Guía de ejercicios - Ciclo de vida de componentes (I)

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada N° 1: Compartir información entre Fragments usando ViewModel

En la siguiente actividad usaremos un ViewModel para enviar datos entre dos Fragments

1. Creamos un proyecto nuevo, y seleccionamos Empty Activity, luego debemos dejar el MainActivity como el codigo a continuacion:

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
}
```

2. Debemos agregar en build.gradle en el módulo de App la siguiente entrada:

```
implementation "androidx.fragment:fragment-ktx:1.5.2"
```

3. Para evitar la creación de código que no usaremos, esta vez crearemos dos Fragments sin hacer uso del asistente de Android Studio:

    a. Creamos un Fragment (Kotlin class) y lo llamaremos UpperFragment, :

```
class UpperFragment : Fragment() {

    val viewModel: MainViewModel by activityViewModels()
    private lateinit var binding: FragmentUpperBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
```

```
    ): View? {
        binding = FragmentUpperBinding.inflate(layoutInflater,
container, false)
        return binding.root
    }
}
```

b. Creamos la vista de diseño en XML y la llamamos "`fragment_upper`". La vista XML del Fragment solo contendrá un `ConstraintLayout` y un `AppCompatTextView`:

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/white"
    tools:context=".LowerFragment">

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/tv_click"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/black"
        android:textSize="98sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

c. Repetimos los pasos anteriores, pero esta vez llamamos al Fragment LowerFragment

```
class LowerFragment : Fragment() {
```

```
    val viewModel: MainViewModel by activityViewModels()
    private lateinit var binding: FragmentLowerBinding

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentLowerBinding.inflate(layoutInflater,
container, false)
        return binding.root
    }
}
```

d. Al igual que en el caso anterior, creamos la vista XML como en el siguiente snippet:

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/black"
    tools:context=".LowerFragment">

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/tv_click"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click!"
        android:textColor="@color/white"
        android:textSize="98sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

4. A continuación necesitamos crear el ViewModel, en él, haremos uso de funciones que nos permitan:

```
class MainViewModel : ViewModel() {

    private val counterMutableStateFlow: MutableStateFlow<Int> =
MutableStateFlow(0)
    val counterStateFlow: StateFlow<Int> =
counterMutableStateFlow.asStateFlow()

    fun increase() {
        counterMutableStateFlow.value += 1
    }
}
```

5. Ahora, necesitamos hacer lo siguiente: cada vez que el usuario presione el TextView "Click!", en el `LowerFragment`, éste debe enviar un dato al `UpperFragment` indicando que el contador debe incrementar en 1 unidad.

   No podemos simplemente usar una variable a la cual le seteamos un valor cada vez que se presione, y luego leerlo en otro fragment. Para esto necesitamos usar LiveData o StateFlow, ya que es un valor que va a estar cambiando constantemente.

```
private val counterMutableStateFlow: MutableStateFlow<Int> =
MutableStateFlow(0)
    val counterStateFlow: StateFlow<Int> =
counterMutableStateFlow.asStateFlow()

    fun increase() {
        counterMutableStateFlow.value += 1
    }
```

6. Ahora ya tenemos la función que se encarga de incrementar en 1 unidad cada vez que se presiona el TextView, y ya tenemos la variable que leeremos desde el `UpperFragment`.

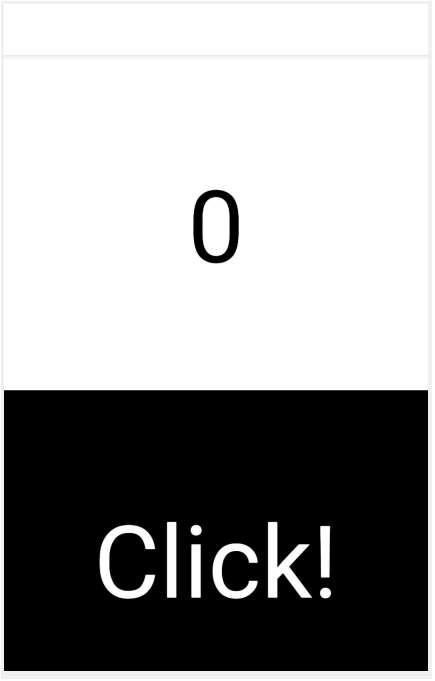
   Primero debemos hacer uso de una función definida en el ViewModel.

```
binding.tvClick.setOnClickListener {
    viewModel.increase()
}
```

7. Luego necesitamos leer la variable que está definida en el ViewModel y que se actualiza cada vez que el usuario presiona el TextView.

```
lifecycleScope.launchWhenCreated {
    viewModel.counterStateFlow.collect {
        binding.tvClick.text = it.toString()
    }
}
```

A continuación se muestra un screenshot de la app corriendo, la idea es simple, cada vez que el usuario presionar "Click!" En el `LowerFragment` este incrementa un contador definido en el `MainViewModel` y `UpperFragment` está siempre atento para actualizar con el nuevo valor del contador.



Con esto finaliza la actividad, el archivo **Apoyo Guía de ejercicios - Ciclo de vida de componentes (I).zip** contiene el código completo, puedes descargar el archivo y comparar.

¡Manos a la obra! - Solicitud de Permisos

En la siguiente actividad haremos solicitud de permisos para el uso del Storage.

1. Define el permiso en el Manifest.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
```

2. Crea una funcion que retorne un Toast mostrando el mensaje “Permiso concedido” y llámala `permissionGranted()`

```
private fun permissionGranted(){
    Toast.makeText(this, "Permiso concedido", Toast.LENGTH_LONG).show()
}
```

3. Crea una funcion que retorne un Toast mostrando el mensaje “Permiso denegado” y llámala `permissionDenied()`

```
private fun permissionDenied(){
    Toast.makeText(this, "Permiso denegado", Toast.LENGTH_LONG).show()
}
```

4. Define un callback para la solicitud del permiso, si el permiso es concedido, entonces llama a la función `permissionGranted()` si no llama la función `permissionDenied()` (puedes guiarte con la slide “**Solicitud de permisos sensibles**” de la sesión *Android (Parte II)*)

```
registerForActivityResult(ActivityResultContracts.RequestPermission()) {
isGranted ->
    if (isGranted) {
        permissionGranted()
    } else {
        permissionDenied()
```

```
    }
}
```

5. Crea una funcion que se llame `requestPermission()`, llámala en `onCreate()` y copia el siguiente código en ella:

```
when {
ContextCompat.checkSelfPermission(this,Manifest.permission.READ_EXTERNAL
_STORAGE) == PackageManager.PERMISSION_GRANTED -> {
         // Permiso concedido
    }

ActivityCompat.shouldShowRequestPermissionRationale(this,Manifest.permis
sion.READ_EXTERNAL_STORAGE) -> {
        // Preguntar otra vez por el permiso
    }
    else -> {
requestPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL_STORA
GE)
    }
}
```

6. Reemplaza el comentario "`// Permiso concedido`" con la función `permissionGranted()`

7. Reemplaza el comentario "`// Preguntar otra vez por el permiso`" con un Toast con el texto "**Preguntar otra vez por el permiso**".

8. Compáralo con el código final.

Respuestas - Solicitud de Permisos

1.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
```

2.

```
private fun permissionGranted(){
    Toast.makeText(this, "Permiso concedido", Toast.LENGTH_LONG).show()
}
```

3.

```
private fun permissionDenied(){
    Toast.makeText(this, "Permiso denegado", Toast.LENGTH_LONG).show()
}
```

4.

```
registerForActivityResult(ActivityResultContracts.RequestPermission()) {
isGranted ->
    if (isGranted) {
        permissionGranted()
    } else {
        permissionDenied()
    }
}
```

5.

```
when {
ContextCompat.checkSelfPermission(this,Manifest.permission.READ_EXTERNAL
_STORAGE) == PackageManager.PERMISSION_GRANTED -> {
         // Permiso concedido
    }

ActivityCompat.shouldShowRequestPermissionRationale(this,Manifest.permis
sion.READ_EXTERNAL_STORAGE) -> {
        // Preguntar otra vez por el permiso
    }
```

```
    else -> {
requestPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL_STORA
GE)
    }
}
```

6.

```
private fun requestPermission() {
        when {
            ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) == PackageManager.PERMISSION_GRANTED -> {
                permissionGranted()
            }
            ActivityCompat.shouldShowRequestPermissionRationale(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) -> {

            }
            else -> {

requestPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL_STORA
GE)
            }
        }
    }
```

7.

```
private fun requestPermission() {
        when {
            ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) == PackageManager.PERMISSION_GRANTED -> {
                permissionGranted()
            }
            ActivityCompat.shouldShowRequestPermissionRationale(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
```

```
            ) -> {
                Toast.makeText(this, "Preguntar otra vez por el
permiso", Toast.LENGTH_LONG).show()
            }
            else -> {

requestPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL_STORA
GE)
            }
        }
    }
```

8.

```
class MainActivity : AppCompatActivity() {

    private val requestPermissionLauncher =

registerForActivityResult(ActivityResultContracts.RequestPermission()) {
isGranted ->
            if (isGranted) {
                permissionGranted()
            } else {
                permissionDenied()
            }
        }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        requestPermission()
    }

    private fun requestPermission() {
        when {
            ContextCompat.checkSelfPermission(
                this,
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) == PackageManager.PERMISSION_GRANTED -> {
                permissionGranted()
            }
            ActivityCompat.shouldShowRequestPermissionRationale(
                this,
```

```
                Manifest.permission.READ_EXTERNAL_STORAGE
            ) -> {
                Toast.makeText(this, "Preguntar otra vez por el
permiso", Toast.LENGTH_LONG).show()
            }
            else -> {

requestPermissionLauncher.launch(Manifest.permission.READ_EXTERNAL_STORA
GE)
            }
        }
    }

    private fun permissionGranted() {
        Toast.makeText(this, "Permiso concedido",
Toast.LENGTH_LONG).show()
    }

    private fun permissionDenied() {
        Toast.makeText(this, "Permiso denegado",
Toast.LENGTH_LONG).show()
    }
}
```

¡Manos a la obra! - Jugando con el ciclo de vida

En la siguiente actividad vamos a jugar un poco con el ciclo de vida, para esto, necesitamos crear un proyecto con un Activity vacío o Empty Activity y nómbralo “Android LifeCycle”. Haremos uso de Logcat para entender sobre el ciclo de vida, podemos crear una entrada en el log de la siguiente forma:

```
Log.d(TAG, "onCreate")
```

TIP: Puedes escribir “logd” y Android Studio te ayudará a crear el log

TAG: Es un string que nos ayudará a encontrar nuestro log, asi que lo definiremos de la siguiente forma:

1. Crea un val y llamalo TAG, debe ser accesible por todos las funciones y métodos en el Activity:
   `private val TAG = "MainActivity >>>>>>>>>>> "`

2. Crear un log en `onCreate()` y ejecutar la app, compara el resultado de Logcat con la respuesta.

3. Sobreescribe el método `onStart()` y repite el paso 2 (Crear un log y ejecutar la app, compara el resultado de Logcat con la respuesta).

4. Sobreescribe el método `onResume()` y repite el paso 2.

5. Sobreescribe el método `onPause()` y repite el paso 2, pero esta vez agrega salir de la app y volver a entrar.

6. Sobreescribe el método `onStop()` y repite el paso 2, pero esta vez agrega salir de la app y volver a entrar.

7. Sobreescribe el método `onDestroy()` y repite el paso 2, pero esta en vez de salir de la app, usa el administrador de apps para matarla.

8. Compara todo tu código con la respuesta.

Respuestas - Jugando con el ciclo de vida

1.

```
class MainActivity : AppCompatActivity() {

    private val TAG = "MainActivity >>>>>>>>>>> "

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

2. Respuesta onCreate()

```
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
```

3. Respuesta onStart()

```
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
```

4. Respuesta onResume()

```
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
```

5. Respuesta onPause()

```
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
androidlifecycle D/MainActivity >>>>>>>>>>>: onPause
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
```

6. Respuesta onStop()

```
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
androidlifecycle D/MainActivity >>>>>>>>>>>: onPause
androidlifecycle D/MainActivity >>>>>>>>>>>: onStop
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
```

7. Respuesta onDestroy()

```
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
androidlifecycle D/MainActivity >>>>>>>>>>>: onPause
androidlifecycle D/MainActivity >>>>>>>>>>>: onStop
androidlifecycle D/MainActivity >>>>>>>>>>>: onDestroy
androidlifecycle D/MainActivity >>>>>>>>>>>: onCreate
androidlifecycle D/MainActivity >>>>>>>>>>>: onStart
androidlifecycle D/MainActivity >>>>>>>>>>>: onResume
```

8. Todo el código

```
class MainActivity : AppCompatActivity() {

    private val TAG = "MainActivity >>>>>>>>>>> "

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Log.d(TAG, "onCreate")
    }

    override fun onStart() {
        super.onStart()
        Log.d(TAG, "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume")
    }
```

```
    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }

    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }
}
```

Preguntas de proceso

Reflexiona:

- A partir de lo aprendido hasta aquí ¿Qué contenido se me ha hecho más difícil? ¿Y cuál más sencillo?
- ¿Existe algún ejercicio de los resueltos previamente que deba repetir o desarrollar otra vez para poder dominarlo mejor?
- Nombra al menos un uso que pueda darle a lo aprendido hasta ahora.

Preguntas de cierre

- A tu criterio, ¿cómo se deberían distribuir los ViewModels en una app?
- ¿Qué crees que pasaría si ejecutamos una función la cual hace cambios en el UI y llamamos esa función en el método `onDestroy()`?
- ¿Si llamamos a un Toast dentro del método `onDestroy()` se ejecuta o no?
- Después que una app rota desde Portrait mode a Landscape Mode, ¿cuáles son los métodos del ciclo de vida que son llamados?
- El permiso para el uso de Internet en una app, ¿es considerado como permiso sensible?