Advanced Mobile Application Development
Week 14: Fragments

Superheroes
Create a new project called Superheroes
Minimum SDK: API 17
Check Phone and Tablet, leave the rest unchecked
Empty Activity template
Activity name: MainActivity
Check Generate Layout File
Layout Name: activity_main

Hero java class
Create a new Java class called Hero.

```
public class Hero {
   private String universe;
   private ArrayList<String> superheroes = new ArrayList<>();

   //constructor
   private Hero(String univ, ArrayList<String> heroes){
      this.universe = univ;
      this.superheroes = new ArrayList<String>(heroes);
   }

   public static final Hero[] heroes = {
         new Hero("DC", new ArrayList<String>(Arrays.asList("Superman", "Batman", "Wonder
Woman", "The Flash", "Green Arrow", "Catwoman"))),
         new Hero("Marvel", new ArrayList<String>(Arrays.asList("Iron Man", "Black Widow",
"Captain America", "Jean Grey", "Thor", "Hulk")))
   //source: http://www.ign.com/articles/2013/11/19/the-top-25-heroes-of-dc-comics
   //source: http://www.ign.com/articles/2014/09/10/top-25-best-marvel-superheroes
   };

   public String getUniverse(){
      return universe;
   }

   public ArrayList<String> getSuperheroes(){
      return superheroes;
   }

   public String toString(){
      return this.universe;
   }
}
```

Universe List Fragment

Let's create a fragment that will be used for our first view which will have an image and a list of the superhero universes.

File | New | Fragment | Fragment(blank)
Name: UniverseListFragment
Check Create layout XML
Fragment layout name: fragment_universe_list
Uncheck both include check boxes
Finish

Android Studio creates a new fragment file called UniverseListFragment.java in the app/src/main/ java folder, and a new layout file called fragment_universe_list.xml in the app/src/res/layout folder.

Go into fragment_universe_list.xml
You can see that fragment layout code looks a lot like activity layout code. You can use all the same views and layouts you've been using in activities when you're creating fragments.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

<ImageView
    android:layout_width="175dp"
    android:layout_height="76dp"
    android:layout_gravity="center"
    android:id="@+id/imageView"
    android:src="@drawable/superheroes"
    android:contentDescription="@string/app_name" />

<ListView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingTop="10dp"
    android:id="@+id/listView" />

</LinearLayout>
```

Add image into the drawables folder.

Look at the code generated in UniverseListFragment.java

```
public class UniverseListFragment extends Fragment {
```

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
   return inflater.inflate(R.layout.fragment_universe_list, container, false);
}
public UniverseListFragment() { }
}
```

Note it extends the Fragment class instead of Activity.
The onCreateView() method gets called each time Android needs the fragment's layout, and it's where you say which layout the fragment uses. This method is optional, but you'll need to implement it almost every time you create a fragment because you need to implement it whenever you're creating a fragment with a layout.
inflater.inflate() is a fragment's equivalent to an activity's setContentView() method. It determines which layout the fragment uses.

All fragments must have a public constructor with no arguments. Android uses it to reinstantiate the fragment when needed, and if it's not there, you'll get a runtime exception. The Java compiler automatically adds a public no- argument constructor for you if your class contains no other constructors, so you only need to add it if you include another constructor with arguments.

Now we need to add the fragment to the main activity which was created for us when we created the project. In activity_main.xml remove the TextView and add the fragment

```
<fragment
   android:layout_width="match_parent"
   android:layout_weight="2"
   android:layout_height="match_parent"
   android:id="@+id/universe_frag"
   class="com.example.aileen.superheroes.UniverseListFragment" />
```

The class attribute specifies the fully qualified name of your fragment.

Now lets update UniverseListFragment.java to load our values into our fragment's listview.

```
@Override
public void onStart(){
   super.onStart();
   View view = getView();
   if (view != null){
      //load data into fragment
      //get the list view
      ListView listUniverse = (ListView) view.findViewById(R.id.listView);
      //define an array adapter
      ArrayAdapter<Hero> listAdapter = new ArrayAdapter<Hero>(getContext(),
android.R.layout.simple_list_item_1, Hero.heroes);
      //set the array adapter on the list view
      listUniverse.setAdapter(listAdapter);
   }
}
```

Need to use onStart() because in onCreateView() the view isn't created yet so we can't access it.

Hero Detail Fragment

Now let's create our detail fragment to hold the list of superheroes. This is going to only hold a list so just like we used the ListActivity class, fragments have a ListFragment class when there is only a list in the layout. Just like with ListActivity, it's automatically bound to a list view so you don't need to create a layout yourself. We also won't need to implement our own event listener because the ListFragment class is already registered as a listener on the list view.

File | New | Fragment | Fragment(blank)
Name: HeroDetailFragment
Uncheck Create layout XML
Uncheck both include check boxes
Finish

In HeroDetailFragment.java change the class so it extends ListFragment

```
public class HeroDetailFragment extends ListFragment { }
```

For now let's populate the list view with the heroes of the first universe (DC) just to get it working.

```
@Override public void onStart(){
    super.onStart();

   // get hero data
   ArrayList<String> herolist = new ArrayList<String>();
   herolist = Hero.heroes[0].getSuperheroes();

   //create array adapter
   ArrayAdapter<String> adapter = new ArrayAdapter<String>( getActivity(),
android.R.layout.simple_list_item_1, herolist);

   //bind array adapter to the list view
   setListAdapter(adapter);
}
```

When we used an array adapter to populate a list view with an activity we could use "this" to get the current context because activity is a type of context, the Activity class is a subclass of the Context class. The Fragment class is not a subclass of the Context class so we must use getActivity() to get the activity that the fragment is associated with. (using getContext() like we did in our other class, requires API 23)

Eventually we'll want the universe fragment to tell the hero fragment which superheroes to display. To do this we'll need the id of the universe and a setter method to set it. Later we'll use it to update the fragment view.

```
private long universeId; //id of the universe chosen

//set the universe id
public void setUniverse(long id){
   this.universeId = id;
}
```

Now we need to add the fragment to the main activity so it appears to the right of UniverseListFragment. We'll use a linear layout with horizontal orientation in activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.aileen.superheroes.MainActivity">

    <fragment
        android:layout_width="0dp"
        android:layout_weight="2"
        android:layout_height="match_parent"
        android:id="@+id/universe_frag"
        class="com.example.aileen.superheroes.UniverseListFragment" />

    <fragment
        android:layout_width="0dp"
        android:layout_weight="3"
        android:layout_height="match_parent"
        android:id="@+id/detail_frag"
        class="com.example.aileen.superheroes.HeroDetailFragment" />

</LinearLayout>
```

You might need to change the dimensions of your image if it's cut off.

Cntrl fn F11 to rotate the emulator on a Mac; cntrl F11 on a PC.

Update Detail List

Now lets get the detail view to show the heroes based on which universe the user selects.
We want the fragments to be reusable so it's best if they rely as little as possible on the activity using it.
We'll decouple the fragment and the activity by using an interface.
When we define an interface, we're saying what the minimum requirements are for one object to talk usefully to another. It means that we'll be able to get the fragment to talk to any kind of activity, so long as that activity implements the interface.
Update UniverseListFragment.java to create an interface, define a listener, assign the listener to the context, and tell the listener when an item is clicked.

```
//create interface
static interface UniverseListListener{
    void itemClicked(long id);
}
```

```
//create listener
private UniverseListListener listener;

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

@Override public void onItemClick(AdapterView<?> parent, View view, int position, long id){
   if (listener != null){
      //tells the listener an item was clicked
      listener.itemClicked(id);
   }
}
```

And update onStart() to attach the listener to the listview (add this after setAdapter())

```
      //attach the listener to the listview
      listUniverse.setOnItemClickListener(this);
```

Then we need to have our main activity implement the interface.
Update MainActivity.java
public class MainActivity **extends** AppCompatActivity **implements**
UniverseListFragment.UniverseListListener

When an item is clicked in the fragment, the itemClicked() method in the activity will be called so that's where we can have the fragment update its details. Instead of updating the fragment details we're going to replace the detail fragment with a brand-new detail fragment each time we want its text to change. This is so the back button does what a user would expect it to and go to the previous data if they had clicked on different items, and not automatically the previous view.

To do this we first change the activity_main.xml layout file. Instead of inserting WorkoutDetailFragment directly, we'll use a frame layout.

A frame layout is a type of view group that's used to block out an area on the screen. You define it using the <FrameLayout> element. You use it to display single items—in our case, a fragment. We'll put our fragment in a frame layout so that we can control its contents programmatically. Whenever an item in the WorkoutListFragment list view gets clicked, we'll replace the contents of the frame layout with a new instance of WorkoutDetailFragment that displays details of the correct workout:

```
<FrameLayout
   android:layout_width="0dp"
   android:layout_weight="3"
   android:layout_height="match_parent" />
```

Next we update MainActivity.java to create a new HeroDetailFragment when the user clicks on a universe.

```
@Override public void itemClicked(long id){
    //create new fragment instance
    HeroDetailFragment frag = new HeroDetailFragment();
    //create new fragment transaction
    FragmentTransaction ft = getFragmentManager().beginTransaction();
    //set the id of the universe selected
    frag.setUniverse(id);
    //replace the fragment in the fragment container
    ft.replace(R.id.fragment_container, frag);
    //add fragment to the back stack
    ft.addToBackStack(null);
    //set the transition animation-optional
    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
    //commit the transaction
    ft.commit();
}
```

Fragments don't have a onBackPressed() method so when you press the back key the app looks for the previous activity or if there is none, exits. To have the back button work correctly with fragments we need to override the onBackPressed() method and have it check if there are fragments on the backstack to pop them out, otherwise follw the default behavior.

```
@Override public void onBackPressed() {
    if (getFragmentManager().getBackStackEntryCount() > 0 ){
        getFragmentManager().popBackStack();
    } else {
        super.onBackPressed();
    }
}
```

Now that we have that set up let's update the hero detail fragment so it shows the data based on which universe the user selected. We just need to change the one line where we have Hero.heroes[0] now use the universeId.

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

Rotation

You'll notice that if you select any item other than the first, when you rotate the device the list automatically goes back to the first one. That's because the detail fragment is getting destroyed and recreated along with the activity. (control fn F11 to rotate)

Save the current universeId being displayed in the fragment.

```
@Override public void onSaveInstanceState(Bundle savedInstanceState){
    savedInstanceState.putLong("universeId", universeId);
}
```

Then update onCreateView() to check to see if there was a saved instance and retrieve the universe id.

```
if (savedInstanceState !=null){
    universeId = savedInstanceState.getLong("universeId");
}
```

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 table 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 directory called layout-sw400dp to 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 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 layout folder.

Remove the FrameLayout so all you have is the one fragment for the universe list fragment.

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.

Update activity_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    class="com.example.aileen.superheroes.HeroDetailFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

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.

Update DetailActivity.java

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

    //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);
}
```

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

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

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);
}
```

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. 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.