CMSC 436 Lab 4

Building a Dynamic UI with Fragments (Advanced)

Overview

- This lab will cover two topics- creating background Fragments without a UI and communicating between Fragments and Activities
- The information you will need for this lab can be found on the Android developer site at

  http://developer.android.com/guide/components/fragments.html

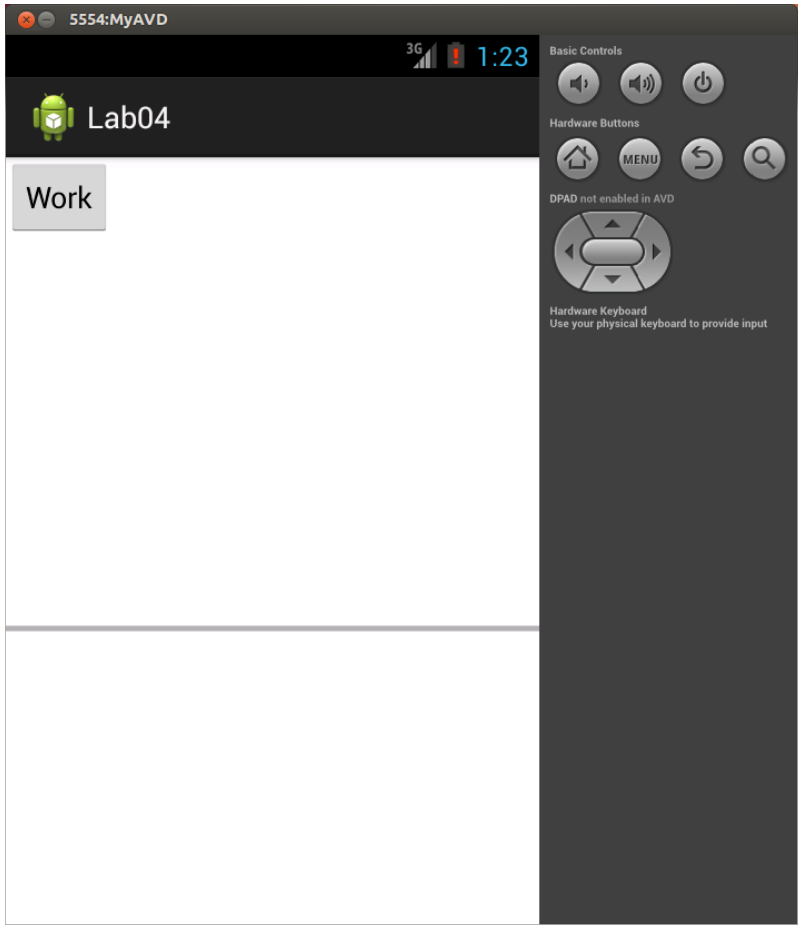
  http://developer.android.com/training/basics/fragments/

Overview

- For this lab you will create three Fragments- a Fragment that displays a Button, a Fragment that displays a ProgressBar, and an invisible background Fragment that does some work

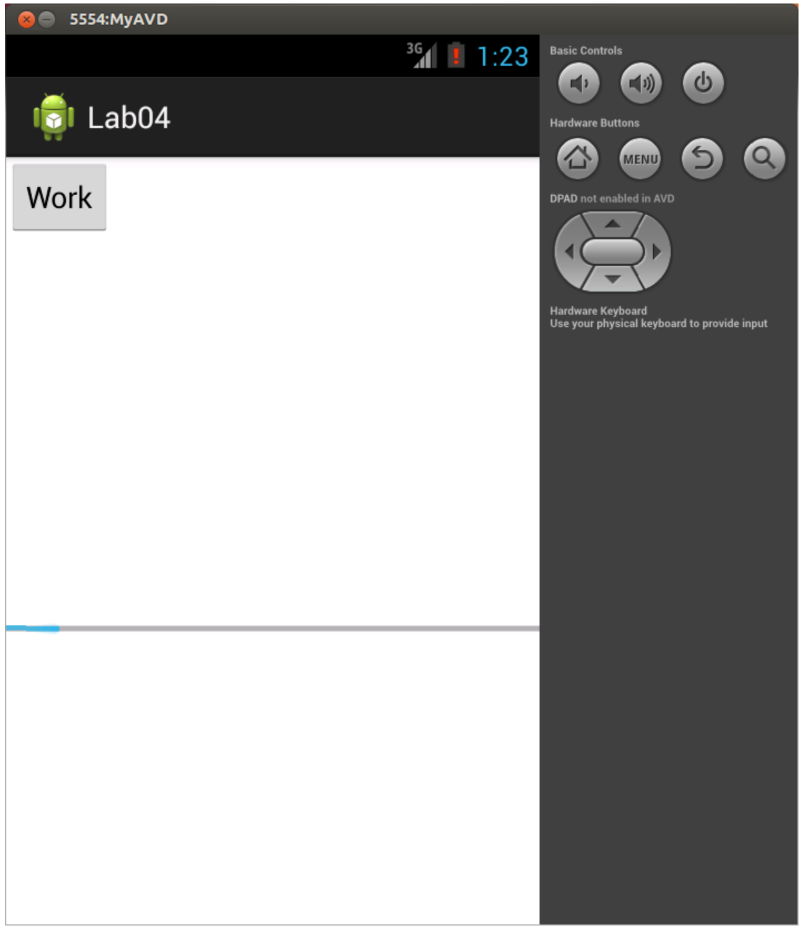
Overview

- When the app starts, the button and progress bar fragments are displayed
- The progress bar starts as empty



Overview

- Every time the button is clicked, the background fragment performs some work and the progress bar increases by 10%



Layout

- In the previous lab you reserved space for the Fragments with FrameLayout tags in the xml and dynamically added and removed them with transactions
- In this lab the Fragments will be a permanent part of the interface (except the background worker Fragment), so you should add them to the main Activity's layout using fragment tags

Layout

- The background worker thread should be added in a Transaction during the Activity's onCreate with the syntax add(Fragment, String)
- The String is a tag that can be used to get a reference to the Fragment later using findFragmentByTag
- Before creating the Fragment you should call findFragmentByTag to see if it has already been created (this happens for example if onCreate is called after the device it rotated)

Communication

- To support a modular design, Fragments should not directly communicate with each other- instead, they should communicate with the host Activity, which can then pass messages to other Fragments
- The Activities and Fragments can have various callback methods to support this

Communication

- If an Activity is going to receive callbacks from a Fragment, the Activity should implement an interface that contains these callbacks
- When the Fragment receives a reference to the host Activity in onAttach, the Fragment can verify that the host Activity implements the required interface by casting the reference to that interface and catching an exception if the cast fails- this is demonstrated on the Android developer site

Communication

- The flow of communication should be
  - The work button is clicked, triggering a callback in the button Fragment
  - The button Fragment makes a call to the host Activity
  - The host Activity makes a call to the worker Fragment to start performing the work
  - The worker Fragment finishes the work and makes a call to the host Activity
  - The host Activity makes a call to the progress bar fragment to update the progress bar

Communication

- Note that if you try to define a callback for the work Button using the onClick attribute in the xml tag, Android will look for the callback in the host Activity, not in the Fragment
- To specify a callback method in the Fragment, you will need to instead call the Button's setOnClickListener method during the Fragment's onActivityCreated callback
- You can get a reference to the Button by using getActivity().findViewById()

Communication

- A common way to implement callback listeners is with an anonymous class, such as

```
workButton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                        //Handle button click
            }
});
```

Work

- When the work button is clicked, the background worker Fragment will need to simulate doing some work
- In this lab you can just have it call Thread.sleep(1000) to pause for 1 second
- Note that this will cause the UI thread to block- later we will cover how to use threads to perform work like this asynchronously