# 1º Trimestre

usecase · repository(es como el dao) · service(la interfaz) · response(construye el objeto respuesta?)

Inyeccion de dependencias

@HiltAndroidApp en la clase de la aplicacion y poner tambien el timber

@AndroidEntryPoint en cada activity

el viewmodel lo declaras como una variable poniendo by viewmodels al final

@HiltViewModel en cada viewmodel mas @Inject antes del constructor, lo que este dentro del constructor tambien tiene que tener el inject

Adapter

Solo se crea un solo

Debe tener una interfaz con todas las cosas que va a hacer el recyclerview, en el constructor de este declaras la interfaz

Llamas al adapter en el activity, porque hay cosas a las que no tiene acceso, por lo que haces override de los metodos y metes su logica alli

Una sealed class es una clase en la que sus hijos son los que nosotros especifiquemos en el padre, no cualquiera puede heredarla sin mas.

El event es un intermediario entre el viewmodel y el activity, solo se llama al viewmodel desde el activity en los observables con el handleevent(la clase o el objeto que queramos dentro del event)

El viewmodel solo tiene de metodos publicos el handleevent y el state. El handleEvent es un metodo que tiene como parametros las clases del Event, que hace un menu para dependiendo de la clase que ha llamado el activity, el viewmodel hace algo.

Cuidado con los colores dinamicos en el material que no lo soportan

si quiero mis propios colores, tiene que poner noactionbar en el style del themes.xml

sacar los iconos del topappbar desde fontawesome, poner esto en un pdf

para filtrar las personas con lo de la busqueda pones el listener para que cada vez que escribas algo llameal viewmodel para filtrar, y el viewmodel cambiara la lista del state, que lo observa el activity y lo cambia en pantalla

en el swipegesture cambias las animaciones como quieres que se comporten los elementos cuando hagas el swipe

* - - -- - - -- - -- - -- - - - - -- - - - -- - - -

Data, es como el dao, tienes dentro

model que es para el objeto que recibes del api

repositorios que es como el impl

sources que ahi dentro va lo del jakarta y las interfaces

Domain

contiene los objetos a los que conviertes del response

contiene los usecases que es la capa de servicio

Framework, contiene todas las pantallas

cada pantalla tiene lo siguiente

Activity, a lo que se refiere el xml, tiene listeners de cuando

se le de click a un boton de la pantalla llame al viewmodel,

observa el state y hace cambios en la pantalla a partir de los

cambios en el state

State, contiene los datos que quieres conservar y cambiar.

Viewmodel, contiene toda la logica que llama a los usecases

y cambia el state con lo que ha recibido, todos sus metodos

son privados para que desde el activity solo puedas llamar a

handleevent.

Event, es una especie de interfaz con las "clases" que luego

recojes en el viewmodel que cuando se hayan llamado hace

un metodo especifico

Adapter, sirve para que el recyclerview identifique los elementos y los meta en la lista,

NECESITARAS

Examen

NachrcoRestaurante

NachoFragments

NavigationDeCero

reciclerviewenhanced de la solucion

es una parte del xml para componer el activity en varias partes y reutilizarlo,

vamos a hacer un sustituto de un activity por el fragment y le vamos a poner ahi la pantalla, y van a haber fragments como superpuestos en un activity, te permite poner menus para todo

vamos a hacer un activity con un botonsheet

este activity en el xml tiene un fragmentcontainerview que es como el anchorpane y un bottomnavigation que es como un menunavigation

el bottomnavigation se refiere a un menu que esta declarado en un xml a parte que te gestiona austomatico todo cuando le das click, pero lo tienes que llamar igual al fragment al que te refieres

para crear un fragment primeto tienes que crearlo a la altura de los activity, lo copias del del profe PrimerFragment, hereda de fragment, que tiene un binding que es como un state pero distinto, lo copias del profe

dentro de la clase del fragment, le oncreateview se lo copias, el onviewcreated pones los listener de los botones, o cambias todo lo del activity,

una vez creado con su xml y todo, te sale en el navigation, que es un gestor de todos los fragmentes y ahi puedes configurarlos

el navigation tiene una seccion de arguments que es para pasar datos entre pantallas, y la seccion de actions que es para definir los accesos de una pantalla a otra, pero que luego tienes que llamar ese cambio de pantalla desde los listeners de cada fragment

los argumentos los defines en cada fragment tambien, pero luego los tienes que especificar, como al cargar la pantalla pues esto igual

cambiando de una pantalla a otra en el observer, es declarar el action llamando al actionNombreFragment y con eso ya puedes ver el cambio a pantalla que habias definido antes, lo autocomplementas al que quieras y listo, entre parentesis metes los argumentos, que es como el intent pero mucho mas facil

una vez tienes el action, llamas a la funcion cargarPantalla suya que es findNavController... y listo

en el mainactivity llamas al fragmentcontainer que es el grupo de todos los fragments, se lo copias al profe

para recibir el argumento una vez lo ha enviado el otro, vas al fragment en el que lo recibes y te declaras una variable (54:20) que ya la puedes utilizar sin mas args.variable1, variable 1 recuerda que tiene que estar declarado en el navigation tambien

LIBRERIA PA QUE FURULE

navigation-safe-agrs-plugin en el proyecto

navigationi-fragment-ktx y navigation-ui-ktx en el modulo

el plugin de el safeargs en el modulo tb para pasar argumentos

en el popbehaviour de cada fragment defines que pasa si le das atrás

# 2º Trimestre

## ROOMS Y FLOWS – MOVILES

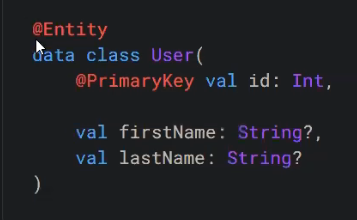
### ROOM

La vamos a usar para cachear datos en local.

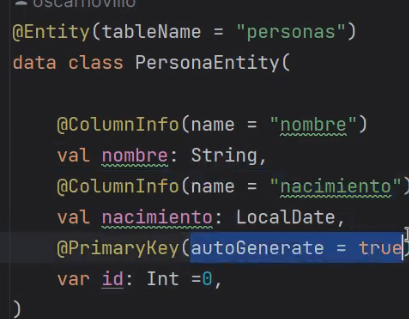
Proyecto del que basarnos para librerías: flows

Es una base de datos con tablas y relaciones y todo, pero que maneja poca información, esta diseñada para eso, necesitamos las entities que son clases convertidas a tablas, las interfaces que sirven para hacer el insert update etc y la base de datos que hay que hacerla con inyección de dependencias porque es mas fácil.

Las Entidades o tablas de los rooms no son mas que clases, se declaran con anotaciones @Entity y @PrimaryKey en la clave primaria

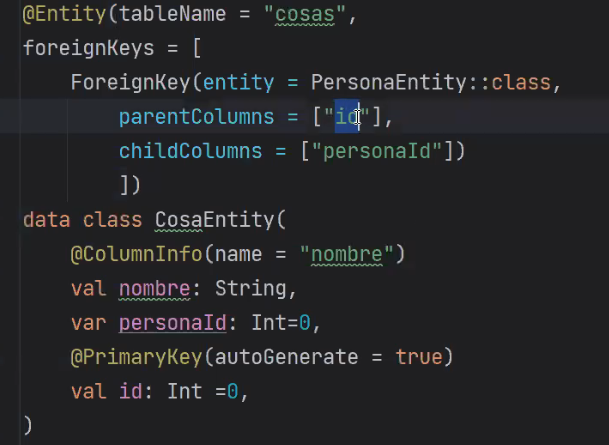


@columninfo es para que el nombre de las variables que quieras que este distinto en la base de datos a el nombre en la clase declarada:



Todo lo que guardemos con rooms no es seguro.

Vamos a acabar teniendo un modelo para el data y otro modelo para el domain

Para declarar una clave foránea, en este ejemplo vemos como vincula el id del padre(“id”, que es el nombre de la pk de persona declarado en otra clase) con el id del hijo (presonaid):  


DataMappers son los que pasaban de la entidad al objeto declarado en nuestro proyecto, por si no queríamos ciertas variables o algo asi, pero esto nos sirve en room para pasar del modelo de datos del data al modelo de datos del domain. Estos antes los hacíamos en el constructor de cada clase pero esta vez lo vamos a tener que hacer en una clase a parte porque es mejor tener todos los datamappers en el mismo sitio. 26:58

es una base de datos en un fichero que hace todo lo repetitivo de base de datos para ahorrar codigo

entitys, interfaz y base de datos

una tabla se crea con @Entity, con @PrimaryKey

datamappers en el modelo que se refiere al domain para hacer los converters, se hace ahi porque el domain es tonto

el dao se anota con @Dao

tb se anota cada query

se define la base de datos con una clase que hereda de RoomDatabase

las fechas y los uuids se gestionan en una clase comberter, que luego la referencias con la anotacion @TypeConvertert

9:23 empieza

dependencias: 9:43

un flow es como el single para hacer programacion reactiva, pero es nativo de koltin

hay dos tipos de flows, uno que se guarda el dato para siempre y otro que si el dato no lo escuchas lo tira

## COMPOSE