Assignment 4: Android Development

In this assignment you will build a simple Android app similar in spirit to the one shown at the end of the demo for the Android lesson. The app is a cipher tool that:

- Takes as input:
  - A *Message* to be encoded.
    - This input should be a **non-empty string** and must contain **at least one letter**.
    - This input should be provided to the app through an EditText widget, **initially blank**.
  - A *Key Number*.
    - This input should be an **integer >= 1 and < 26.**
    - This input should be provided to the app through EditText widget, **initially set to '0'**.
  - An *Increment Number*.
    - This input should be an **integer >= 1 and < 26.**
    - This input should be provided to the app through EditText widget, **initially set to '0'**.
- Produces as output
  - The *Ciphertext*, which is the string resulting from using the specified key letter and skip number to encrypt the input string.
    - The first letter in the message (a-z, A-Z) will be shifted by the number of characters specified by the *Key Number*, as in a Caesar Cipher.
    - Each following letter in the message will be shifted by a new key number, provided by adding the *Increment Number* and preceding key number.
    - All non-alphabetic characters remain unchanged.. Capitalization from the original message is preserved.

    As described on the linked page, a Caesar cipher shifts each letter in the original *Message* by a number of places along the alphabet which depends on the corresponding key (so an A plus 3 is a D).
    EXAMPLE:
    "Cat & Dog" shifted according to the Key Number 3 and Increment Number 2 would be "Ffa & Mzt" (C+3, a+5, t+7, ' ', &, ' ', D+9, o+11, g+13).

    This output should be shown using a non editable text field initially blank and (re)computed automatically whenever the *Encode* button is pressed. **If any input is invalid when the button is pressed, the output should then be set to "" (i.e., the empty string), and ALL applicable error messages should be set.**

Error Messages

If the user provides an **invalid** input (see Item 1 above for what constitutes correct input values), the app should display an error message on the appropriate EditText widget using the error capabilities provided by the EditText widget. The error messages should be **exactly** (1) "**Invalid Message**", applied to the *Message* field, (2) "**Invalid Key Number**", applied to the *Key Number*

field, or (3) **“Invalid Increment Number”**, applied to the *Increment Number* field for (1) an empty or letterless string *Message*, (2) a blank or out of range *Key Number*, and (3) a blank or out of range *Increment Number*. If you do this correctly, the results should be a floating text error message (whenever the field has focus), with error mark **“”**, right next to the EditText , as shown in the error Screenshots below. If you have set multiple applicable errors, then all error marks will show, but the error message will only show whenever the field has focus. You may limit the input in the numerical fields to positive numbers, or provide the same errors for negative numbers and non-numerical input.

Below are several (slightly cropped) screenshots of the app that provide examples of normal behavior and errors. You should also treat the screenshots as example test results and use them to check that your app behaves as intended.

We suggest that you try to keep your user interface (UI) similar to the one shown, but you don’t have to. However, **you must make sure to use exactly the same identifiers that we show in the next figure for the key widgets in the UI**. This is very important, as we will use these identifiers to automatically test your app. The labels shown in the example UI are optional, but if included they must be separate from your required input and output fields. The identifiers are also listed next to the next figure for your (copy-and-paste) convenience.

Identifiers:

- `"messageInput"`
- `“keyNumber”`
- `“incrementNumber”`
- `“encryptButton”`
- `"cipherText"`

For example, in the XML layout file for your app, the entry for the text field used to input the *Message* should have the following ID: `android:id="@+id/messageInput"`.

Please note that, as described in the detailed instructions below, we took some steps to help you make sure that you used the right identifiers and strings.

Detailed Instructions

To complete the assignment you must perform the following tasks:

- In the root of **your assigned individual GitHub repository**, create a directory called `Assignment4`. Hereafter, we will refer to this directory in your local repo as `<dir>`.
- Using Android Studio, create an Android app project called “SDPEncryptor” in `<dir>`. Check that this generates a directory SDPEncryptor under `<dir>` as the save location.

Ensure company domain is entered as "seclass.gatech.edu", which will result in the package **`edu.gatech.seclass.sdpencryptor`**. If your project set up asks for the package name instead, simply enter **`edu.gatech.seclass.sdpencryptor`**. Verify that the package name is correct after your project is created.

- Select "**API 23: Android 6.0 (Marshmallow)**" as the **minimum sdk** for your app.

  This `minSdkVersion` should be reflected in your build.grade file in the project.
- Create an "Empty Activity" and call it `SDPEncryptorActivity`
- Download and save the archive available here, which contains three files provided to help you make sure that you used the right identifiers and strings, as we mentioned above. **These files will not check everything for you**, but if you use them, they will prevent many simple errors. The files, which we describe how to use in the rest of the steps, are:
  - **`SanityCheck.java`** prevents your app from compiling if certain identifiers, your activity name, or your package name are incorrect.
  - **`strings.xml`** provides some constant strings that you can reference to avoid typos in error messages. You are welcome to edit this file or add additional strings to it, but please keep in mind that **our tests will expect the identifiers and error messages to match the ones provided, exactly**.
  - **`AssignmentExamples.java`** contains Espresso tests similar to the tests we will run on your code for grading.
- Extract `strings.xml` from the archive and copy it to your project at `<dir>/SDPEncryptor/app/src/main/res/values`
- Implement the primary functionality of the app in `SDPEncryptorActivity`. Note: You will need to make any function that your button calls 'public', not 'private' in order to use the Espresso tests.
- Define the IDs for the key widgets in the app as described above.
- Extract `SanityCheck.java` from the archive and copy it to your project at `<dir>/SDPEncryptor/app/src/main/java/edu/gatech/seclass/sdpencryptor`
  - Rebuild the project in Android Studio.
  - If the project does not compile, and you have added all the required identifiers to your layout, it should mean that there are issues with your activity name/package, your identifiers, or both. **Make sure to correct your project, and not the provided `SanityCheck.java` file.**
- Extract `AssignmentExamples.java` from the archive and copy it to your project at `<dir>/SDPEncryptor/app/src/androidTest/java/edu/gatech/seclass/sdpencryptor`
  - Add the necessary dependencies to your build.gradle, including:

```
androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
androidTestImplementation 'androidx.test:runner:1.1.1'
androidTestImplementation 'androidx.test:rules:1.1.1'
```

  - Rebuild the project in Android Studio
  - Run the tests by either setting up a new run configuration or right clicking the file and choosing "Run 'AssignmentExamples'".
- Create a `manual.md` file (in Github flavored MD format) that describes how to use the app. Put the file in `<dir>`, not in `SDPEncryptor`. Think of this file as a (very concise) user manual. You should not need more than one page for the manual. **It should use at least some markdown formatting.** Feel free to add screenshots to the manual, but that is not mandatory. You can view your markdown file in Github or using an MD viewer.

- Commit and push your project from within Android Studio (or from the command line, if you know what you are doing). Doing it from Android Studio should help ensure that all the required files are committed. **Be sure to use your existing, assigned repository, rather than creating a new repository from Android Studio.** No matter how you commit and push your project, we strongly recommend that you (1) clone your repo in a different directory, (2) open the project in Android Studio from this new directory, (3) compile the project, and (4) run it on a (virtual) device that does not already have the app installed (i.e., you may have to remove the app from the device if you have run the app there before).
- As usual, submit your solution by doing the following:
  - Pushing your code to your assigned remote GitHub repository.
  - Submitting the final commit ID for your submission on Canvas. (You can get your commit ID by running "git log -1" and copying the hexadecimal ID it produces.)

Notes

- **You should commit early and often**. Verify that you are able to correctly push your code to the assigned repository as soon as possible. You can perform multiple commits and work on multiple branches as you produce your solution. This is not only fine, but actually encouraged. Just make sure that your final solution is committed to the `master` branch.
- You should complete the assignment using Android Studio 3.*. Earlier versions of Android Studio may work as well, but we have not tested our solution there.
- Feel free to **take inspiration** from online resources when developing your app. However, be careful not to copy and paste entire pieces of functionality. The tool we use to identify cases of plagiarism is likely to have access to the same online resources that are available to you.
- **Do not add field labels or any extra text into the designated EditText fields.** Each EditText field should contain only the relevant input or output. You may use other fields for optional labels or UI elements.
- In case you want to use automated testing for your app, you may find Espresso and either Barista (documentation here) or Android Studio Test Recorder useful. This is completely optional; that is, developing tests for your app, whether manual or automated, is not required.
- If you decide to use Espresso yourself, or run our provided tests, here's a couple of potentially useful tips:
  - If anything covers your fields or buttons (even invisible boxes), the tests may fail to complete.
  - If you use buttons that call private methods, Espresso will be unable to click the button.
  - Turn off animations in your AVD, particularly if the tests return `AmbiguousViewMatcherException`.
  - You may need to turn off the autofill feature or spellcheck in your AVD if your tests fail due to auto-completion of text input.