
- END_TAG: an XML end tag was read
  - the tag name can be retrieved using the getName() method
- END_DOCUMENT: end of documents, no more events

The method next() advances the parser to the next event.
You basically walk through the document and test for the tags that you're interested and then extract that data.

**Superheroes**
Let's add the ability to add heroes.
In fragment_hero_detail.xml change the FrameLayout to a ConstraintLayout since we're going to add a button. Change the FrameLayout, which is meant to display a single item, to a ConstraintLayout by going into the Design view and right click on FrameLayout in the component tree and pick Convert FrameLayout to ConstraintLayout, selecting to flatten the hierarchy.
Make the ListView a little shorter and add a button below it with an id of addButton and text, using a string resource, that says "Add Hero"

Add the string resource <**string name="add_dialog_title"**>Add Hero</**string**>

Update HeroDetailFragment.java to create an interface, define a listener, assign the listener to the context, and tell the listener when the button is clicked.
HeroDetailFragment.java

```
//create interface
interface ButtonClickListener{
   void addheroclicked(View view);
}

//create listener
private ButtonClickListener listener;

@Override public void onAttach(Context context){
   super.onAttach(context);
   //attaches the context to the listener
   listener = (ButtonClickListener)context;
}

import android.content.Context;
```

Implement OnClickListener for the button(this will show an error until you implement the onClick() method)
**public class** HeroDetailFragment **extends** Fragment **implements** View.OnClickListener{

```
@Override public void onClick(View view){
    if (listener !=null){
        listener.addheroclicked(view);
    }
}
```

And update onStart() to attach the listener to the button

```
Button addHeroButton = (Button) view.findViewById(R.id.addHeroButton);
addHeroButton.setOnClickListener(this);

import android.widget.Button;
```

Create addHero() method

```
public void addhero(){
    //create alert dialog
    AlertDialog.Builder dialog = new AlertDialog.Builder(getActivity());
    //create edit text
    final EditText edittext = new EditText(getActivity());
    //add edit text to dialog
    dialog.setView(edittext);
    //set dialog title
    dialog.setTitle("Add Hero");
    //sets OK action
    dialog.setPositiveButton("Add", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            //get hero name entered
            String heroName = edittext.getText().toString();
            if(!heroName.isEmpty()){
                // add hero
                Hero.heroes[(int) universeId].getSuperheroes().add(heroName);
                //refresh the list view
                HeroDetailFragment.this.adapter.notifyDataSetChanged();
            }
        }
    });
    //sets cancel action
    dialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            // cancel
        }
    });
    //present alert dialog
    dialog.show();
}

import android.app.AlertDialog;
import android.content.DialogInterface;
```

Then we need to update MainActivity.java to have our main activity implement the interface.

Update MainActivity.java
**public class** MainActivity **extends** Activity **implements** UniverseListFragment.UniverseListListener, HeroDetailFragment.ButtonClickListener (will show an error until you implement the following method)

```
@Override public void addheroclicked(View view){
   HeroDetailFragment fragment =
(HeroDetailFragment)getFragmentManager().findFragmentById(R.id.fragment_container);
   fragment.addhero();
}

import android.view.View;
```

Now when you run it you should be able to add a superhero.

Delete Heroes

When the user does a long-press on a hero we want them to be able to delete it. We'll do this though a context menu so they can chose to delete the hero or not.
In HeroDetailFragment.java add the following

Create a context menu on the long press

```
@Override public void onCreateContextMenu(ContextMenu menu, View view,
ContextMenu.ContextMenuInfo menuInfo){
   super.onCreateContextMenu(menu, view, menuInfo);
   //cast ContextMenu.ContextMenuInfo to AdapterView.AdapterContextMenuInfo since we're using an
adapter
   AdapterView.AdapterContextMenuInfo adapterContextMenuInfo =
(AdapterView.AdapterContextMenuInfo) menuInfo;
   //get hero name that was pressed
   String heroname = adapter.getItem(adapterContextMenuInfo.position);
   //set the menu title
   menu.setHeaderTitle("Delete " + heroname);
   //add the choices to the menu
   menu.add(1, 1, 1, "Yes");
   menu.add(2, 2, 2, "No");
}

import android.view.ContextMenu;
import android.widget.AdapterView;
```

Then we have to handle the delete when they chose a menu item selection

```
@Override public boolean onContextItemSelected(MenuItem item){
   //get the id of the item
   int itemId = item.getItemId();
   if (itemId == 1) { //if yes menu item was pressed
      //get the position of the menu item
      AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo)
item.getMenuInfo();
      //remove the hero
```

```
        Hero.heroes[(int) universeId].getSuperheroes().remove(info.position);
        //refresh the list view
        HeroDetailFragment.this.adapter.notifyDataSetChanged();
    }
    return true;
}

import android.view.MenuItem;
```

Register the context menu in onStart()
registerForContextMenu(listHeroes);

XML file
In order to read data in from an XML file we need to change our heroes array to an ArrayList so we can add whatever data the file holds.
Hero.java

```
public static ArrayList<Hero> heroes = new ArrayList<Hero>();
```

HeroDetailFragment.java
You can't access arraylists with indices, you need to use get instead. Change the 3 instances of

```
herolist = Hero.heroes[(int) universeId].getSuperheroes();
```

to

```
herolist = Hero.heroes.get((int) universeId).getSuperheroes();
```

To add our XML file we need to create an XML directory in resources and add it there.
Right click on res folder File | New | Android resource folder
Xml
Put xml file in there

MainActivity.java
We'll load the XML data in our Main Activity. Update onCreate(). We test to see if heroes is empty otherwise it loads the XML file every time onCreate() runs, which happens on every device roataion.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if(Hero.heroes.isEmpty()) {
        try {
            loadXML(this);
        } catch (XmlPullParserException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Then we'll create a function that loads the XML, parses it, and adds the data to our heroes arraylist. The StringBuffer class is only used for debugging purposes.

```
private void loadXML(Activity activity) throws XmlPullParserException, IOException {
   String new_universe = new String();
   ArrayList<String> new_heroes=new ArrayList<String>();
   //string for debugging puposes only
   StringBuffer stringBuffer = new StringBuffer();
   //get xml file
   XmlResourceParser xpp = getResources().getXml(R.xml.superheroes);
   //advances the parser to the next event
   xpp.next();
   //gets the event type/state of the parser
   int eventType = xpp.getEventType();
   while (eventType != XmlPullParser.END_DOCUMENT) {
      switch (eventType) {
         case XmlPullParser.START_DOCUMENT:
            // start of document
            break;
         case XmlPullParser.START_TAG:
            if (xpp.getName().equals("universe")) {
               stringBuffer.append("\nSTART_TAG: " + xpp.getName());
            }
            if (xpp.getName().equals("name")) {
               stringBuffer.append("\nSTART_TAG: " + xpp.getName());
               eventType = xpp.next();
               new_universe = xpp.getText(); //gets the name of the universe
            } else if (xpp.getName().equals("hero")) {
               stringBuffer.append("\nSTART_TAG: " + xpp.getName());
               eventType = xpp.next();
               //add to arraylist
               new_heroes.add(xpp.getText()); //gets the name of the hero
            }
            break;
         case XmlPullParser.END_TAG:
            //if end of universe add the new hero
            if (xpp.getName().equals("universe")) {
               //at the end of the universe
               //create new Hero object
               Hero new_hero = new Hero(new_universe, new_heroes);
               //add hero
               Hero.heroes.add(new_hero);
               //clear universe name and list of heroes
               new_universe = "";
               new_heroes.clear();
            }
            break;

         case XmlPullParser.TEXT:
```

```
                break;
            }
            //advances the parser to the next event
            eventType = xpp.next();
        }
        System.out.println(stringBuffer.toString());
    }
```

Phone and tablet layouts

One of the reasons we're using fragments is so our app can look different on a tablet than on a phone. On a tablet we want to see both fragments, as we are now. On a phone since the screen is smaller we want to only see one at a time. Just like with other resources you can create multiple layout folders with names to target specific specifications.

Switch to the Project view hierarchy and go to app/src/main/res and create a new resource directory.

Directory name: layout-sw400dp

Resource type: layout

Source set: main

This will target devices with the smallest available width of 400dp.

You can also use layout-w500dp(available width) or layout-land for landscape phone and tablet but not portrait phone or tablet.

(starting with Android 3.2, API 13, layout folders no longer use screen size values like –large)

http://developer.android.com/guide/practices/screens_support.html#DeclaringTabletLayouts

http://labs.rampinteractive.co.uk/android_dp_px_calculator/

Select your activity_main.xml file in your layout folder and copy/paste it into your new layout-sw400dp folder. This layout with side by side fragments will be used for devices with the smallest available width of 400dp.

Now we need to create a new Activity called DetailActivity that will use HeroDetailFragment. So instead of using both fragments in MainActivity, MainActivity will use UniverseListFragment and DetailActivity will use HeroDetailFragment when the user clicks on a universe.

Use the wizard to create a new empty activity called DetailActivity with a layout file named activity_detail. Make sure activity_detail.xml is in your layout folder so any device can use it.

Now let's update activity_main.xml in the layout folder to change the layout for smaller devices. Make sure you open activity_main.xml in the original layout folder.

Remove(cut because you're going to paste) the FrameLayout so all you have is the one fragment for the universe list fragment.

Change the layout_width for the one fragment to match_parent so it takes up the full width.

Paste it into activity_detail.xml as a fragment and change the width to match_parent so it takes up the full width.

```
<fragment
    android:id="@+id/fragment_container"
    android:name="com.example.aileen.superheroes.HeroDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
app:layout_constraintRight_toRightOf="parent"
tools:layout="@layout/fragment_hero_detail" />
```

If the app is running on a phone MainActivity will need to start DetailActivity with an intent and pass the universe id to the HeroDetailFragment using its setUniverse() method.

In DetailActivity.java add to onCreate()

```
//get reference to the hero detail fragment
   HeroDetailFragment heroDetailFragment = (HeroDetailFragment)
getFragmentManager().findFragmentById(R.id.fragment_container);
   //get the id passed in the intent
   int universeId = (int) getIntent().getExtras().get("id");
   //pass the universe id to the fragment
   heroDetailFragment.setUniverse(universeId);
```

Since DetailActivity.java will be running the HeroDetailFragment we need DetailActivity to implement the ButtonClickListener as well.

```
public class DetailActivity extends Activity implements HeroDetailFragment.ButtonClickListener

@Override public void addheroclicked(View view){
   HeroDetailFragment fragment =
(HeroDetailFragment)getFragmentManager().findFragmentById(R.id.fragment_container);
   fragment.addhero();
}
```

DetailActivity will get the id from the intent that starts it. We need MainActivity to start DetailActivity only when the app is running on a phone. So we want MainActivity to perform different actions if the device is using activity_main.xml in the layout or layout-sw400dp folder.
If the app is running on a phone it will be running activity_main.xml in the layout folder which doesn't include the HeroDetailFragment so we'd want MainActivity to start DetailActivity.
If the app is running on a tablet it will be running activity_main.xml in the layout-sw400dp folder which includes a frame layout with the id fragment_container so we'd need to display a new instance of HeroDetailFragment in the fragment container(as we have been doing).

Update MainActivity.java

Update itemClicked()

```
//get a reference to the frame layout that contains HeroDetailFragment
View fragmentContainer = findViewById(R.id.fragment_container);
//large layout device
if (fragmentContainer != null) {
** put the rest of the existing code in the body of the if statement**
} else{ //app is running on a device with a smaller screen
   Intent intent = new Intent(this, DetailActivity.class);
   intent.putExtra("id", (int) id);
   startActivity(intent);
}
```

```
import android.view.View;
import android.content.Intent;
```

Define an AVD that is a tablet so you can test it on a phone and tablet. Chose category Tablet and pick one(I picked Nexus 7 API 23).

You should see both fragments side by side on the tablet and only 1 fragment at a time on a phone. You can probably modify the layout_width for the fragments in your layout-w500dp/activity_main layout file.
Make sure you've tested rotation as well and if you've clicked on an item other than the first when you rotate it stays on that data.

Don't forget launcher icons.