DAO

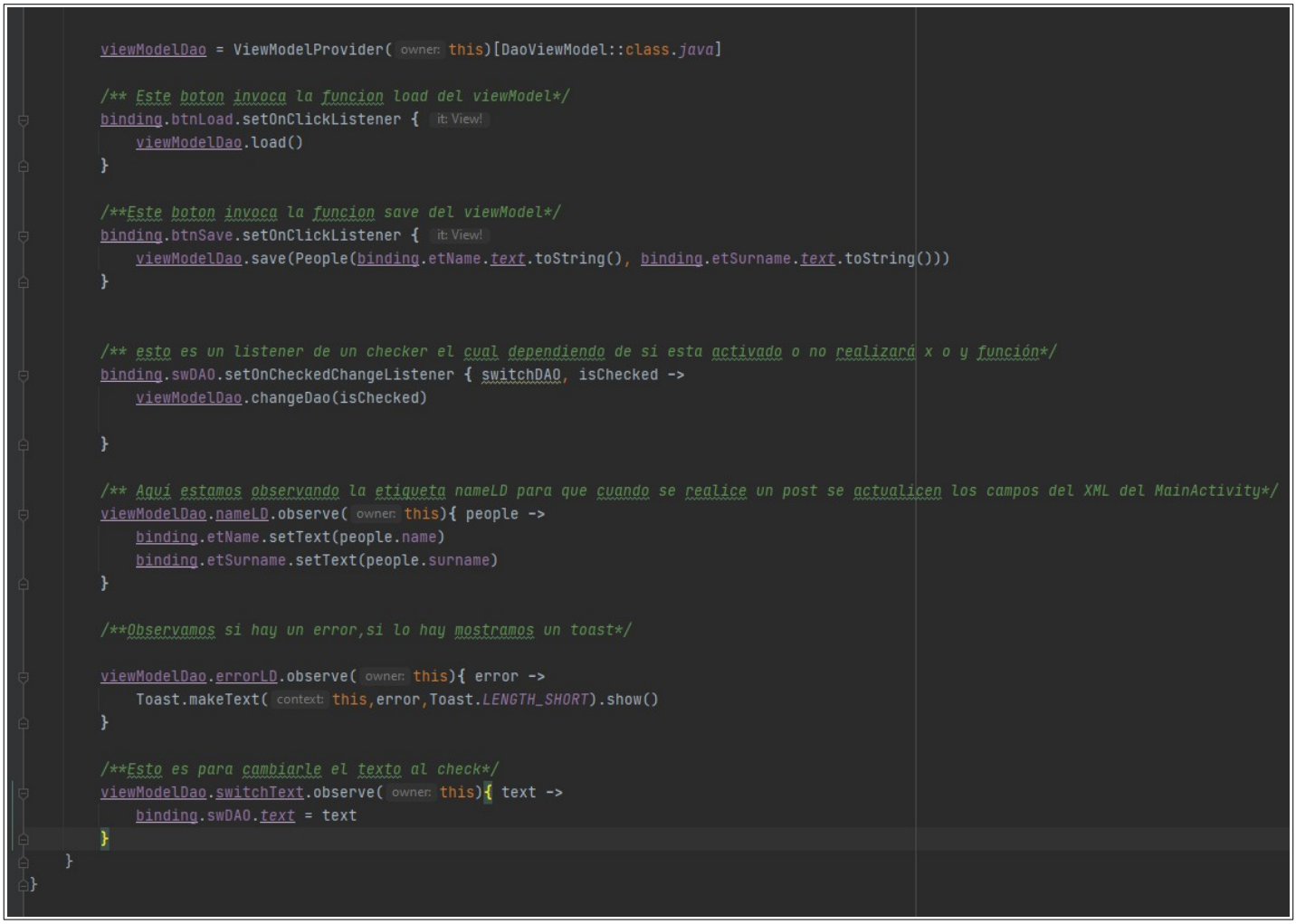
1-Lo primero que hacemos como siempre es activar el viewbinding y añadir las dependencias

```
    buildFeatures{
        viewBinding true
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.2'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    def viewModelVersion ="2.4.0"
    implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$viewModelVersion"
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:$viewModelVersion"

    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9")
}
```

2-Ahora vamos a ver el MainActivity en el cual solo tenemos el metodo onCreate pero tenemos varias cosas dentro.
Lo primero es el inflate del binding y el viewModel

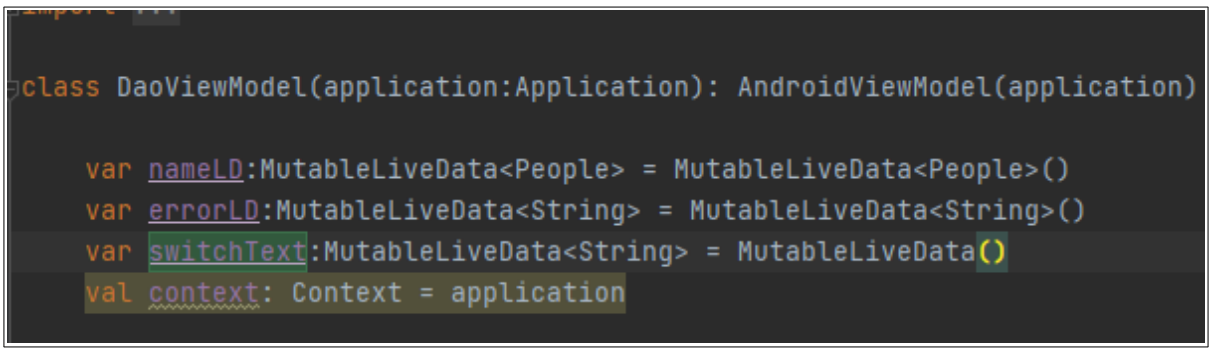
```
class MainActivity : AppCompatActivity() {

    private lateinit var viewModelDao:DaoViewModel
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(ActivityMainBinding.inflate(layoutInflater).also { binding = it }.root)
```

3-Esto es MUY complejo de explicar,asi que voy a ir paso a paso explicando el recorrido del programa,para empezar voy a mostrar el MainActivity entero con unos comentarios de las funciones .



```
        viewModelDao = ViewModelProvider( owner: this)[DaoViewModel::class.java]

        /** Este boton invoca la funcion load del viewModel*/
        binding.btnLoad.setOnClickListener {  it: View!
            viewModelDao.load()
        }

        /**Este boton invoca la funcion save del viewModel*/
        binding.btnSave.setOnClickListener {  it: View!
            viewModelDao.save(People(binding.etName.text.toString(), binding.etSurname.text.toString()))
        }


        /** esto es un listener de un checker el cual dependiendo de si esta activado o no realizará x o y función*/
        binding.swDAO.setOnCheckedChangeListener { switchDAO, isChecked ->
            viewModelDao.changeDao(isChecked)

        }

        /** Aquí estamos observando la etiqueta nameLD para que cuando se realice un post se actualicen los campos del XML del MainActivity*/
        viewModelDao.nameLD.observe( owner: this){ people ->
            binding.etName.setText(people.name)
            binding.etSurname.setText(people.surname)
        }

        /**Observamos si hay un error,si lo hay mostramos un toast*/

        viewModelDao.errorLD.observe( owner: this){ error ->
            Toast.makeText( context: this,error,Toast.LENGTH_SHORT).show()
        }

        /**Esto es para cambiarle el texto al check*/
        viewModelDao.switchText.observe( owner: this){ text ->
            binding.swDAO.text = text
        }
    }
}
```

4-Lo siguiente que vamos a ver es el ViewModel que en este caso se llama DaoViewModel y que hemos visto que ha sido invocado en el MainActivity,lo primero que tenemos es la declaración de las variables observables y el contexto de la aplicación(que se le pasa por que es un AndroidViewModel(application))



```
class DaoViewModel(application:Application): AndroidViewModel(application)

    var nameLD:MutableLiveData<People> = MutableLiveData<People>()
    var errorLD:MutableLiveData<String> = MutableLiveData<String>()
    var switchText:MutableLiveData<String> = MutableLiveData()
    val context: Context = application
```

5-Ahora viene una explicación dura , esta linea de codigo te crea una variable de una interfaz llamada IMyDAO,la cual cuenta con unas funciones(save y load) que son suspends(para ejecutarse en corrutines)

```
//By default we work with Files
var iMyDAO:IMyDAO = DaoFiles.getInstance(context)
```

-SAVE: para crear la funcion save esta interfaz hace falta pasarle una **persona** en este caso(Clase creada que solo guarda nombre y apellido,aquí seria donde si quisieramos poner otra cosa tendriamos que modificar en si) ,también le pasa un string (OnError que marca si hay error) , un saved(que es un booleano)

-LOAD: En esta como el propio nombre indica cargamos la persona ya guardada al texto,por ello simplemente tenemos que pasarle un OnLoaded(type alias de People) y un onError por si falla.

```
import com.catata.directoriesfilessharedddao.model.People
typealias OnError = (error:String)->Unit
typealias OnSaved = (isSaved:Boolean)->Unit
typealias OnLoaded = (people:People)->Unit

interface IMyDAO {
    suspend fun save(people: People, onSaved: OnSaved, onError: OnError?):Unit
    suspend fun load(onLoaded: OnLoaded,onError: OnError?): Unit
}
```

6-Ya hemos acabado con la interfaz ImyDAO,y aunque queda codigo por ver en el viewModel ahora lo que vamos a hacer es ver DaoFiles,que es la clase ImyDAO la cual le pasa a la Interfaz anterior todo lo relacionado con DaoFiles.

```
//By default we work with Files
var iMyDAO:IMyDAO = DaoFiles.getInstance(context)
```

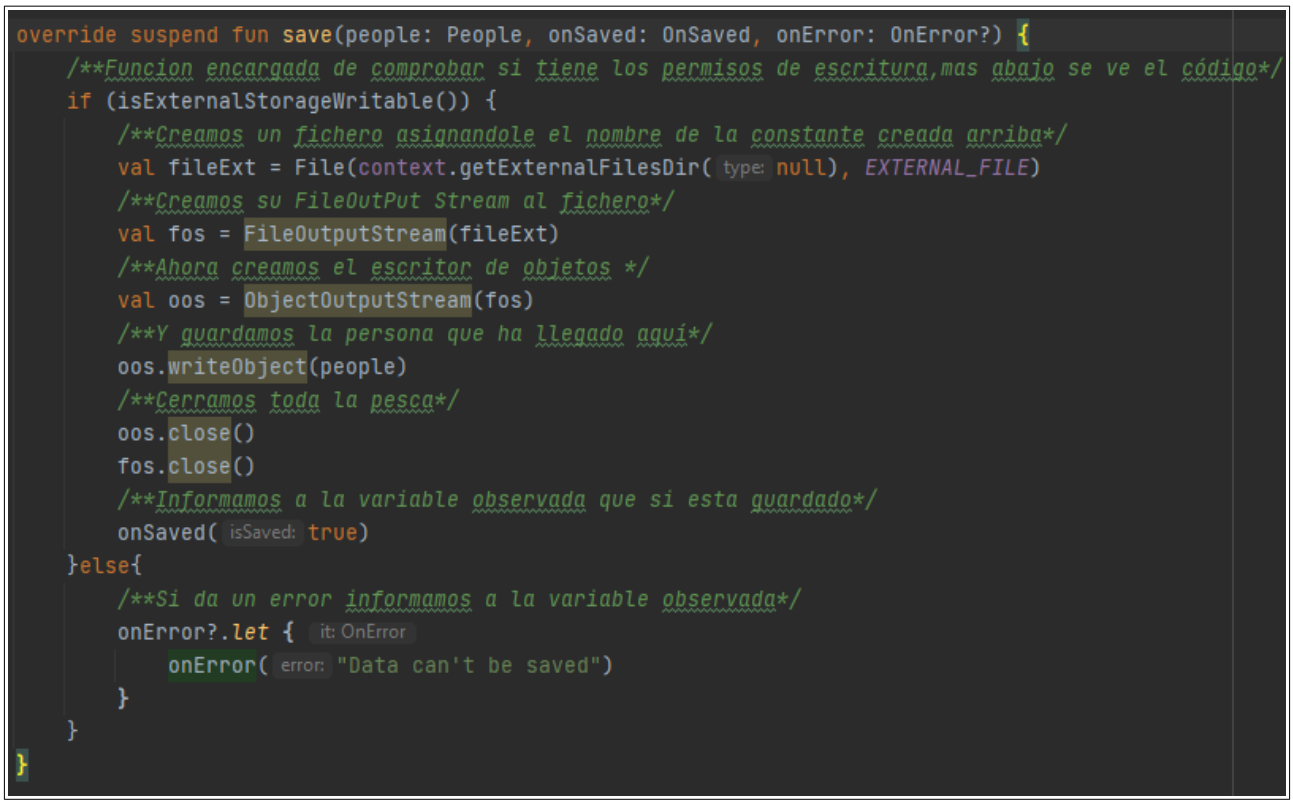
7-Lo primero que tenemos es una constante con el nombre del fichero donde se va a guardar el objeto People.

```
const val EXTERNAL_FILE = "External_File.txt"
```

8-Ahora tenemos el metodo getInstance que pasandole un contexto devuelve un DaoFiles para crear el objeto de la interfaz que he comentado antes.
Todo esto esta encapsulado en un companion object para que en todas las instancias hace lo mismo.

```
companion object{
    var daofile:DaoFiles? = null

    fun getInstance(c:Context):DaoFiles {
        daofile?.let {} ?: run {  this: Companion
            daofile = DaoFiles(c)
        }
        return daofile!!
    }

}
```

9-Por fin tenemos el primer metodo llamado Save el cual se encarga de guardar,esta explicado incode



```
override suspend fun save(people: People, onSaved: OnSaved, onError: OnError?) {
    /**Funcion encargada de comprobar si tiene los permisos de escritura,mas abajo se ve el código*/
    if (isExternalStorageWritable()) {
        /**Creamos un fichero asignandole el nombre de la constante creada arriba*/
        val fileExt = File(context.getExternalFilesDir( type: null), EXTERNAL_FILE)
        /**Creamos su FileOutPut Stream al fichero*/
        val fos = FileOutputStream(fileExt)
        /**Ahora creamos el escritor de objetos */
        val oos = ObjectOutputStream(fos)
        /**Y guardamos la persona que ha llegado aqui*/
        oos.writeObject(people)
        /**Cerramos toda la pesca*/
        oos.close()
        fos.close()
        /**Informamos a la variable observada que si esta guardado*/
        onSaved( isSaved: true)
    }else{
        /**Si da un error informamos a la variable observada*/
        onError?.let {  it: OnError
            onError( error: "Data can't be saved")
        }
    }
}
```

10-Ahora vamos a ver el metodo load que también estará explicado en código

```
override suspend fun load(onLoaded: OnLoaded, onError: OnError?) {
    /**Si tenemos permitido Leer*/
    if(isExternalStorageReadable()){
        /**Creamos el fichero inSystem*/
        val fileExt = File(context.getExternalFilesDir( type: null), EXTERNAL_FILE)
        /**Si el fichero existe*/
        if(fileExt.exists()){
            /**Creamos el FileInput del fichero*/
            val fis = FileInputStream(fileExt)
            /**Creamos el lector de objetos del fichero*/
            val ois = ObjectInputStream(fis)
            /**Guardamos la persona utilizando el lector de objetos*/
            val p: People = ois.readObject() as People
            ois.close()
            fis.close()
            /**Devolvemos la persona mediante la observable data de People llamada onLoaded*/
            onLoaded(p)
        }


    }else{
        onError?.let{ it: OnError
            onError( error: "Data can't be loaded")
        }
    }
}
```

11-Por último tenemos dos funciones que se encargan de comprobar los permisos de lectura/escritura

```
// for read and write:
private fun isExternalStorageWritable(): Boolean {
    return Environment.getExternalStorageState() == Environment.MEDIA_MOUNTED
}

// Checks if a volume containing external storage is available to at least read.
private fun isExternalStorageReadable(): Boolean {
    return Environment.getExternalStorageState() in
            setOf(Environment.MEDIA_MOUNTED, Environment.MEDIA_MOUNTED_READ_ONLY)
}
```

12-Ahora volvemos al VIEWMODEL recuerda que había explicado la manera de guardar mediante ficheros DaoFiles,ahora vamos ver como trabajan estas funciones (load y save) en el viewmodel

Como podemos ver lo unico que hacen es “observar” el ImyDao y hacer postValue en las variables observables

13-Y aquí tenemos un metodo con MUCHISIMA chicha,este metodo es el que reacciona al check que hay en el xml del MainActivity.
Si tu invocas esta funcion,la variable iMyDAO que he creado antes pasa de ser un DaoFiles a ser un daoShared.
Es decir,esta funcion hace que accedas a las funciones de otra clase TOTALMENTE DISTINTA que guarda los datos de una manera disinta.

```
fun changeDao(type: Boolean){
    if(type){ //SharedPreferences
        iMyDAO = DaoShared(context)
        switchText.postValue( value: "Shared")
    }else{ //Files
        iMyDAO = DaoFiles(context)
        switchText.postValue( value: "Files")
    }
}
```

14-Aqui hacemos lo mismo que en el metodo getInstance de DaoFiles,no esta en utilización por que solo sirve cuando se crea una variable NUEVA ,cuando reasignas valor con poner
iMyDAO = DaoShared(context) sirve.

```
ass DaoShared(val context:Context): IMyDAO {

 companion object{
     var daoShared:DaoShared? = null
     fun getInstance(c:Context):DaoShared{
         daoShared?.let {} ?: run {  this: Companion
             daoShared = DaoShared(c)
         }
         return daoShared!!
     }
 }
```

15-Ahora vienen comentarios un poco teoricos,un SharedPreferences apunta a un archivo de keys el cual se puede escribir,leer y decir que es privado, a continuación muestro el archivo en cuestion.
Es strings xml en el que hemos creado nuevos atributos

```
oViewModel.kt × DaoShared.kt × strings.xml ×
anslations for all locales in the translations editor.
<resources>
    <string name="app_name">DirectoriesFiles</string>
    <string name="my_shared_file">com.catata.directoriesfiles</string>
    <string name="key_preference">key_name</string>
    <string name="key_preference_surname">Surname_KEY</string>
    <string name="key_preference_name">name_KEY</string>
</resources>
```

16-Entonces lo que hace en el metodo save es crear el sharedPred pasandole el contexto de la key que marca la carpeta del trabajo (com.catata.directoriesfiles) y diciendole que sea privado.
Luego este lo editamos con .edith() y asignamos a la key nombre el nombre de la persona y a la key surname el apellido de la persona.
Si todo va bien le dice a la observable onSaved que todo correcto,si no,marca error

```
override suspend fun save(people: People, onSaved: OnSaved, onError: OnError?) {
    val sharedPref2 = context.getSharedPreferences(
        context.getString(R.string.my_shared_file), /**Direccion?*/
        Context.MODE_PRIVATE
    )
    with(sharedPref2.edit()){ this: SharedPreferences.Editor!
        putString(context.getString(R.string.key_preference_name),/**Nombre*/
            people.name)
        putString(context.getString(R.string.key_preference_surname),/**Apellido*/
            people.surname)
        if(commit())
            onSaved( isSaved: true)  ^with
        else {
            onError?.let {  it: OnError
                it( error: "Data can't be saved on Shared Preferences")
            }  ^with
        }
    }
}
```

17-En el load hace algo parecido pero utilizando getString para coger los valores creando una Persona que devolveremos con la variable observable onLoaded.

```
override suspend fun load(onLoaded: OnLoaded, onError: OnError?) {
    val sharedPref2 = context.getSharedPreferences(
        "com.catata.directoriesfiles",
        Context.MODE_PRIVATE
    )
    val name = sharedPref2.getString("name_KEY",  defValue: "")
    val surname = sharedPref2.getString("Surname_KEY",  defValue: "")
    if(name == "" || surname==""){
        onError?.let {  it: OnError
            onError( error: "Data from shared can't be loaded")
        }
    }else{
        onLoaded(People(name, surname))
    }
}
```