Step 1. Android Programming

In this step, you are required to write an Android App and build your signed app. You should get familiar with Android programming, Android Studio and basic Android knowledge after this lab.

Tasks

Task 1. What's in Background

KEY: Android component, Service, Receiver, Permission, Thread, Handler

There are always lots of services start with the system and run in background when you use your phone, in this task you are going to let your application gain this ability.

You need to finish all the requirements below:

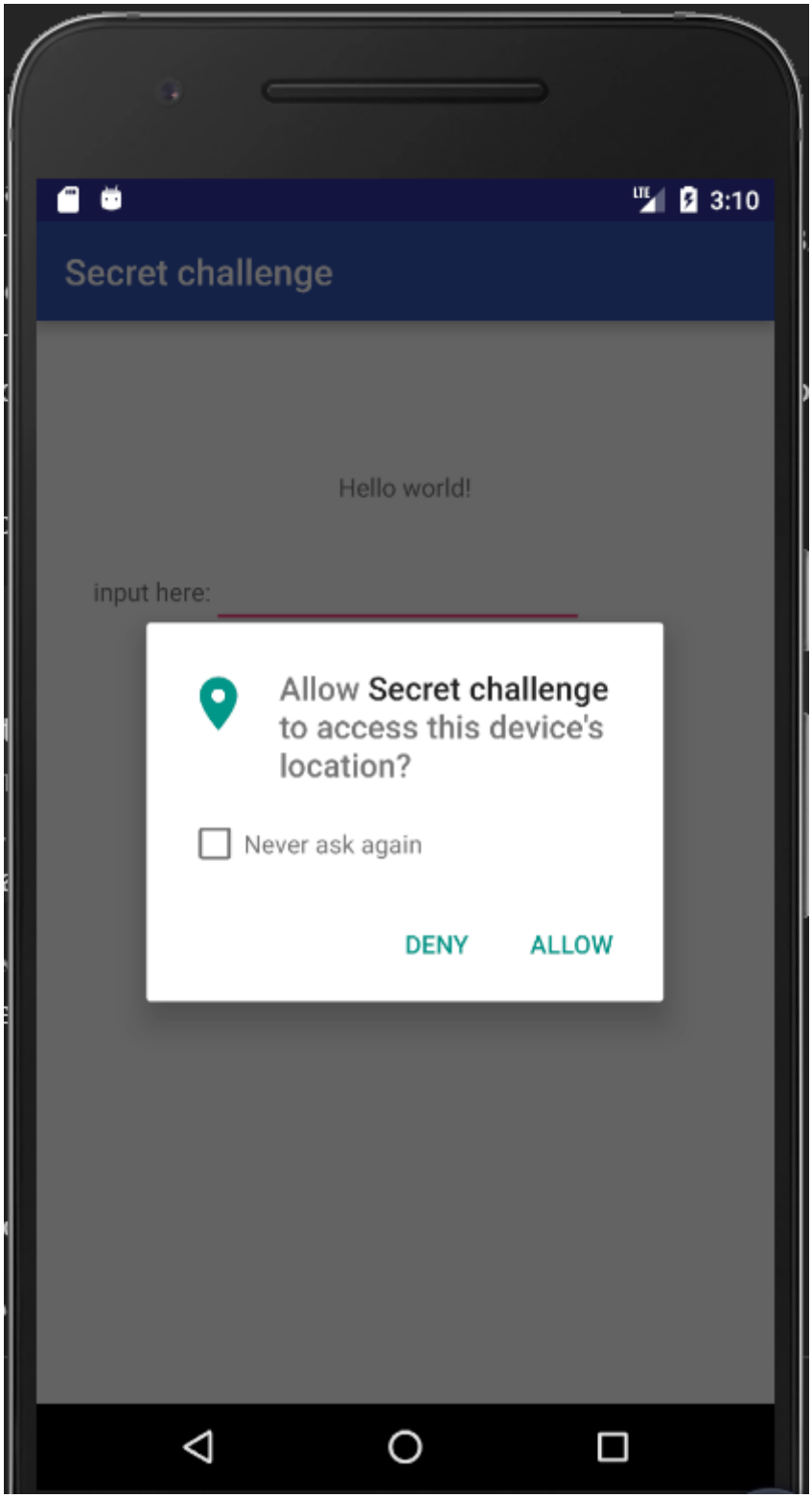
1. Create a new receiver class named `SecretBootReceiver` in package `com.smali.secretchallenge`.
   The function of this receiver has to
   - Let the application **autostart** when Android System start up.
   - Start a service in the background without any user operation.

> #### Tips
>
> 1. You can use `adb shell am broadcast -a android.intent.action.BOOT_COMPLETED -p com.smali.secretchallenge` to pretend the device is just reboot. You can use the **Terminal** window below the Android Studio to send this adb command.
> 2. Another useful tool is **Logcat** window which is also below the Android Studio. You can see the log information and you can also filter the log information with some key words.

2. The service started by `SecretBootReceiver` should be named `SecretService` and also located in package `com.smali.secretchallenge`.
   The function of this service has to
   - Get the **GPS location** of this device.
   - Toast the **Latitude** and **Longitude** on the screen every 3s.
     Example

3. Generally, app's permissions are requested before running. However, some sensitive permissions like Location permission should be requested from the user since SDK 23. In this task, if our app does not request location permission from user, it can't get location data from system. Therefore your task is to request location permission in `MainActivity` located in package `com.smali.secretchallenge` as soon as the app launches.

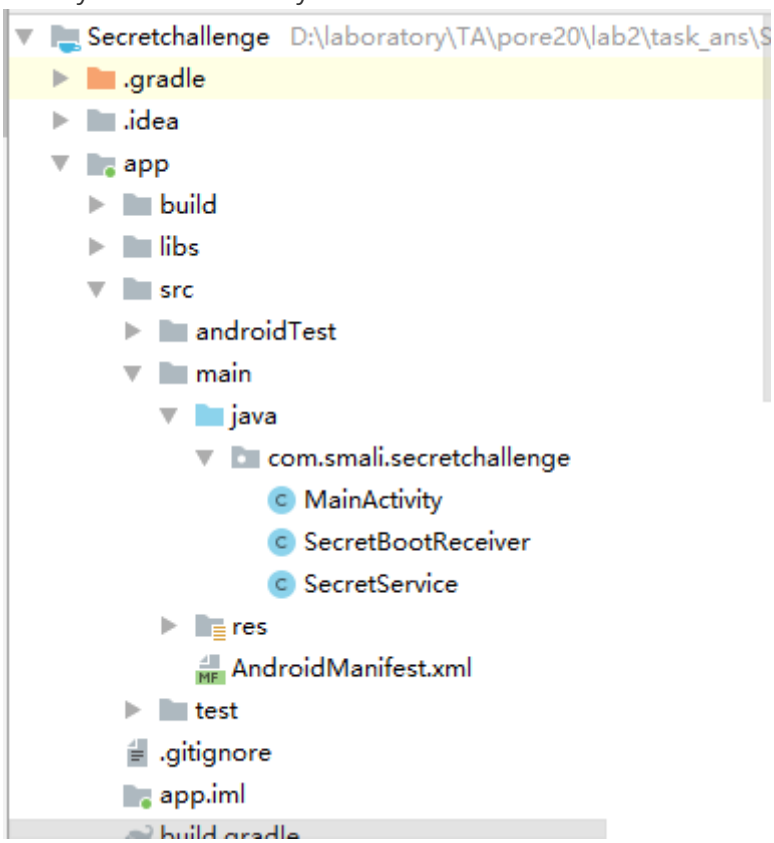
   Your app will look like the picture below:



Tips

3. If you have not finished the task of dynamic requesting permission, you can temporarily set the app with Location permission manually. Find **Settings->Apps->Secret challenge->Permissions**, and you can see the Location permission is off, turn on it and you can get Location data now.

Note that there should only be three classes under the package `com.smali.secretchallenge` when you have done all your task.



Note:

- Never forget to add permission you need.
- Search engine and official document can always help you a lot when you meet truobles.
- See the log to get error info.
- Make your application more stable

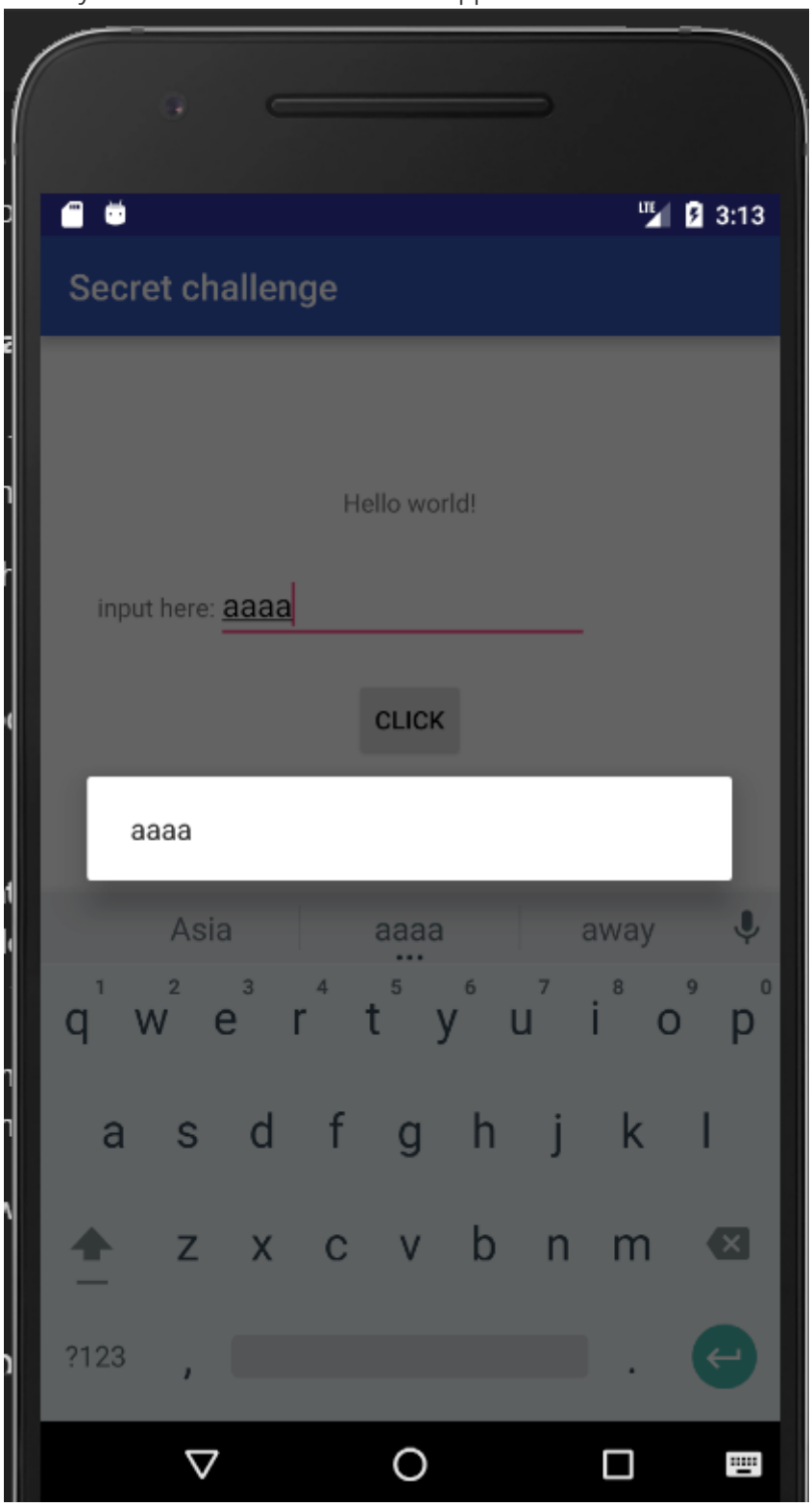
Task 2. Change UI in the thread.

KEY: Asynchronous message transfer mechanism, Looper, Message,Handler

In this task you need to change the UI in an thread. Actually we are not allowed to change the UI in the other thread. We can only change it in the main thread. So you need to figure out a way to solve this.

Specifically, you need to make a **Dialog** when the button is clicked and show the content that you just put in the text. **Notice**, the code in current project that setting the textview is not this task target. That code is just aimed to show you that the app will crash if you want to change the UI of the main thread in child thread without some effort.

When you finish this task and run the app it look like this.



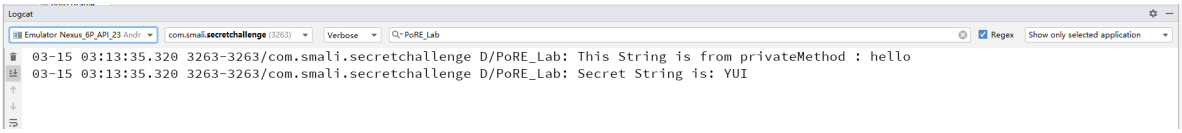
Task 3. How to call that method?

KEY: Java Reflection

As we all know, the keywords **private** is aimed to protect the fileds or the methods from visiting outside the class. But that does not always work because of the Java Reflection. So in this task you are supposed to learn this mechanism and do the following in `MainActivity` located in package `com.smali.secretchallenge` :

- Get a private field's value named curStr in PoRELab class of classes.jar.
- Invoke a private method named privateMethod in PoRELab class of classes.jar.

Your prove of work will be logged with a log function which can be observed in **Logcat** window menthoned above.



Task 4. Generate Signed Application

Now the app you test is debug version, if you want to protect your app and publish to some other palce, you have to sign it.

1. Click **Build->Generate Signed APK** in Android Studio.
2. Create your own Key Store at anywhere you prefer.
   Remember the password.
3. Choose your key store, enter your password, click **Next**.
4. Choose APK Destination Folder, use your student ID to name the apk, click **Finish**.
5. Just wait and check the apk after generated.

Step 2. Smali2Java

In this step, you are required to learn the skill of reversing smali code and converting smali code to Java.

Tasks

In this part, you are required to read the given smali code and write java programs accordingly. Your Java code should implement the same functions as smali.

Smali code files: DOWNLOAD

Task 1. Checker

1. Compile and run the smali code as you learned in step 2. One thing you have to pay attention to is that because you need to run more than one smali files together in this lab, when you try to compile smali to dex, put all the files you need in one folder and compile them together using command `java -jar smali-2.1.3.jar -o dexName.dex FOLDER`.

   Input a String, and the CheckBox will check whether the string passes the check in class Checker and output true/false accordingly.

   This is my output in smali:

   ```
   root@generic_x86:/data/local/tmp # dalvikvm -cp Box.dex CheckBox
   input: abc
   Task 1: false
   ```

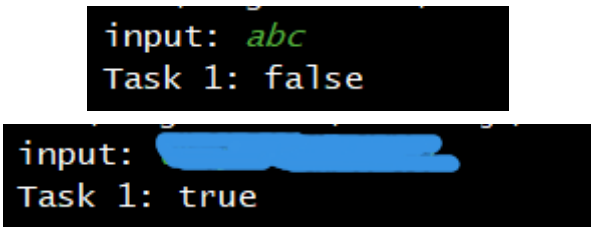
   ```
   root@generic_x86:/data/local/tmp # dalvikvm -cp Box.dex CheckBox
   input:
   Task 1: true
   ```

2. Read **Checker.smali** and understand the processing procedure of it. (The following is a part of Checker.smali)

```
.class public LChecker;
.super Ljava/lang/Object;
.source "Checker.java"


# static fields
.field private static secret:Ljava/lang/String;


# direct methods
.method static constructor <clinit>()V
    .registers 1

    .prologue
    .line 2
    const-string v0, "key"

    sput-object v0, LChecker;->secret:Ljava/lang/String;

    return-void
.end method

.method public constructor <init>()V
    .registers 1
```

3. Write the Java code to implement the functions as **Checker.java**.
4. Once you finish your Java code, you can check whether your code goes correctly by running the Main method in CheckBox.java and comparing the output in Java with that in smali.

   This is my output in Java:



Task 2. Encoder

1. Compile and run the smali code as in Task1.
   1. Input your student ID, and the CheckBox will output the encoded msg.

```
root@generic_x86:/data/local/tmp # dalvikvm -cp CheckBox.dex CheckBox
input: 18307130341
Task 2: (Encoded msg) 419607119a0990c11ef02e0ae0c406c1a715407c12d0bb15
```

   2. Use the encoded msg in 1.1 as the args of CheckBox and input your Student ID, the CheckBox will check whether the msg passes the check in class Encoder and output true/false accordingly.

   This is my output in smali:

```
alvikvm -cp CheckBox.dex CheckBox 419607119a0990c11ef02e0ae0c406c1a715407c12d0*
input: 18307130341
Task 2: true
```

```
alvikvm -cp CheckBox.dex CheckBox 419607119a0990c11ef02e0ae0c406c1a715407c12d0*
input: 18000000000
Task 2: false
```

2. Read **Encoder.smali** and understand the processing procedure of it.
3. Write the Java code to implement the functions as **Encoder.java**.

4. Once you finish your Java code, you can check whether your code goes correctly by running the Main method in CheckBox.java and comparing the output in Java with that in smali.

```
input: 18307130341
Task 2: (Encoded msg) 81ac18105e0d90e700c0191ed0441bc06f11a1d100d0940c
```

```
input: 18307130341
Task 2: true
```