Networking & NDK

ICT2105 Mobile Application Development Spring 2020

Overview

This lab provides exercises and instructions to familiarize students with

1. networking with a RESTful API and parsing of JSON data, and
2. working with the Android Native Development Kit (NDK).

Outcomes

Upon completion of the session, you should be able to:

- Practice more coroutines for threaded background work
- Use an Android networking API to download some data
- Create a simple application that calls native methods using JNI
- Implement native code to use a simple C++ class in Android
- Compare the performance between native and Java methods

Familiarization: Hello JNI

The latest Android Studio has made it very easy to create an example C++ hello world application using JNI with the Android NDK. A tutorial can be found at the following link:

https://developer.android.com/studio/projects/add-native-code.html#new-project

Inspect the project to understand how the parts connect together.

Part 1: Basic Factorials

The goal of this exercise is to compare different ways to obtain the simple factorial function, (1) using a cloud API as well as (2) locally using iterative and recursive methods in Java and C.

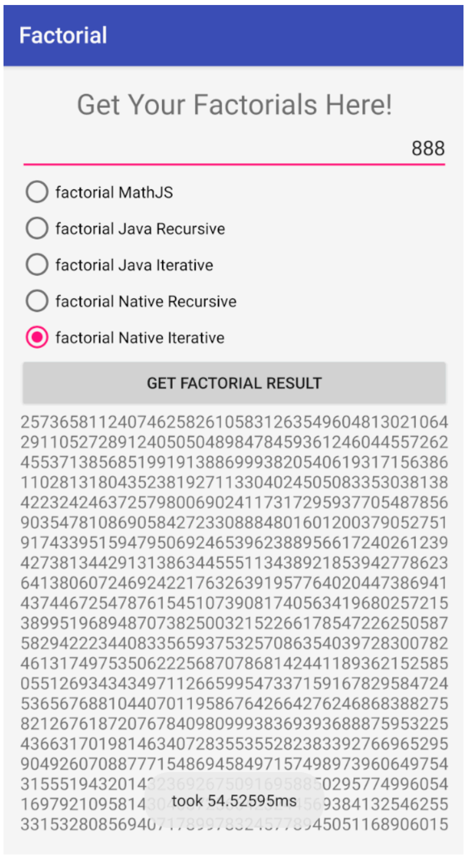
Fork the repo **ict2105-lab07-2020** and inspect the code within the project. This code is incomplete.

Refer to the screenshot on the next page. There are 5 different versions of the factorial function that can be selected from the radio buttons:

- Factorial function in MathJS API, called through the network
- Factorial function in Java, implemented recursively
- Factorial function in Java, implemented iteratively
- Factorial function in C, implemented recursively
- Factorial function in C, implemented iteratively

On click of the Button, the factorial of the number from the TextEdit input at the top should be displayed in the TextView output at the bottom. A toast should be shown to display the amount of time taken to run the function in seconds. See the screenshot for an example.

Note that in this lab the **use of AsyncTasks is FORBIDDEN**. If there is a valid working AsyncTask derived class detected in your code, you will get zero. I mean, why use something that is deprecated when there are better solutions.



Factorial

Get Your Factorials Here!

888

factorial MathJS

factorial Java Recursive

factorial Java Iterative

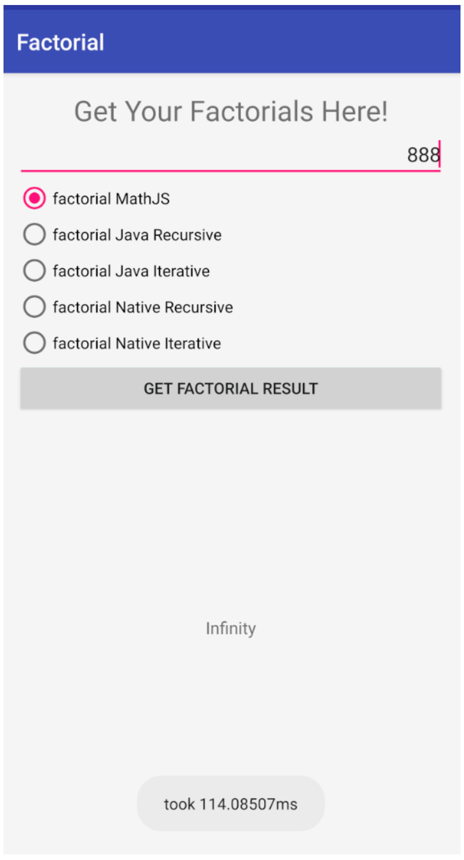
factorial Native Recursive

[✓ factorial Native Iterative]

GET FACTORIAL RESULT

```
25736581124074625826105831263549604813021064
29110527289124050504898478459361246044557262
45537138568519919138869993820540619317156386
11028131804352381927113304024505083353038138
42232424637257980069024117317295937705487856
90354781086905842723308884801601200379052751
91743395159479506924653962388956617240261239
42738134429131386344555113438921853942778623
64138060724692422176326391957764020447386941
43744672547876154510739081740563419680257215
38995196894870738250032152266178547226250587
58294222344083356593753257086354039728300782
46131749753506222568707868142441189362152585
05512693434349711266599547337159167829584724
53656768810440701195867642664276246868388275
82126761872076784098099938369393688875953225
43663170198146340728355355282383392766965295
90492607088777154869458497157498973960649754
31555194320143236926750916958850295774996054
169792109581430[illegible]569384132546255
33153280856940717899783245778945051168906015
```

took 54.52595ms



Factorial

Get Your Factorials Here!

888

[✓ factorial MathJS]

factorial Java Recursive

factorial Java Iterative

factorial Native Recursive

factorial Native Iterative

GET FACTORIAL RESULT

```
Infinity
```

took 114.08507ms

Part 2: Working with Large Numbers

After you work on part 1, you should realize that the large output shown in the screenshot is impossible to achieve using primitive Long types.

In this part, you should make use of large number libraries to help you get much larger factorial numbers. In particular you will use:

- The **BigInteger** class in Java's internal library. There are loads of information on the web for this class.
  Use this class for the iterative and recursive implementations in Java.
- A naïve (and native ☺) **bigint** C++ class inspired from
  https://sites.google.com/site/indy256/algo_cpp/bigint
  Note that this class has already been fully implemented for you so you just need to use it appropriately. See the link for an example.
  Use this class (look for the **cpp** folder) for the iterative and recursive implementations in C++.

Basically, the goal of this part is to enable the calculation and display of factorials results for numbers above 100. See all the TODO hints given in the code.

You will have to implement the changes for all 5 versions of the factorial functions. Note that to keep the interface consistent, the numbers will be passed around as Strings (which can represent arbitrarily large number of digits), but within each function you will make use of the appropriate big number library classes to help you compute the factorials.

Note also that for the MathJS API will not be able to handle these large numbers and you will get "Infinity" as the result, and this is what I will be expecting.

The runtimes of each function, as well as the StackOverflows (not that most-popular-amongst-ICT-students website) will reveal some very interesting things for you to ponder over.

> **IMPORTANT:**
> **Ensure that the UiInstrumentedTest.kt test file compiles without errors. Do not edit any existing package, class, variable, method or layout file names. You may add to, but do not edit existing dependencies in, the gradle files.**

Lab Exercise 7

Due Date: Sun Mar 22, 2020 2359 hrs

1. Fork the repo **ict2105-lab07-2020**, and then clone it. Inspect the code within the project. This code is incomplete as usual.
2. Implement code that will access the RESTful API URL and download asynchronously.
3. Implement the Java version of the factorial function both iteratively and recursively.
4. Implement the native C++ version of the factorial function both iteratively and recursively, in a separate library loaded and called via JNI.
5. Add timing code that will measure the amount of time taken for a call to the factorial function to complete.
6. Display the result and timing of the factorial computation on the TextView and Toast box respectively.

7. Choose a sufficiently large value of n (not too large), the input value to the factorial function. Observe the running time of each of the 4 functions for the same value of n.
8. Commit and push all changes to your forked repository **ict2105-lab07-2020.**

END OF DOCUMENT