Lab.02 Fragments and Dialogs

We will test how fragments share information and can be managed dynamically.

1. Interaction between fragments: Interface

1. Create a new project *Lab02.Surname* with main activity *FragmMainActiv*.

   a) Create a new fragment *FragmOne*[1], without the *callbacks* nor the *factory methods*. Add a button (*but_url*) and a spinner (*sp_url*) to its layout.

   b) Create a *FragmTwo* fragment with a WebView (*webView*) and Internet permission.

   c) Statically add these two fragments to the main activity's layout.

2. **Manage events: interface**

   a) **Declare the interface of the fragment that generates the event**. In our case, FragmOne, with the name*onArticleSelectedListener*, which include a single method (*onArticleSelected*) with a single parameter (*string*):

   b) **Implement the interface in the activity**, with the code that manages the event. In our case, the activity will call a public method on FragmTwo: *load(name)*, where *name* is the argument (string) of the *onArticleSelected* method (it will be an *url*).

   c) In fragments it is usual to **"handle" the interface** in the onResume () method (that is, when the fragment returns to the foreground and the user can use it).
   In our case, we associate the interface method (*onArticleSelected()*) with the event of pressing the button (*but_url*) and passing a parameter (*url*) selected in (*sp_url*). The spinner *sp_url* has a list of urls (the first entry is empty)

3. **Executes external actions**: FragmTwo displays the url selected in FragmOne:

   a) Define the WebView (*webView*), e.g. in the onResume () method.

   b) Redefine the callbacks generated in this WebView (not to leave the widget).

   c) Create the public method *load(string url)* which loads the url.

   d) If the *url* parameter is null or empty, the content of *webView* must be deleted.

```
public void onResume() {
    super.onResume();
    Log.d(TAG, "[FRAGMENT 2]: onResume()");
    webView = (WebView)getView().findViewById(R.id.webView);
    webView.setWebViewClient(new Callback()); // Book  W-M Lee pag. 290-5
}
private class Callback extends WebViewClient {// Book  W-M Lee pag. 290-5
    @Override
    public boolean shouldOverrideUrlLoading (WebView view, String url) {
          return false;     }
}
public void load(String url){webView.loadUrl(url);}//clean if url is empty
```

[1] (*New > Android Component > New Blank Fragment*).

2. Add fragments dynamically

Create the fragment(s), interface(s), layout(s), code (etc) to carry out these actions:

1. Add 3 elements to FragmOne: Button (*but_clear*), SeekBar (*sb_f1*) and EditText (*et_f1*).

   When moving *sb_f1*, and if the contents of *et_f1* **is not empty**, the text of *et_f1* is sent to FragmThree (a string). The value of *sb_f1* (an integer between 0 and 100) it also sent.

2. **Create the FragmThree fragment** whose layout has two TextViews (*tv_f3_id, tv_f3*).

   tv_f3 shows the string sent by FragmOne with the font size of the integer parameter.

   The main activity, must record the number of times that a FragmThree is created or added to its layout. This number is showed in *tv_f3_id*

   Very important: Initially, the main activity do not have FragmThree. It only can be added/replaced dynamically. Avoid that any fragment can be completly hidden.

3. When *but_clear* is enabled in FragmOne:

   a) If the FragmThree fragment exists in the main activity: it is deleted.

   b) If not, only show a Toast anda Log indicating that FragmThree does not exist.

4. **Add a new interface** to the FragmThree fragment:

   a) Add a Toggle button (*tgbut_disable*) to FragmThree, whose initial state is disabled.

   b) When *tgbut_disable* is enabled, the main activity must "block" the changes in FragmThree **disabling** the changes in FragmTwo (of *sb_f1* and *et_f1*).

   c) When you disable *tgbut_disable*, the main activity again allows normal operation.

   d) **Attention**: avoid the possible blockage between both fragments.

3. Create an AlertDialog

AlertDialog is an easy way to ask the user for action's confirmation (or request information).

1. Add a button with id *but_exit*, text *Exit* and *drawable/ic_lock_power_off* icon.

2. Add the corresponding *Listener* to create a dialog alert to confirm the order (i.e. two options or actions: *OK* and *Cancel*).

3. If the command is confirmed (*OK*), end the execution of the application (*finish()*).

4. Add an EditText to AlertDialog. When canceling, collect the data of EditText and display it in a log and in a Toast.