Livedata example in android

In the name of the Great God, the most merciful of late, Google IO 17 has announced a new component of architecture and in this article we will talk about two of them, namely ViewModel and LiveData. First of all, suppose we have an app like this when you press the Add One button will add 1 to the value in textView, and the layout code will be as follows. The activity is that the public class mainActivity zlt;android.support.constraint.Constraint.Constraint.Constraint.Constraint.ConstraintLayout xmlns:android/ xmlns:app schemas.android.com/apk/res-auto xmlns:tools/ android:layout_width/match_parent android:layout_height/match_parent tools:context/me.a3zcs.mvp.architecturecurecomponent.MainActivity TextView android:layout_width'wrap_content android:layout_height'wrap_content app:layout_constraintbottom_tobottomof'parent app:layout_constraintleft_toleftof'parent app:layout_constraintright_torightof'parent app:layout_constrainttop_totopof'parent android:id'id/number'gt;'lt;lt;//TextView'gt; button android:id'id/button android:layout_width'wrap_content android:layout_height'wrap_content android:text'button app:layout_constraintleft_toleftof'parent app:layout_constraintright_torightof'parent'android:layout_margintop'0dp :layout_constrainttop_tobottomof/number/number/button/lt;/android.support.constraint.ConstraintLayout'gt; expands appCompatActivity (TextView number); AddOne button; int i No 0; @Override protected void onCreate (Bundle savedInstanceState) - super.onCreate (savedInstanceState); ContentView (R.layout.activity_main); number - findViewById (R.id.number); addOne - findViewById (R.id.button); setNumber (i); addOne.setOnClickListener (new View.OnClickListener)- @Override public void onClick (View) - setNumber (i); The code is very simple, and works correctly, but when you change the screen mode to the landscape instead of the portrait the value of textView will return to 0, because the activity got its destruction and then create again, previously such problems that are related to configChanges solved by using onSaveInstStateance and restored onInstanceState, very nicely, we will talk about one of the new solutions to the problem, which Is ViewModel. ViewModel: This is a class designed to work with view data, and the added benefit is that it stores data in a state of configChanges, very nice to start refactoring for the previous code creation model, which extends to the viewModel public class Expands ViewModel private int count; Public emptinesscount this.count count; Public int getCount Countdown; Very nice now we will use this class in MainActivity as an object of composition, and then create it in onCreate to become a form of activity as follows: the public class MainActivity expands AppCompatActivity (Textnumber number; addOne button; CounterViewModel counterViewModel; @Override protected void onCreate (Bundle savedInstanceState) - super.onCreate (savedInstanceState); setContentView (R.layout.activity_main); ViewModel - ViewModelProviders.of number - findViewById (R.id.number); addOne - findViewById (R.id.button); setNumber (counterViewModel.getCount)); addOne.setOnClickListener (new View.OnClickListener.) - @Override public void on Click (View View) - counterViewModel.setCount (counterViewModel.getCount) ;); setNumber (counterViewModel.getCount(); Please note that we use The ViewModelProviders, and it will store and store ViewModel for this activity, i.e., if the status of configChanges occurs - as a change of orientation - for the activity class ViewModelProviders will store ViewModel data and therefore we give it this information during viewModelProviders.of (get) now if we do rotations impact data that has been lt;UI controller LiveData: LiveData allows lifeCycleOwner, whether it's activity or snippet, to receive data from ViewModel by monitoring -observation - any modification to view Model Class, Very nice if instead of updating the user interface every time using setters and getters, we'll look at it in the following example first to make a refactor for your viewModel built, to become the next public class CounterViewModel expands to the publicViewModel ()setValue We have changed the type of data we are dealing with. The button is zlt;CounterViewModel counterViewModel; The government @Override @Nullable Integer'lt;Integer'integer (integer) @Override; R.layout.activity_main: R.layout.activity_main view model - ViewModelProviders.of (it).get (CounterViewModel.class); counterViewModel.getCount.observe number - findViewById (R.id.number); addOne - findViewById (R.id.button); setNumber (counterModel.getView); @Override is also a key element in the development of the new approach. I made a comment before each line was deleted, to make it easier for you to see the edit, starting we made the Observer has only one feature that is on the change, called, if there is any data change, then we have observed on this line qlt;Object'gt; in onCreate counterViewModel.getCount.observe), as our notes observer will call qlt;LifeCycleerOwn'gt; Provided that the state for lifeCycleOwner in STARTED or RESUMEED, LifecycleOwner is in a STATE OF START on two occasions after calling the onStart function, secondly, before calling the onPause function, and LifecycleOwner is in a resumeed state after

calling the onResume function. So when we observed, we gave the function of an observer, in addition to the life cycle owner who is associated with it. A very important point to note is if the application is running, but in the background, and the system is exposed to what we call low memory, the system will reuse resources by eliminating - kill - from the apps in the background, and if you rely only on ViewModel you will just lose your data, as opposed to onSavestance State, which solved the problem. Here I hope that I have managed to simplify this concept, and you will no doubt have to read further to deal with more complex data than mentioned in the example. Peace is upon you and God's mercy and blessings. Sources of Android Architecture Components from ExampleAndroid Architecture ViewModels Components: A Simple Example of Android ViewModel Architecture Component(Harmful page 2 تحب تنضم لعالم البرمجة؟ وتنشئ عالمك الخاص، تنشر المقالات، الدورات، تشارك المبرمجين وتساعد الآخرين، اشترك الآن بخطوات يسيرة ! A few years ago, Google I/O announced that Android was taking a new approach to software architecture, starting with the idea that android app development should be more of an opinion. Revealed at the conference was a pair of libraries developed by Google for Android engineers to be included in their apps. I immediately started reading about Jetpack and creating apps that included new libraries, ViewModel, and LiveData. In this blog we will see how the pieces fit together and how we can build our first app using both of these classes. Ready? Start! To start using ViewModel and LiveData, you first need to add the necessary dependencies to our build.gradle file. Open the Gradle file and insert it into the next one. apply plugin: 'com.android.application' apply plugin: 'kotlin-android' apply plugin: 'kotlin-android-extensions' // Daniel says add this line: apply plugin: 'kotlin-kapt' android { compileSdkVersion 29 defaultConfig { applicationId com.danielmalone.dating minSdkVersion 21 targetSdkVersion 29 versionCode 1 versionName 1.0 testInstrumentationRunner androidx.test.runner.AndroidJUnitRunner } buildTypes { release { minifyEnabled false proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro' } } compileOptions { sourceCompatibility JavaVersion.VERSION_1_8 targetCompatibility JavaVersion.VERSION_1_8 } kotlinOptions { jvmTarget = JavaVersion.VERSION_1_8.toString() } } dependencies { implementation fileTree(dir: 'libs', include: ['*.jar']) implementation org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version implementation 'androidx.appcompat:appcompat:1.1.0' implementation 'androidx.core:core-ktx:1.2.0' implementation 'androidx.constraintlayout:constraintlayout:1.1.3' implementation 'com.google.android.material:material:1.1.0' testImplementation 'junit:junit:4.12' androidTestImplementation 'androidx.test.ext:junit:1.1.1' androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0' // Daniel says add these four line: implementation androidx.lifecycle:lifecycle-viewmodel-ktx:2.2.0 implementation androidx.lifecycle:lifecycle-livedata-ktx:2.2.0 kapt androidx.lifecycle:lifecycle-compiler:2.2.0 implementation androidx.activity:activity-ktx:1.1.0 } Create a new class that extends ViewModel : package com.danielmalone.dating import androidx.lifecycle.ViewModel class MainActivityViewModel : ViewModel () - Now this class does nothing yet. Let's add some LiveData variables that will later be noted in our MainActivity. With MVVM it's best practice to have one ViewModel per activity or snippet, just like you would one for a class in MVP. package com.danielmalone.dating import androidx.lifecycle.MutableLiveData import androidx.lifecycle.ViewModel class MainActivityViewModel : ViewModel () - Val firstName - MutableLiveData) val lastName - MutableLiveData () val age - Here's our activity: package com.danielmalone.dating import android.os.Bundle import androidx.activity.activityModels import androidx.appcompat.app.appCompatActivity import androidx.lifecycle.Observer import kotlinx.android.synthetic.main.activity_main. class MainActivity: AppCompatActivity () - super.onCreate (savedInstanceState) setContentView (R.layout.activity_main) setSupportActionBar (toolbar) viewModel.firstName.observe (it, Observer) Please note how we set up the viewModel variable at the top, using viewModels, part of the KTX library provided by Google. See the build.gradle file above for the import line (it's at the bottom of the file). Once the setup is complete, we can freely use viewModel in onCreate. To observe the data, simply enter viewModel.firstName.observe by going through this (or activity if you use a snippet) and then the Observer lambda. Every time firstName changes, we can process user interface changes between characters. Let's give it a try! viewModel.firstName.observe (this, Observer - firstNameTextView.text) If your XML layout has a TextView with firstNameTextView ID, the first name will be TextView text, appearing on the screen for users to see. But we can see more than just one copy of LiveData. We can observe and update several LiveData objects, no matter how many times we want to. Here's an example of ViewModel: package com.danielmalone.dating import androidx.lifecycle.MutableLiveData import androidx.lifecycle.ViewModel class MainActivityViewModel : ViewModel<lt;gt;tth;t.t.-gt;-lt.gt;) While we'll fore-fake it. val apiFirstName - Daniel val apiLastName - Malone val apiAge - 28 firstName.value - apiFirstName lastName.value - apiLastName age.value - apiAge - And example Activities, which monitors these values: package com.danielmalone.dating import android.os.Bundle import androidx.activity.viewModels import androidx.appcompat.app.app.AppCompatActivity import androidx.lifecycle.Observer import kotlinx.android.synthetic.main.activity_main. import kotlinx.android.synthetic.main.content_main.'lt;lt;int'gt; qgt; String: AppCompatActivity () - Private Val-ViewModel: MainActivityViewModel by viewModels () redefine pleasure onCreate (saved: Bundle?) - super.onCreate (savedInstanceState) setContentView (R.layout.activity_main) setSupportActionBar (toolbar) viewModel.first Observer - firstNameTextView.text) viewModel.lastName.observe (this, Observer - lastNameTextView.text) Stay tuned for Part 2 as we dive more into ViewModel and LiveData! LiveData! live data example in android. live data simple example in android. mutable live data example in android