Assignment 4: SDPEncryptor

Summer 2025

In this assignment, you will develop a simple Android app, *SDPEncryptor*, that encrypts messages using a simple [affine cipher](https://en.wikipedia.org/wiki/Affine_cipher). Before you make an ed post or communicate with your peers about this assignment, read [‘Guidelines for Communicating’](#_mn8ur8uv0qnv) near the end of this assignment spec; **failure to follow said guidelines may result in a points deduction**. Configuring your Android Studio development environment is itself a key learning objective; it lays the groundwork for the group project. Spending 70 to 90 % of your time on this setup is normal.

## INPUTS:

1. *Message Text:* message to be encoded.

* This input should be a **non-empty string** and must contain **at least one letter or number**.
* This input should be provided to the app through an [EditText](https://developer.android.com/reference/android/widget/EditText) widget, **initially blank**.

1. *Scale Input:* first encryption parameter.

* This input should be an integer [coprime](https://en.wikipedia.org/wiki/Coprime_integers) to 62 between 0 and 62: **1, 3, 5, 7, 9, 11, 13, 15, ...**
* This input should be provided to the app through an EditText widget, **initially set to '1'**.

1. *Shift Input:* second encryption parameter.

* This input should be an **integer >= 1 and < 62.**
* This input should be provided to the app through an EditText widget, **initially set to '1'**.

## OUTPUT:

1. *Encoded Text*, the text resulting from applying the following cipher:

* Each character in the alphabet is assigned a numeric value between 0 and 61 based on its position in the alphabet (i.e., "0"=0, "1"=1, ... "9"=9, "a"=10, "b"=11, ... "z"=35, "A"=36, ..., "Z"=61). Note that the alphabet contains letters and numbers.
* For each character in the alphabet, where the numeric value is , the encoded value of the letter is defined as where and are the values of Scale Input and Shift Input, respectively, as in an [Affine Cipher](https://en.wikipedia.org/wiki/Affine_cipher).
* The encoded character for the input character is calculated by taking the encoded number, which is a value between 0 and 61, and translating it back into a character (again, where "0"=0, "1"=1, ... "9"=9, "a"=10, "b"=11, ... "z"=35, "A"=36, ..., "Z"=61).
* All non-alphanumeric characters must remain unchanged.
* The output should be shown using a *non-editable* [TextView](https://developer.android.com/reference/android/widget/TextView) that is initially blank and (re)computed when the "Encode Message Text" 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 generated (see below).

## EXAMPLE

* Inputs:
  + *Message Text* = "Cat & 5 DogS"
  + *Scale Input* = 5
  + *Shift Input* = 3
* Output:
* *Encoded Text* = "7Ro & s cZlp"
* Explanation:
  + "C" has a value of 38, (38 \* 5 + 3) % 62 = 7, 7 corresponds to "7".
  + "a" has a value of 10, (10 \* 5 + 3) % 62 = 53, 53 corresponds to "R".
  + …
  + " ", "&", " " are unchanged.
  + "5" has a value of 5, (5 \* 5 + 3) % 62 = 28, 28 corresponds to "s".
  + " " is unchanged.
  + …

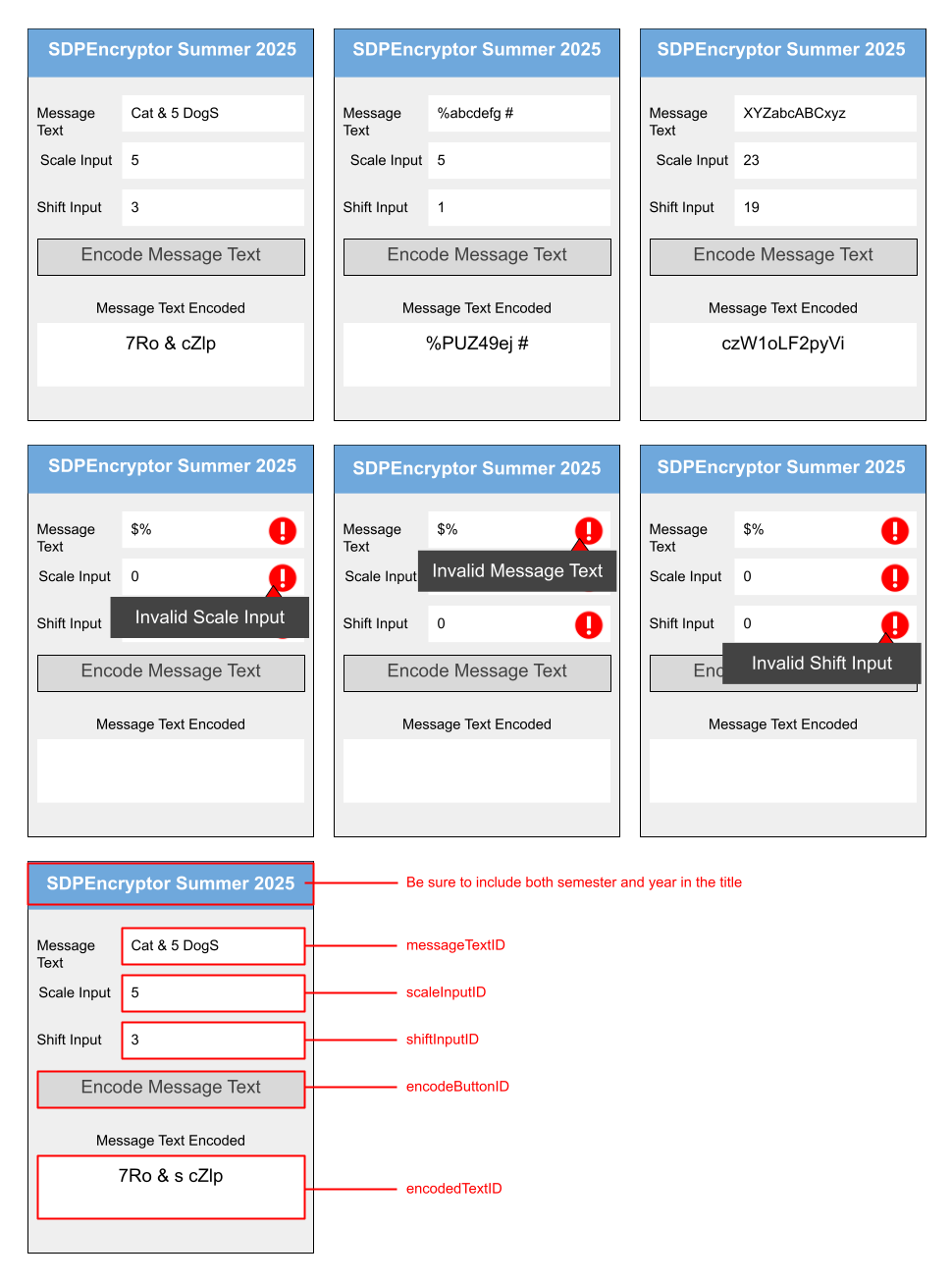
## ERROR MESSAGES

The app should generate suitable error messages by calling [EditText's setError method](https://developer.android.com/reference/android/widget/TextView.html#setError(java.lang.CharSequence)) (inherited from TextView) on the appropriate EditText widget when the computation is triggered (i.e., the button is pressed). If done correctly, this will result in (1) an error mark () on the right-hand side of the text field and (2) a floating error message whenever the field has focus, as shown in the error screenshots below. It is possible to have more than one error active at the same time, as shown in the screenshots below.

There are three error situations:

1. "Invalid Message Text", related to the *Message Text* field, for any entry both letter-less and number-less. (Examples: “”, “&$%#^$&^”)
2. "Invalid Scale Input", related to the *Scale Input* field, for a blank or unacceptable value  
   (i.e., not coprime to 62).
3. "Invalid Shift Input", related to the *Shift Input* field, for a blank or out-of-range value.

For illustration, we are providing several mockups for a possible implementation of the app:



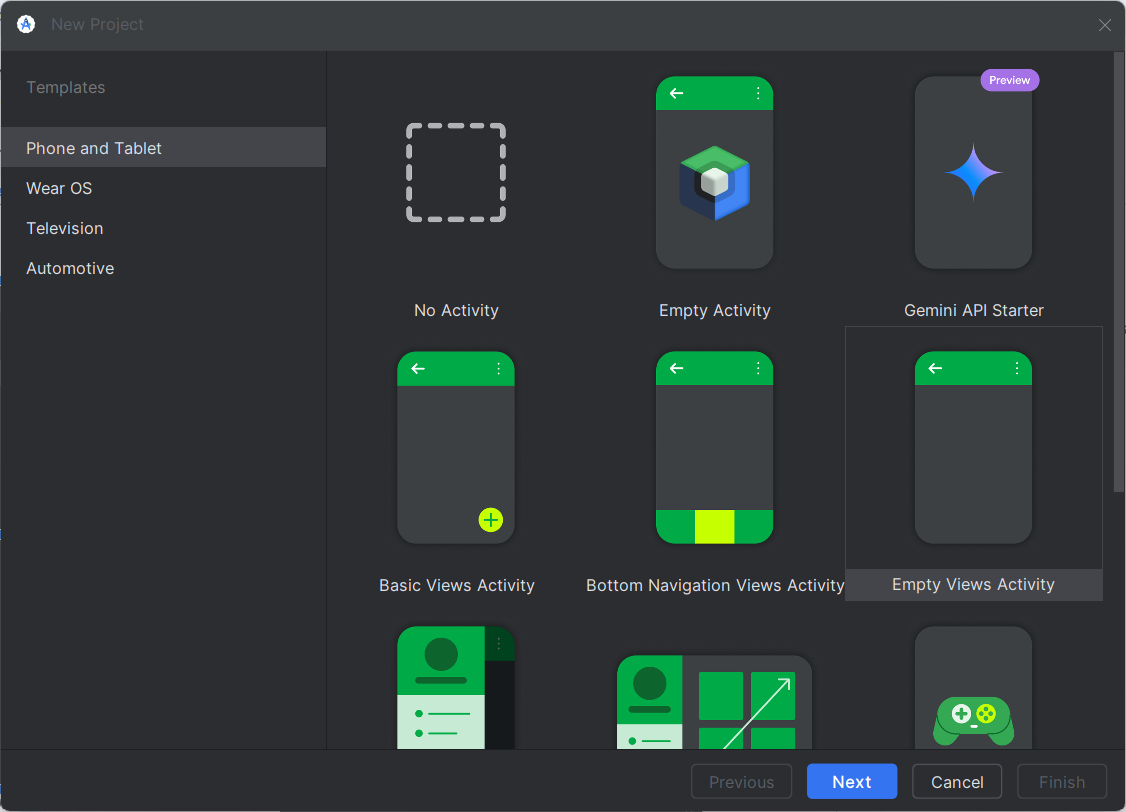
We suggest that you try to generate a user interface (UI) similar to the one shown above, but you don't have to. However, you must make sure to use the exact same identifiers we provide below for your widgets. This is very important, as we will use these identifiers to check and auto-grade your app.

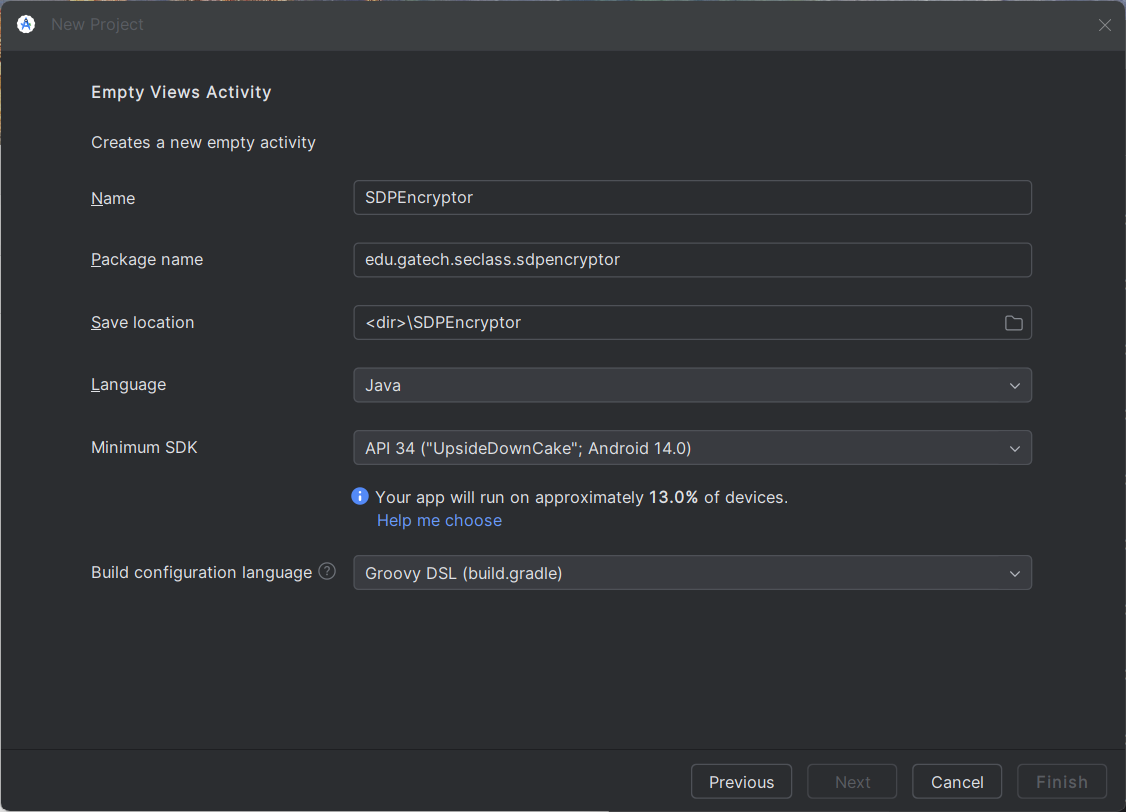
## IDENTIFIERS

| * messageTextID * scaleInputID * shiftInputID * encodeButtonID * encodedTextID | ***PRO-TIP: The identifiers are typed here for you. Just copy and paste them to be safe!***  For example, in the XML layout file for your app, the entry for the text field used to input the *Message Text* should have the following ID: android:id="@+id/messageTextID" |
| --- | --- |

## INSTRUCTIONS[[1]](#footnote-0)

1. Make sure to watch the [Android Studio demo](https://www.youtube.com/watch?v=jqB3r_16WqA) (part of the lessons on Android).
2. In the root of the assigned private GitHub repository that we made for you (https://github.gatech.edu/gt-omscs-se-2025summer/6300Summer25<your GT username>.git), create a directory called "Assignment4". Hereafter, we will refer to this directory in your local repo as <dir>.
3. Copy and paste (appending) the following into your .gitignore file in the root (<dir>/..)of your repo:  
    \*.iml  
    .gradle  
    local.properties  
    .idea/workspace.xml  
    .idea/libraries  
    .idea  
    .DS\_Store  
    build  
    captures  
    .externalNativeBuild
4. Create an Android app project called "SDPEncryptor" in <dir>. (**Make sure that this results in a directory called SDPEncryptor in <dir>**, as shown in the "Configure Your Project" snapshot below--you will likely need to manually modify the "Save location" entry for this to happen.)
   * Choose "Empty Views Activity" as your project template.



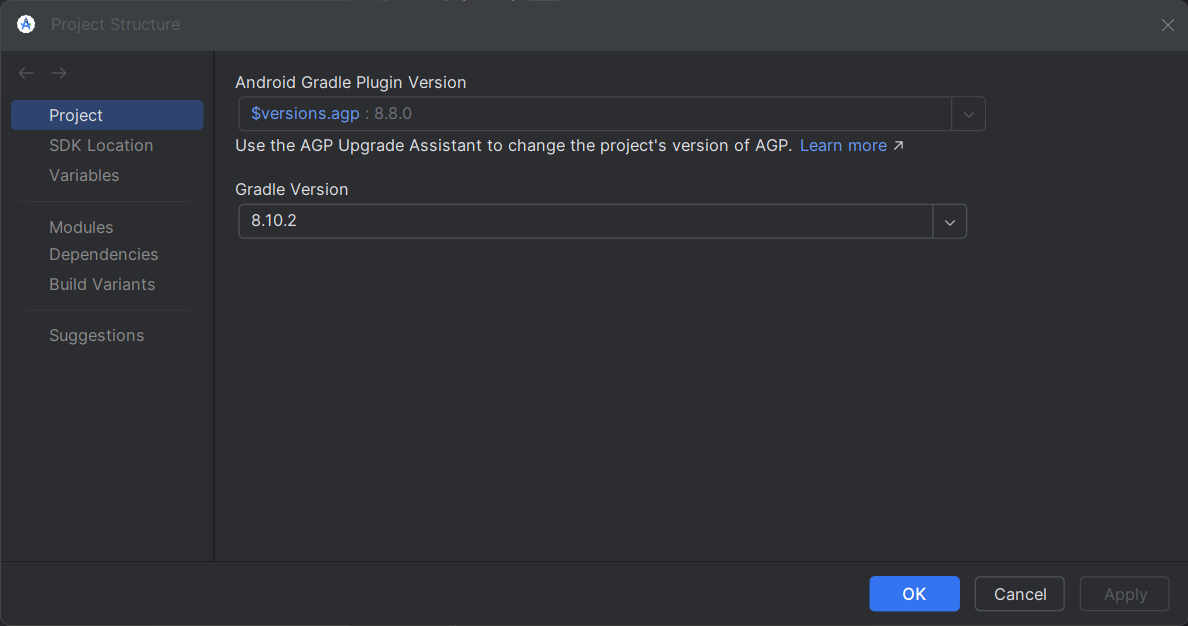


Note the package name: "edu.gatech.seclass.sdpencryptor"

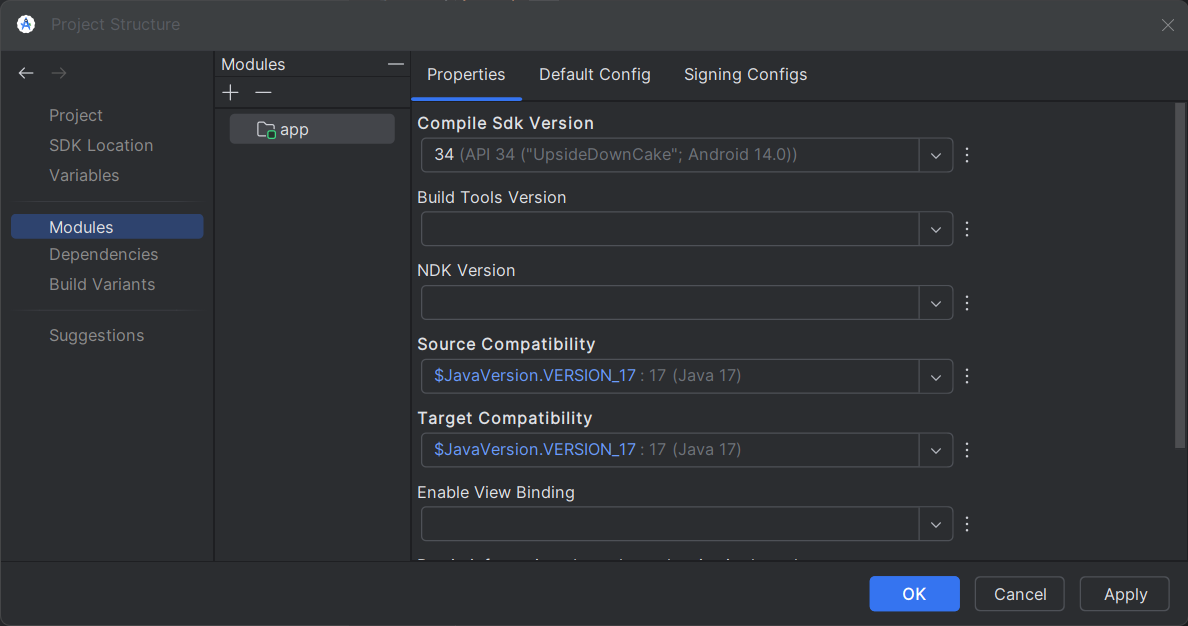
Note the language: “Java” (Kotlin is allowed, but our ability to support problems you may encounter will be limited)  
 Note the minimum SDK: "API 34: Android 14"

Note the build configuration language: “Groovy DSL (build.gradle)” ***(You must use this exact build configuration language)***

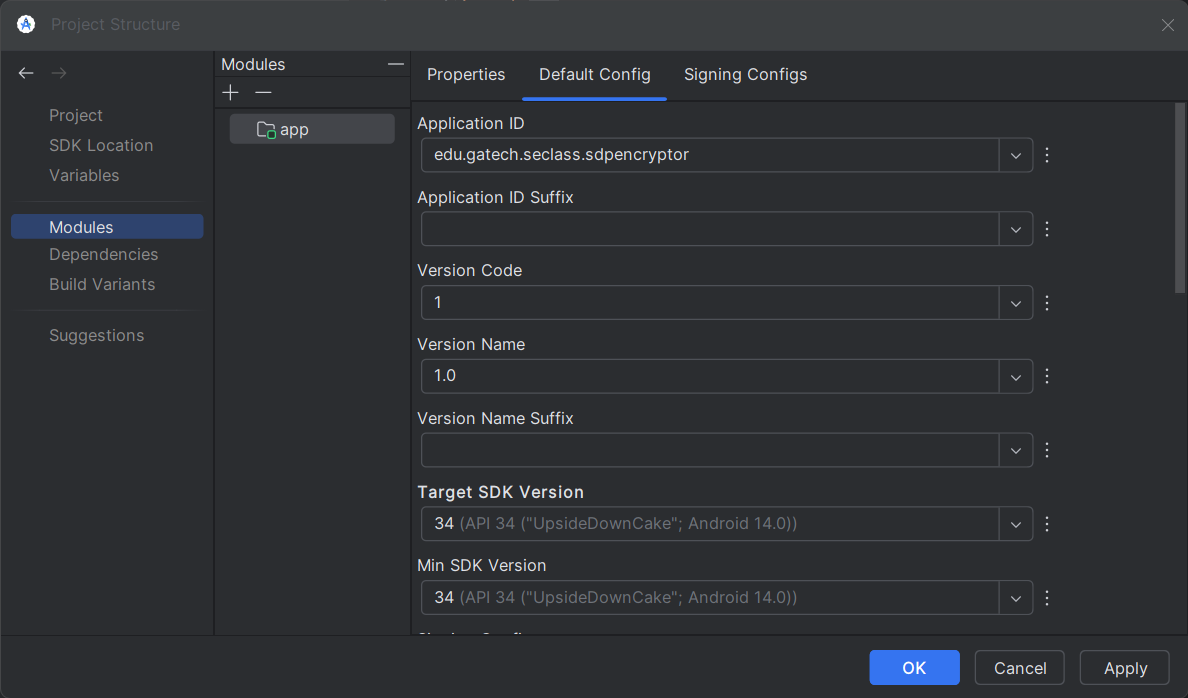
1. You should test your app against a virtual “Pixel 6 with API 34 (UpsideDownCake) Installed.” You can add this device in the ‘device manager’ view.
2. Ensure you are using Gradle and Android Gradle Plugin versions compatible with Java 17. Go to File > Project Structure > Project and select 8.8.0 for the Android Gradle Plugin Version and 8.10.2 for the Gradle Version



1. Choose Modules in the Project Structure dialogue, and check to see that you are using a Compile SDK Version of 34 and Source Compatibility and Target Compatibility of Java 17.



1. In the same dialogue, choose the Default Config tab and ensure the Target SDK version is 34.



1. Check the build.gradle file for ***module app*** to see that the settings were applied:

android {

**// ...**

**compileSdk 34**

**defaultConfig {**

**// ...**

**minSdk 34**

**targetSdk 34**

**// ...**

**}**

**// ...**

**compileOptions {**

**sourceCompatibility JavaVersion.VERSION\_17**

**targetCompatibility JavaVersion.VERSION\_17**

**}**

}

1. Make sure that your build.gradle file for ***module app*** contains the following dependencies (**including the specific versions listed**):

implementation 'androidx.appcompat:appcompat:1.4.1'

implementation 'com.google.android.material:material:1.5.0'

implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

testImplementation 'junit:junit:4.13.2'

testImplementation 'org.robolectric:robolectric:4.11.1'

* + If some dependencies are missing, make sure to add them.
  + There may be newer versions of some of the dependencies available, but you should use the ones we recommend anyway.
  + You will have to rebuild the project if you added dependencies (the IDE should let you know through a banner notification).

1. To ensure compatibility with the tests used in autograding, add the following in the build.gradle for the ***module app***:

android {

// ...

**testOptions {**

**unitTests {**

**includeAndroidResources = true**

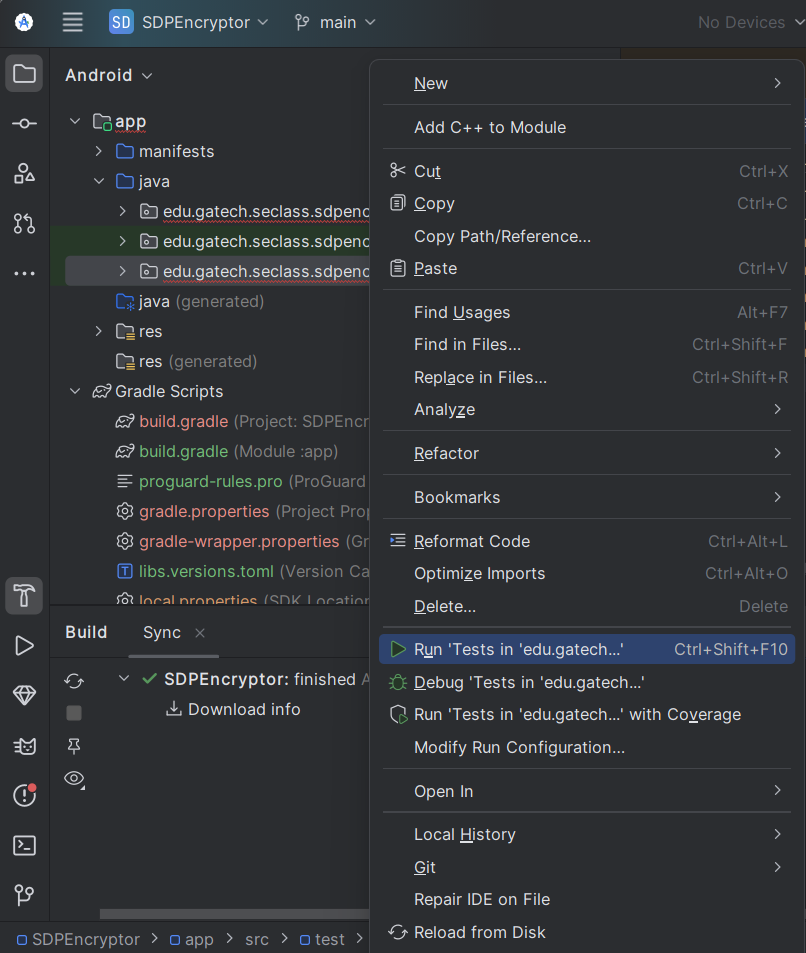
**}**

**}**

// ...

}

1. Implement your solution based on the requirements provided above using **Java 17**. Be sure your app includes both the semester and year in the Title as seen in the wireframes above.
2. **Mandatory:** **check your solution before submitting it by following these directions**. Download [this archive](https://drive.google.com/file/d/174icjdVJTLy5q8ThKeIeFb1APn3V8kOv/view?usp=drive_link) and unpack its contents in <dir>,[[2]](#footnote-1) which should create the following files:
   * SDPEncryptor/app/src/main/java/edu/gatech/seclass/sdpencryptor/SanityCheck.java: This file prevents your app from compiling if the identifiers, activity name, or package name are incorrect. **Do not edit this file. If compilation fails after you add this file it’s indicative of a bad project structure. Carefully follow direction 1-4 again.**
   * SDPEncryptor/app/src/test/java/edu/gatech/seclass/sdpencryptor/RobolectricViewAssertions.java: This file contains a helper library to run robolectric tests and enrich their output.
   * SDPEncryptor/app/src/test/java/edu/gatech/seclass/sdpencryptor/SmallTestExamples.java: This file contains a set of Robolectric tests similar to those we will run on your code for grading, for your convenience. Run the tests on the command line by executing "./gradlew test" in the project directory (you may have to install gradle to do so). During development and before submission, you can also run the tests from within Android Studio, by selecting "Run Tests in…" as shown below in the image to follow:



* + One key learning outcome of this assignment is to develop the skills to read, interpret, and resolve failed test cases. If you encounter failing test cases, it is *your responsibility* to determine the issue and implement a solution. Should you ask a question on ED, our guidance will often be consistent: "What do you understand about the output? Why do you think this assertion is being made? How does it align with the specifications?" If an assertion appears unexpected, your first reference should be the specifications rather than posting a query on ED. Below is an example of the "Expected" and "View Details" sections from the test failure output, along with some guidance on how to interpret them:
    - *Expected: "wBM8B lBG" but was: "v6SF6 b6B"*  
      *View Details: ID: 2131231064, res-name: messageTextID, visibility: VISIBLE, enabled: true, focusable: false*
    - In this case, the test is saying that the text in the field corresponding with the “*messageTextID”* identifier is "**v6SF6 b6B**" but should have been "**wBM8B lBG**". In this case, the most likely cause is that your affine cipher code isn’t working properly, as the cipher should be what populates that ‘*messageTextID’* field.

1. If your build fails with an error "Task :app:lint FAILED" or similar, you should be able to fix the issue by "inferring constraints" for your layout. Android Studio will do it for you automatically, just go to your activity\_main.xml file, hover over the error reported, and the IDE should give you the option to fix it. Alternatively, you can also disable the lint task for your project by adding the following to build.gradle:

android {

//...

lintOptions {

tasks.lint.enabled = false

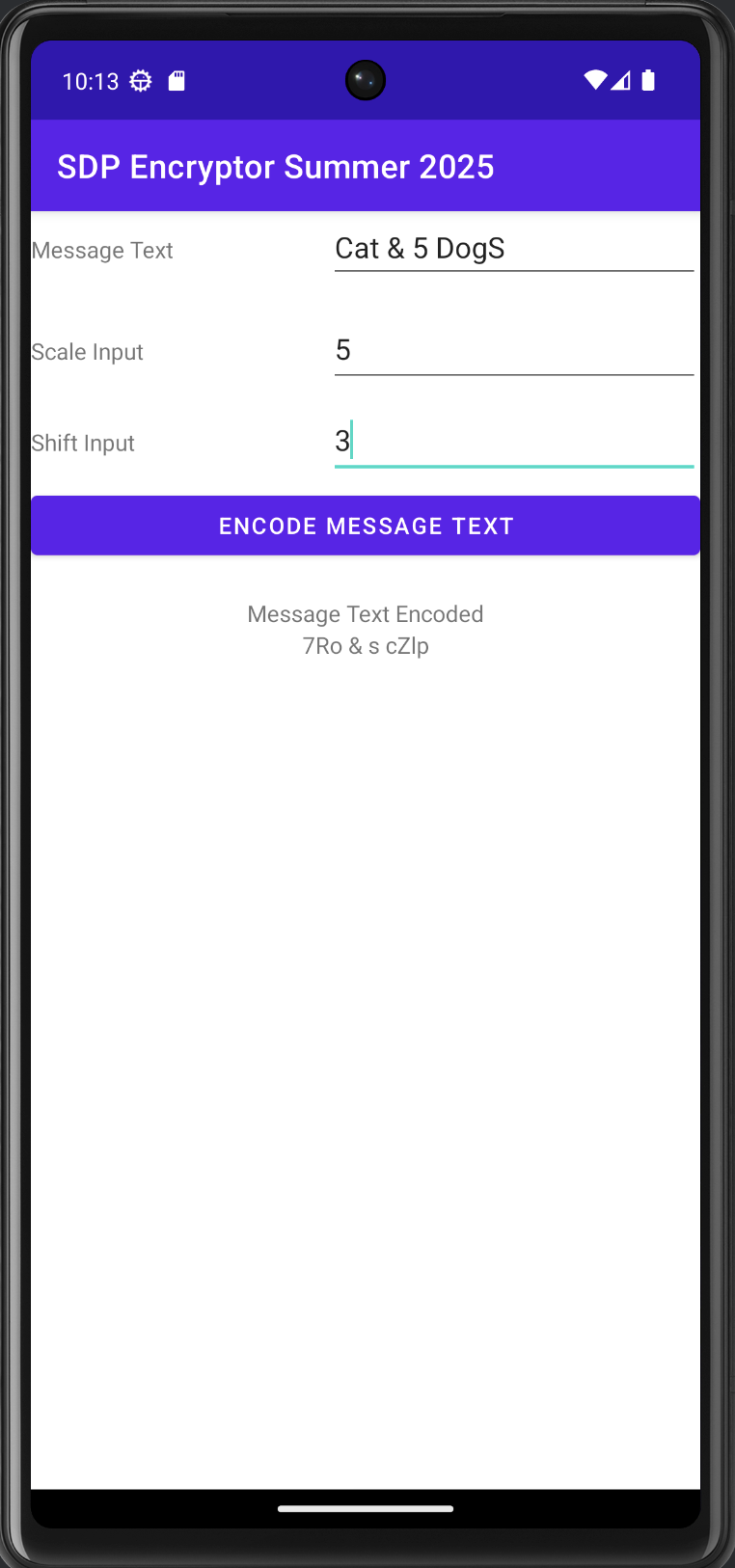
abortOnError false

}

}

1. For automated testing, check out [Robolectric](https://robolectric.org/). Here are some Robolectric tips, based on past experience:
   * Make sure your unit test config is set up so that Android resources are included. (See step 9 of these instructions)
   * Avoid using buttons that call private methods, as Robolectric may be unable to click them.
   * Avoid elements that cover your fields or buttons, even if these elements are invisible, as they may result in tests that fail to complete.
   * 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.
   * Do not add extra field labels or 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.
2. Commit the gradle-wrapper.jar, <dir>/SDPEncryptor/gradle/wrapper/gradle-wrapper.jar. If, for whatever reason, this file doesn’t exist run the following in <dir>/SDPEncryptor:   
    gradle wrapper  
   and include it in your private GitHub repository. If your .gitignore excludes jar files, you will need to force-add it.

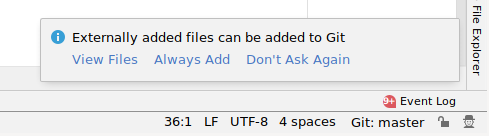
1. Include two screen captures of the build result of the app (app\_compile.png) and of the app running in the emulator (app\_run.png). Put both image files in your "Assignment4" folder (see below). The name must be exact and all lowercase. Every semester our image validation process has improved – if the auto grader tells you there’s a problem, there’s a high likelihood that there’s a problem – follow these steps carefully.
   * app\_compile.png must be taken from a stand-alone terminal/CMD (not Android Studio built-in terminal) and must show both the "./gradlew build" (or "gradlew build") command with "BUILD SUCCESSFUL", as seen from the example below. Ensure that you terminal window is 100% opaque (i.e not translucent/transparent/see-through), the color difference between background and the text is significant, the window size is wide enough, the font size is appropriate (not too big, not too small); all of this is to ensure that we can easily read the content of your screenshot. Failure to do so will result in a points deduction. Example Image:
   * app\_run.png must show the semester and year in the app title, as seen in the example below. Failure to adhere to the app title format text will result in a points deduction. Example Image:



Note: "SDPEncryptor" (without space) and "SDP Encryptor" (with space) are both acceptable

* + Hints for passing the image capture portion of the autograder (**Do not open an ED post** **asking us to prevalidate any images** – the previous instructions and what follows is the only advice we have on the matter): To pass the app\_run image process, consider factors like [contrast](https://support.google.com/accessibility/android/answer/7158390?hl=en), resolution, the size of the image. For the app\_compile image process, ensure you don’t have a partially transparent terminal, ensure you follow directions to the letter, it’s best if your view is large enough that you don’t have wrapping text, and lastly don’t take your image so close to the text block as to cut off edges or to make part of the output unreadable.

## SUBMISSION

1. Add/commit/pugsh your app to your assigned, *private* repo (the one we made for you). To do so, and make sure that all the necessary files are committed, do the following:
   * Go to <dir> and execute:
     + git commit -a
     + git push
2. **As an alternative to Item 1, and only if you know what you are doing,** you could also commit and push your app from within Android Studio, which should actually give you an option to do so:  
     
   **But please use the instructions in Item 1 if you need clarification.**
3. **Submit on Gradescope a file, called submission.txt that contains, in two separate lines (1) your GT username and (2) the commit ID for your submission.** For example, the content of file submission.txt for George P. Burdell could look something like the following:  
    submission.txt

| gpburdell1 |
| --- |
| 81b2f59 |

* **As soon as you submit, your assignment will be scored** by compiling it and running it against a set of test cases written by the instructors.
* **You can resubmit as many times as you want before the deadline**. However, we would not recommend a trial-and-error process, as Gradescope takes a non-negligible amount of time even to compile and check your code. You may want to complete and submit the assignment sooner rather than later and avoid rushing at the end (when the autograder may also be overloaded).

## 

## 

## GUIDELINES FOR COMMUNICATING

1. Should my post be public or private?
   1. You can only post publicly about environment-related questions (failure to launch Android Studio, the emulator, or build the project etc.) or questions related to failing *tests that you run on your local machine.* (eg. SmallTestExample)
   2. ***You cannot under any circumstance share or discuss any gradescope output publicly***, you may only do so privately. If you share output of failed tests on ED with your peers, the minimum deduction will be a failure of each test case shared. This includes discussing the tests by their ‘name’ in Gradescope. These names are there to help guide you, but not to be used in discussion.
   3. When in doubt, post privately – we’ll let you know if it can be public.
2. What should be in my private post?
   1. A link to the Gradescope result
   2. If you are asking about a failed autograder test, give us a demonstration that you:
      1. Tried to understand the test results (see [this section from the assignment spec](#xvfakow74ksh))
      2. Confirmation that you’ve taken that understanding back to the assignment spec to see if there is anything you missed.
   3. any information that may be relevant (e.g., inlined snippets of relevant code--no screenshots or commit IDs, please).

## PLAGIARISM

As for all other assignments, we check for plagiarism. Unfortunately, there are plagiarism cases every semester. Please keep in mind that the tools we use to identify cases of plagiarism have access to the same online resources that are available to you.

TL;DR (Checklist)

1. Build an Android App using **Java 17 and Android SDK Version 34**. You should test your app on a **Virtual Pixel 6 device with API 34 installed**.
2. Push everything to your assigned, *private* repo (the one we created for you).
3. Clone a fresh copy of your app and run the tests we provided on it.
4. Submit on Gradescope.
5. If your submission fails the Gradescope checks, it will not be graded and will receive a 0.

1. We performed our testing and created these instructions using [Android Studio Ladybug Feature Drop | 2024.2.2](https://developer.android.com/studio). Your windows may look slightly different from the ones we provide if you are using a different version. [↑](#footnote-ref-0)
2. Be careful when extracting the files, as a conservative program may create a second "SDPEncryptor" directory (e.g., "SDPEncryptor 2"). If that happens, make sure to move the files to their right location. [↑](#footnote-ref-1)