**بسم الله الرحمن الرحيم**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

****

**Submitted to:**

Sir Haq nawaz

**Submitted by:**

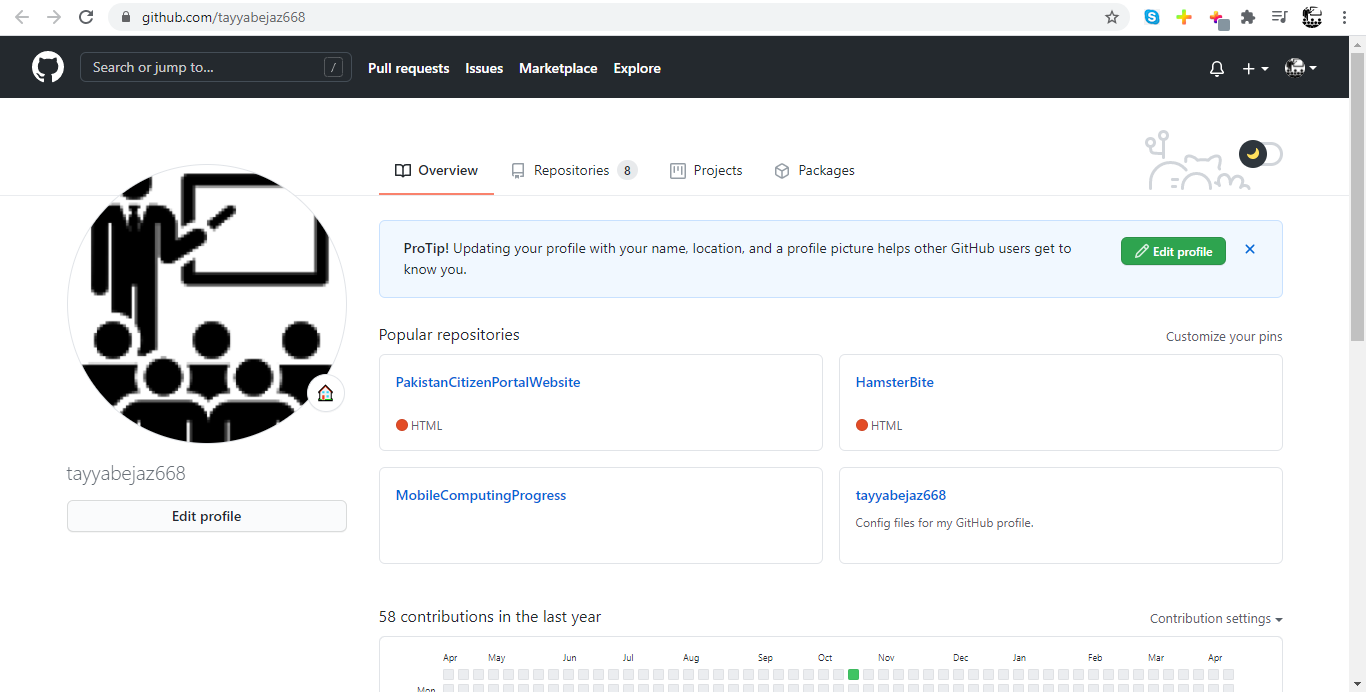
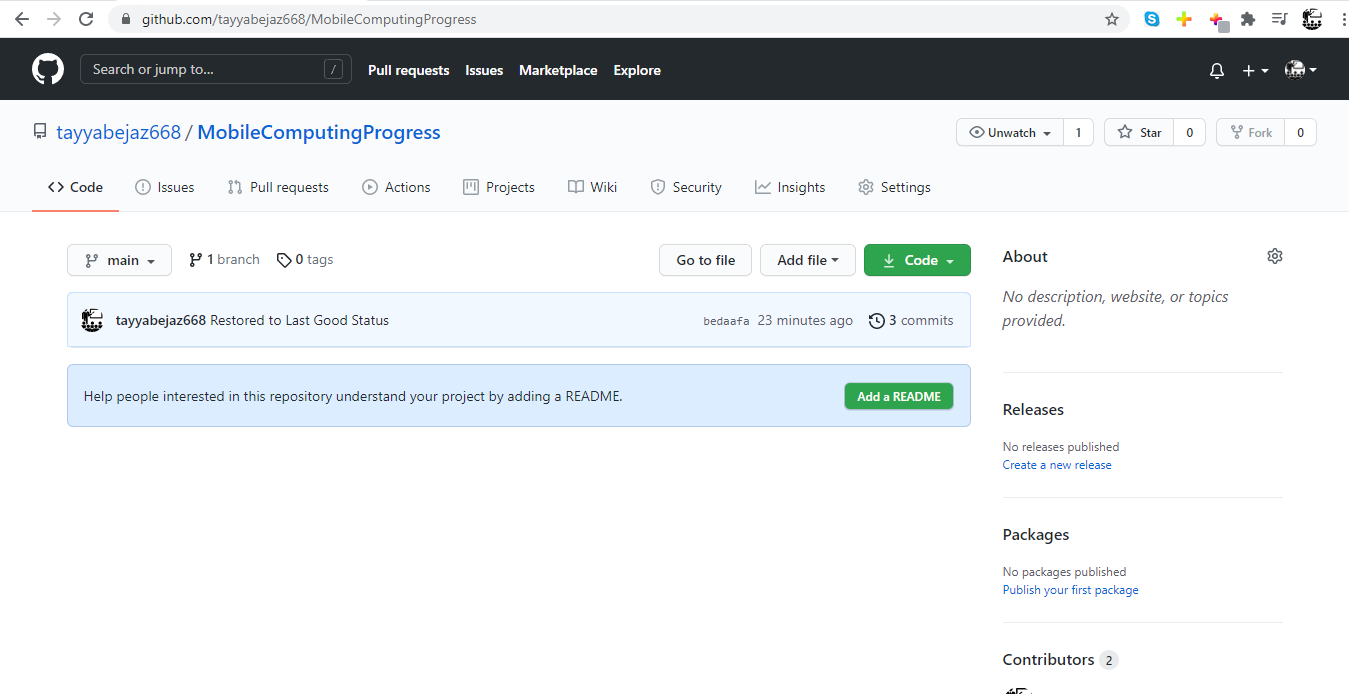
Tayyab Ejaz

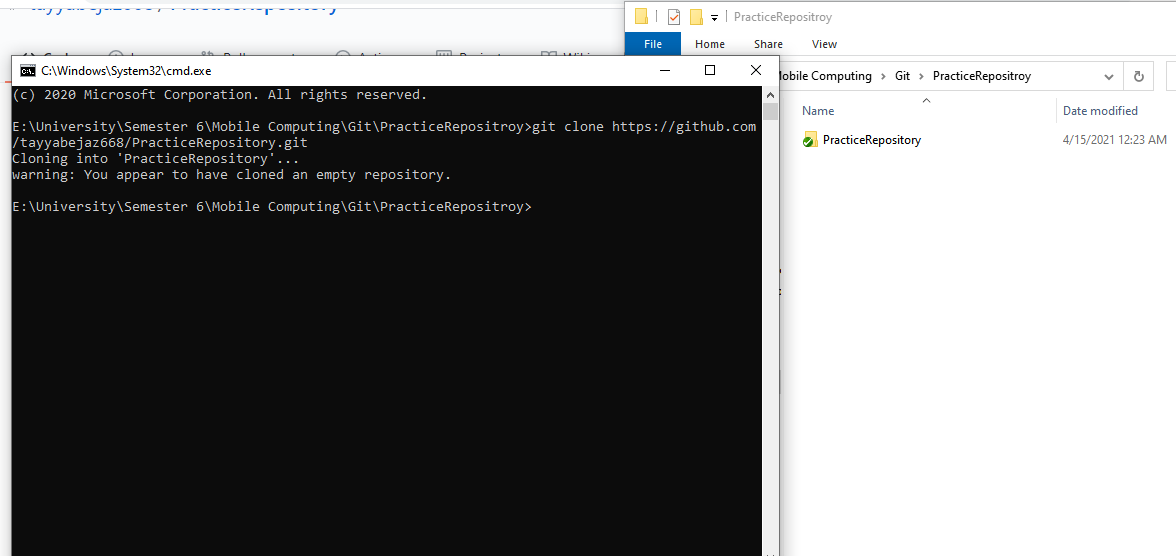
**Roll no:** BSEF18A014

**Punjab University College of Information Technology,**

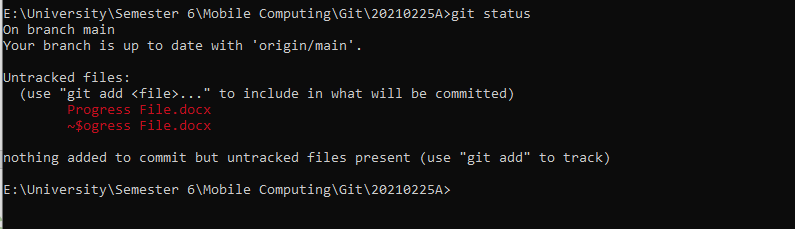
**PUCIT (Old Campus), LAHORE**

**Version Control System**

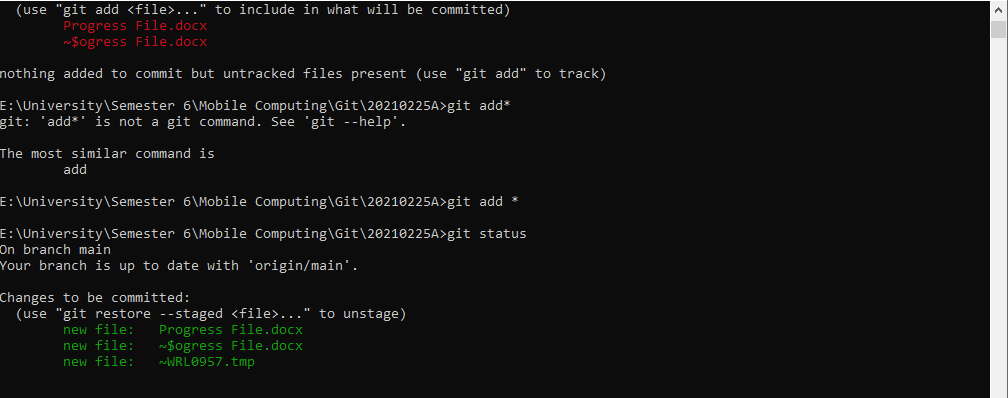
* **Features:   
  1)** Code Synchronization (code sharing among different programmers)  
  **2)** Change tracking (track the history of all previous committed changes)  
  **3)** Provides the remote access to central repository
* **Account Created:**
* **Create Repository for Progress File:**
* **Git Clone:** command to clone a repository in local machine from remote (if authorized to access the repository)



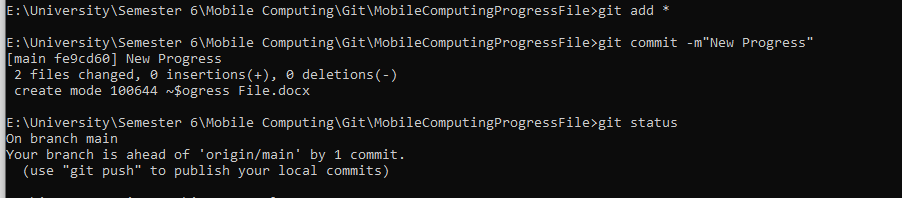
* **Git Status:** Used to check the current status of the local repository



* **Git Add:** Used to add the changed files to staged area



* **Git Commit:** Used to submit staged changes to the local branch



\*m option for giving the caption for commit

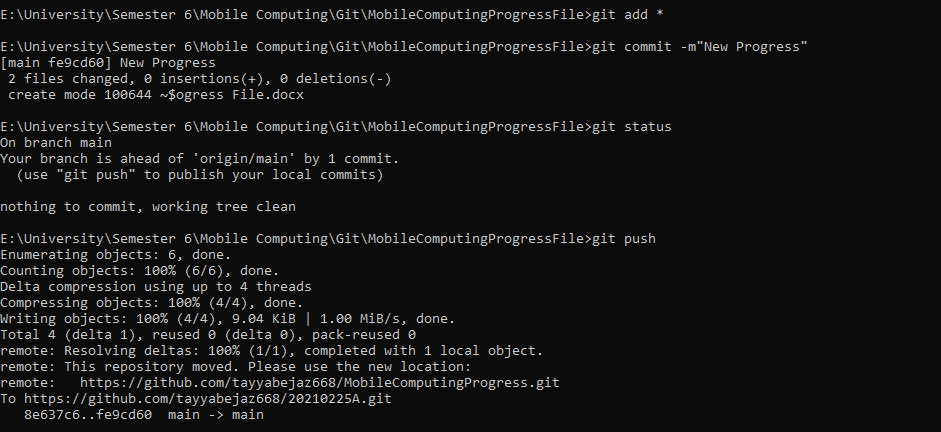
* **Git Pull:** command to pull (get) remote repository changes to the local repository
* **Git Push:** command to push (put) local repository changes to the remote repository
  + **Steps:**

**1)** Do Changes  
**2)** Git Add \*  
**3)** Git Commit

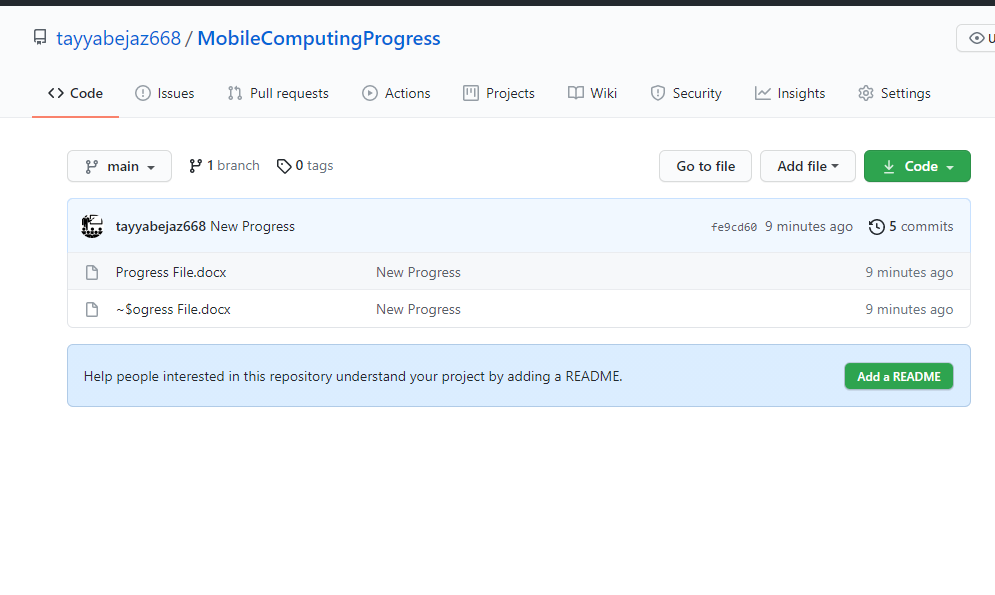
**4)** Git Pull (must to update your local with the new changes in remote)

**5)** Now use **Git Push** command to push code in remote repository

**Local:**

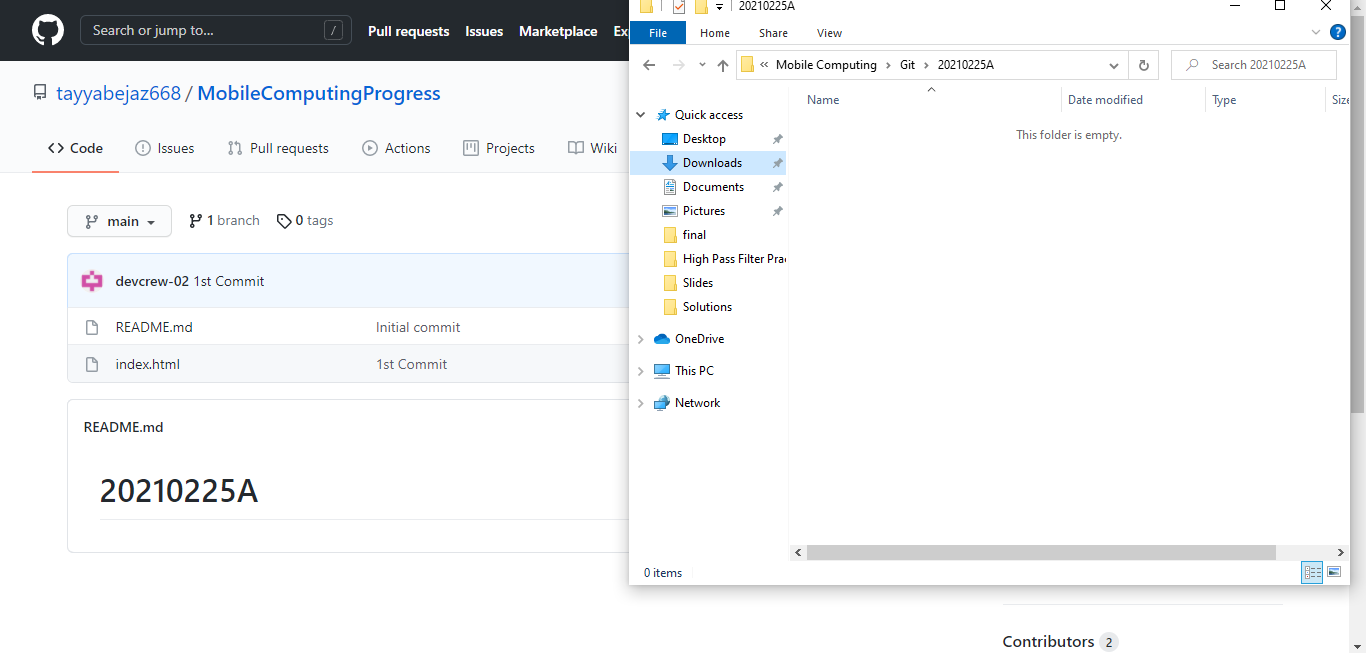


**Remote:**

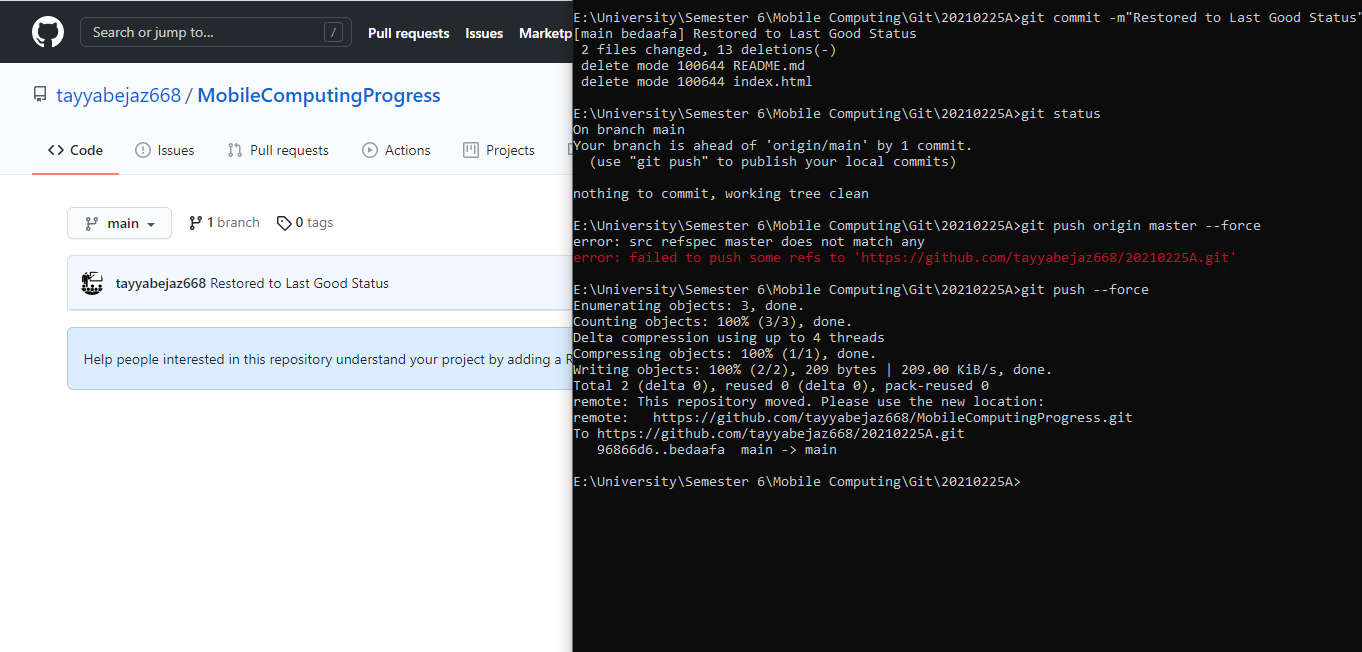


* **Git Push Origin –force:** command works like git push the difference is it just replaces the remote code with the local repository code without resolving any conflict

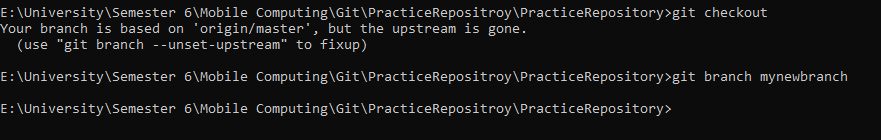
**Before:**



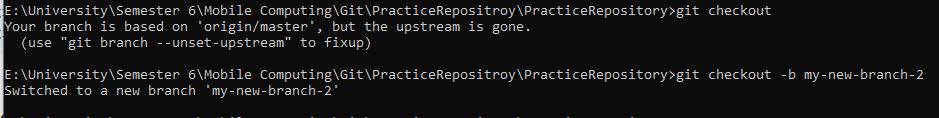
**After:**

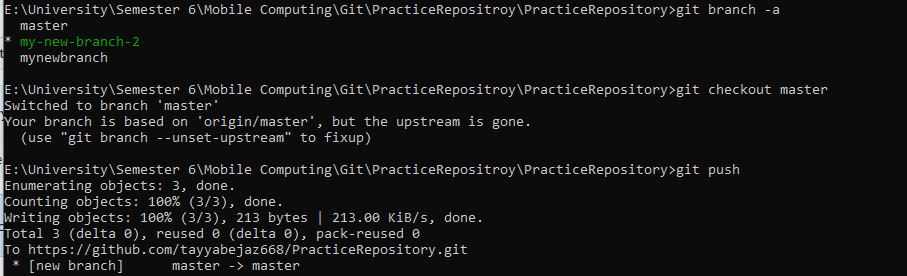


* **Git Branch <Branch Name>:** command to create a new branch in local repository, or to delete a branch when used with -d option.

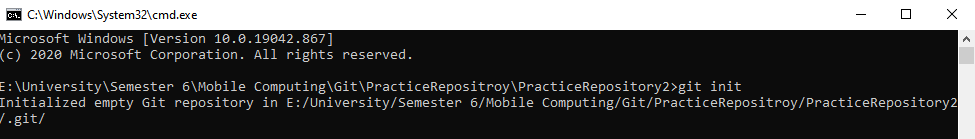


* **Git Checkout <Branch Name>:** command to see current branch, switch an existing branch and to create a new branch when used with the option -b.





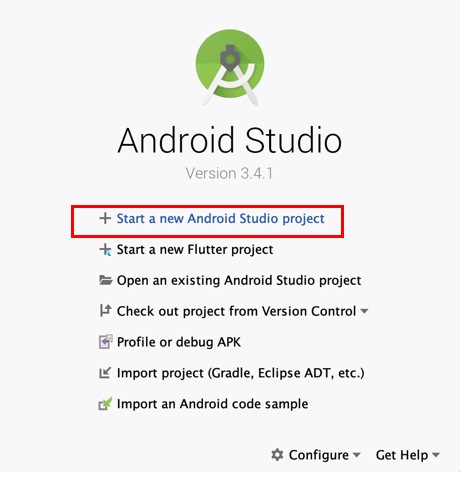
* **Git Init:** To initialize current directory as a new local git repository.

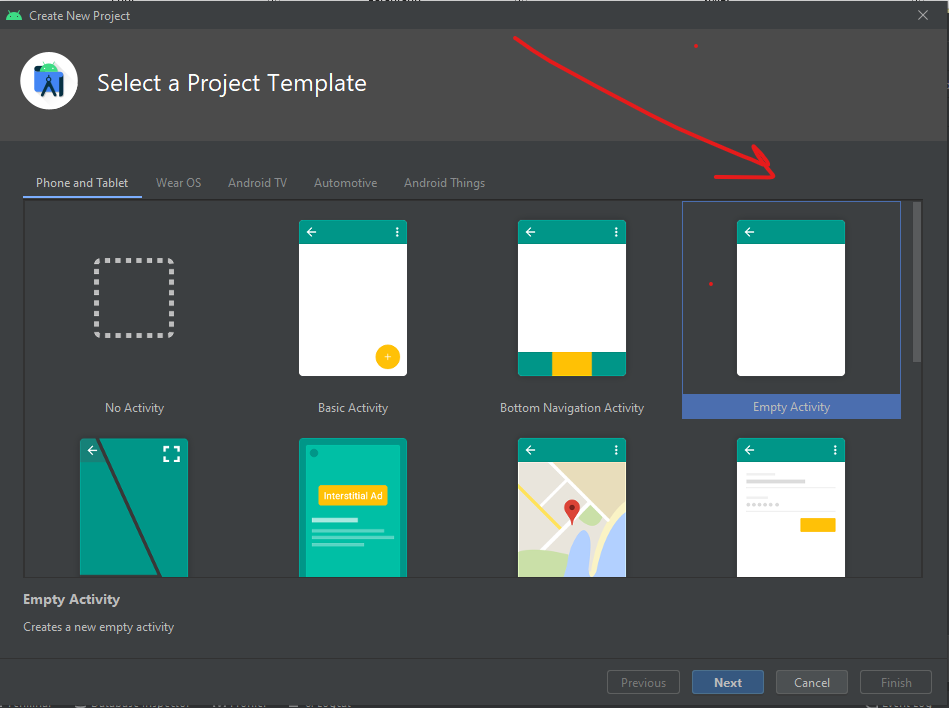


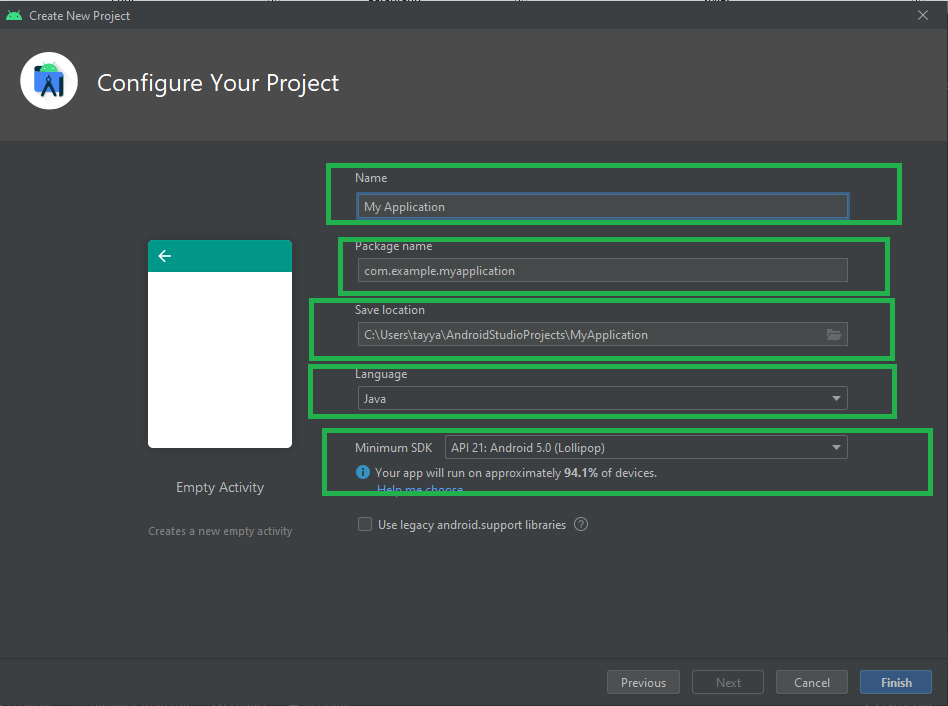
* **Git Merge**: command to Integrate branches together. Git merge combines the changes from one branch to another branch

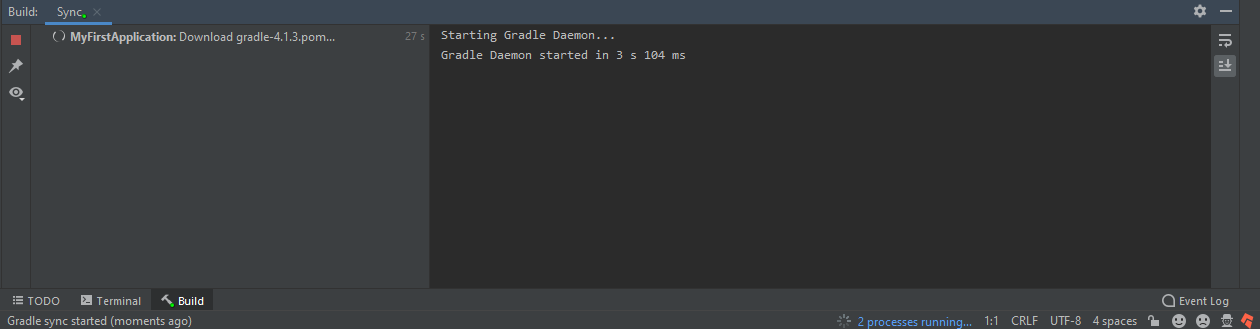
**IDE Introduction And First Application**

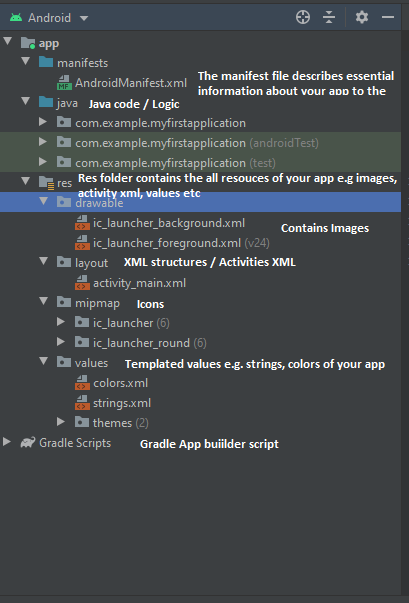
* **New Project**:





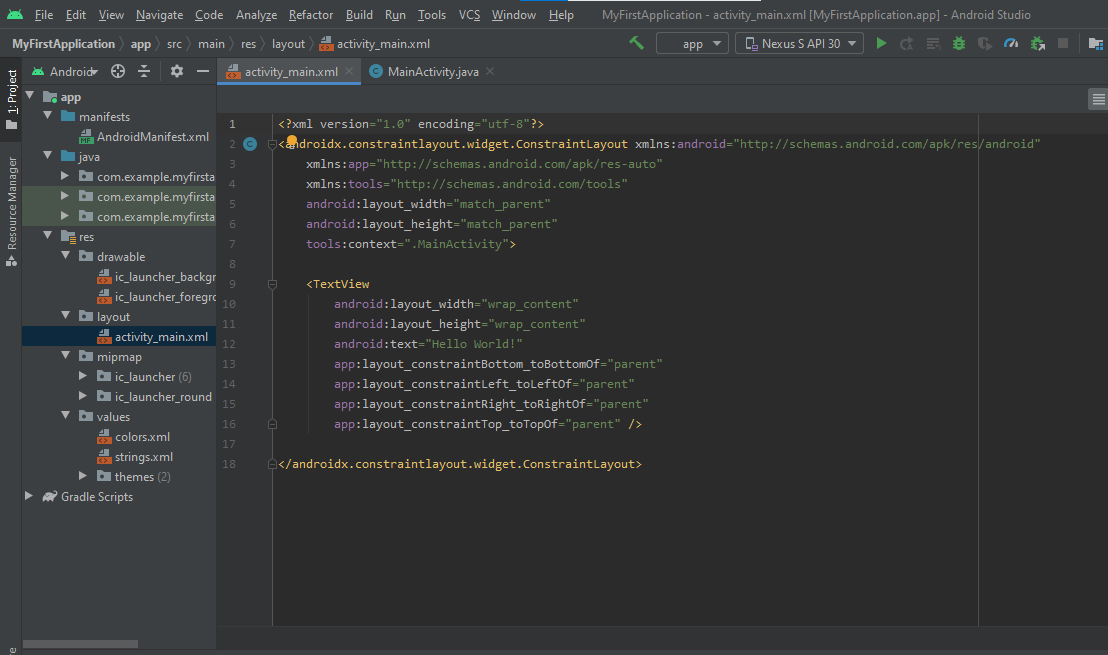


* Gradle is building project in background processes  
  
* **Directory Structure:**

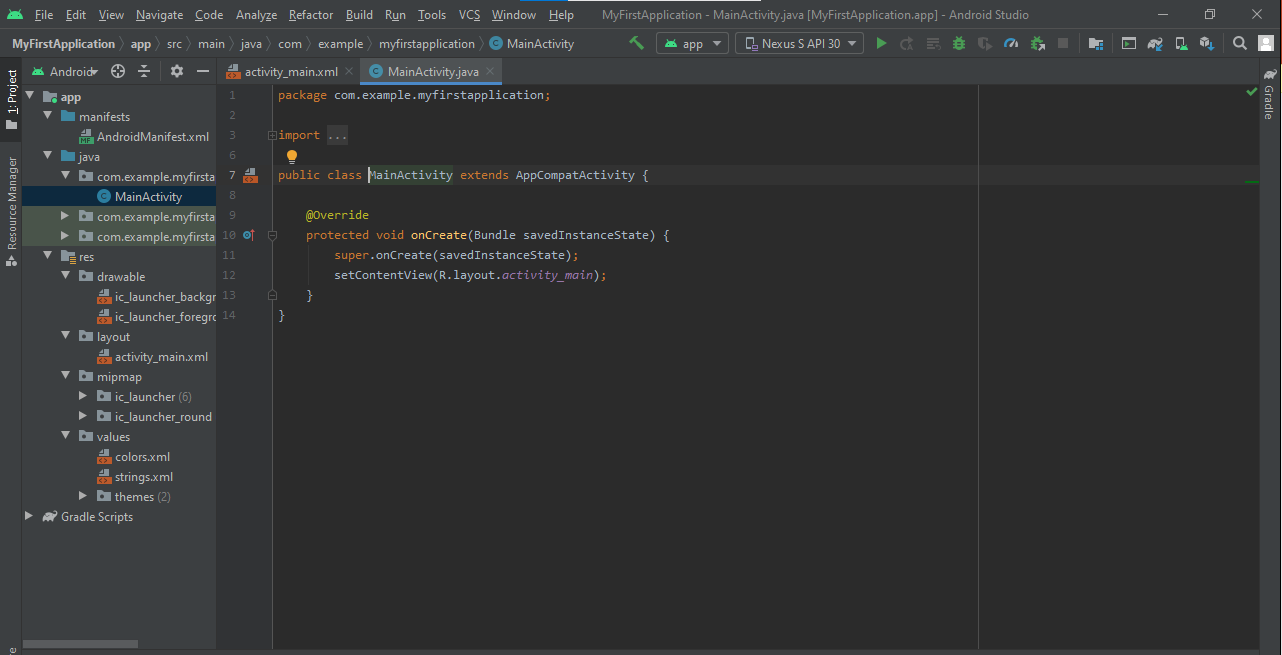


* **Activity:**

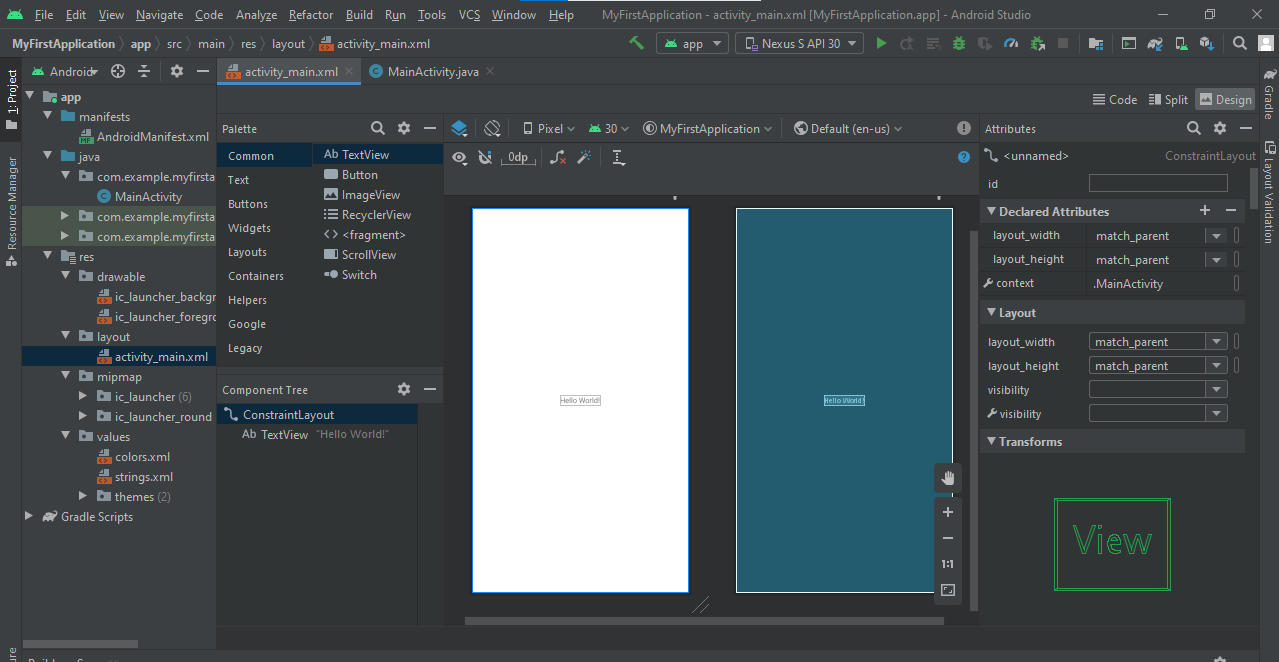
**XML File**



**JAVA File**

****

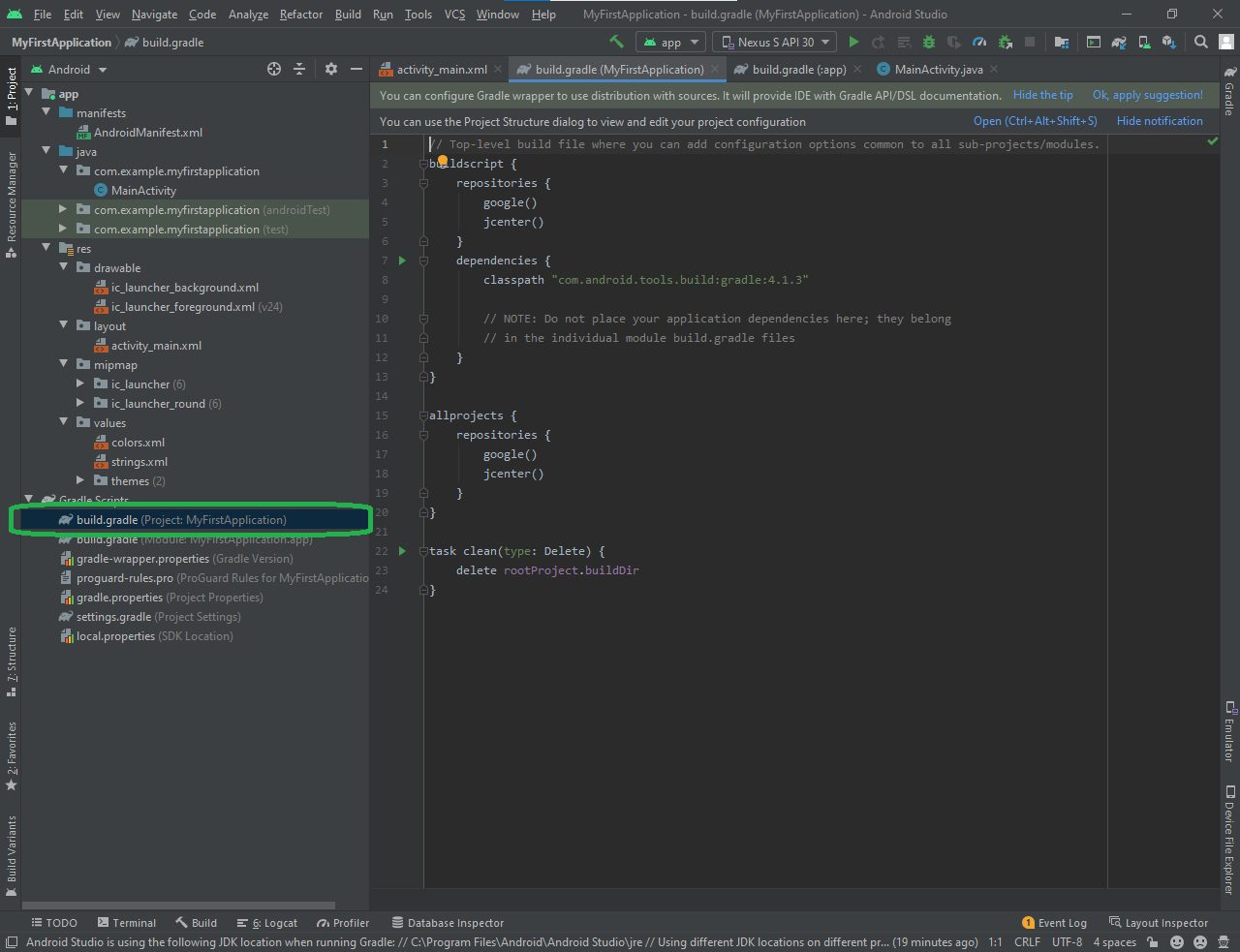
**GUI View**

****

* **Build.Gradle Files:** Defines the configuration of the application being coded.

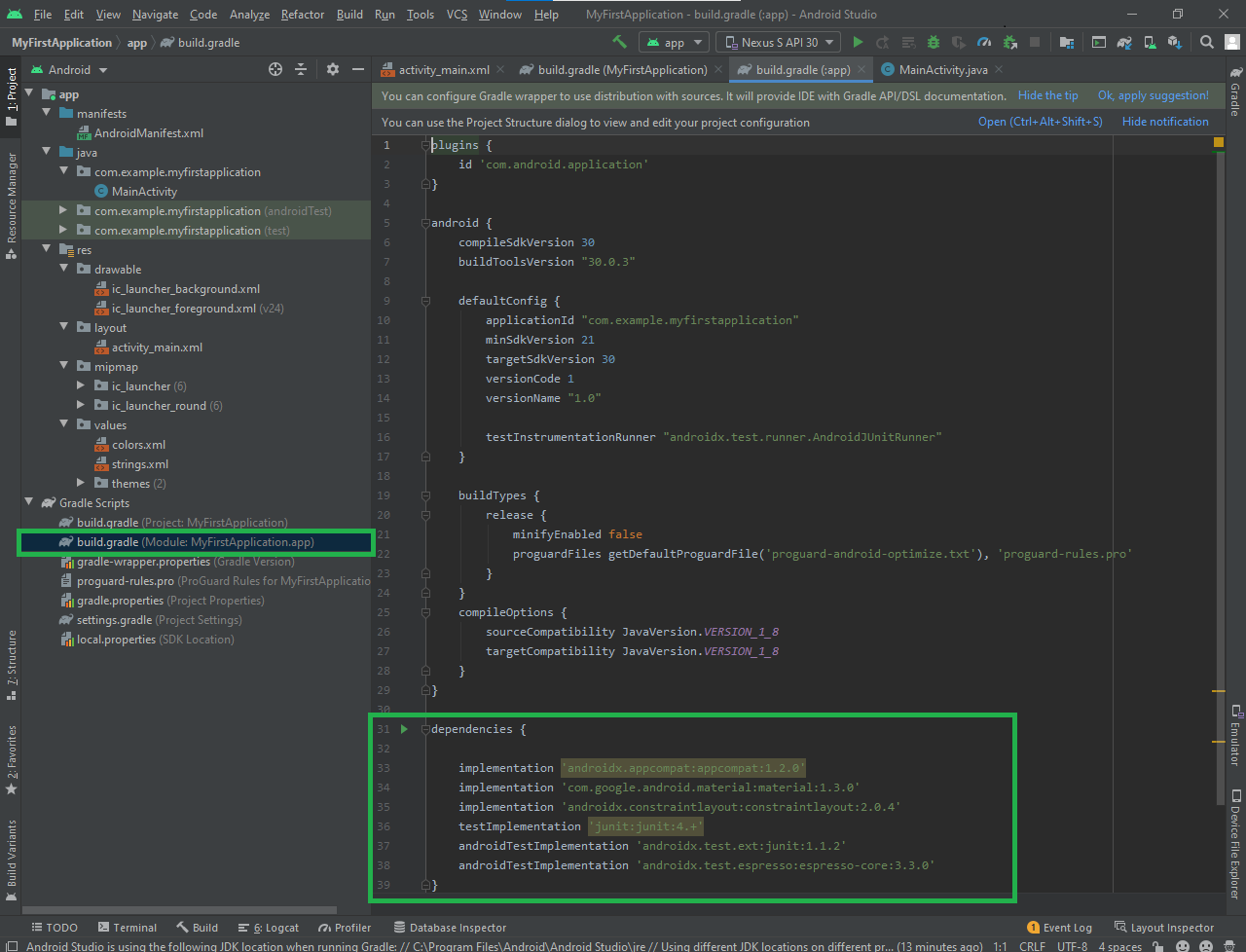
**Build.Gradle (project)**

The top-level build.gradle file, located in the root project directory, defines build configurations that apply to all modules in your project. By default, the top-level build file uses the buildscript block to define the Gradle repositories and dependencies that are common to all modules in the project. The following code sample describes the default settings and DSL elements you can find in the top-level build.gradle after creating a new project.

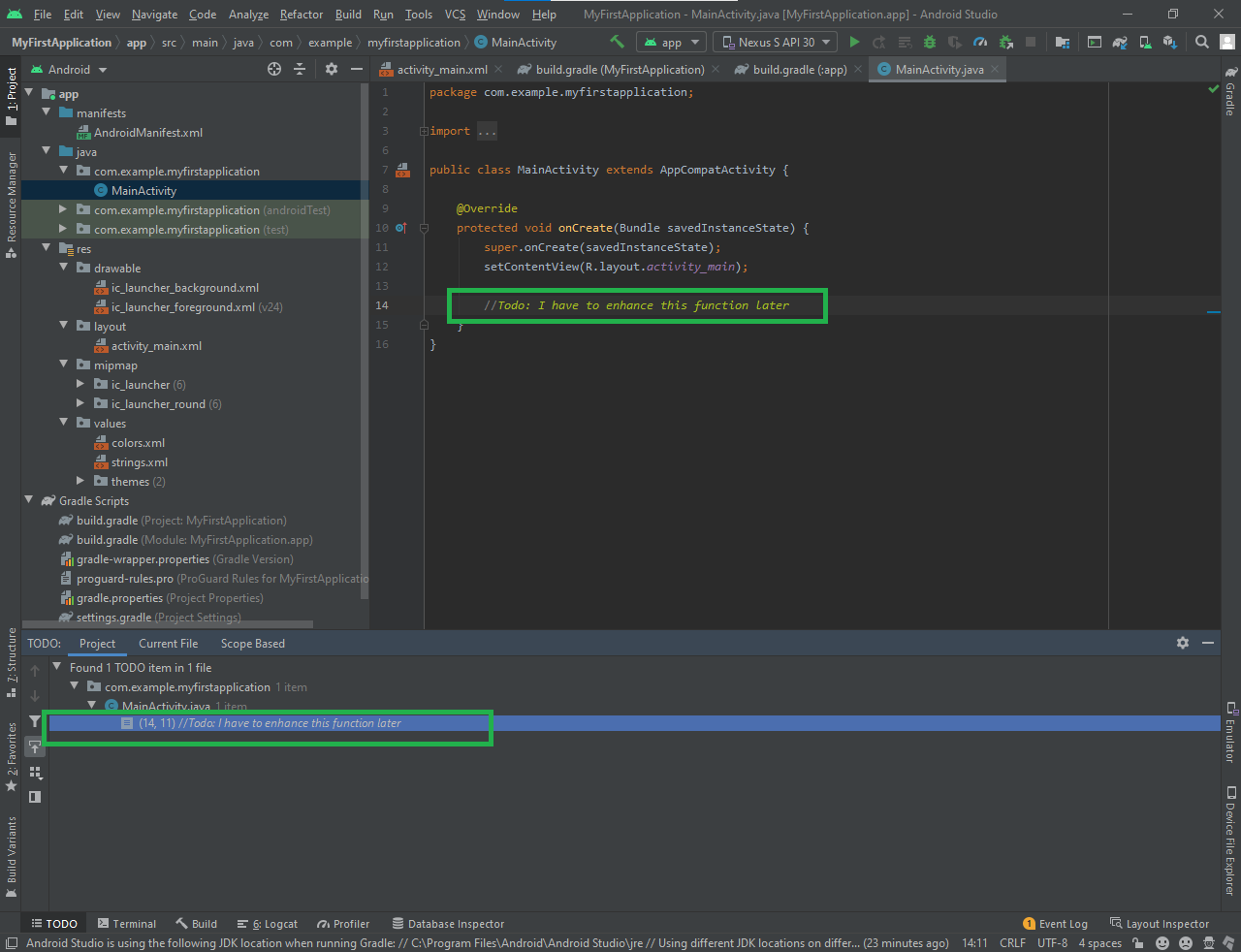


**Build.Gradle (Module)**

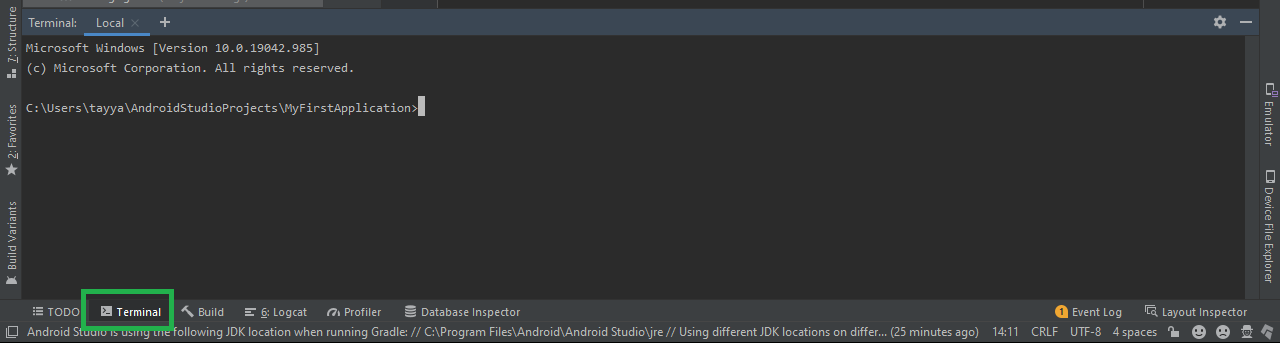
The module-level build.gradle file, located in each project/module/ directory, allows you to configure build settings for the specific module it is located in. Configuring these build settings allows you to provide custom packaging options, such as additional build types and product flavors, and override settings in the main/ app manifest or top-level build.gradle file.



* **TODOs:** Android studio provides a feature of TODOs which is just like a comment in the code but all TODOs are listed in a panel at the bottom. To add a TODO, just comment anywhere in your code starting with the word TODO.

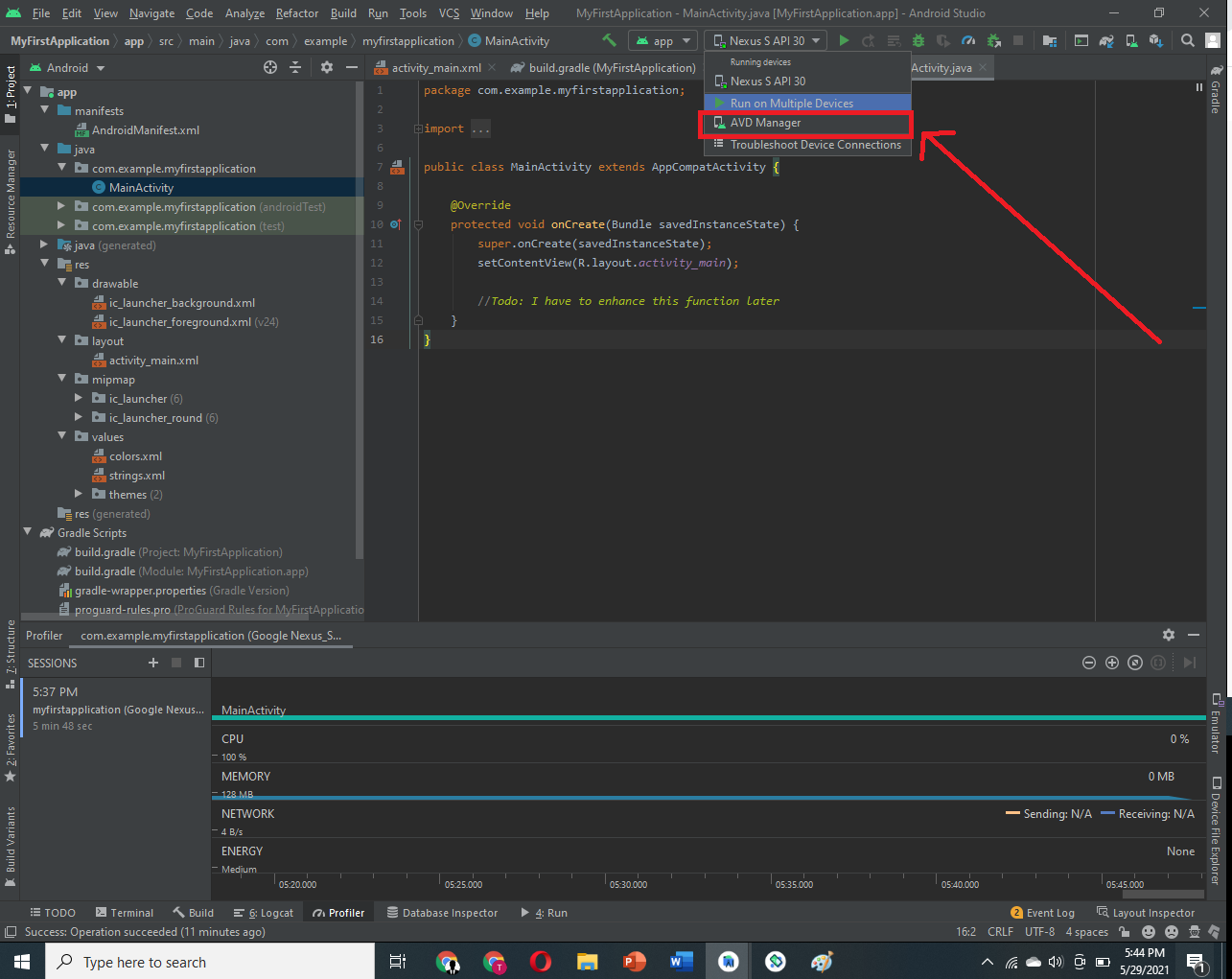


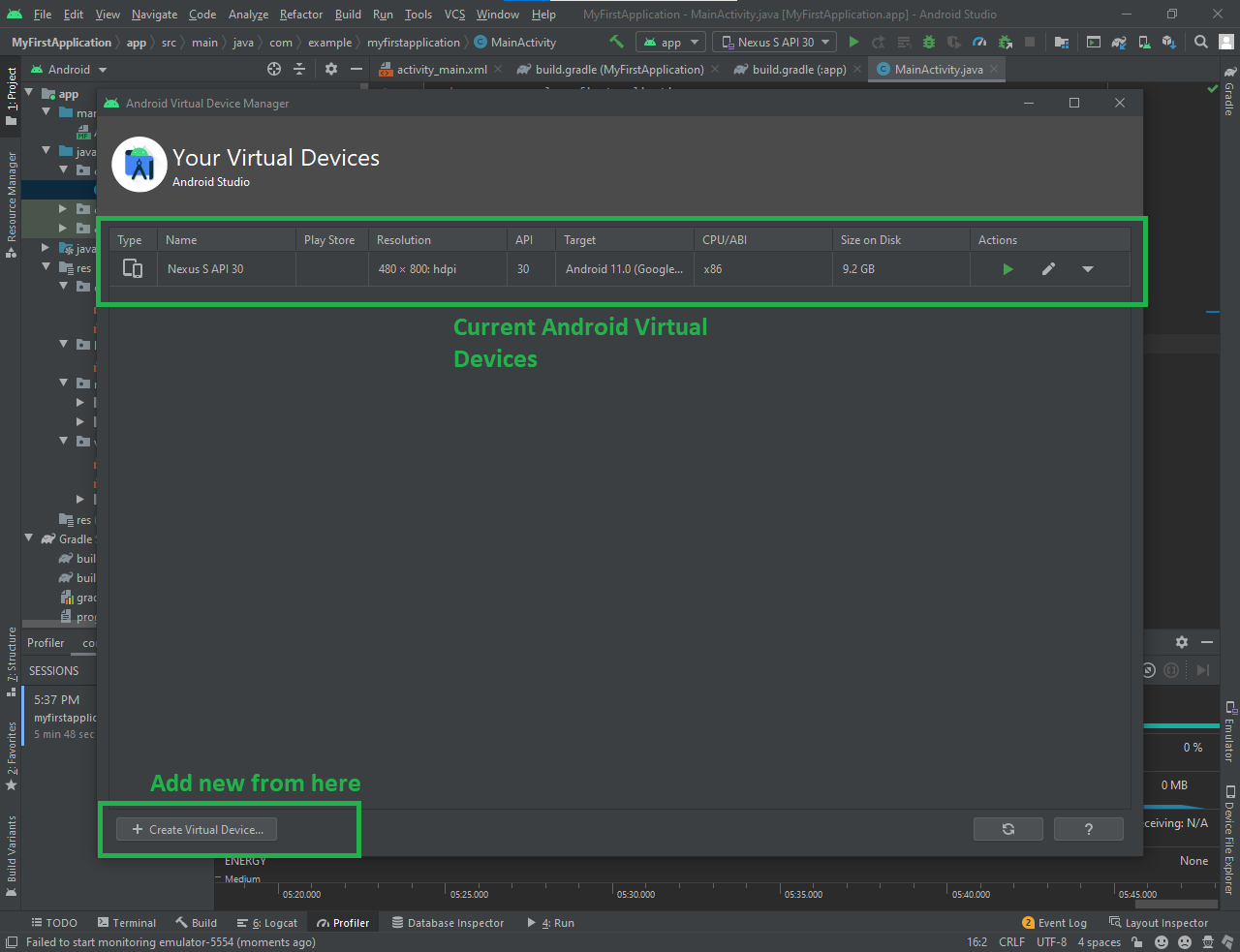
* **Terminal:**

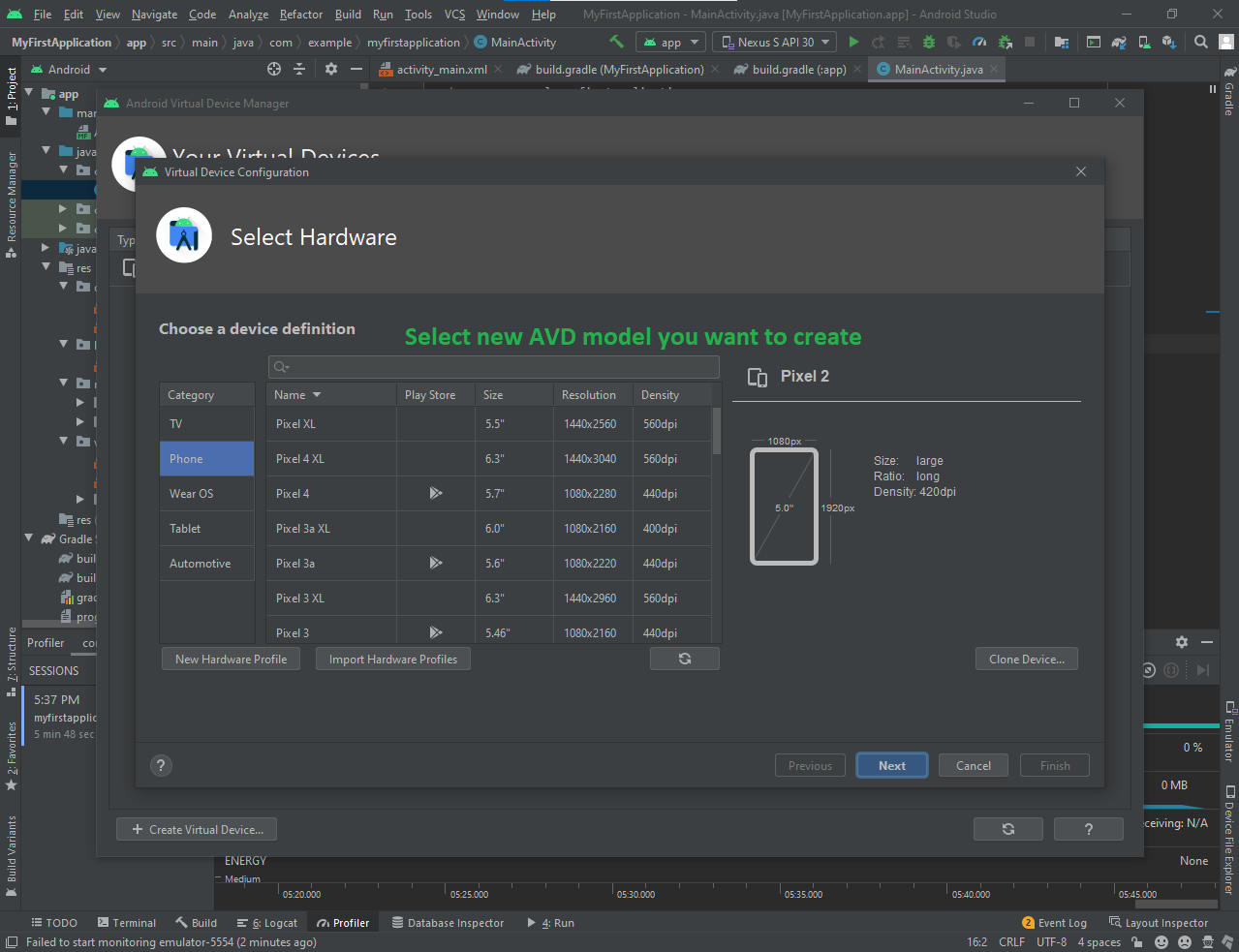


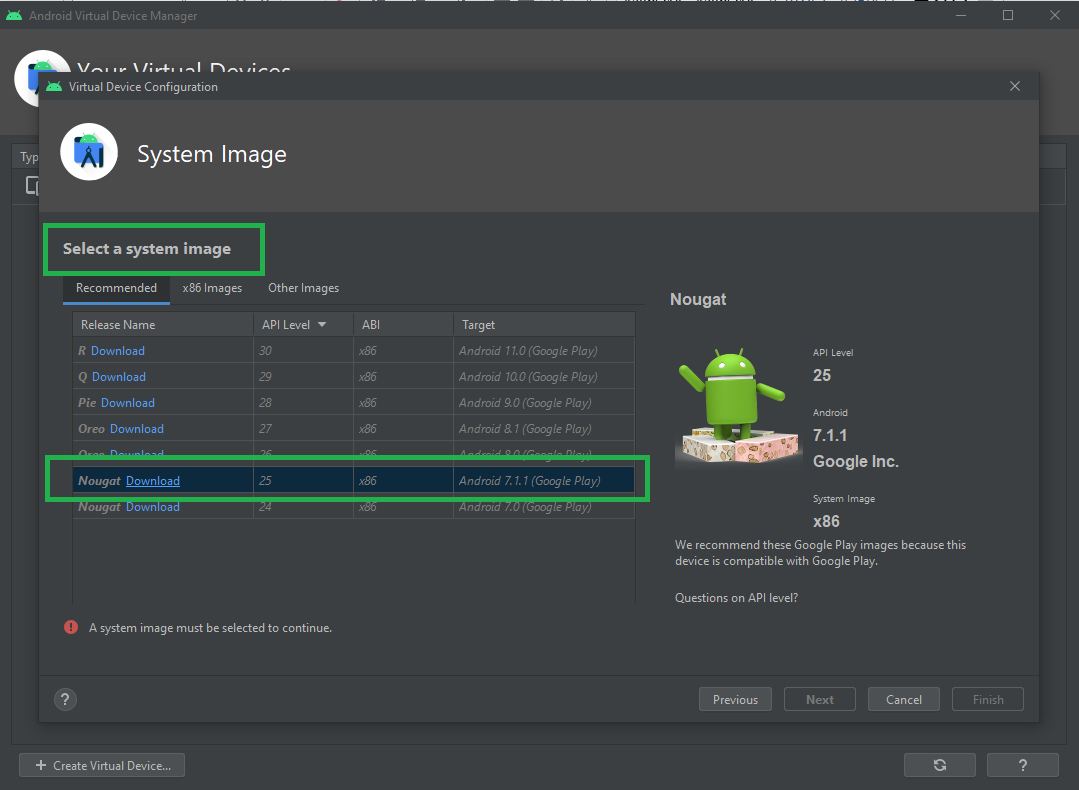
* **Add an AVD to run application:**

**Open AVD Manager**



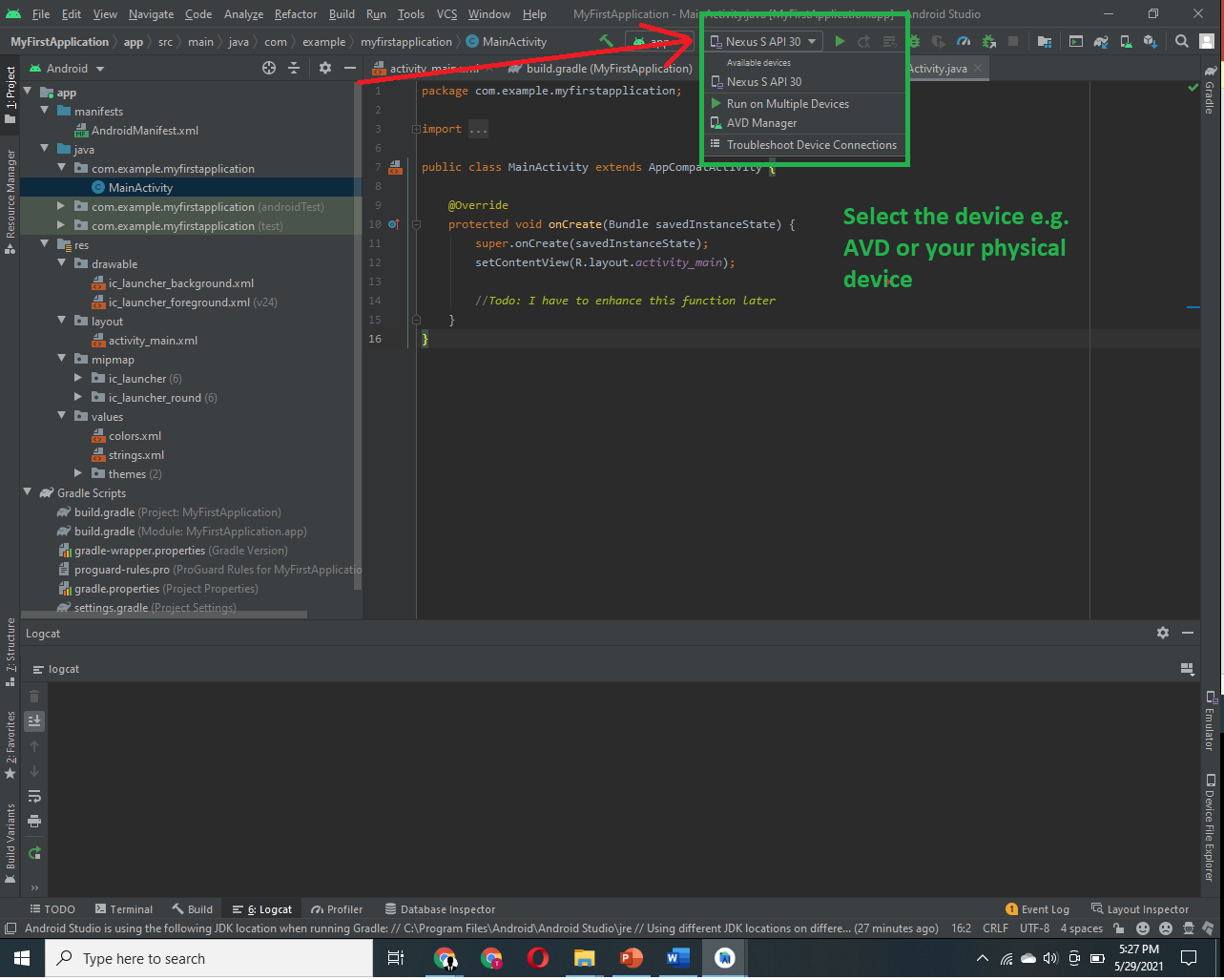






* **Build your App:**

**Select the device**



**Click the play button**



* **Profiler:** To monitor the resources being used by the application



**Application running on AVD**

****

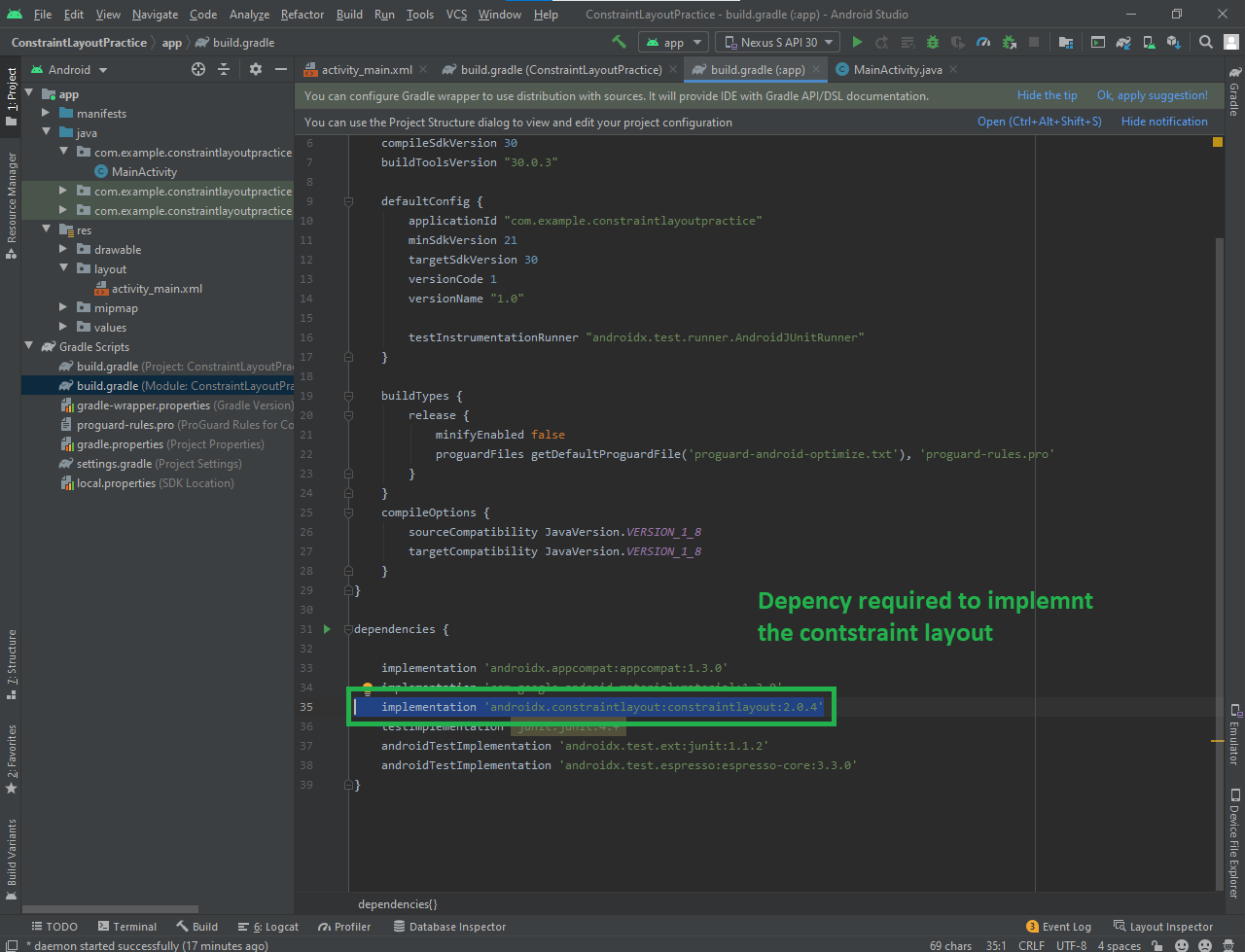
**Layouts**

* **Layout (View Group):** Layout is just an invisible container that’s used to structure our activity or interface or to align out views/components in our desired way. For example, ConstraintLayout or LinearLayout.
* **View:** A view is something the use can see and interact with. For example, a TextView, Button etc. A view is also called a widget.

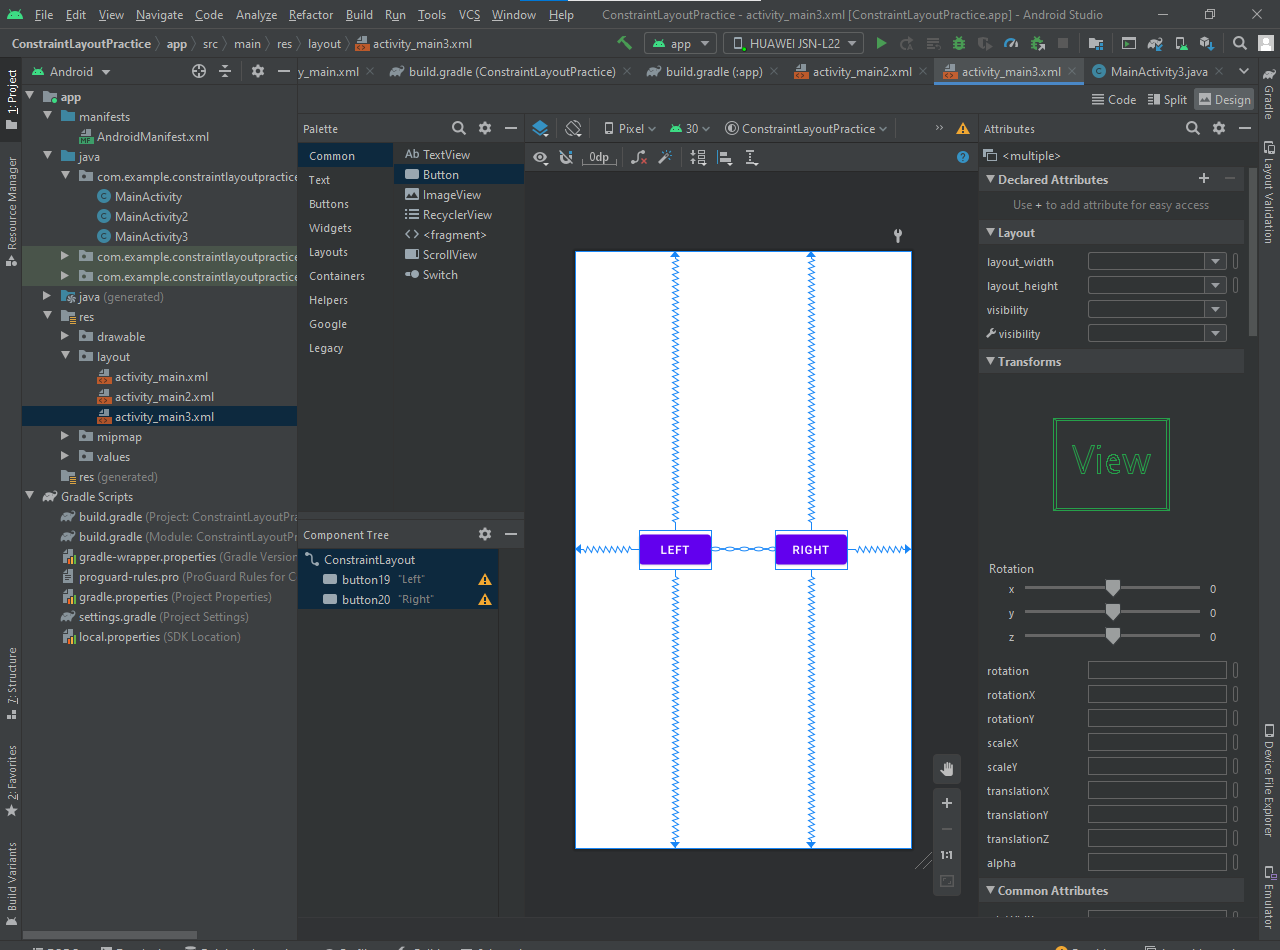
**Note:** Each layout file must contain a layout or view as a root element.

* **ContstraintLayout:**

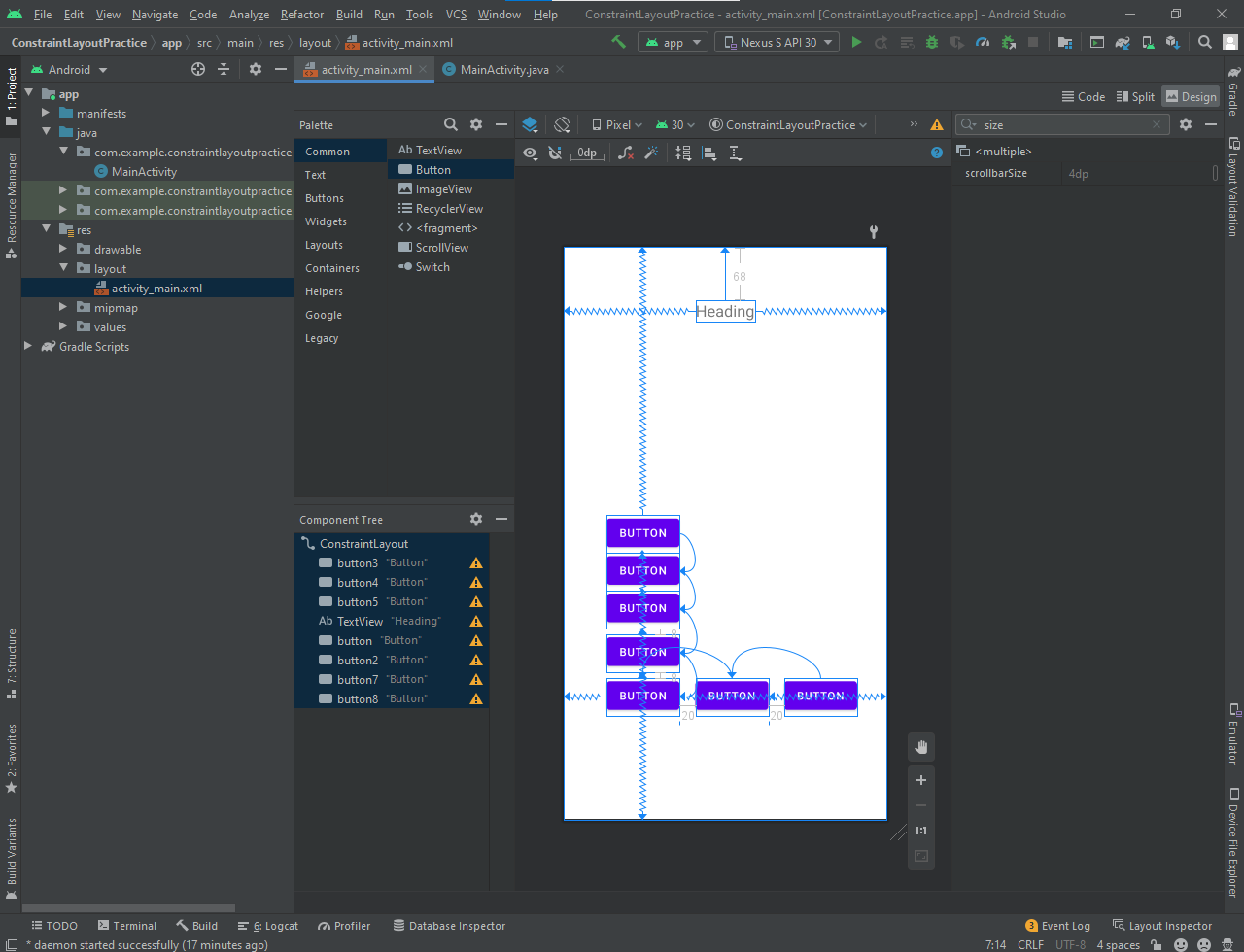
**Required Dependency**

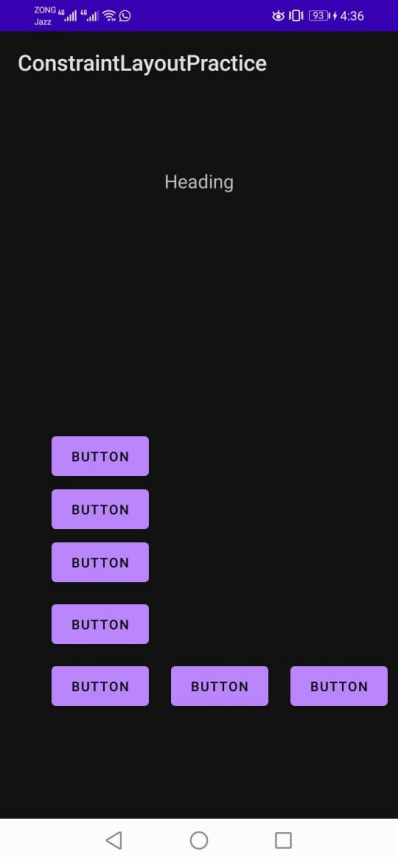
****

**Center Constraint**

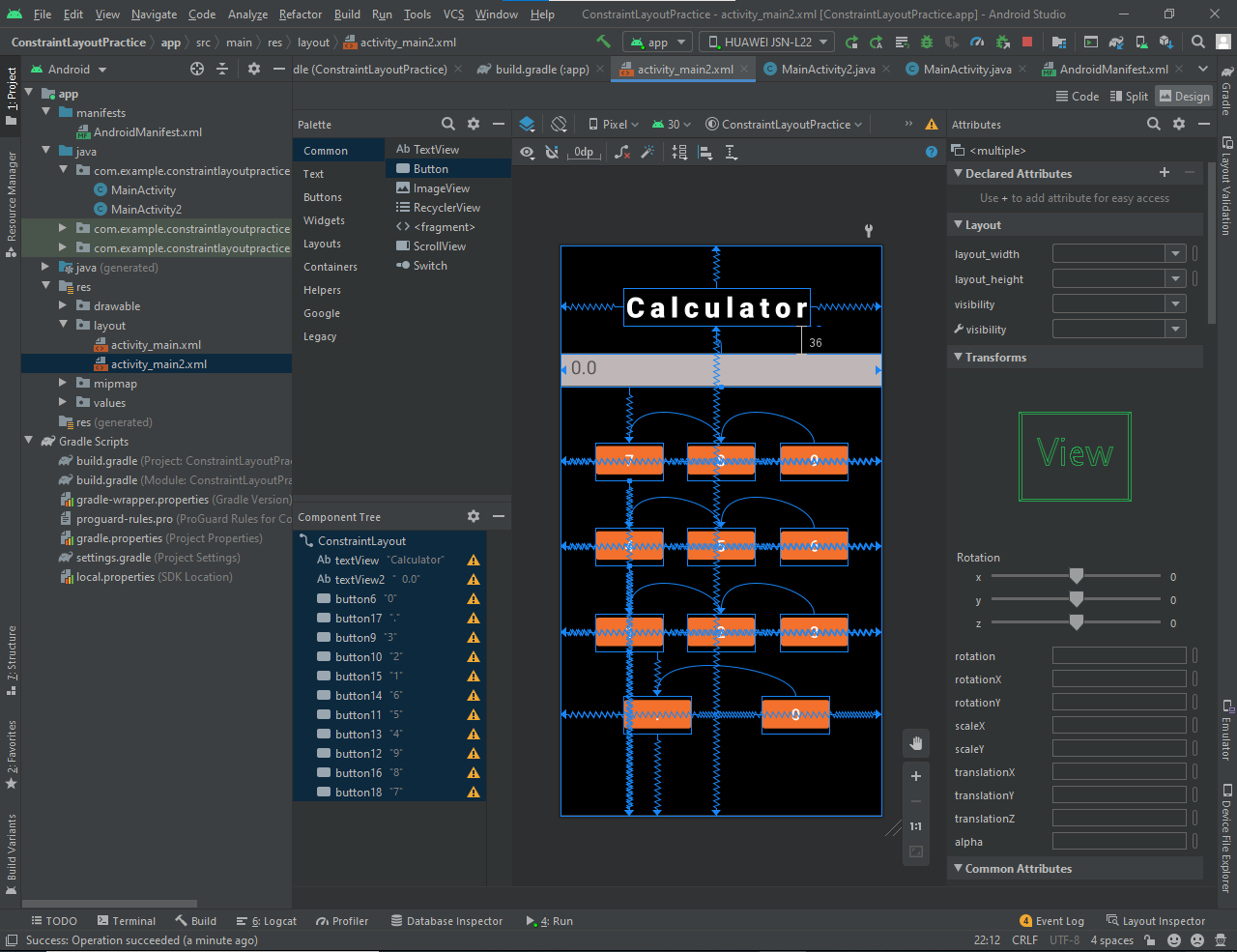
****

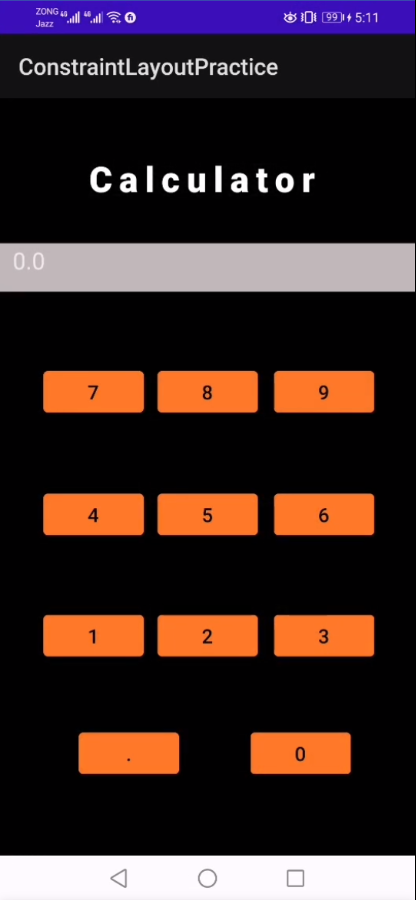
**Designed Layout-1**

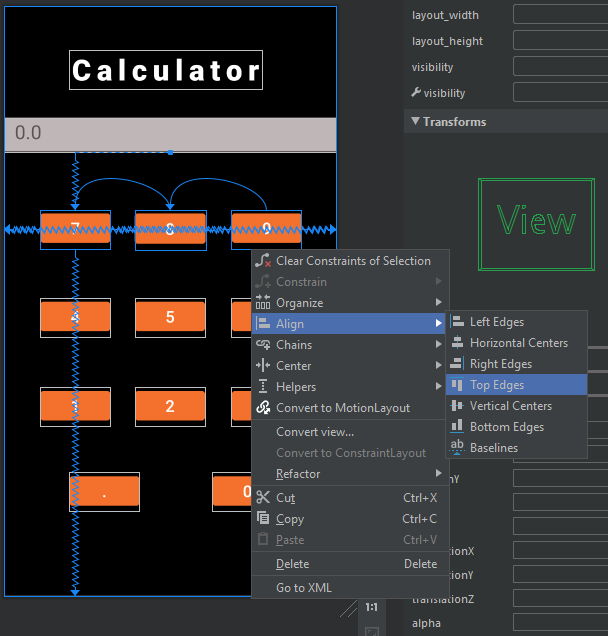
****

****

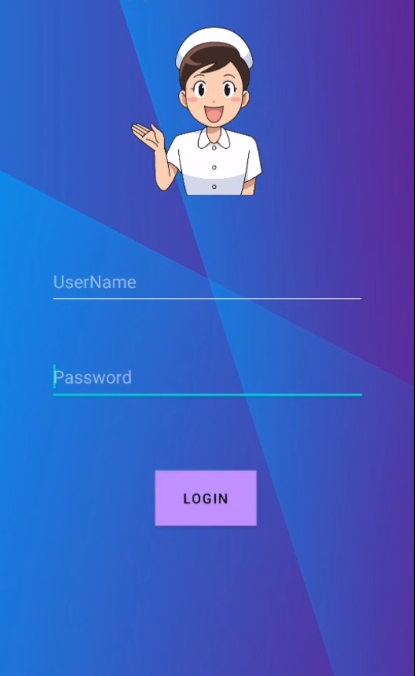
**Designed Layout-2**

****

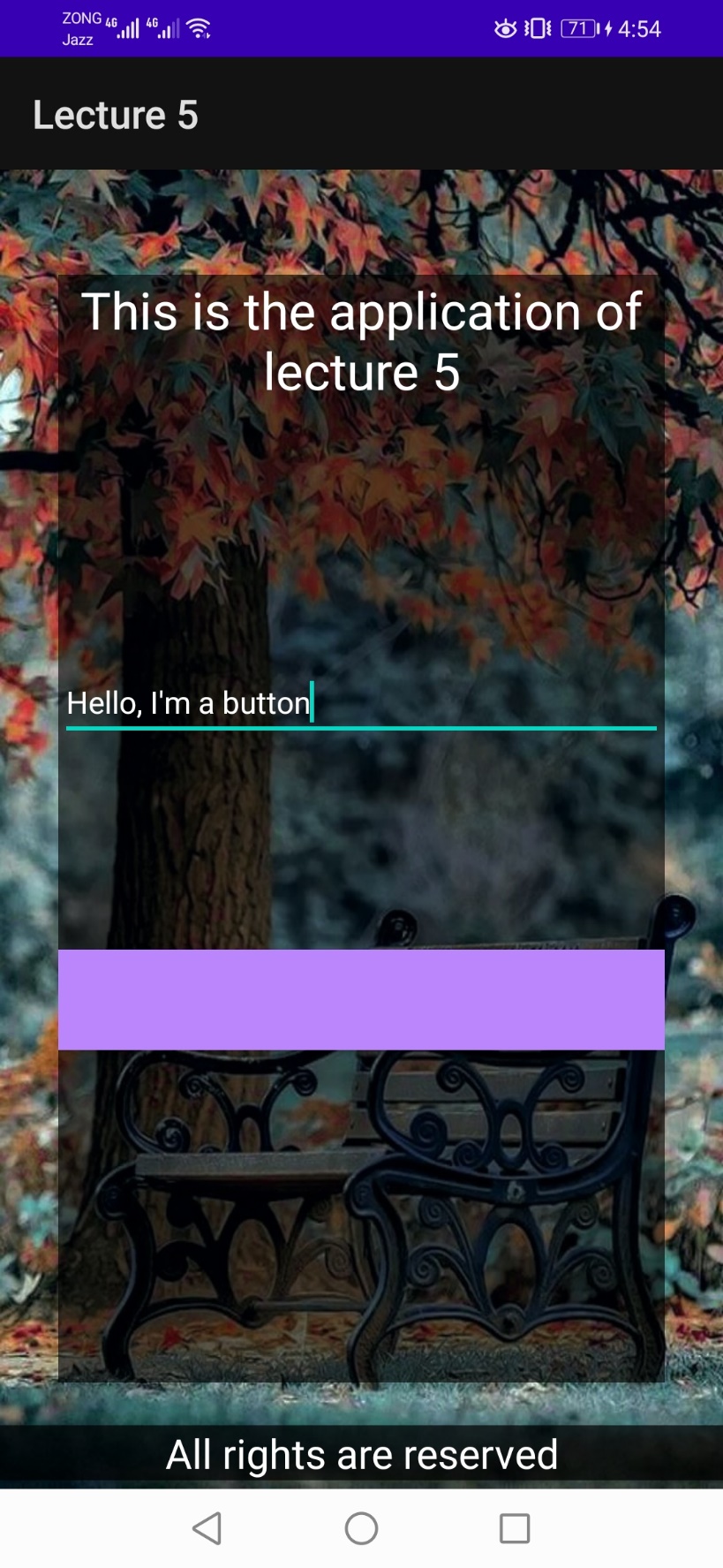
****

**We can align the components with respect to each other.   
**

**LOGIN Page**

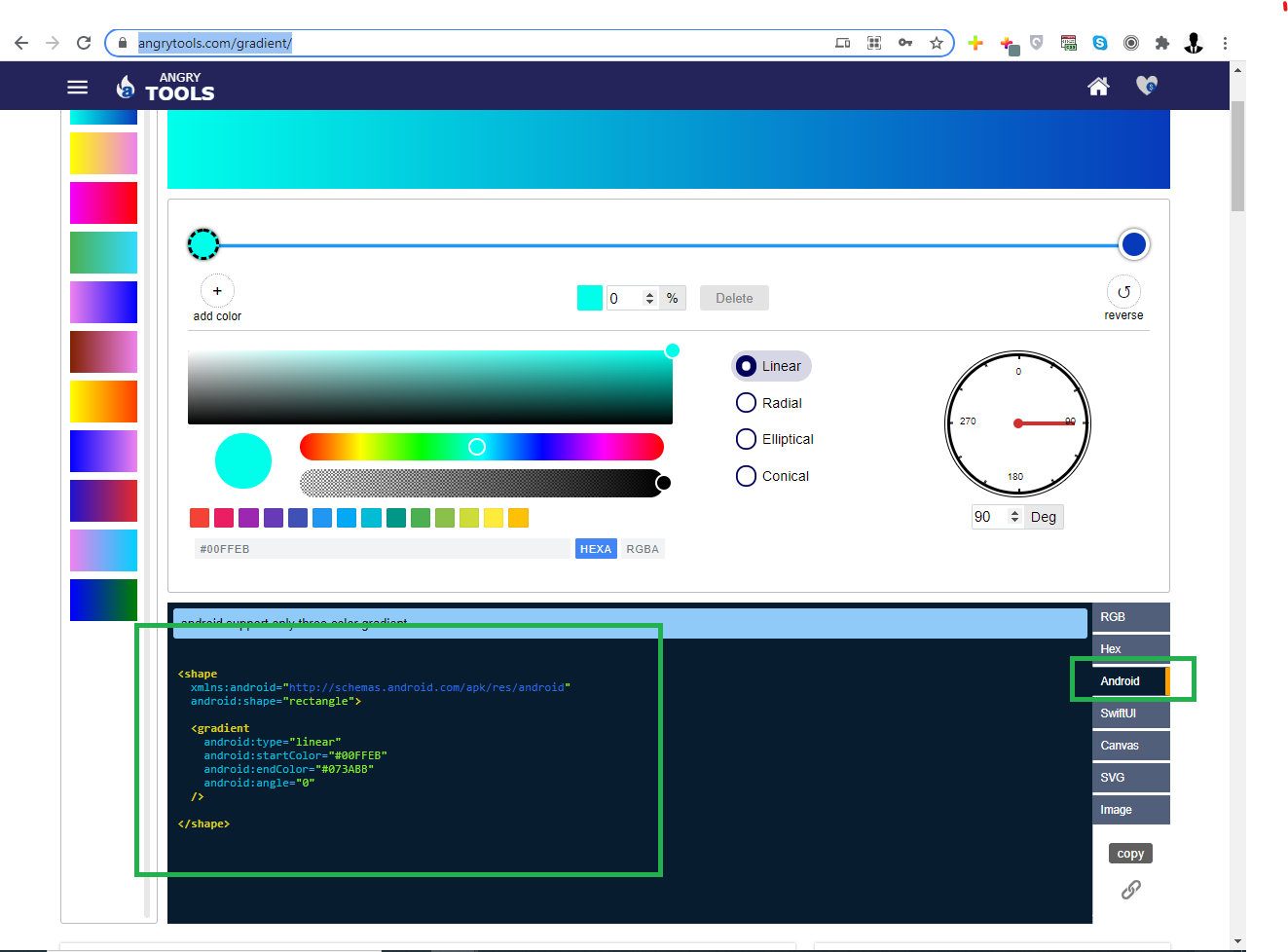
****

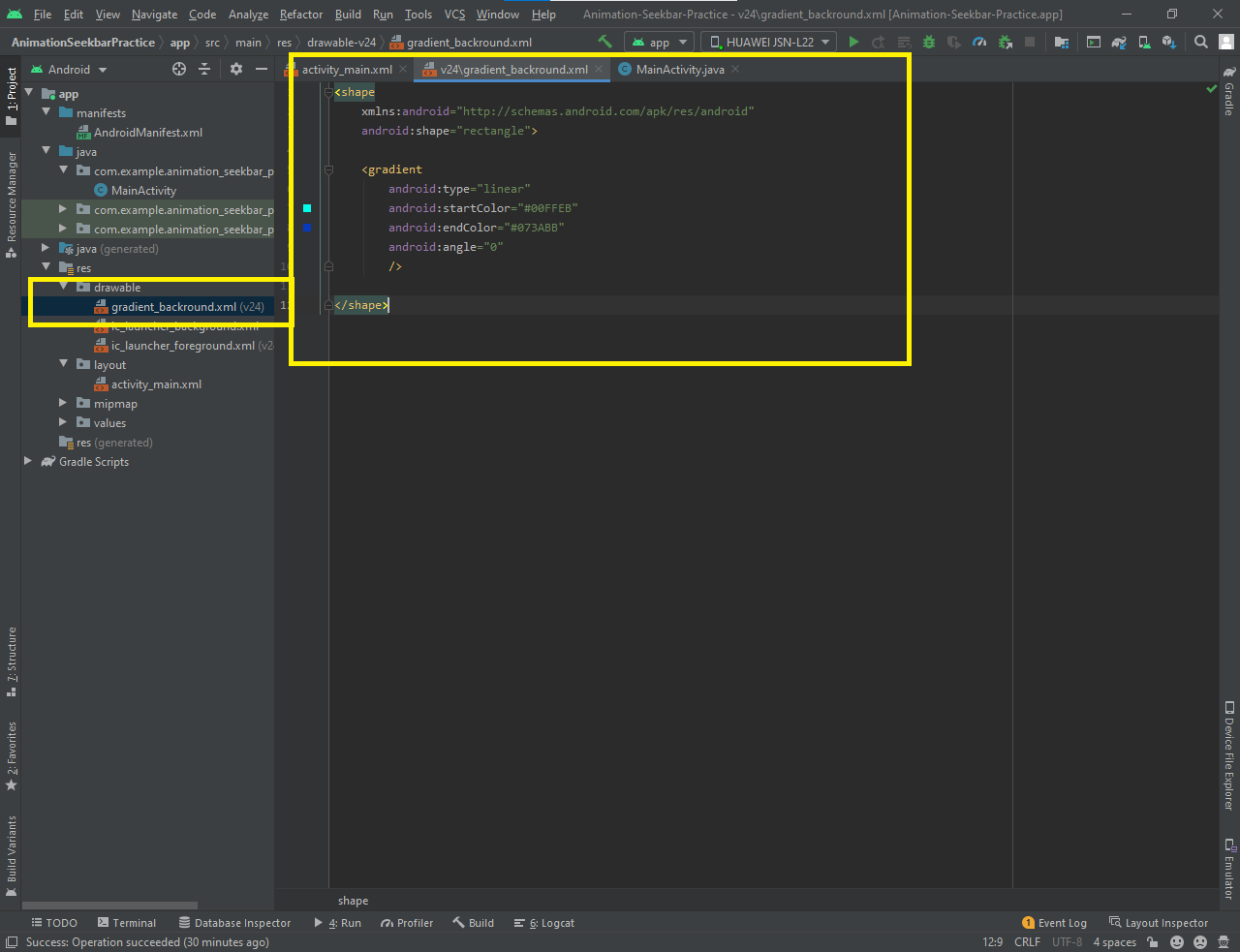
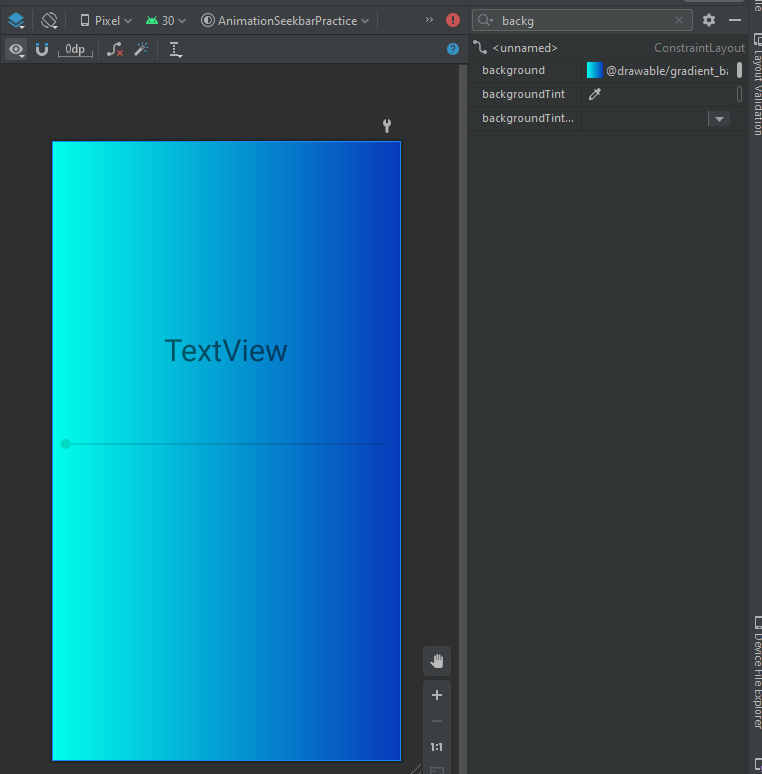
**Display text in button when clicked**



**Add Gradient In Background**

* **Make Gradient In XML file:** Go to this site and make your custom gradient.   
  <https://angrytools.com/gradient/>
* **Copy the code from code palette:**

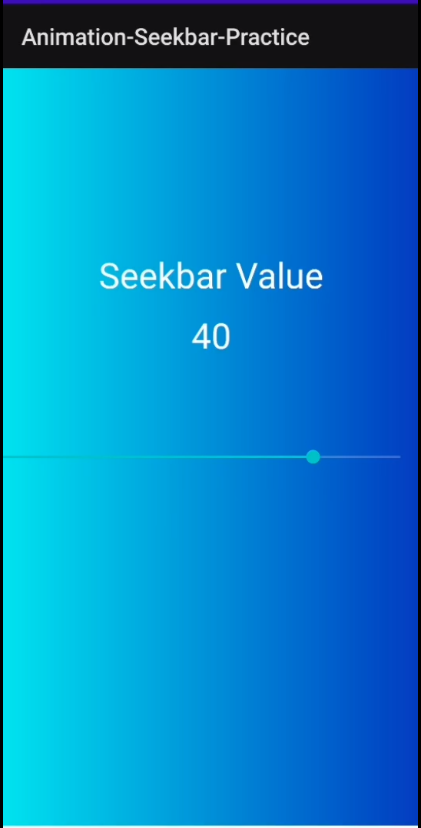
****

* **Make an XML file in drawable directory and paste this code in that file: **
* **Now you can apply this gradient as a background of any component: **

SeekBar

A SeekBar is an extension of ProgressBar that adds a draggable thumb. The user can touch the thumb and drag left or right to set the current progress level or use the arrow keys.

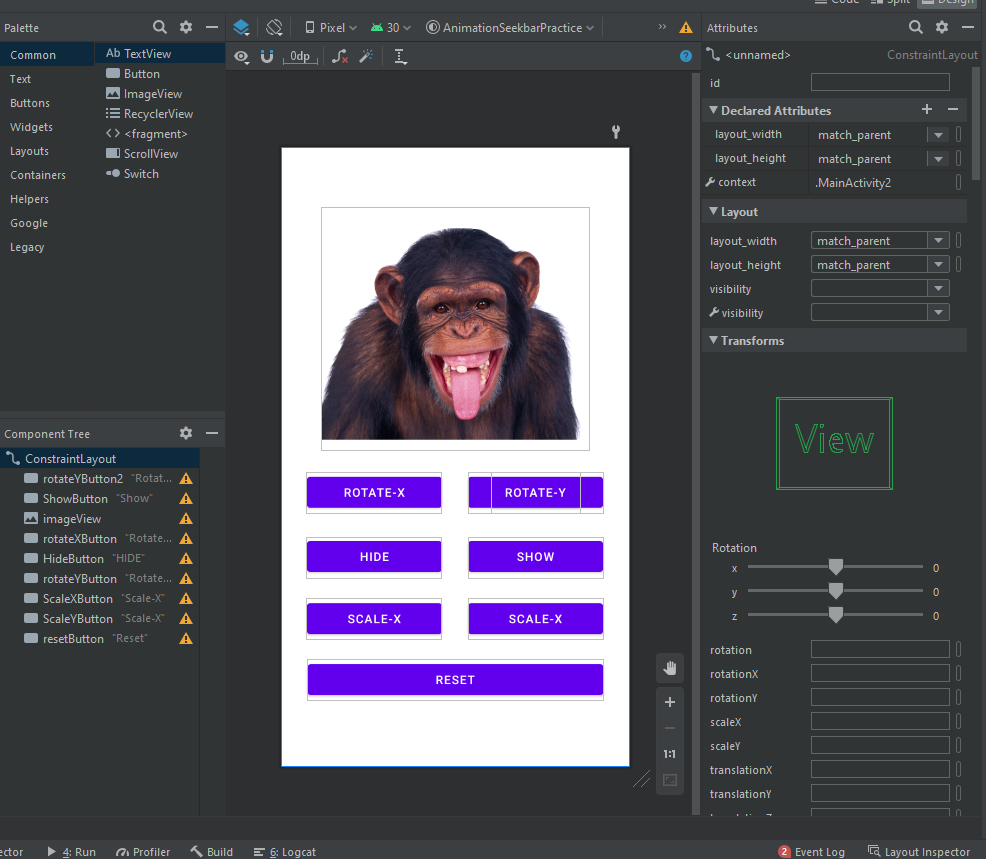
**Application**

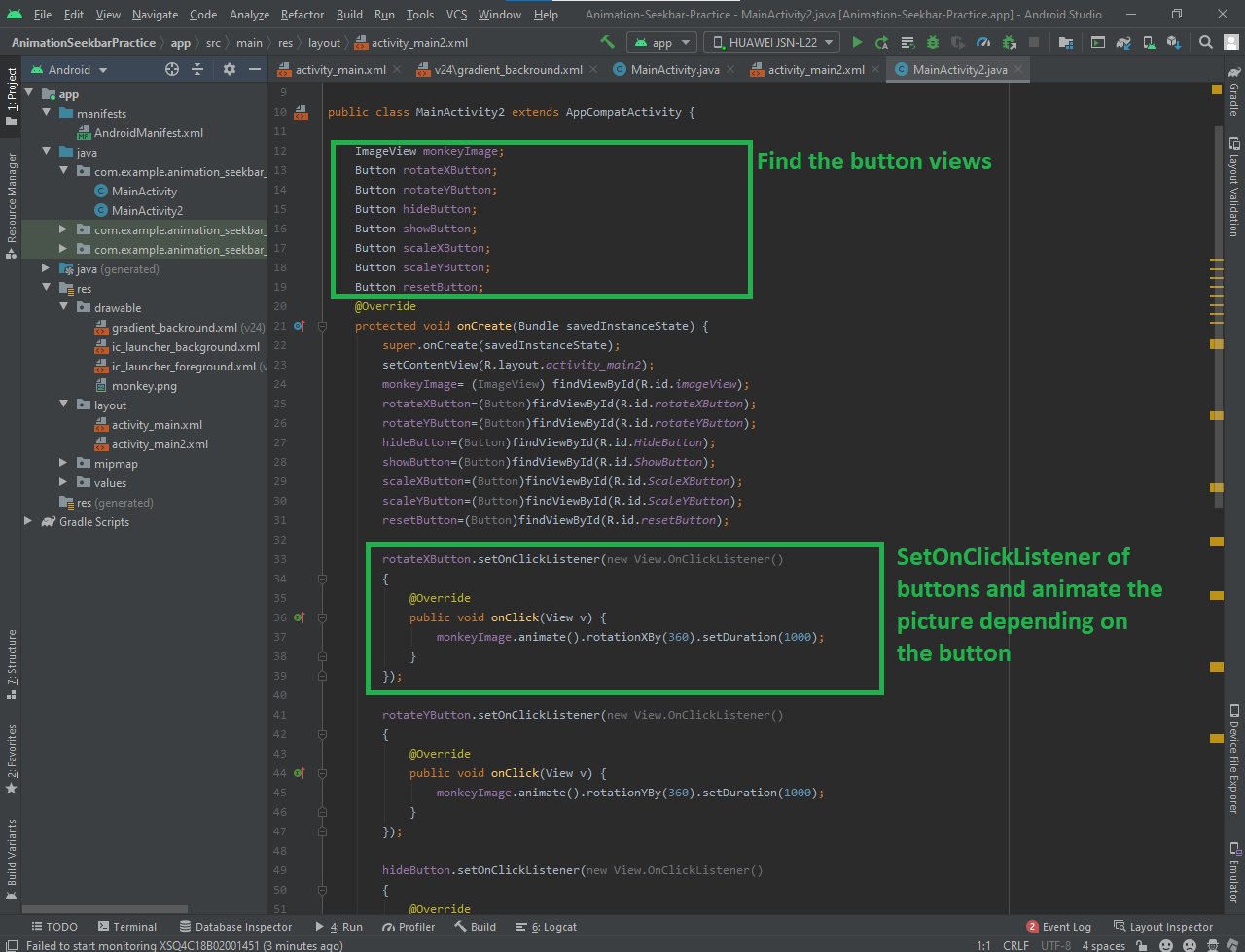
****

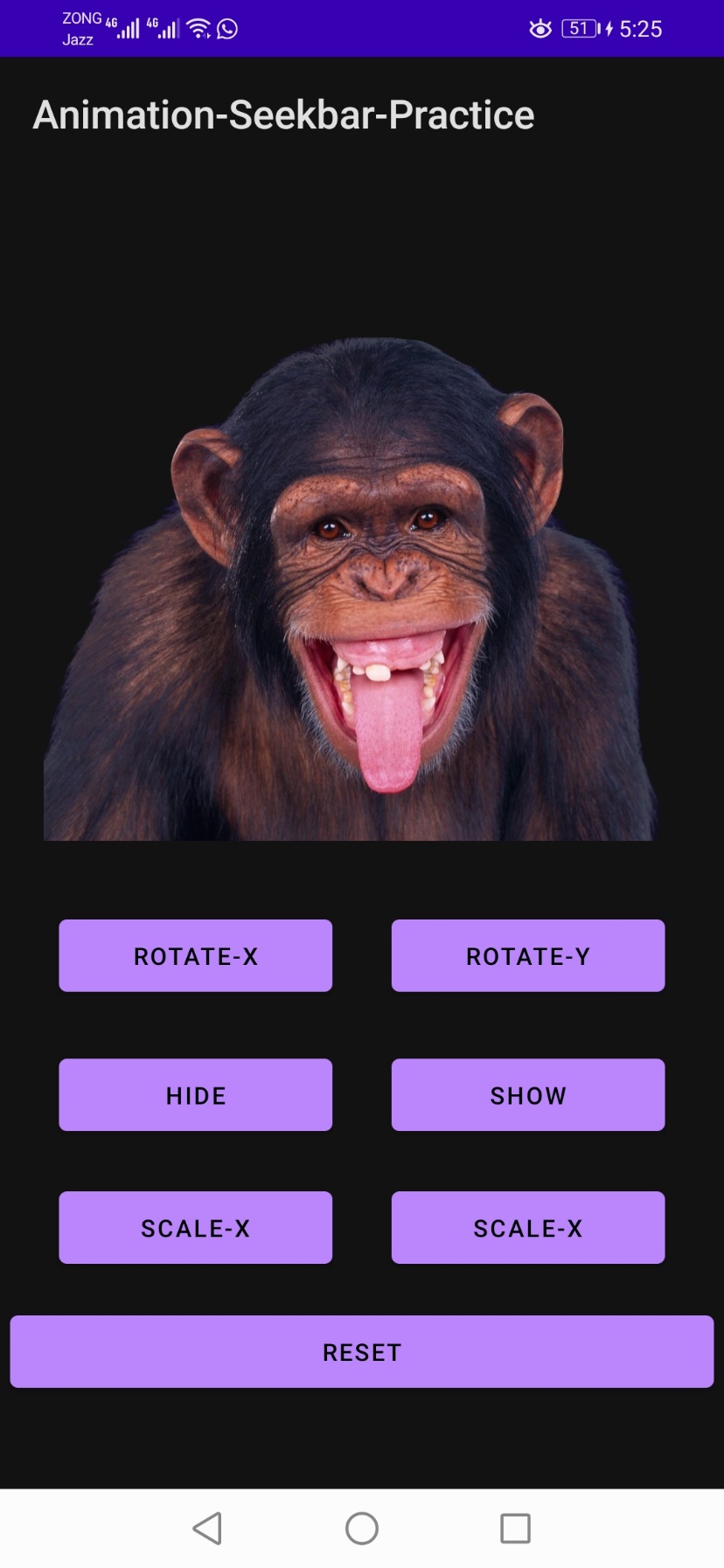
Animation

* To move, reveal, or hide views within the current layout, you can use the property animation system provided by the android.animation package
* **Application 1:**

Code

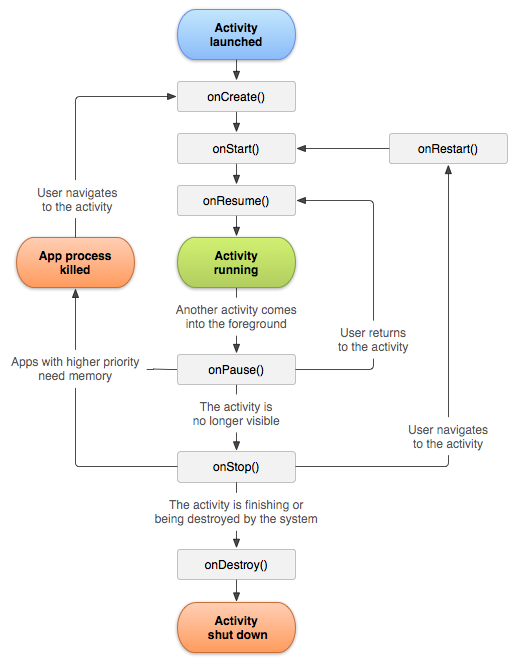


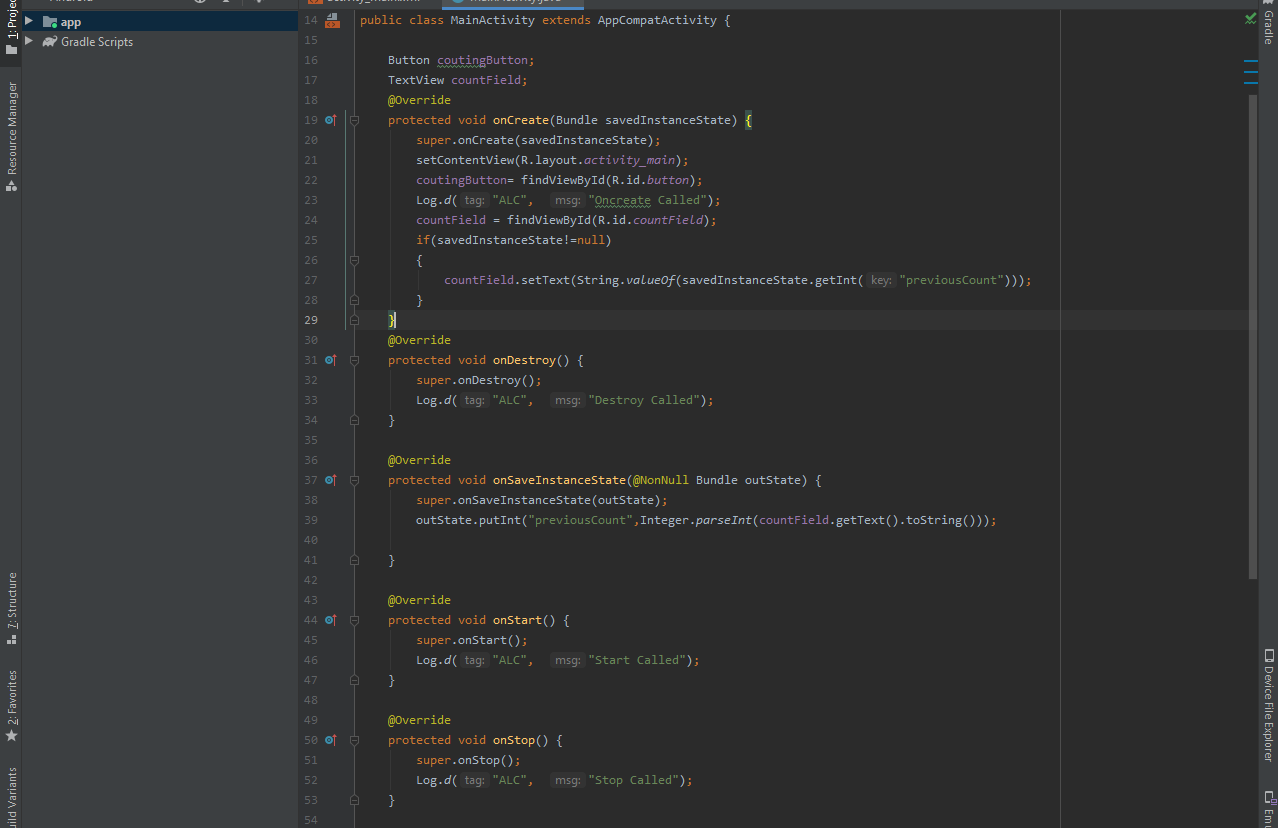




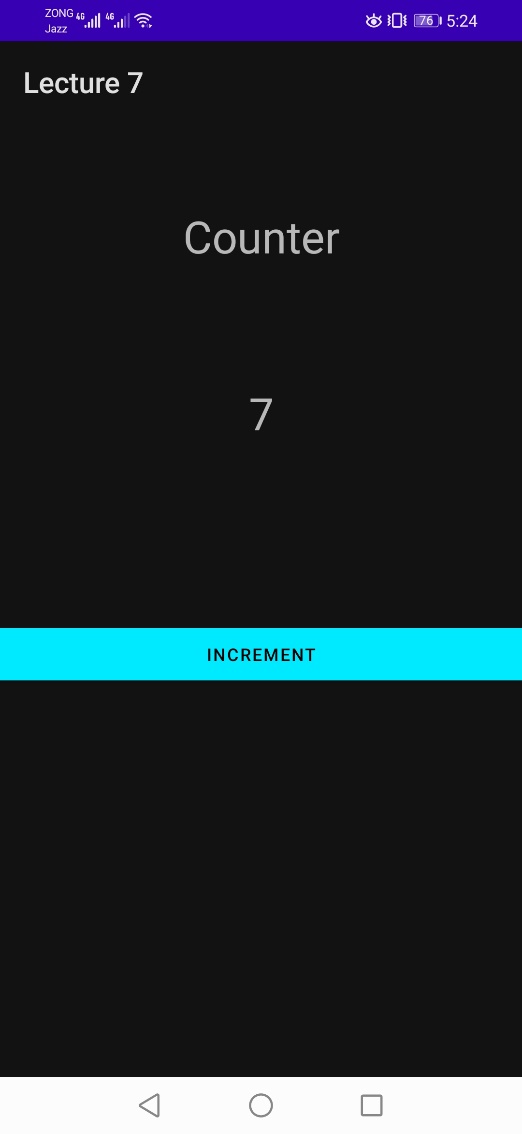
Activity Life Cycle

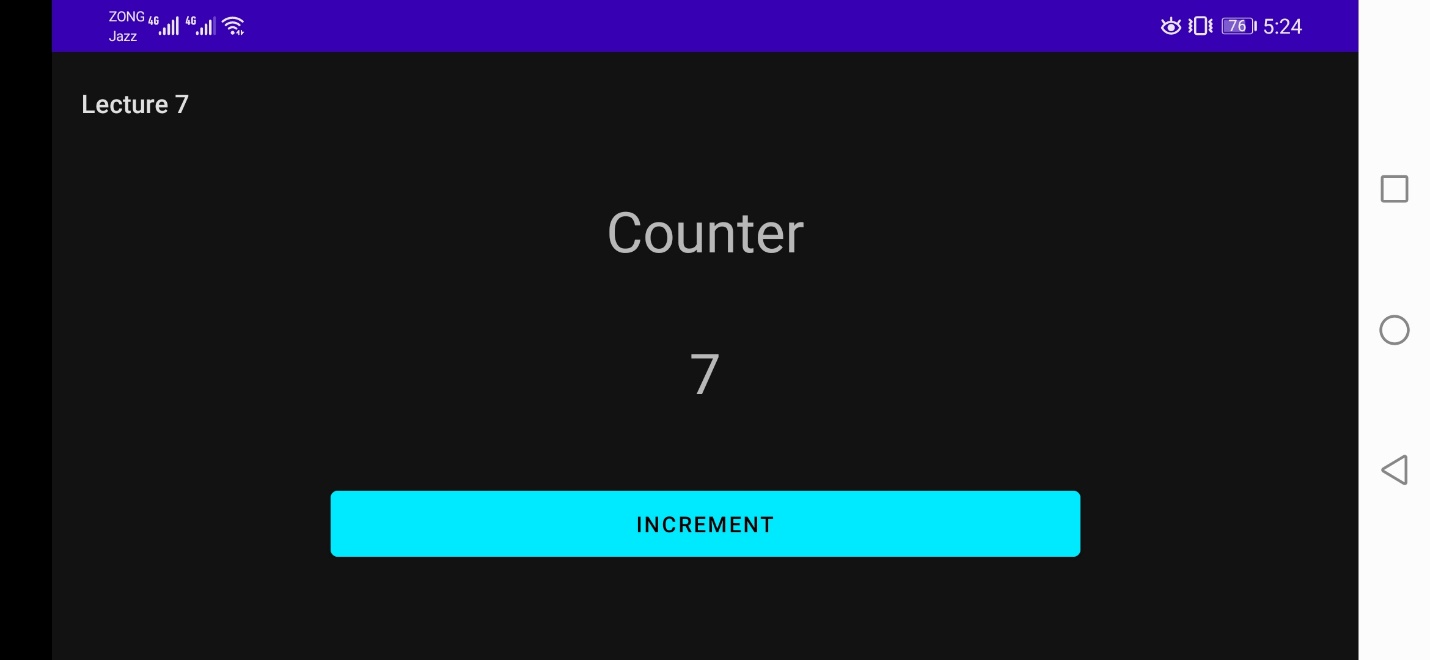
To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.







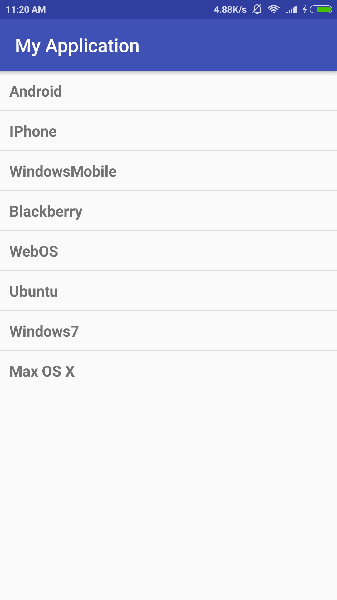
\*State remained same when the device is rotated



ListView

**Android ListView** is a view which groups several items and display them in vertical scrollable **list**. The **list** items are automatically inserted to the **list** using an Adapter that pulls content from a source such as an array or database. ArrayAdaper class acts as an adapter between the data source (array) and the list view. It populates the data on the listview items.

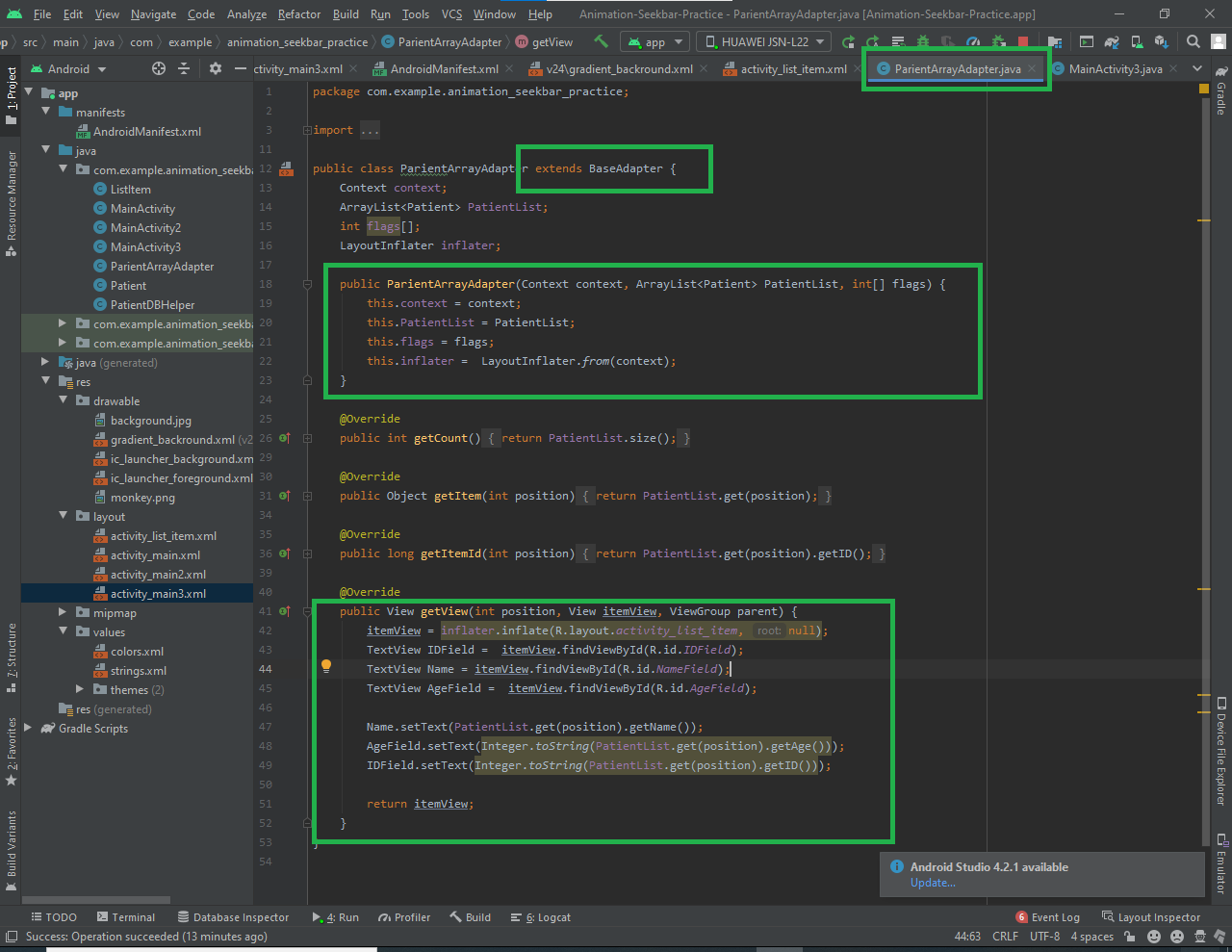
A simple listView with string values a items

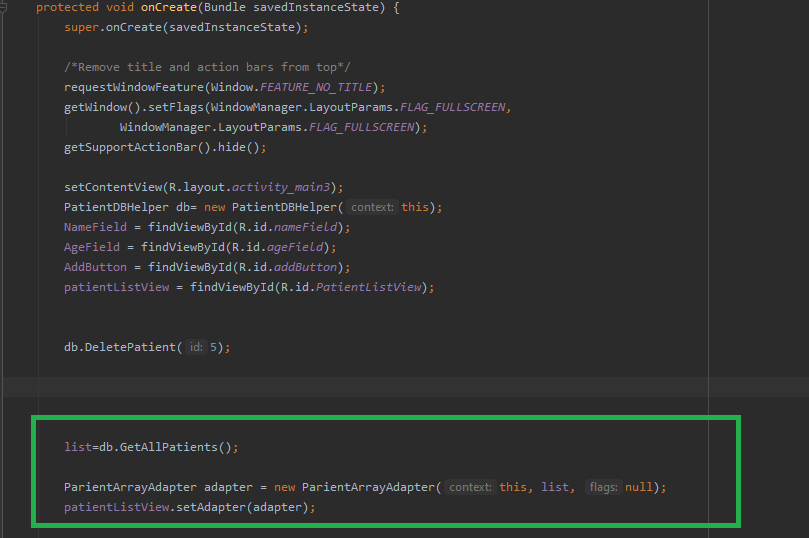


**ArrayAdapter<String> adaper = new ArrayAdapter<String>(getApplicationContext(), R.id.ListViewItem, Array)**

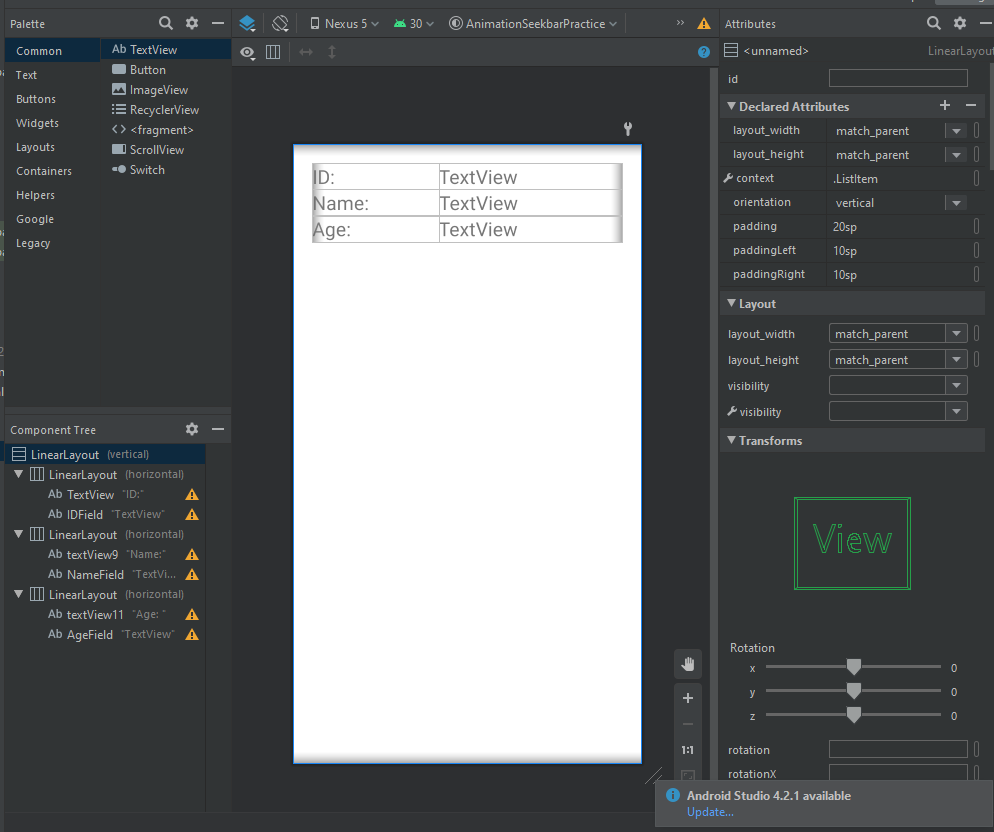
ListView with custom listview item

By default, the array adapter expects a single textview in the listview item. But if we want to make a complex/custom listview item then we have to define our own ArrayAdaper class for our listview which extends the BaseAdaper class and overrides some of its methods.

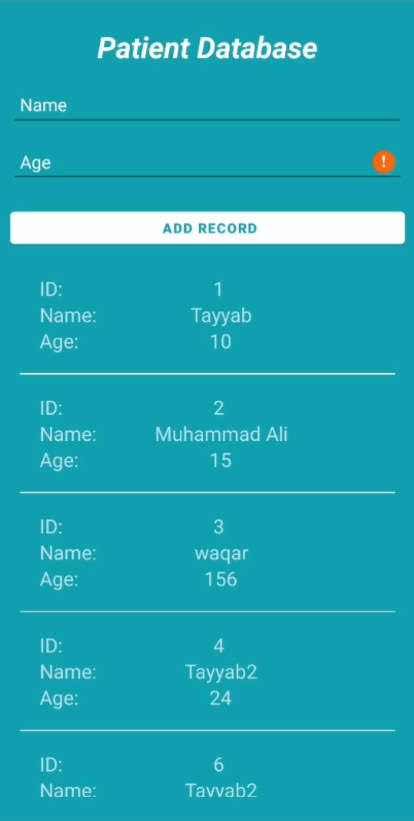




Custom List View



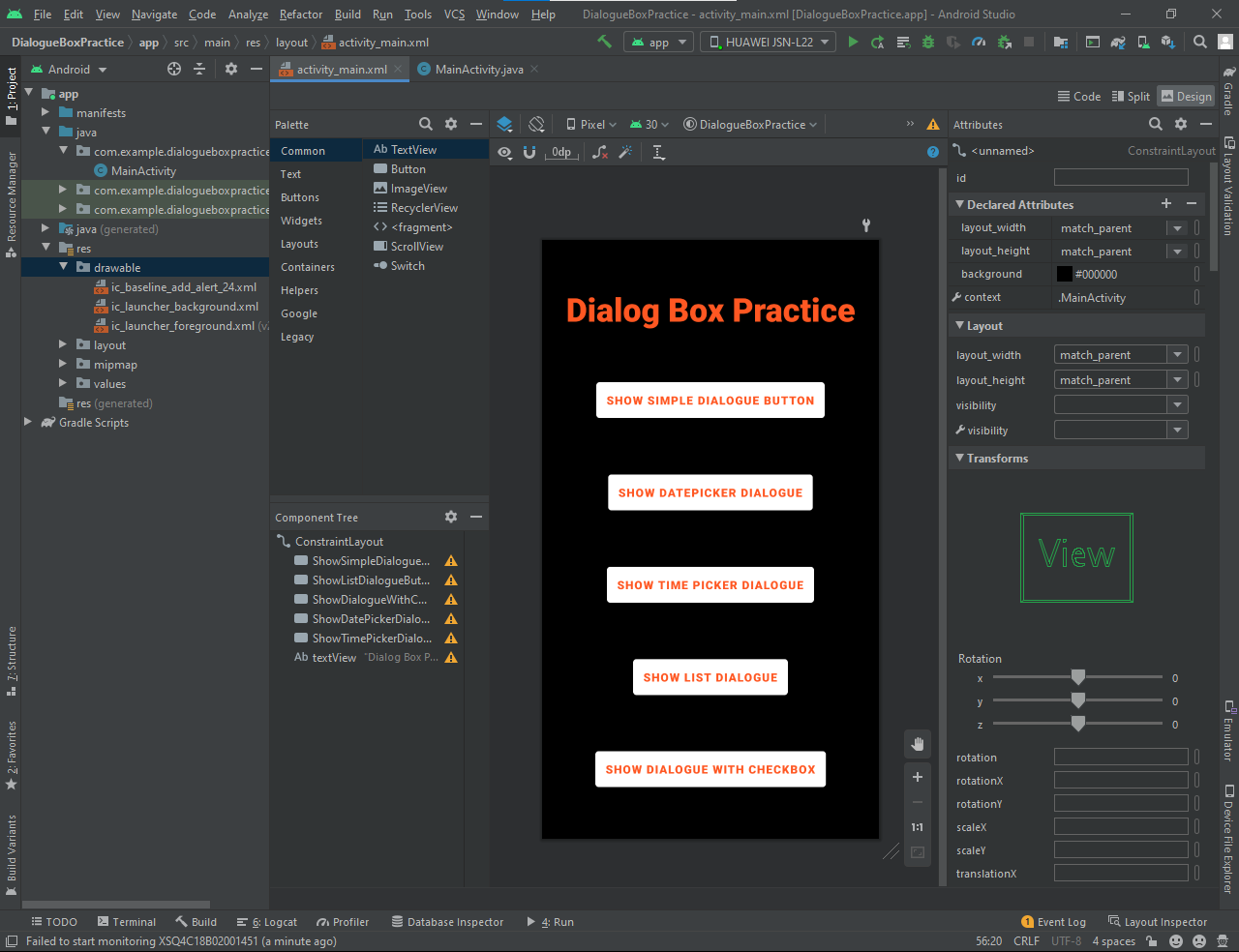
Application



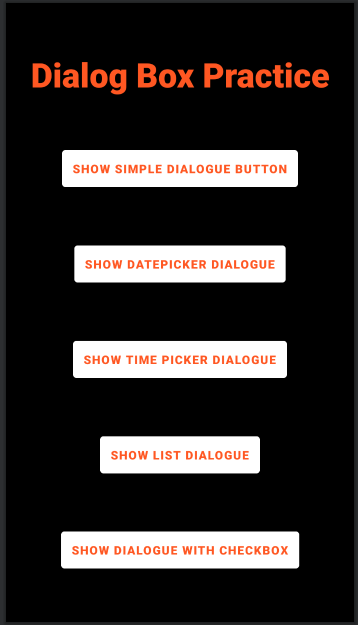
DIALOGUE BOX

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

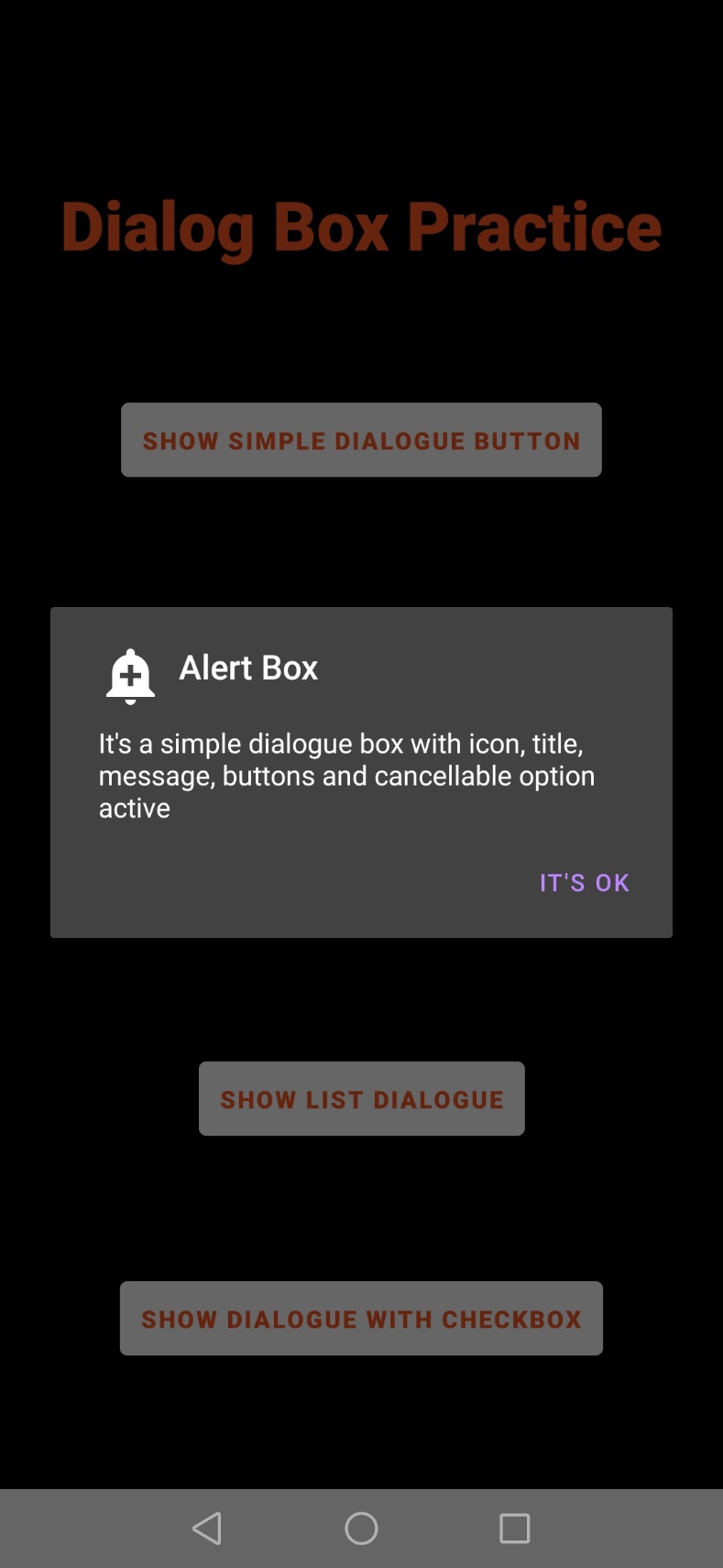


First, we design our desired Dialogue Builder then we make our Dialogue Box from that builder.

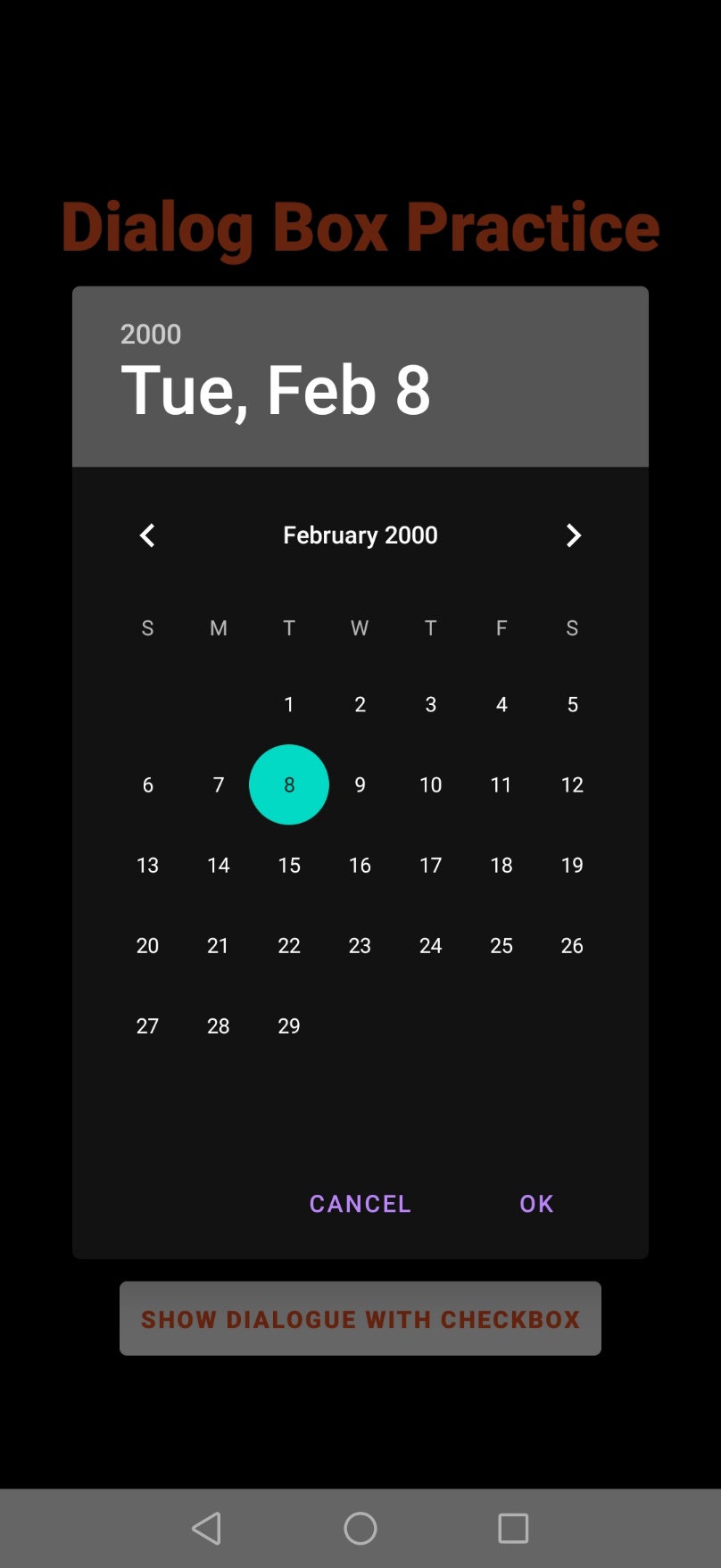
**Interface**



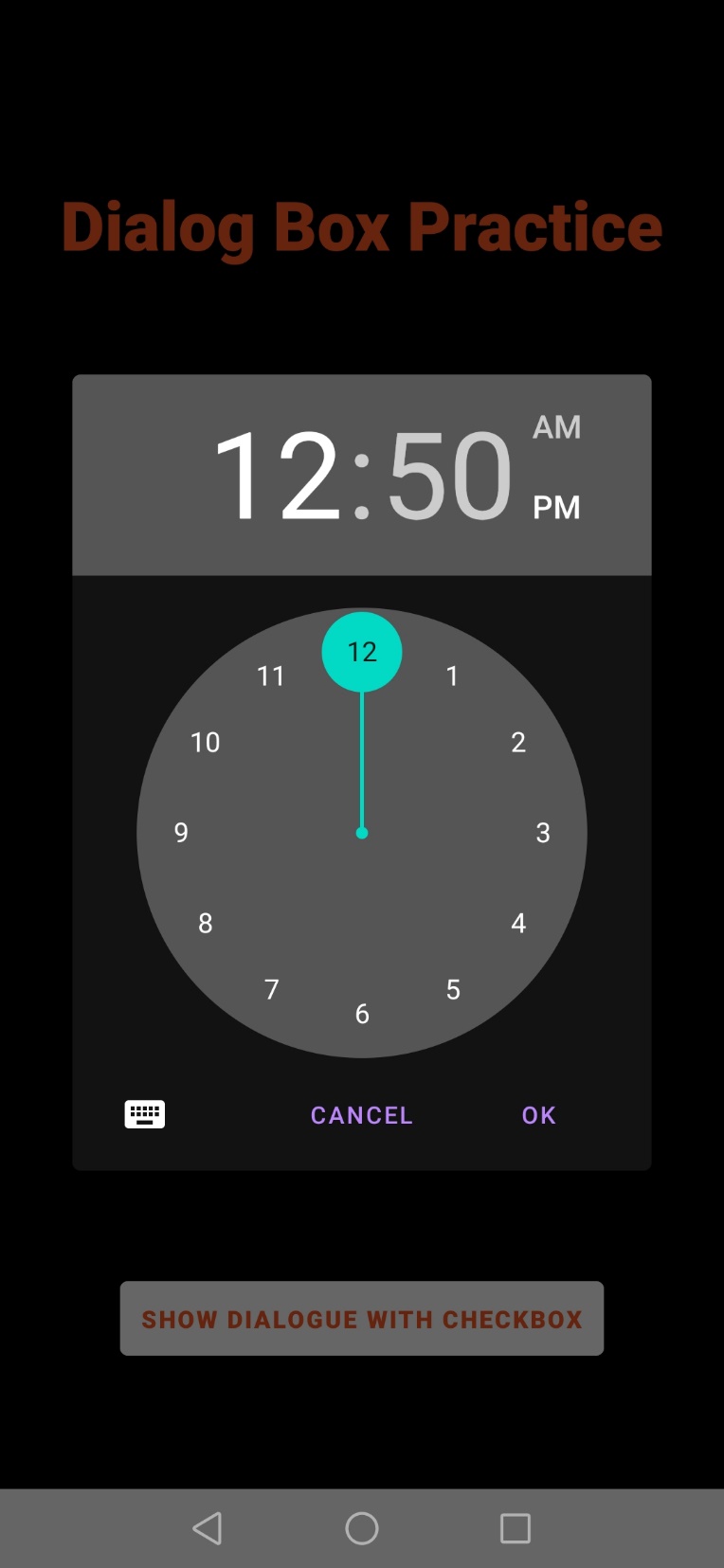
**Simple AlertDialogue**



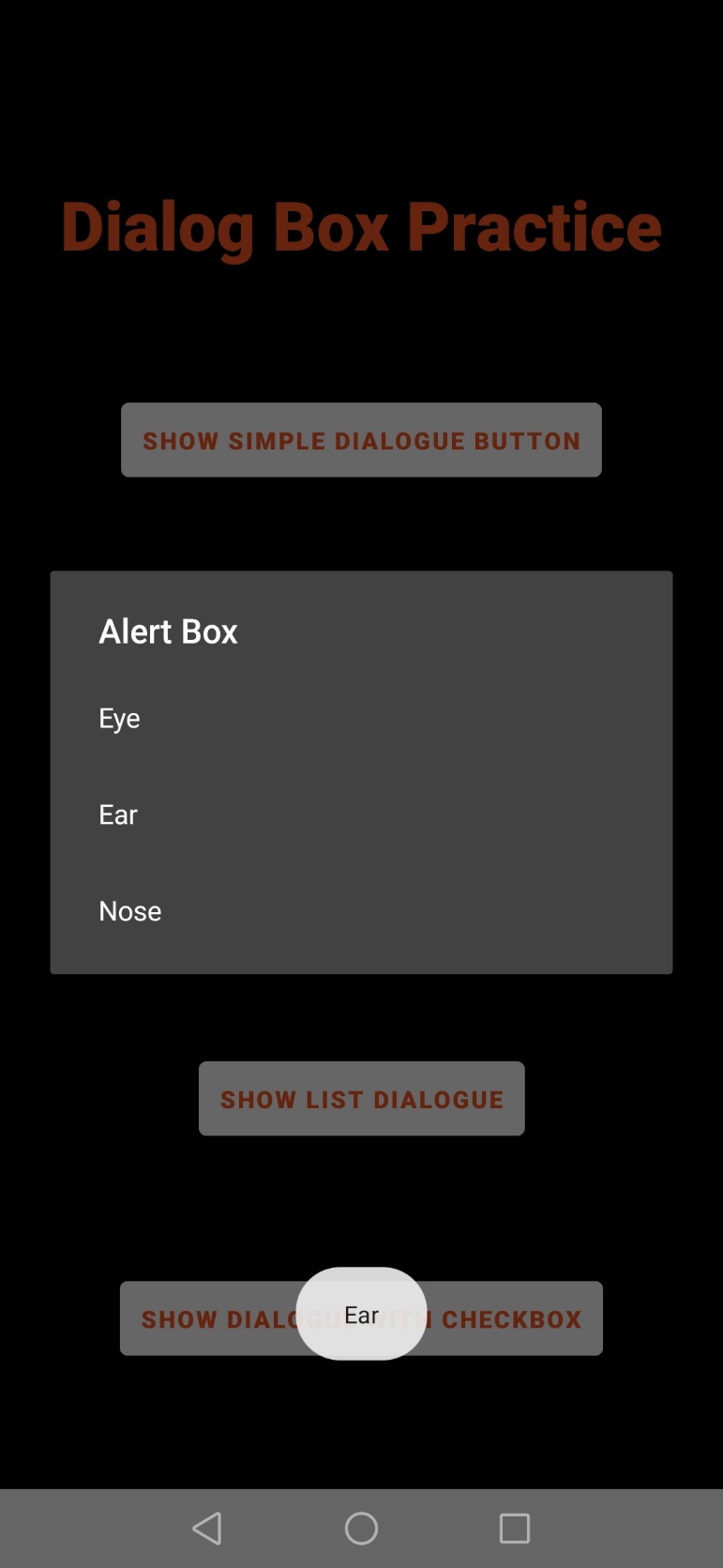
**DatePicker Dialogue Box**



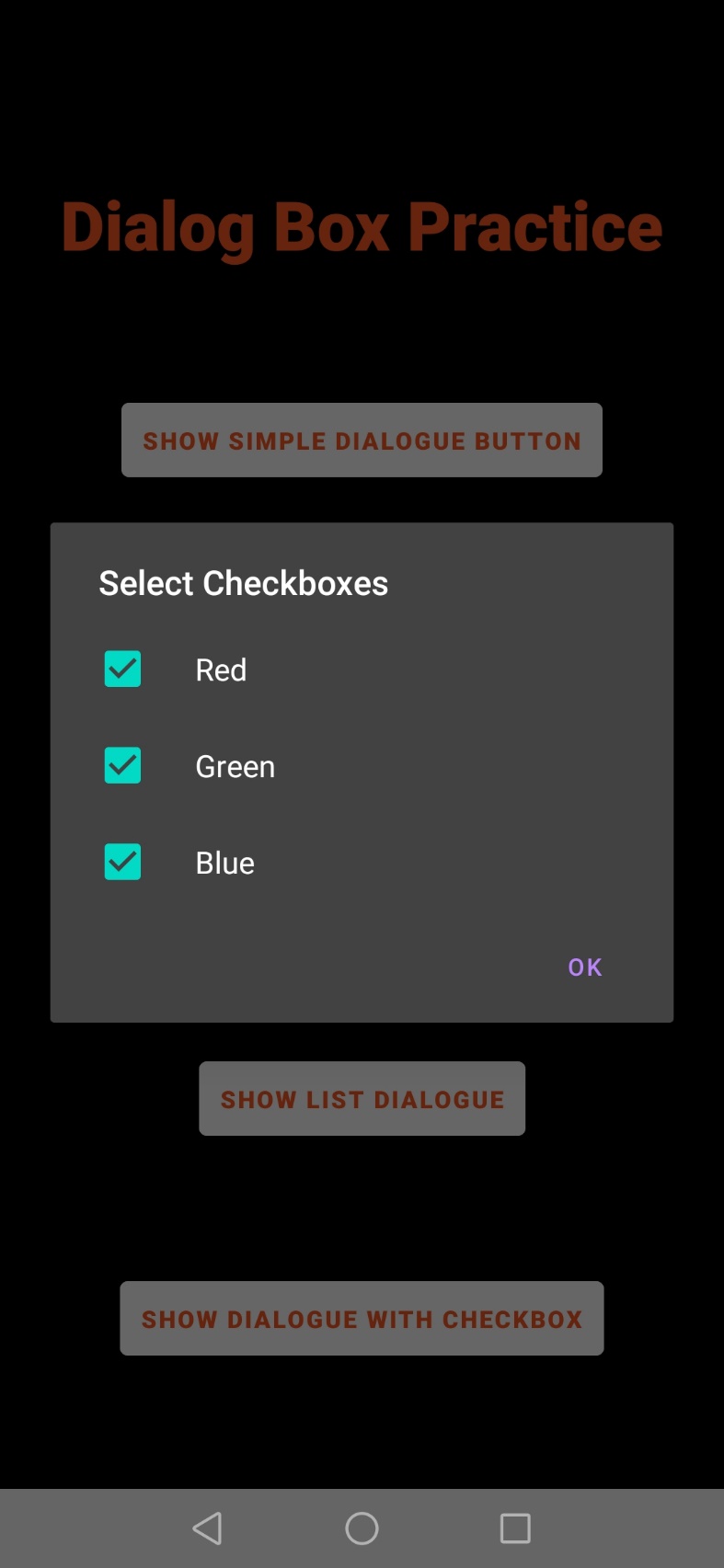
**TimePicker Dialogue Box**



**List Dialogue Box**



**Checkbox Dialogue Box**

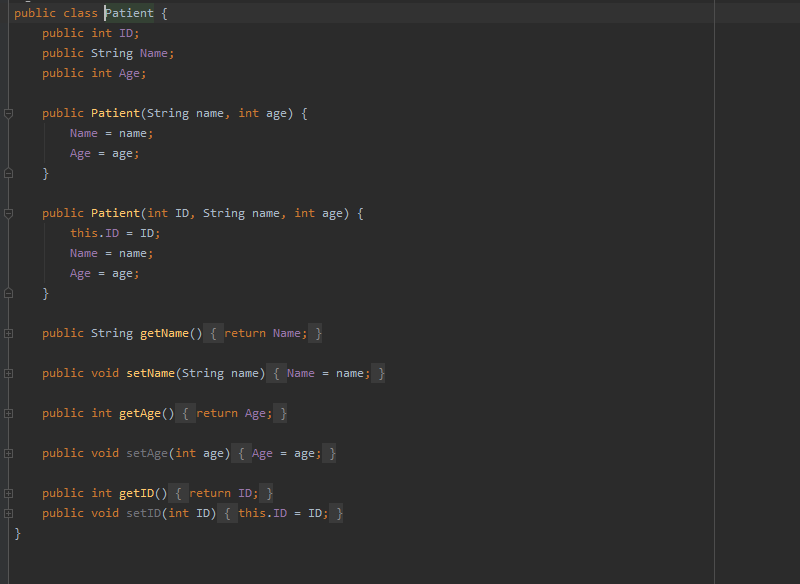


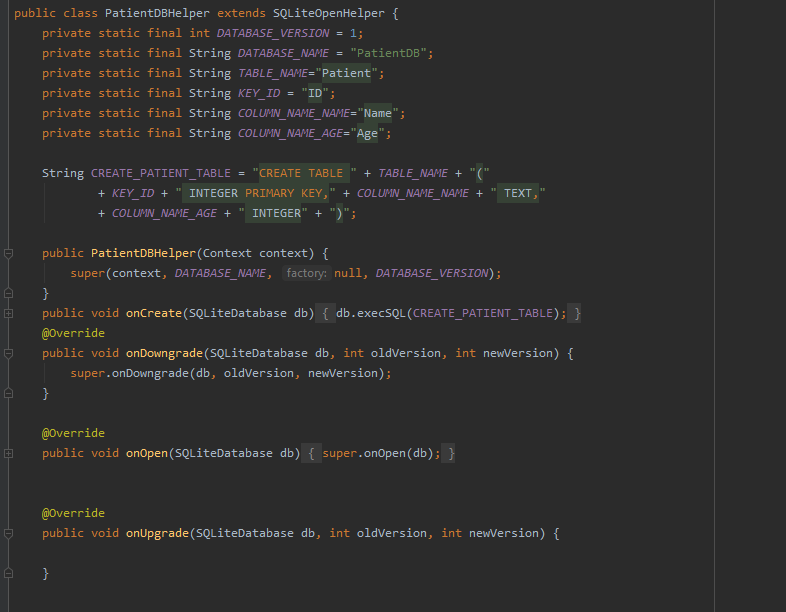
SQLITE DATABASE

SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks. We have to define a DBHelper class for each database to perform CRUD operations in our database tables.

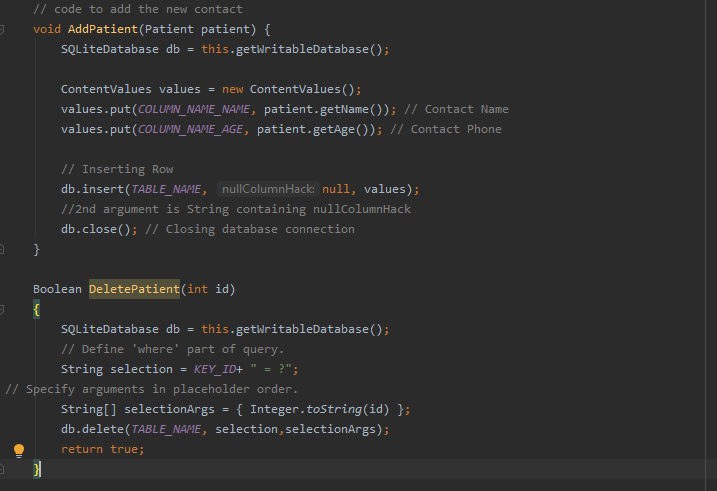
Each DBhelper extends the SQLiteOpenHelper class. We have to override some methods like constructor(), onCreate() etc. In constructor we execute the query to create db and in oncreate we execute the query to make tables. And we’ve to define the CRUD functions by getting readable and writable database from SQLiteOpenHelper classes.

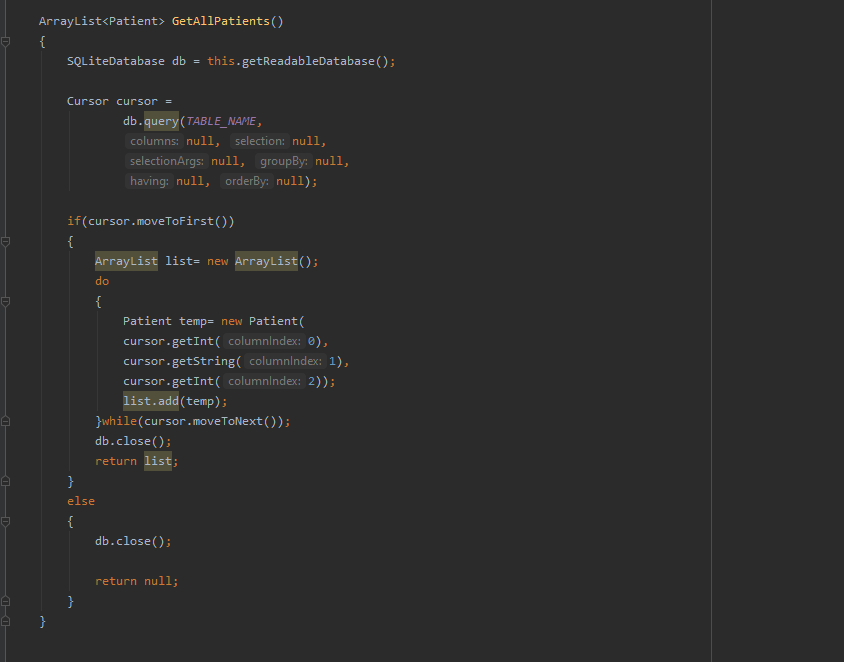
**Model Class**:



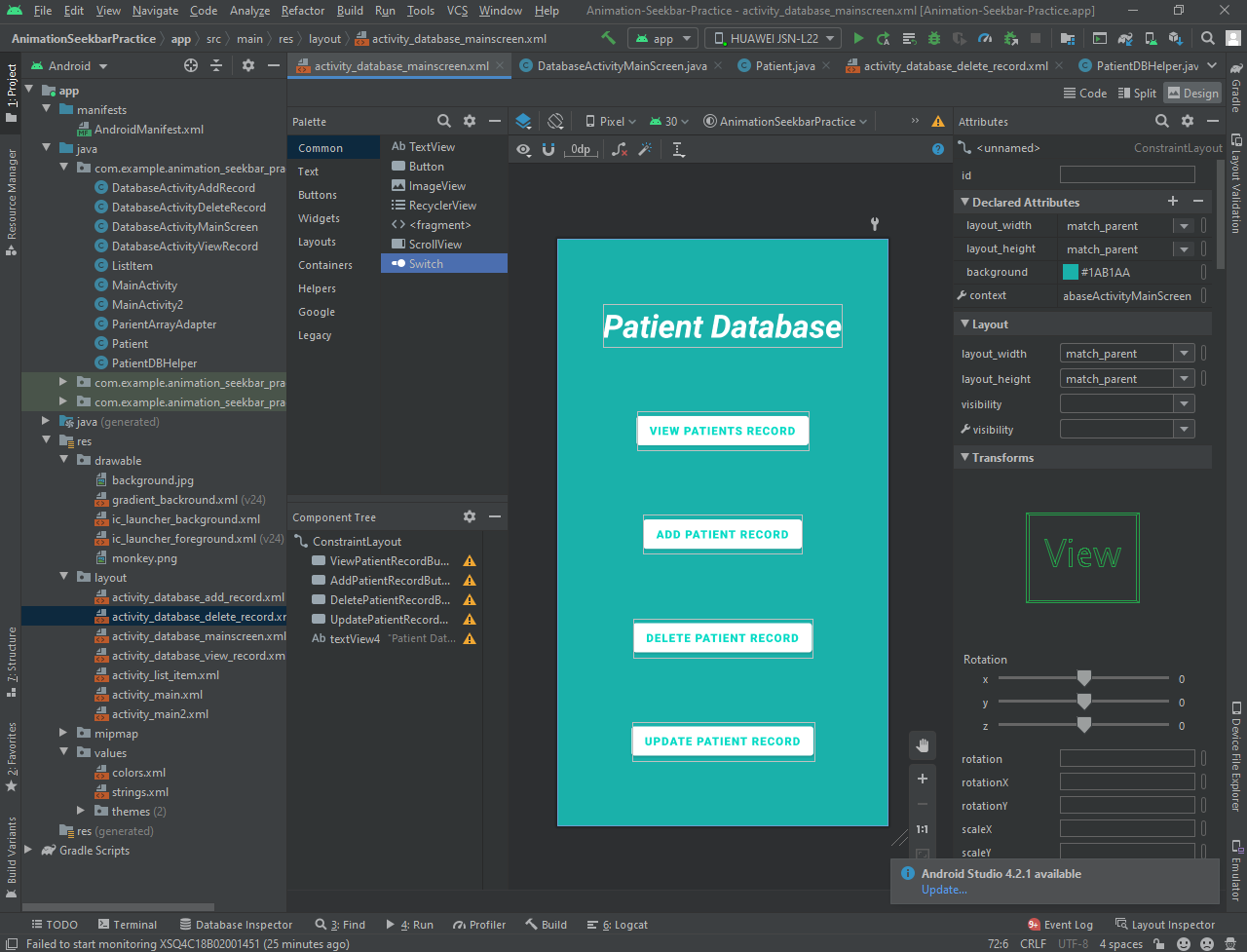
**Override Methods:** 

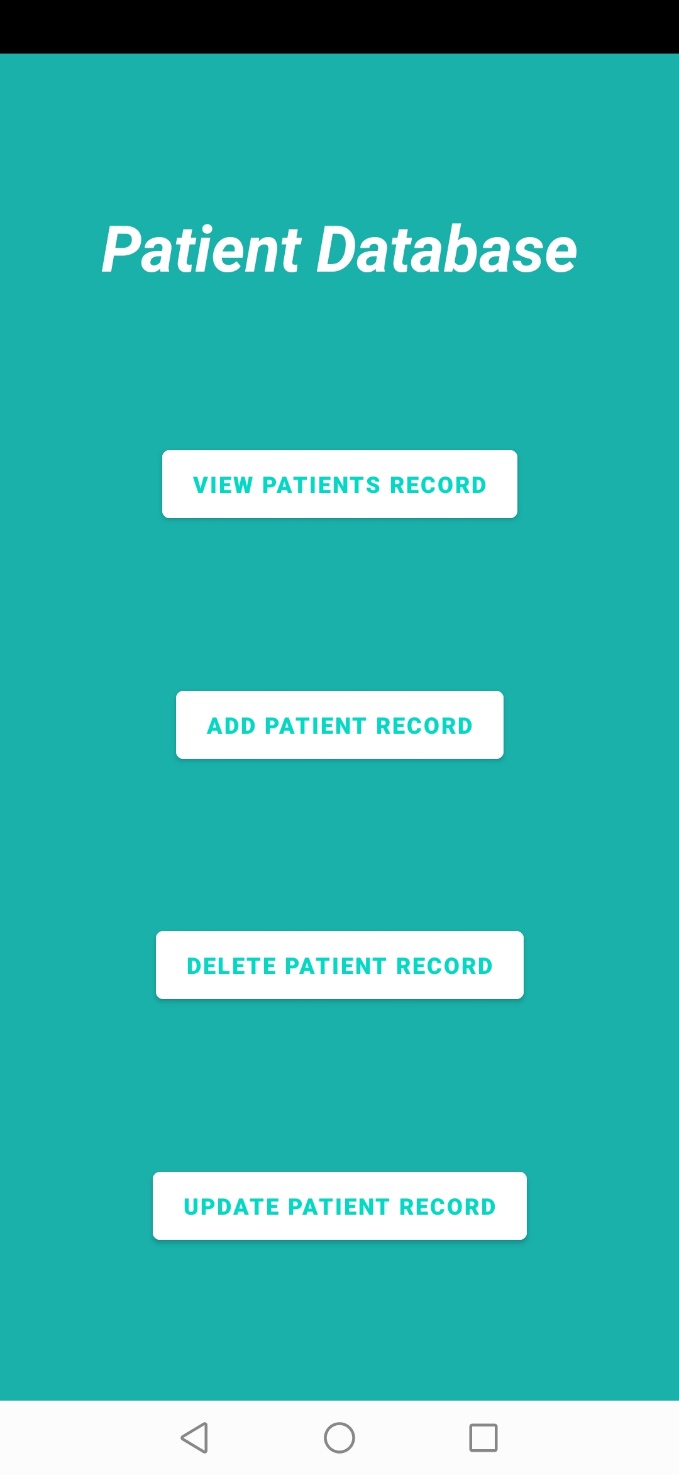
**CRUD Methods:**

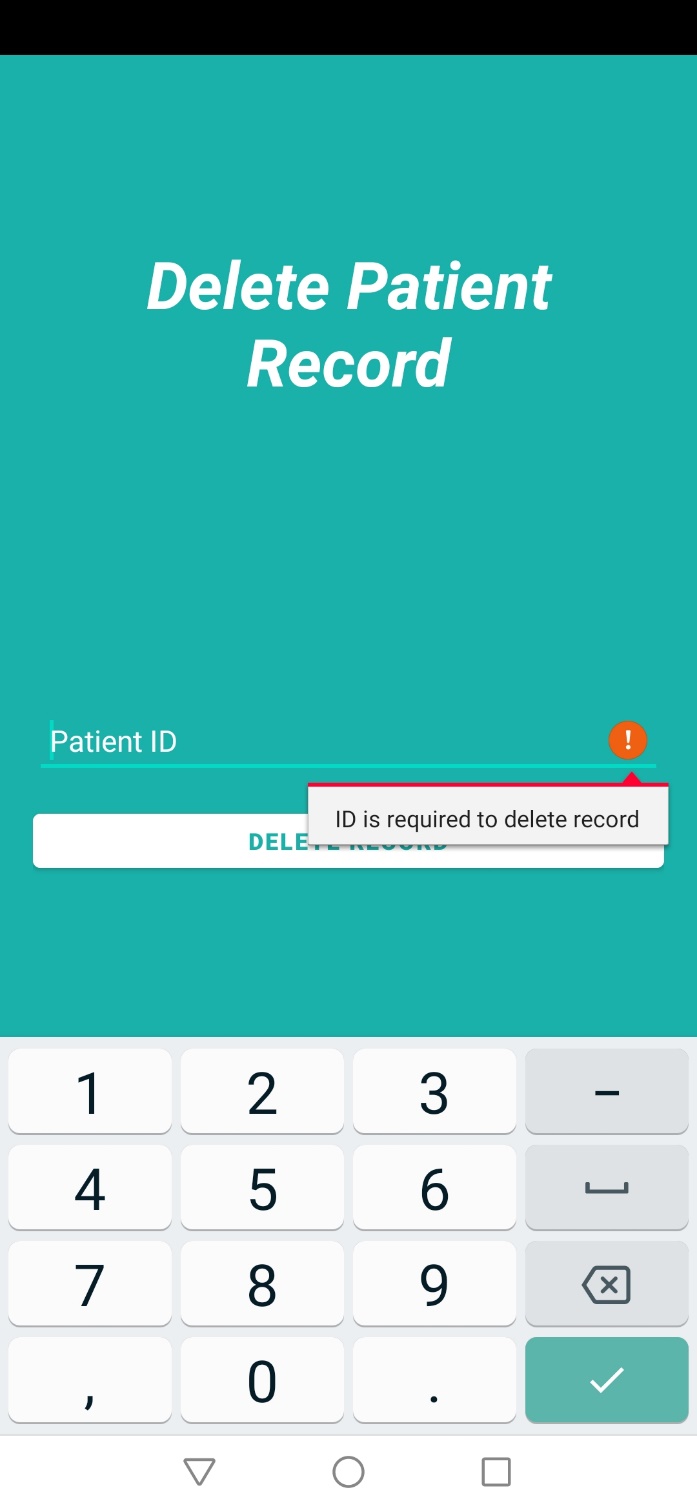
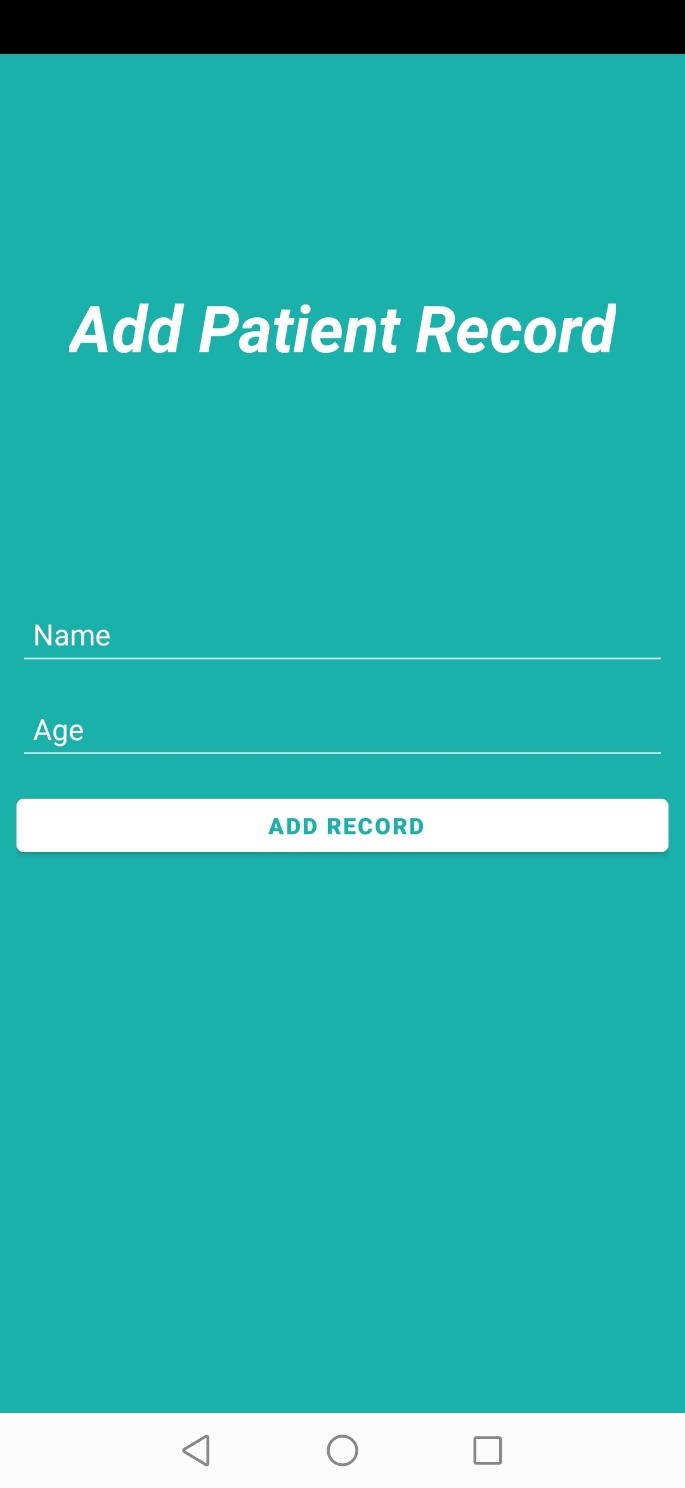




**Application**

****

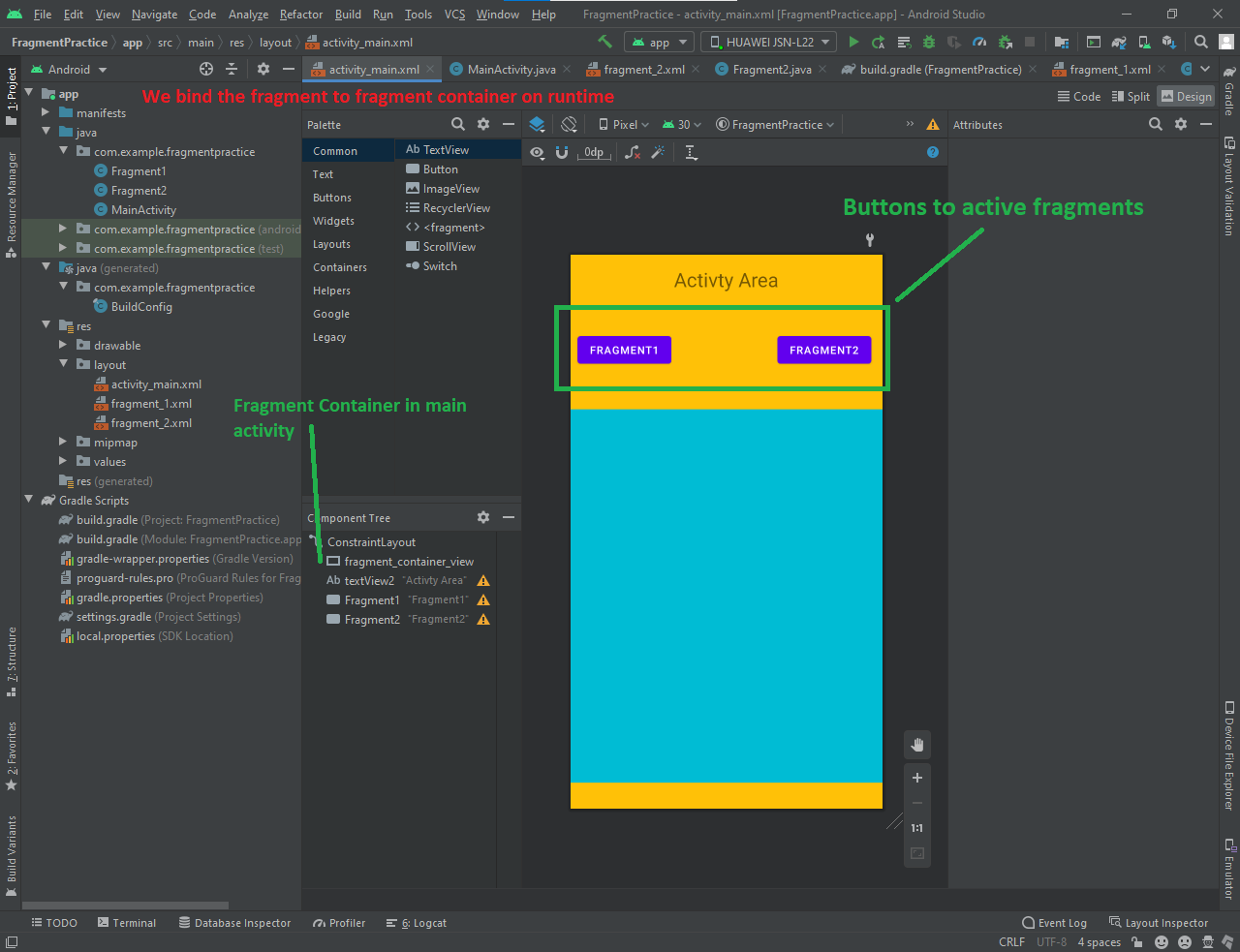




Fragments

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments cannot live on their own--they must be hosted by an activity or another fragment.

**Application**



MainActivty with a fragment container (replace on button click)



 Fragment – 1 Fragment - 2