I need a simple Android app, *SDPEncryptor*, that encrypts messages using a simple substitution cipher.

INPUTS:

1. *Entry 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. *Arg Input 1:* 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, ... 61**
* This input should be provided to the app through an EditText widget, **initially set to '1'**.

1. *Arg Input 2:* 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. *Text Encrypted*, 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., "a"=0, "b"=1, ..., "z"=25, "A"=26, "B"=27, ..., "Z"=51, "0"=52, "1"=53, ..., "9"=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 Arg Input 1 and Arg Input 2 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 "a"=0, "b"=1, ..., "z"=25, "A"=26, "B"=27, ..., "Z"=51, "0"=52, "1"=53, ..., "9"=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 "Encrypt Entry 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:
  + *Entry Text* = "Cat & 5 DogS"
  + *Arg Input 1* = 5
  + *Arg Input 2* = 3
* Output:
* Text Encrypted = "tdK & O ylHL"
* Explanation:
  + "C" has value 28, , 19 corresponds to "t".
  + "a" has value 0, , 3 corresponds to "d".
  + …
  + " ", "&", and " " are unchanged
  + "5" has value 57, , 40 corresponds to "O”.
  + " " is unchanged
  + …

## ERROR MESSAGES

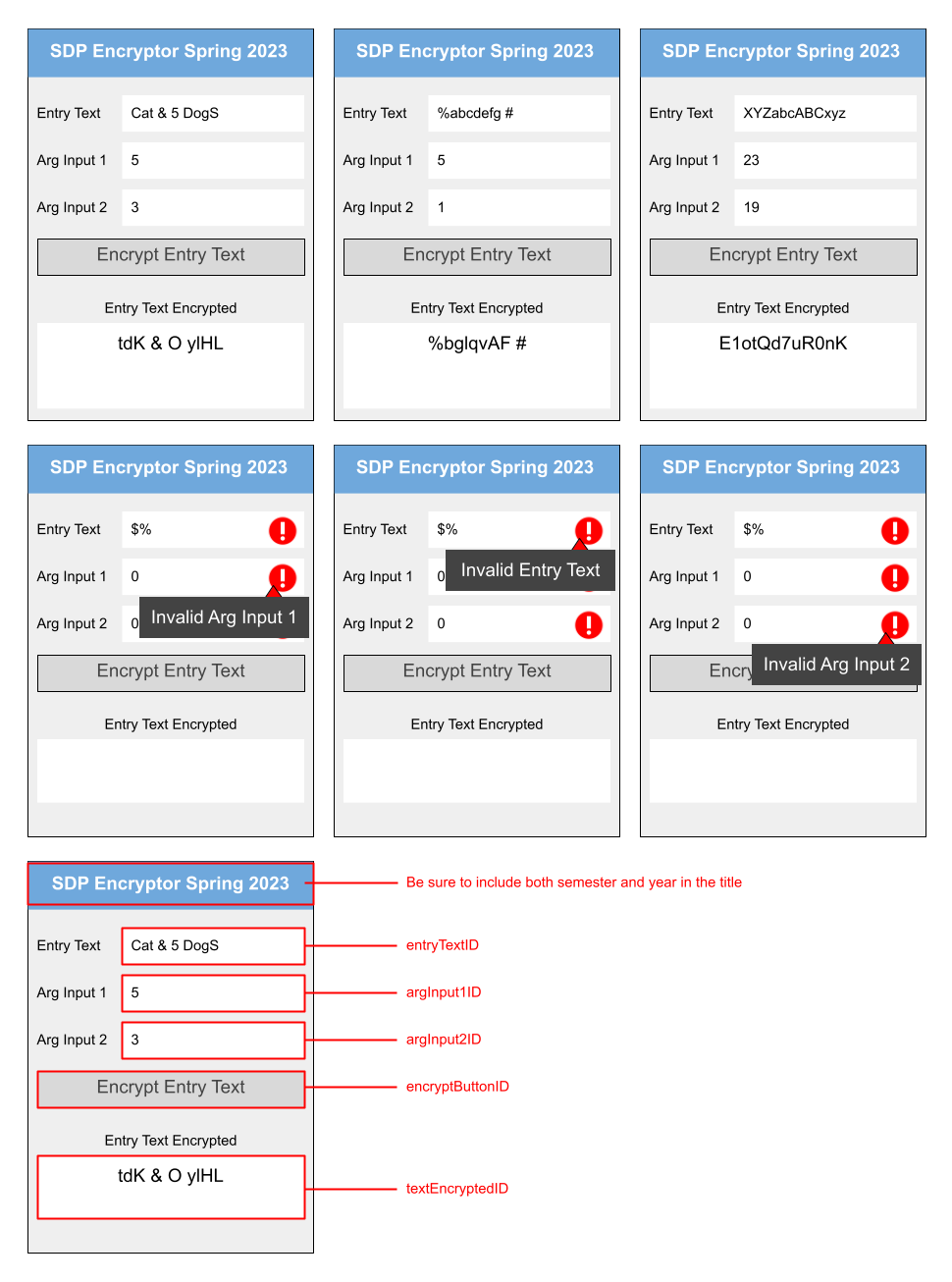
The app should generate suitable error messages by calling [EditText's setError method](https://developer.android.com/reference/android/widget/TextView.html#setError(java.lang.CharSequence)) (inherited from TextView) on the appropriate EditText widget when the computation is triggered (i.e., the button 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 also shown in the screenshots below.

There are three error situations:

1. "Invalid Entry Text", related to the *Entry Text* field, for an empty or letter- or number-less entry.
2. "Invalid Arg Input 1", related to the *Arg Input 1* field, for a blank or unacceptable value  
   (i.e., not coprime to 62).
3. "Invalid Arg Input 2", related to the *Arg Input 2* field, for a blank or out-of-range value.

**Note:** *You may either limit the input in the numerical fields to positive numbers or provide the same errors if negative numbers or non-numerical input are entered.*

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

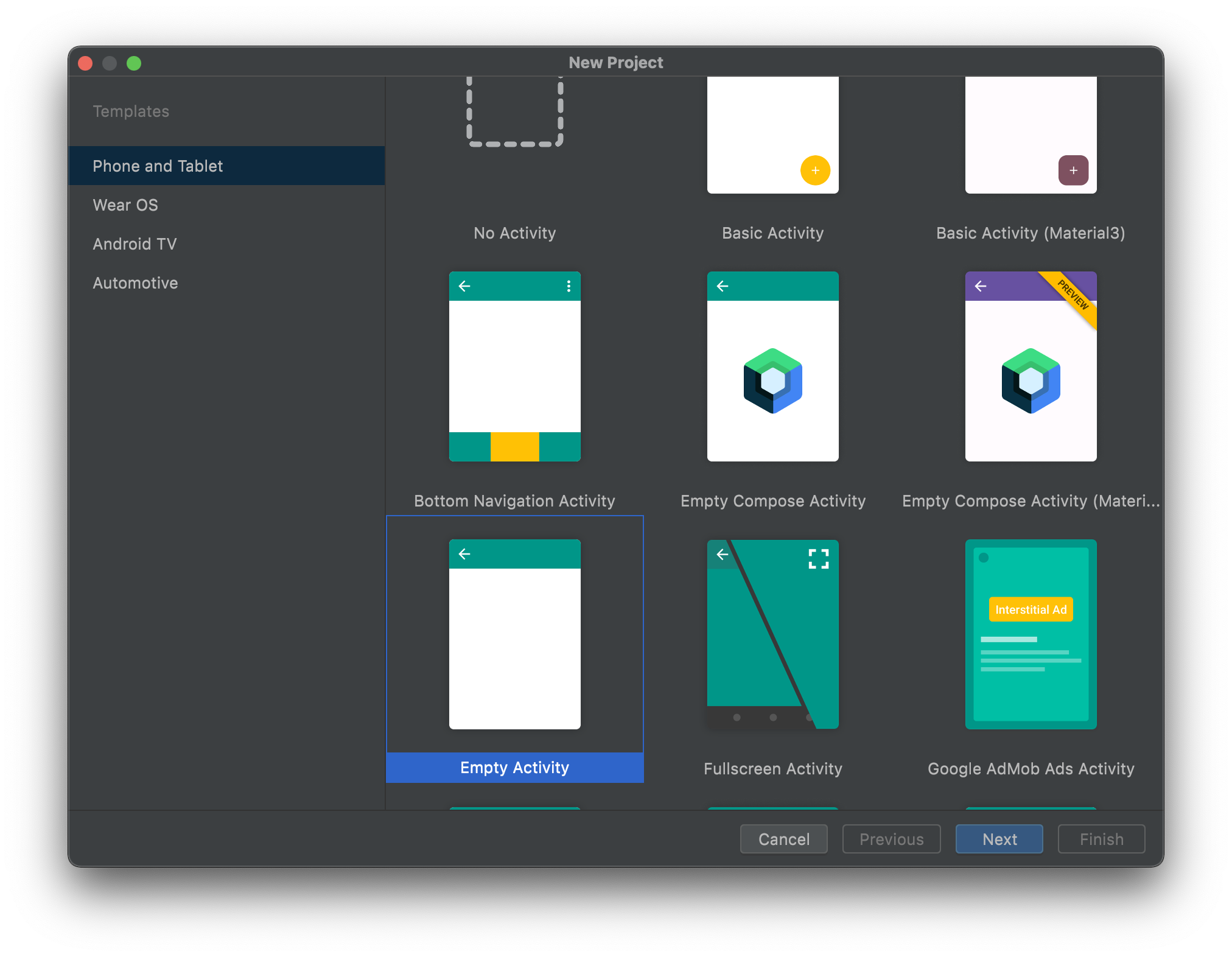


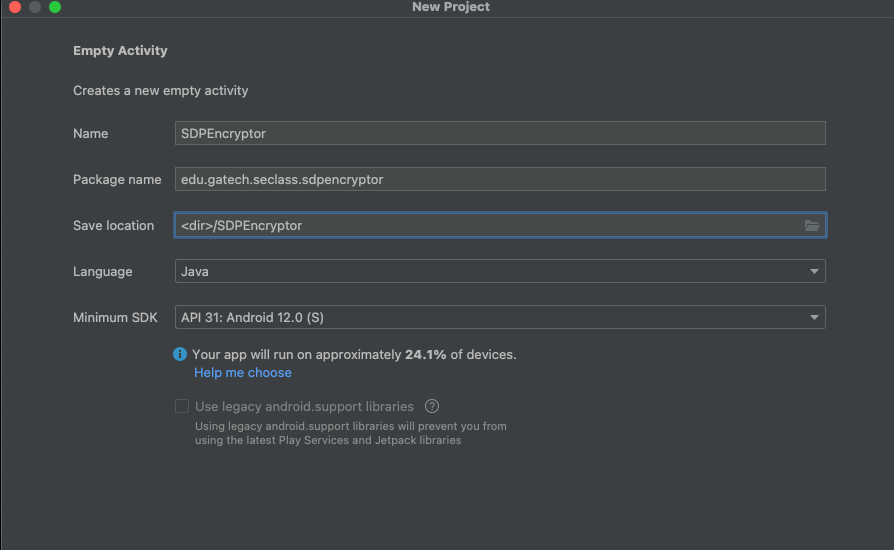
Suggestion: try to generate a user interface (UI) similar to the one shown above, but you don't have to. However, you must make sure to use the exact same identifiers we provide below for your widgets. This is very important, as we will use these identifiers to check and auto-grade your app.

IDENTIFIERS

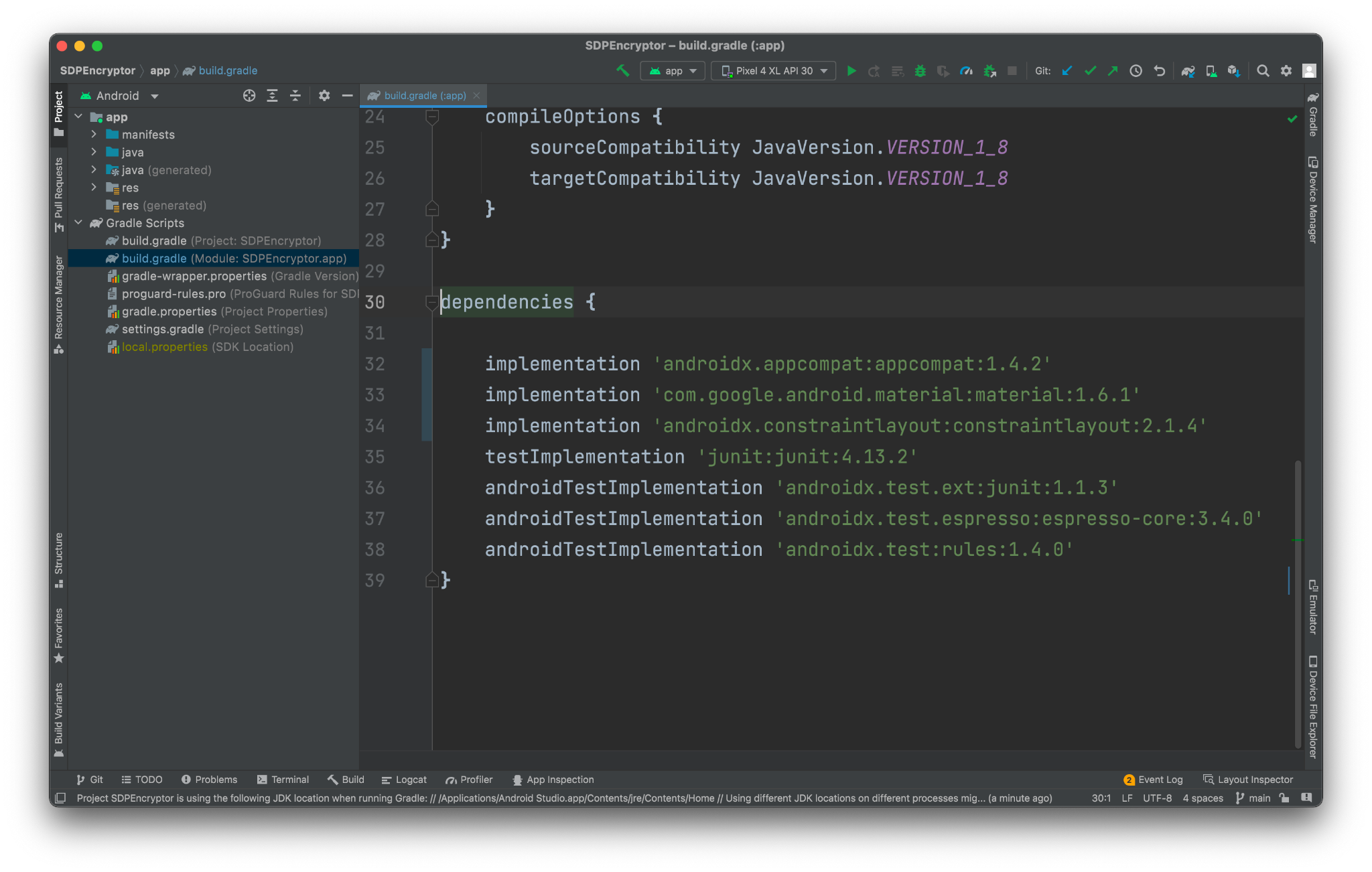
| * entryTextID * argInput1ID * argInput2ID * encryptButtonID * textEncryptedID | ***PRO-TIP: The identifiers are typed here for you. Just copy and paste them to be safe!***  For example, in the XML layout file for your app, the entry for the text field used to input the *Entry Text* should have the following ID: android:id="@+id/entryTextID" |
| --- | --- |

## INSTRUCTIONS

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-2023spring/6300Spring23<your GT username>.git), create a directory called "Assignment4". Hereafter, we will refer to this directory in your local repo as <dir>.
3. Using Android 3.x or higher[[1]](#footnote-0) 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 Activity" as your project template



Note the package name: "edu.gatech.seclass.sdpencryptor"  
 Note the minimum SDK: "API 31: Android 12"

1. Make sure that your build.gradle file for module app contains the following dependencies (**including the specific versions listed**):  
   testImplementation 'junit:junit:4.13.2'  
   androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
   androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'  
   androidTestImplementation 'androidx.test:rules:1.4.0'  
   (as shown in the figure)
   * 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 that through a notification).
2. Make sure to add the following in the build.gradle file for module app:

android {

// ...

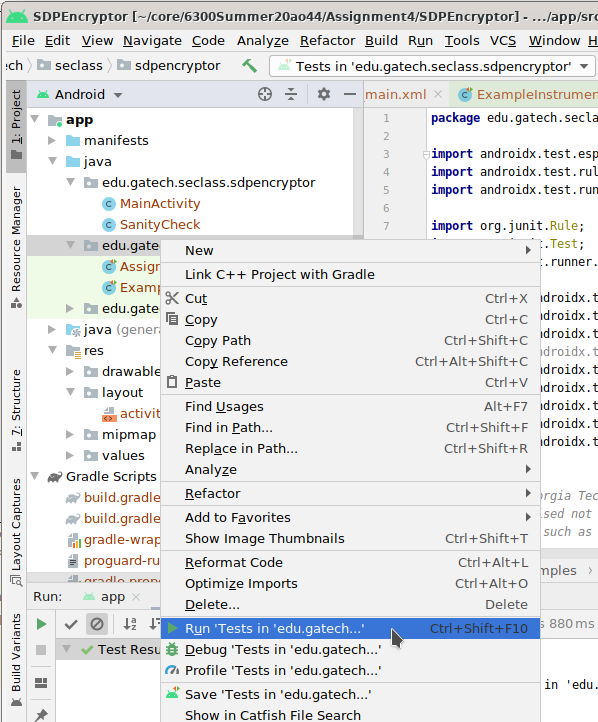
**testOptions {**

**animationsDisabled = true**

**}**

// ...

}

1. Implement your solution based on the requirements provided above using **Java 11**. 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**. The included download should 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.
   * SDPEncryptor/app/src/androidTest/java/edu/gatech/seclass/sdpencryptor/SmallTestExamples.java: This file contains a set of Espresso 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 connectedAndroidTest" in the project directory (you may have to install gradle to do so, and you will need an emulator running or a phone connected to your computer). During development and before submission, you can also run the tests from within Android Studio, by selecting "Run Tests in…" as shown below:  
       
     If a test fails, you can figure out the details by checking the "Expected" and "Got" part of the test output (which is not always the easiest thing to do, admittedly). Here we show the key parts of the output using an example:
     + *Expected*: with text: is "**wBM8B lBG**"
     + *Got*: "AppCompatEditText{id=2131165271, res-name=textEncryptedID, visibility=VISIBLE, width=530, height=113, has-focus=false, has-focusable=true, has-window-focus=true, is-clickable=true, is-enabled=true, is-focused=false, is-focusable=true, is-layout-requested=false, is-selected=false, layout-params=androidx.constraintlayout.widget.ConstraintLayout$LayoutParams@3fabad8, tag=null, root-is-layout-requested=false, has-input-connection=true, editor-info=[inputType=0x61 imeOptions=0x6 privateImeOptions=null actionLabel=null actionId=0 initialSelStart=0 initialSelEnd=0 initialCapsMode=0x0 hintText=null label=null packageName=null fieldId=0 fieldName=null extras=null hintLocales=null contentMimeTypes=null ], x=0.0, y=0.0, text=**mYMDB eDX**, input-type=97, ime-target=false, has-links=false}"
     + In this case, your code generated the result "**mYMDB eDX**" instead of "**wBM8B lBG**".
     + The autograder will grade your assignment using a Pixel 4 XL, Android 12 (API level 31). To confirm you do not hit snags with grading, ensure that you test your application with exactly this emulator and API level. While it may run okay on your machine locally with a different emulator and API level, results may vary when run on the autograder if emulator and API levels differ.
3. 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 {

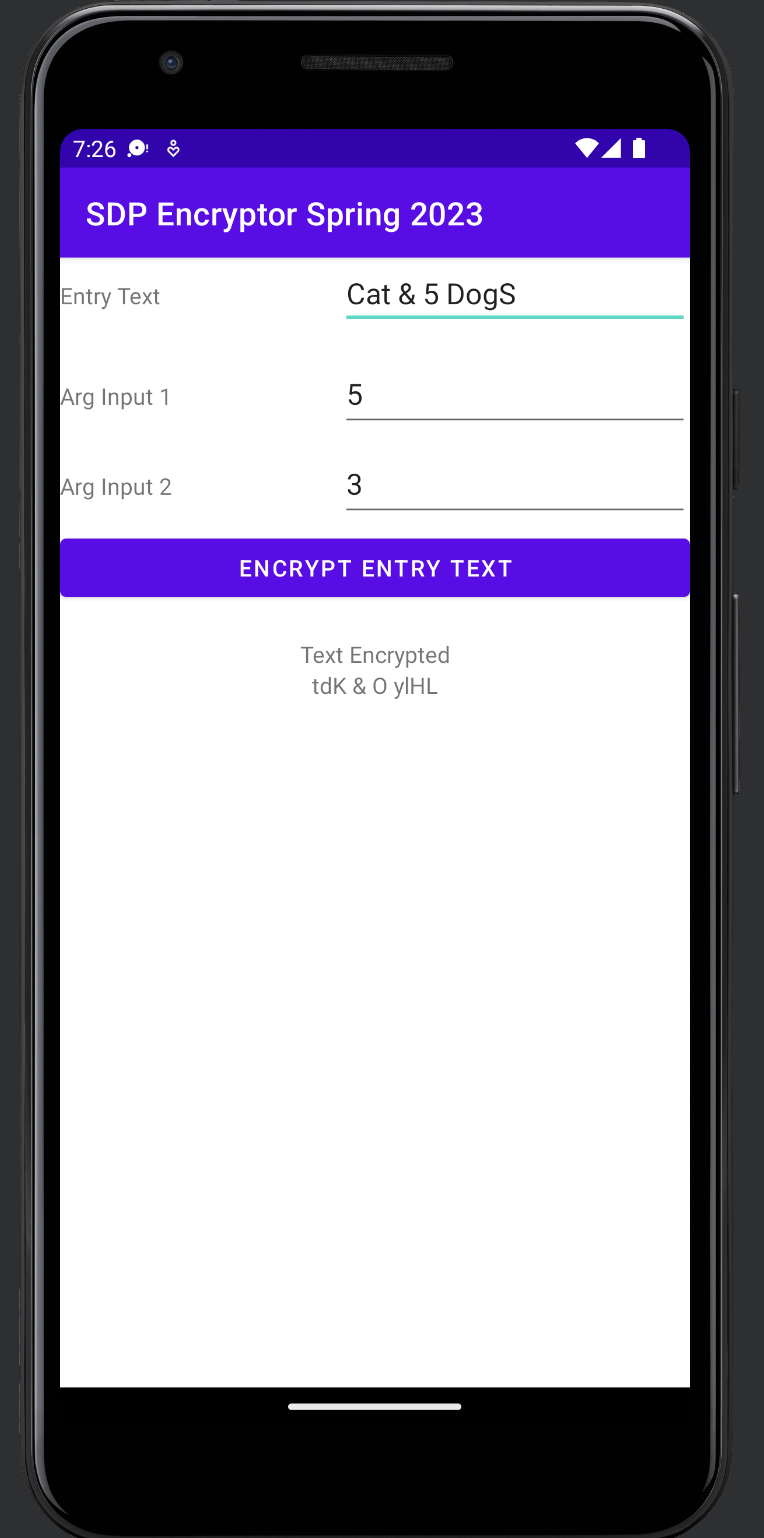
tasks.lint.enabled = false

abortOnError false

}

}

1. For automated testing, check out [Espresso](https://developer.android.com/training/testing/espresso/) and [Android Studio Test Recorder](https://developer.android.com/studio/test/espresso-test-recorder.html). Here are some Espresso tips, based on past experience:
   * Turn off animations in your AVD, particularly if the tests fail with an AmbiguousViewMatcherException.
   * Avoid using buttons that call private methods, as Espresso will 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 and include it in your private GitHub repository. If your .gitignore excludes jar files, you will need to force add it.
3. 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 lower case.
   * 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 to big, not too small); all of this is to ensure that we can easily read the content of your screenshot. Failure to do so might result in points deduction.
   * app\_run.png must show semester and year in the app title, as seen from the example below. Failure to adhere to the app title format text might result in points deduction.



1. Your windows may look slightly different from the ones we provide if you are using a later version than 3.x. [↑](#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)