**Key Points in Android**

* Always use notifyDataSetChanged in fun setData function inside recyclver adapter.
* While iterating if I want to add new things in list and add that list in some data class then instead of clearing it in every iteration always intiate it in every iteration like create new list in every iteration

**Budget Tracker**

In the main layout we have used collapsing layout

<?xml version="1.0" encoding="utf-8"?>

<!--first of all you have to change

the layout as CoordinatorLayout.

This is the first thing we need to do.-->

<androidx.coordinatorlayout.widget.CoordinatorLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<!--AppBarLayout helps the toolbar and other components to react on

scroll changes. I am using Dark theme for AppBarLayout.

Inside this AppBarLayout i

have used CollapsingToolbarLayout. -->

<com.google.android.material.appbar.AppBarLayout

android:id="@+id/appBarLayout"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

<!-- In CollapsingToolbarLayout sompe important attributes are:

i) app:layout\_scrollFlags which is used to specify how collapsing

layout behaves when content is scrolled.I have used

app:layout\_scrollFlags="scroll|snap|exitUntilCollapsed"

so it will scroll until it's is completely collapsed.

ii) app:contentScrim="@color/green" that specifies the color

of the collapsed toolbar -->

<com.google.android.material.appbar.CollapsingToolbarLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:contentScrim="@color/green"

app:layout\_scrollFlags="scroll|snap|exitUntilCollapsed"

app:title="@string/app\_name">

<!--We are adding the image that we have added earlier

in the drawable folder.The attribute app:layout\_collapseMode="parallax"

causes the image to move when user scrolls at a specific ratio. -->

<ImageView

android:layout\_width="match\_parent"

android:layout\_height="250dp"

android:scaleType="centerCrop"

android:src="@drawable/image"

app:layout\_collapseMode="parallax" />

<!-- The attribute app:layout\_collapseMode="pin" is set so that

sticks to the top when the user scrolls the view up-->

<androidx.appcompat.widget.Toolbar

android:layout\_width="match\_parent"

android:layout\_height="?attr/actionBarSize"

app:layout\_collapseMode="pin"

app:popupTheme="@style/ThemeOverlay.AppCompat.Dark" />

</com.google.android.material.appbar.CollapsingToolbarLayout>

</com.google.android.material.appbar.AppBarLayout>

<!--Now add the NestedScollView-->

<androidx.core.widget.NestedScrollView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">

<!--In textview we will be adding the text that i have

added earlier in strings.xml file.This is simply the

the content to be scrolled -->

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:lineSpacingExtra="8sp"

android:padding="16dp"

android:text="@string/content"

android:textSize="20sp" />

</androidx.core.widget.NestedScrollView>

</androidx.coordinatorlayout.widget.CoordinatorLayout>

**Bottom Navigation Bar**

<com.google.android.material.bottomnavigation.BottomNavigationView  
 android:id="@+id/bottom\_navbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="60dp"  
 android:layout\_gravity="bottom"  
 app:menu="@menu/bottom\_nav\_menu"  
 android:scrollIndicators="left">  
  
</com.google.android.material.bottomnavigation.BottomNavigationView>

You have to create a menu first

<?xml version="1.0" encoding="utf-8"?>  
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
 <item  
 android:id="@+id/budget"  
 android:icon="@drawable/ic\_wallet\_budget\_svg"  
 android:title="budget" />  
 <item  
 android:id="@+id/Category"  
 android:icon="@drawable/ic\_category\_svg"  
 android:title="Category" />  
 <item  
 android:id="@+id/add"  
 android:icon="@drawable/ic\_baseline\_add\_24"  
 android:title="add" />  
 <item  
 android:id="@+id/Stats"  
 android:icon="@drawable/ic\_stats\_svg"  
 android:title="Stats" />  
 <item  
 android:id="@+id/more"  
 android:icon="@drawable/ic\_baseline\_more\_horiz\_24"  
 android:title="more" />  
</menu>

Kotlin code of bottom navigation bar we are using when statement it is same like switch statement in java

//------------------------Bottom-Navigation-Bar------------------------------------  
bottomNav.setOnItemSelectedListener **{** when(**it**.*itemId*){  
 R.id.*budget* -> {  
 val budgetIntent=Intent(this,BudgetActivity::class.*java*)  
 startActivity(budgetIntent)  
 true  
 }  
 R.id.*add* -> {  
 val AddIntent=Intent(this,AddNew::class.*java*)  
 startActivity(AddIntent)  
 true  
 }  
  
 else -> {false}  
 }  
**}**

**Date Picker in kotlin**

First of all datepicker listener so that when datepicker dialog open it and user select some date it can save those

val dateSetListener = object : DatePickerDialog.OnDateSetListener {  
 override fun onDateSet(view: DatePicker, year: Int, monthOfYear: Int,  
 dayOfMonth: Int) {  
 cal.set(Calendar.*YEAR*, year)  
 cal.set(Calendar.*MONTH*, monthOfYear)  
 cal.set(Calendar.*DAY\_OF\_MONTH*, dayOfMonth)  
 updateDateInView()  
 }  
}

then open datepicker on button click

todayDateTv.setOnClickListener**{** DatePickerDialog(this@AddNew,  
 dateSetListener,  
 // set DatePickerDialog to point to today's date when it loads up  
 cal.get(Calendar.*YEAR*),  
 cal.get(Calendar.*MONTH*),  
 cal.get(Calendar.*DAY\_OF\_MONTH*)).show()  
**}**

then this code we run to update the textview where we are showing to the user

private fun updateDateInView() {  
 val myFormat = SimpleDateFormat("MMMM")// mention the format you need  
 val year = cal.get(Calendar.*YEAR*)  
 val month = cal.get(Calendar.*MONTH*)  
 val day = cal.get(Calendar.*DAY\_OF\_MONTH*)  
 cal.set(Calendar.*MONTH*, month)  
 val month\_name = myFormat.format(cal.getTime())  
 todayDateTv!!.*text* = "$day $month\_name $year"  
}

**Time picker in android**

1. TimePickerDialog :  Creates a new time picker dialog for the given date.
2. TimeFormat : TimePickerDialog has a boolena is24HourView , whether to show 24 hour view or AM/PM .true to show 24Hr View , false to show AM/PM View
3. TimePickerDialog.OnTimeSetListener is used to listen time select event .when user selects date onTimeSet callback is called .

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | val mTimePicker: TimePickerDialog  val mcurrentTime = Calendar.getInstance()  val hour = mcurrentTime.get(Calendar.HOUR\_OF\_DAY)  val minute = mcurrentTime.get(Calendar.MINUTE)    mTimePicker = TimePickerDialog(this, object : TimePickerDialog.OnTimeSetListener {   override fun onTimeSet(view: TimePicker?, hourOfDay: Int, minute: Int) {    selectedTime.setText(String.format("%d : %d", hourOfDay, minute))   }  }, hour, minute, false) |

TimePicker Dialog has two mode clock and spinner. use android:timePickerMode property to set the mode

file: style.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <resources>      <!-- Base application theme. -->      <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">          <!-- Customize your theme here. -->          <item name="colorPrimary">@color/colorPrimary</item>          <item name="colorPrimaryDark">@color/colorPrimaryDark</item>          <item name="colorAccent">@color/colorAccent</item>          <item name="android:timePickerStyle">@style/MyTimePickerTheme</item>      </style>        <style name="MyTimePickerTheme" parent="android:Widget.Material.Light.TimePicker">          <item name="android:timePickerMode">clock</item>      </style>    </resources> |

Full code

package com.tutorialsbuzz.androidtimepicker

import android.app.TimePickerDialog

import android.os.Bundle

import android.widget.TimePicker

import androidx.appcompat.app.AppCompatActivity

import kotlinx.android.synthetic.main.activity\_main.\*

import java.util.\*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity\_main)

        val mTimePicker: TimePickerDialog

        val mcurrentTime = Calendar.getInstance()

        val hour = mcurrentTime.get(Calendar.HOUR\_OF\_DAY)

        val minute = mcurrentTime.get(Calendar.MINUTE)

        mTimePicker = TimePickerDialog(this, object : TimePickerDialog.OnTimeSetListener {

            override fun onTimeSet(view: TimePicker?, hourOfDay: Int, minute: Int) {

                selectedTime.setText(String.format("%d : %d", hourOfDay, minute))

            }

        }, hour, minute, false)

        selectTime.setOnClickListener({ v ->

            mTimePicker.show()

        })

    }

}

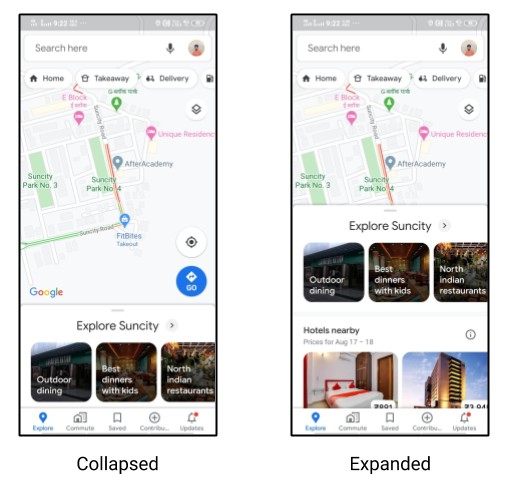
**BottomSheet Bar in android**

**Types of BottomSheet**

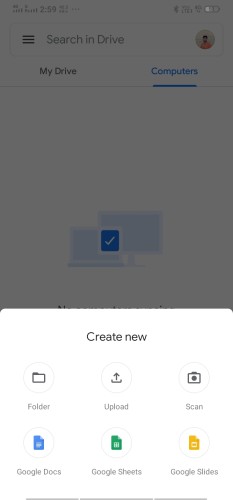
A BottomSheet is of two types:

* **Persistent BottomSheet:**The Persistent BottomSheet is visible along with other UI components on the screen. Initially, some content of this Persistent BottomSheet is visible(you can set the peek height i.e. the initial height of BottomSheet to 0 also) and when you slide it up, then rest of the content will appear along with the BottomSheet. The following is an example of Persistent BottomSheet:

***NOTE:****A Persistent BottomSheet is applied to a child of CoordinatorLayout.*



* **Modal BottomSheet:**This looks similar to some alert-dialog or some dialog fragment but originating from the bottom of the screen. Unlike Persistent BottomSheet, it is totally invisible initially, and on certain action(maybe a button click), it appears from the bottom of the screen. It generally contains some list of items and the items of the list correspond to some action. The following is an example of Modal BottomSheet:



**Example of Persistent BottomSheet**

Let's first make the UI of the Persistent BottomSheet. So, here is the code for layout\_persistent\_bottom\_sheet.xml :

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:id="@+id/bottomSheet"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:behavior\_hideable="false"

app:behavior\_peekHeight="@dimen/dimen\_32"

app:layout\_behavior="com.google.android.material.bottomsheet.BottomSheetBehavior">

<androidx.appcompat.widget.AppCompatTextView

android:id="@+id/tvTitle"

android:layout\_width="@dimen/dimen\_0"

android:layout\_height="wrap\_content"

android:layout\_marginStart="@dimen/dimen\_16"

android:layout\_marginEnd="@dimen/dimen\_16"

android:text="@string/tv\_title"

android:textSize="@dimen/font\_size\_24"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<androidx.appcompat.widget.AppCompatTextView

android:id="@+id/tvSubtitle"

android:layout\_width="@dimen/dimen\_0"

android:layout\_height="wrap\_content"

android:layout\_margin="@dimen/dimen\_16"

android:text="@string/tv\_description"

android:textSize="@dimen/font\_size\_16"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toBottomOf="@+id/tvTitle" />

</androidx.constraintlayout.widget.ConstraintLayout>

Here, we are having two text views, one for the title and one for the description. Few things that you can notice from the above code is:

The behavior of the layout is set to com.google.android.material.bottomsheet.BottomSheetBehavior

Some flags are set such as:

* **behavior\_hideable:**This is used to set if the bottom sheet can be hidden totally when we drag it down or not.
* **behavior\_peekHeight:**This is height up to which the bottom sheet will be visible in the collapsed state.
* **behavior\_skipCollapsed:**When it is set to true and the bottom sheet is hidable, then it will not have a collapsed state.

After that, we need to add this layout to our activity\_main.xml file. Also, as discussed earlier, we will have two buttons on the main page, one for Persistent BottomSheet and other for Modal BottomSheet. So, here is the code for activity\_main.xml file:

<?xml version="1.0" encoding="utf-8"?>

<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MainActivity">

<androidx.appcompat.widget.AppCompatButton

android:id="@+id/btnBottomSheetPersistent"

android:layout\_width="match\_parent"

android:layout\_height="@dimen/dimen\_60"

android:text="@string/btn\_persistent" />

<androidx.appcompat.widget.AppCompatButton

android:id="@+id/btnBottomSheetModal"

android:layout\_width="match\_parent"

android:layout\_height="@dimen/dimen\_60"

android:layout\_marginTop="@dimen/dimen\_60"

android:text="@string/btn\_modal" />

<include layout="@layout/layout\_persistent\_bottom\_sheet" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>

If you see the above code, then you will find that the BottomSheet is the direct child of the CoordinatorLayout.

Now, if you run the app, then you will see the BottomSheet present on the screen. In order to observe various states of the BottomSheet, you can override the onStateChanged() function in your MainActivity.kt file.

The following is the code of MainActivity:

**class** **MainActivity** : **AppCompatActivity**() {

**private** lateinit var bottomSheetBehavior: BottomSheetBehavior<ConstraintLayout>

override fun **onCreate**(savedInstanceState: Bundle?) {

**super**.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

bottomSheetBehavior = BottomSheetBehavior.from(bottomSheet)

bottomSheetBehavior.addBottomSheetCallback(object :

BottomSheetBehavior.BottomSheetCallback() {

override fun **onSlide**(bottomSheet: View, slideOffset: Float) {

*// handle onSlide*

}

override fun **onStateChanged**(bottomSheet: View, newState: Int) {

when (newState) {

BottomSheetBehavior.STATE\_COLLAPSED -> Toast.makeText(**this@MainActivity**, "STATE\_COLLAPSED", Toast.LENGTH\_SHORT).show()

BottomSheetBehavior.STATE\_EXPANDED -> Toast.makeText(**this@MainActivity**, "STATE\_EXPANDED", Toast.LENGTH\_SHORT).show()

BottomSheetBehavior.STATE\_DRAGGING -> Toast.makeText(**this@MainActivity**, "STATE\_DRAGGING", Toast.LENGTH\_SHORT).show()

BottomSheetBehavior.STATE\_SETTLING -> Toast.makeText(**this@MainActivity**, "STATE\_SETTLING", Toast.LENGTH\_SHORT).show()

BottomSheetBehavior.STATE\_HIDDEN -> Toast.makeText(**this@MainActivity**, "STATE\_HIDDEN", Toast.LENGTH\_SHORT).show()

**else** -> Toast.makeText(**this@MainActivity**, "OTHER\_STATE", Toast.LENGTH\_SHORT).show()

}

}

})

btnBottomSheetPersistent.setOnClickListener {

**if** (bottomSheetBehavior.state == BottomSheetBehavior.STATE\_EXPANDED)

bottomSheetBehavior.state = BottomSheetBehavior.STATE\_COLLAPSED

**else**

bottomSheetBehavior.state = BottomSheetBehavior.STATE\_EXPANDED

}

}

}

Here, we have created a variable of BottomSheetBehavior and on click of the button we are changing its state i.e. when it is expanded then we are collapsing it and vice-versa.

Now, run the app and try to observe various states of the BottomSheet.

**Example of Modal BottomSheet**

The implementation of Modal BottomSheet is similar to that of a fragment or simply a DialogFragment. But here, we use BottomSheetDialogFragment.

So, let's first make the UI. In this example, we will have three buttons. So, the code for layout\_modal\_bottom\_sheet.xml will be:

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<androidx.appcompat.widget.AppCompatButton

android:id="@+id/firstButton"

android:layout\_width="@dimen/dimen\_0"

android:layout\_height="@dimen/dimen\_60"

android:text="@string/btn\_one"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent" />

<androidx.appcompat.widget.AppCompatButton

android:id="@+id/secondButton"

android:layout\_width="@dimen/dimen\_0"

android:layout\_height="@dimen/dimen\_60"

android:text="@string/btn\_two"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toBottomOf="@+id/firstButton" />

<androidx.appcompat.widget.AppCompatButton

android:id="@+id/thirdButton"

android:layout\_width="@dimen/dimen\_0"

android:layout\_height="@dimen/dimen\_60"

android:text="@string/btn\_three"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toBottomOf="@+id/secondButton" />

</androidx.constraintlayout.widget.ConstraintLayout>

Now, we have already made one class named CustomBottomSheetDialogFragment . In this class, we will inflate the layout and handle the button clicks of the Modal BottomSheet. The following is the code for the same:

**class** **CustomBottomSheetDialogFragment** : **BottomSheetDialogFragment**() {

companion object {

**const** val TAG = "CustomBottomSheetDialogFragment"

}

override fun **onCreateView**(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

**return** inflater.inflate(R.layout.layout\_modal\_bottom\_sheet, container, **false**)

}

override fun **onActivityCreated**(savedInstanceState: Bundle?) {

**super**.onActivityCreated(savedInstanceState)

firstButton.setOnClickListener {

*//handle click event*

Toast.makeText(context, "First Button Clicked", Toast.LENGTH\_SHORT).show()

}

secondButton.setOnClickListener {

*//handle click event*

Toast.makeText(context, "Second Button Clicked", Toast.LENGTH\_SHORT).show()

}

thirdButton.setOnClickListener {

*//handle click event*

Toast.makeText(context, "Third Button Clicked", Toast.LENGTH\_SHORT).show()

}

}

}

And finally, we need to call this class from the MainActivtiy on the click of the button. So, the code for this will be:

btnBottomSheetModal.setOnClickListener {

CustomBottomSheetDialogFragment().apply {

show(supportFragmentManager, CustomBottomSheetDialogFragment.TAG)

}

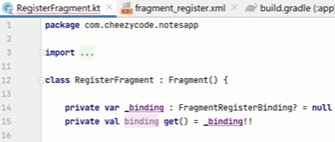
}

**View Binding in Android**

*View binding* is a feature that allows you to more easily write code that interacts with views. Once view binding is enabled in a module, it generates a *binding class* for each XML layout file present in that module. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.

In most cases, view binding replaces findViewById.

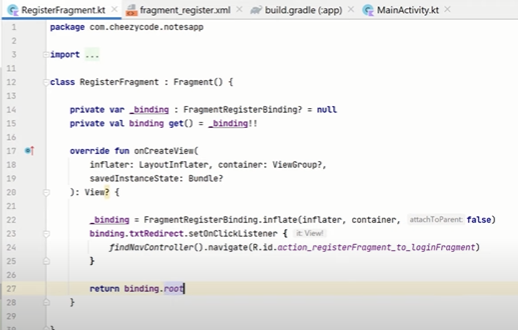
View binding jab hum use karte hai to hume har vari findViewById nahi likhna parrta isse hmara time bhi bachta hai or runtime errors bhi nahi aate



Pehle hume \_binding naam ka variable banana parrta hai or usko initialize krna hota hai us file ki corresponding xml file ke naam se like in this case it is FragmentRegisterBinding or isko null se initialize krna hota hai

Next line hum likh dete hai taki hume har vari null check na krna parre

FULL CODE :-



**TabLayout with ViewPager 2 on bottomsheetFragment**

This code is from my fragment\_layout\_modal\_bottomsheet activity from where I need to open my bottomsheet

selectCategoryLayout.setOnClickListener()**{** CustomBottomSheetDialogFragment().*apply* **{** show(*supportFragmentManager*, CustomBottomSheetDialogFragment.TAG)  
 **}  
}**

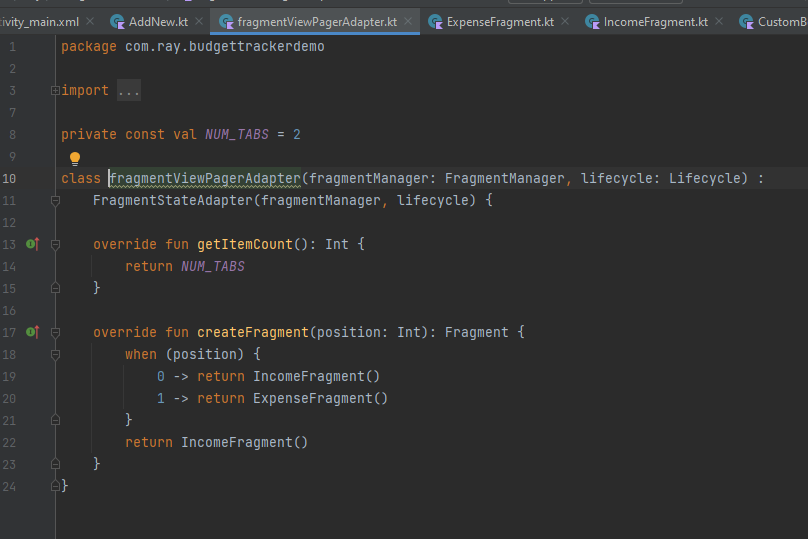
CustomBottomSheetDialog is the fragment which I made

class CustomBottomSheetDialogFragment : BottomSheetDialogFragment() {  
 companion object {  
  
 const val TAG = "CustomBottomSheetDialogFragment"  
  
 }  
 lateinit var tabLayout: TabLayout  
 lateinit var viewPager: ViewPager2  
 var arrayToPass= *arrayOf*("EXPENSE","INCOME")  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 // Inflate the layout for this fragment  
 return inflater.inflate(R.layout.*fragment\_layout\_modal\_bottom\_sheet*, container, false)  
  
  
 }  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 //handle click event  
 tabLayout= view.findViewById<TabLayout>(R.id.*tabLayoutForBottomSheet*)  
 viewPager= view.findViewById(R.id.*viewPagerForBottomSheet*)  
  
 val adapter=fragmentViewPagerAdapter(*parentFragmentManager*,*lifecycle*)  
 viewPager.*adapter*=adapter  
  
 TabLayoutMediator(tabLayout, viewPager) **{** tab, position **->** tab.*text* = arrayToPass[position]  
 **}**.attach()  
  
 Toast.makeText(*context*, "Perform add operation", Toast.*LENGTH\_SHORT*).show()  
  
 }  
}

in the xml we need

<com.google.android.material.tabs.TabLayout  
 android:id="@+id/tabLayoutForBottomSheet"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:background="#ffffff"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:tabIconTint="#ff6600"  
 app:tabIndicatorColor="#ff6600"  
 app:tabSelectedTextColor="#ff6600"  
 app:tabTextColor="#ff6600" />  
  
<ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="8dp"  
 android:layout\_marginEnd="16dp"  
 android:src="@drawable/ic\_baseline\_add\_24"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
<androidx.viewpager2.widget.ViewPager2  
 android:id="@+id/viewPagerForBottomSheet"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/tabLayoutForBottomSheet" />

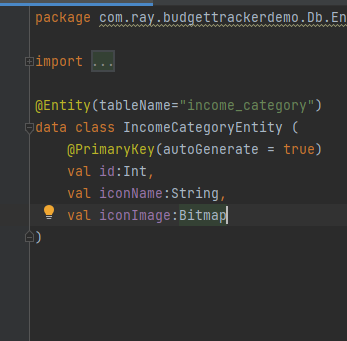
Make a fragment adapter extend it with fragmentStateAdapter



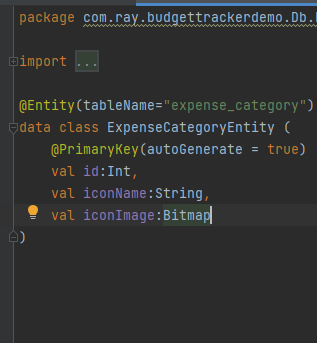
**Now I want to Add Categories**

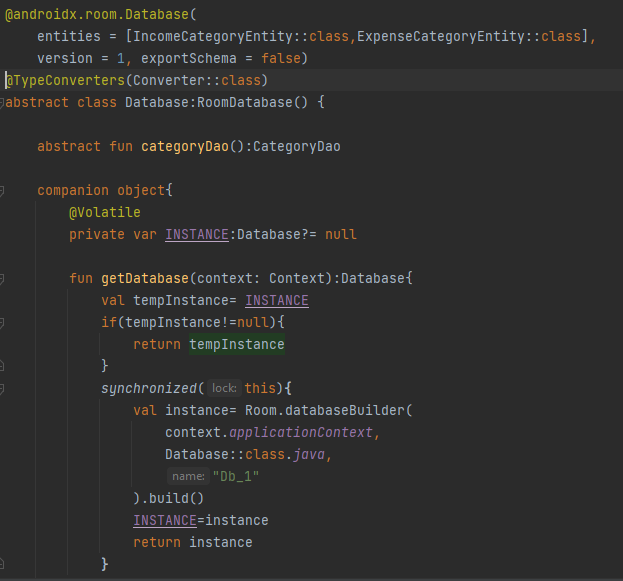
I made room database to add income and exoense categories

Then income category

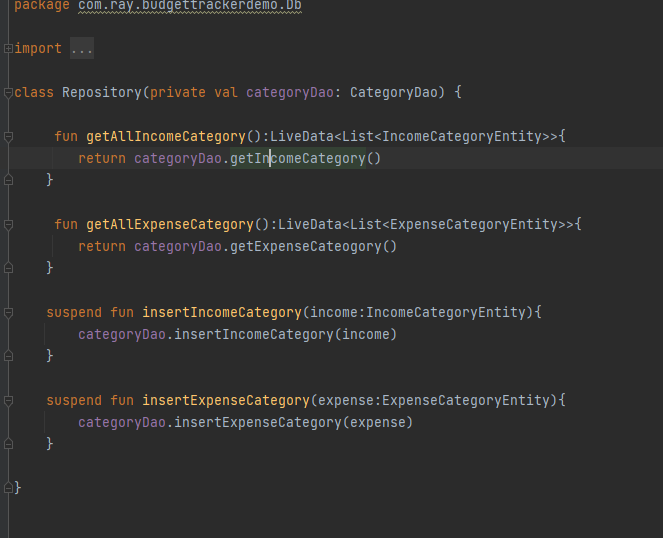


Then expense entity

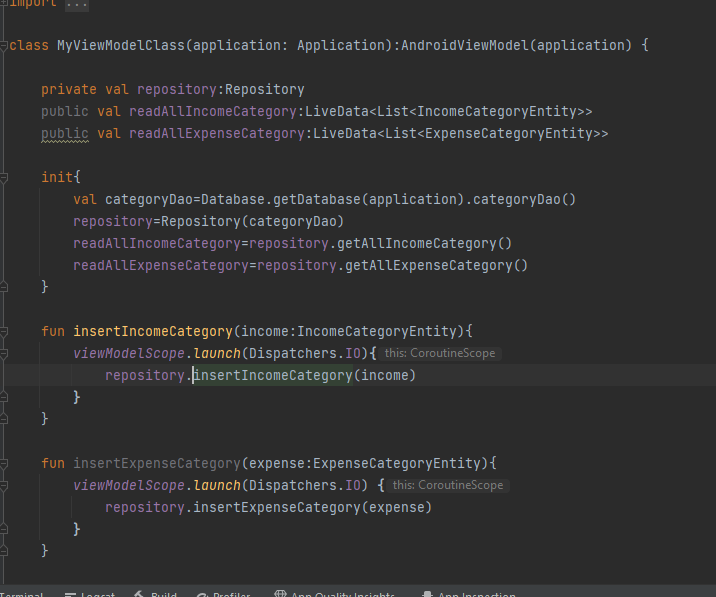




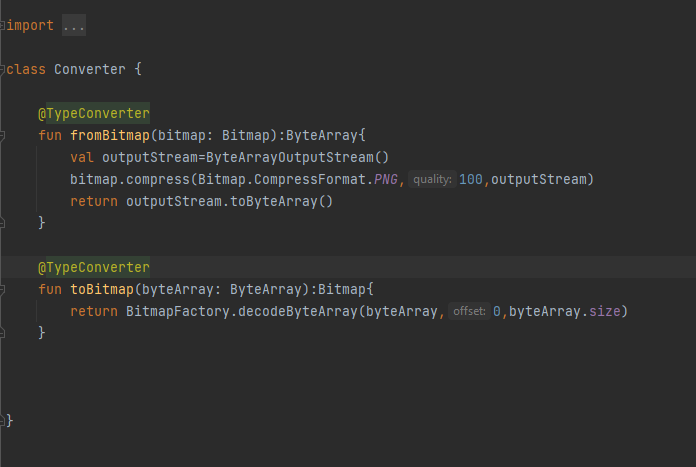
Repository



Then viewmodel



Converter class



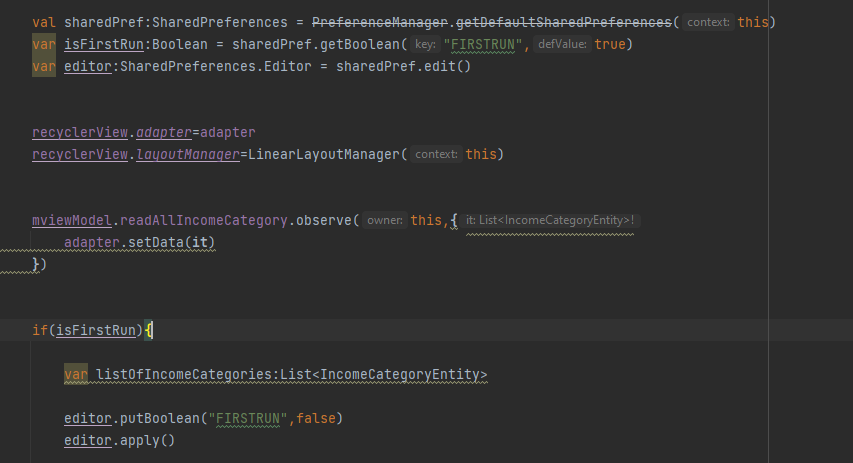
**How to convert Drawable to Bitmap**

private fun getBitmapFromVectorDrawable(context: Context, drawableId: Int): Bitmap {  
 val drawable = ContextCompat.getDrawable(context, drawableId)  
 val bitmap = Bitmap.createBitmap(  
 drawable!!.*intrinsicWidth*,  
 drawable.*intrinsicHeight*, Bitmap.Config.*ARGB\_8888* )  
 val canvas = Canvas(bitmap)  
 drawable.setBounds(0, 0, canvas.*width*, canvas.*height*)  
 drawable.draw(canvas)  
 return bitmap

}

The parameter drawable int is drawable id that is like :- R.drawable.face

**Now I wanted to show this in recyclerview but for just once to do that I used shared preferences**





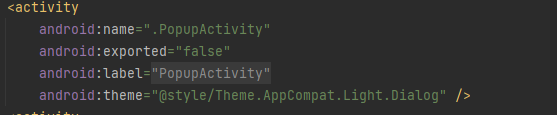
**NOTE:- always make height and width of custom layour that you are going to use in recyclerview fixed like don’t make it wrap content always give some number otherwise it will make your recyclerviee laggy.**

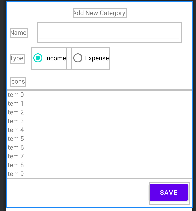
**Popup Window Open on clicking floating button on category Activity**

Popupwindow is a floating view that is displayed on top of an activity. Android provides a PopupWindow class for creating a popup window with the custom design.

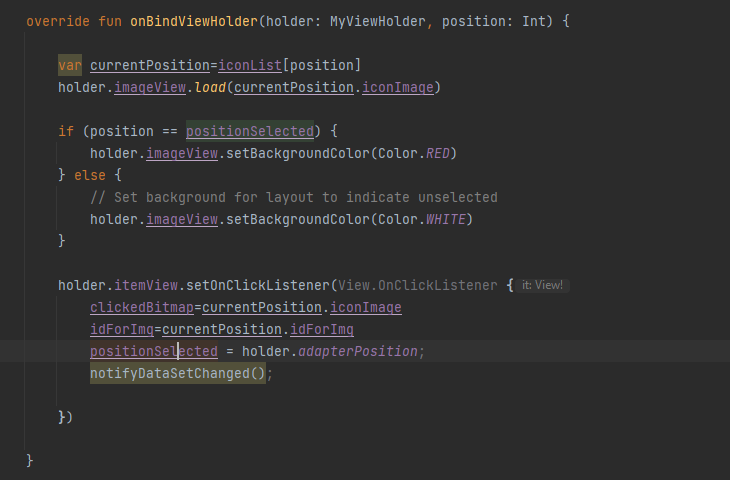
**How to make a popup window in android**

I created a new activity and in manifest



\

Now this activity has this type of layout now I want whenever I click on one item in recyclverview its bacjground color get changed and rest icons bg will remain white



First initialize positionSelected variable to -1

**BottomSheetDialogFragment**

Now I have two fragments in bottom Sheet

1. income fragment
2. expense fragment

I want to load income category in income fragment and expense categories in expense fragment

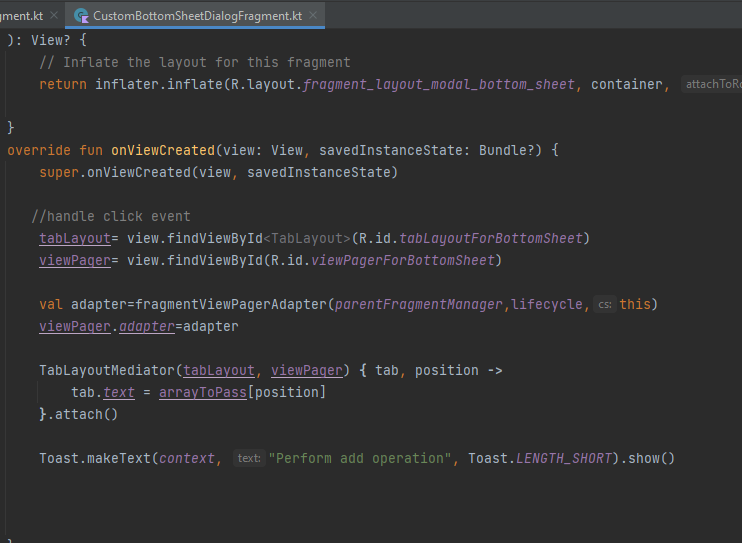
To do that

1. first of all like we usually do I added recyclerview to the fragment and make one layout file for the adapter

now in this the thing was this that I wanted to dismiss the dialog whenever the user click on some item in the list

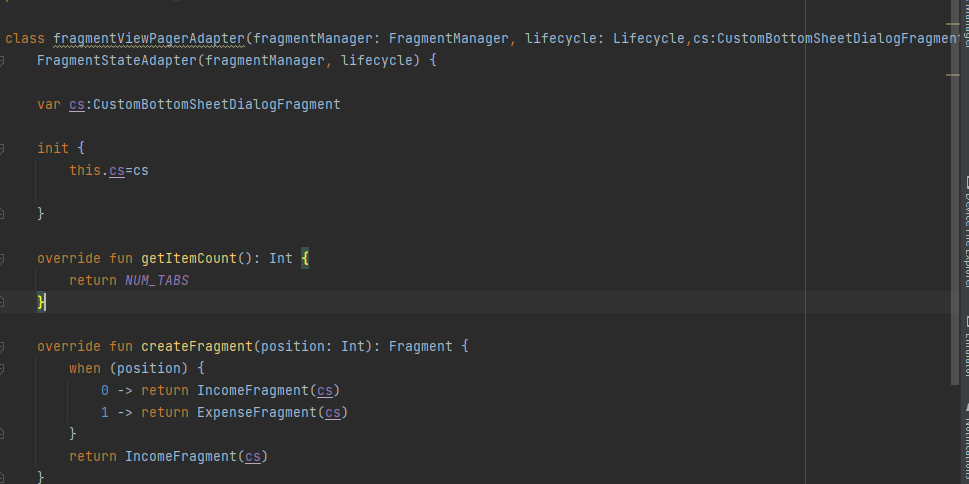
so to do that I need the object of the custombottomsheetdialog that opens and only then I can use the inbuilt function customsheet.dismiss().

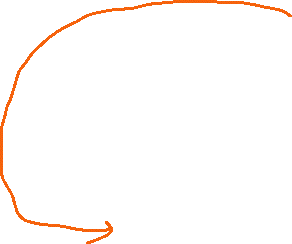
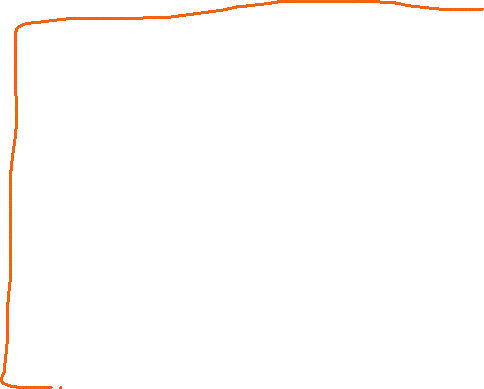
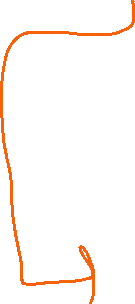
So how I passed it is:





I passed the object this in the adapter of viewpager and in viewpager adapter





Passed In constructor

Initialized it

Passed to te fragment further

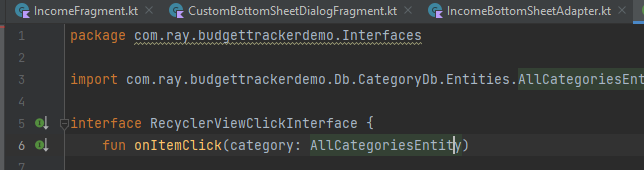
So now we have this object in our fragment class from fragment we can pass this to adapter class and from there on itemClicklisner we can use cs.dismiss().

Similarly I used this for both the fragment and from there now I can able to dismiss the dialog on button click

**Pass Data from adapter to the activity**

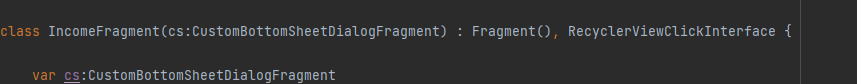
To pass the data from adapter to the activity I use interface method

1. make one interface



Make one func onitemclick which Is taking category as argument

1. make your activity extends this interface in which you want to receive the data



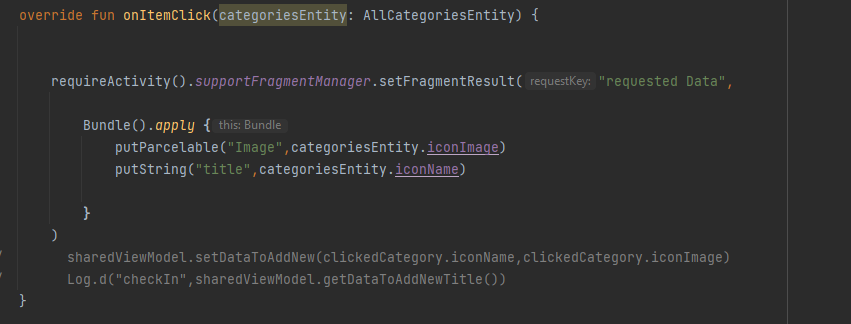
As you can see I extended the fragment with the interface

Now pass it to the adapter



By using this

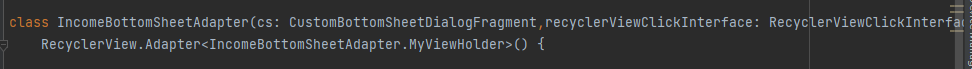
Now we have to override its method in this fragment which is onitemclick

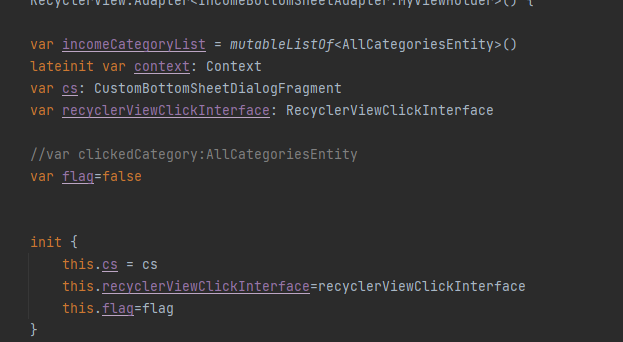


1. in adapter class

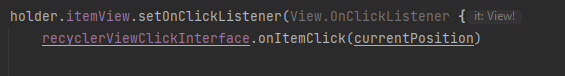
use it like this

first get it as parameter in constructor then





Initialize it



Then use it.

Similary I done this for expense fragment too

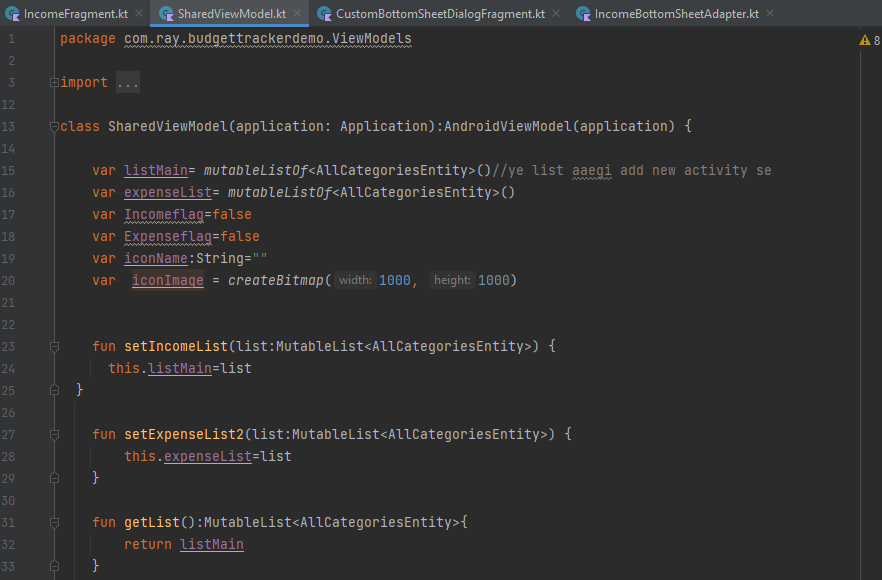
**How I managed to get the same list even after closing the dialog**

**So the requirement was when I click on some item the bottomsheetdialogfragment should get close and when the user again open the fragment the last selected item should be on top**

Steps:

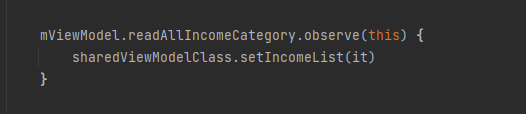
First I knew that I cant make changes In the database like one idea was that to remove the selected item and then add it on the top in the backend by which when the user open the dialog again the fragment reloads and send the list to the adapter and adapter loads the list which contains the clicked item on top but it is not a good approach to change the db

So we used **SharedViewModel**



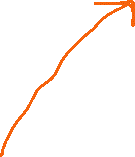
So in this first you make a class and extends it with AndroiViewModel then as you can see I made some functions so lets understand

First I make setIncomeList function to get the whole list from db like in addnew activity I make an object of this viewmodel and using this function I pass the list



Now Inn fragment

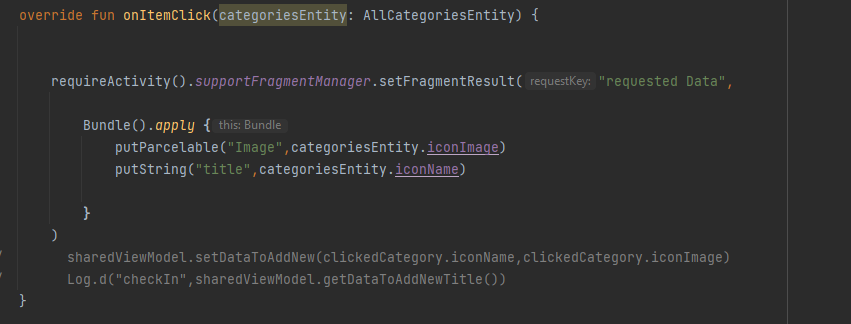




Make sure this inside fragment use requireActivity().

Similarly I did for expense fragment

**To pass the data from fragment to the activity**



From fragment above picture and in activity look below picture

