



Hoja_UT2_11

MVVM

Seguiremos completando la aplicación **App_UT2_06**

Ahora vamos a crear un modelo que pueda propagar los datos entre los distintos fragments de nuestra aplicación.

Cuando terminemos seleccionaremos el tipo de visita, la cantidad de adultos y niños, así como la fecha y nos calculará el importe total.

VIEWMODEL

Antes de comenzar vamos a añadir las dependencias en el fichero build.gradle del módulo

```
//ViewModel
def lifecycle_version = "2.3.1"
implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
implementation "androidx.lifecycle:lifecycle-viewmodel-savedstate:$lifecycle_version"
```

Además, añadiremos estableceremos la vinculación de datos (dataBinding). Esto permite vincular los componentes de la IU de los diseños a las fuentes de datos de la app usando un formato declarativo en lugar de la programación:

```
buildFeatures {
    viewBinding true
    dataBinding true
}
```

Crearemos un paquete llamado modelo. En él crearemos una clase ReservaZooViewModel que herede de ViewModel.

Tendrá como propiedades:

- El tipo de visita (String)
- El número de adultos (Int)
- El número de niños (Int)
- La fecha de visita (Calendar)
- El precio (Double)

```
class ReservaZooViewModel: ViewModel()
{
    val tipoReserva = MutableLiveData<String>()
    val fecha = MutableLiveData<Calendar>()
    val numeroAdultos = MutableLiveData<Int>()
    val numeroNinos = MutableLiveData<Int>()
    val precio = MutableLiveData<Double>()
```



Hoja_UT2_11

El problema que tiene este código es que estamos haciendo públicas las propiedades MutableLiveData, con lo cuál, podrían modificar el contenido de los datos de esta estructura (no se puede volver a asignar la variable al establecerla con val, pero sí se puede modificar su contenido).

Por eso, en un elemento ViewModel, lo recomendable es no exponer los datos mutables. En su lugar, establecemos las propiedades mutables como privadas e implementamos una propiedad de copia y ponemos como pública la versión inmutable de cada propiedad.

La convención es que el nombre de las propiedades mutables vaya precedido de un guion bajo (_)

```
class ReservaZooViewModel: ViewModel()
{
    private val _tipoReserva = MutableLiveData<String>()
    private val _fecha = MutableLiveData<Calendar>()
    private val _numeroAdultos = MutableLiveData<Int>()
    private val _numeroNinos = MutableLiveData<Int>()
    private val _precio = MutableLiveData<Double>()

    val tipoReserva: LiveData<String> = _tipoReserva
    val fecha: LiveData<Calendar> = _fecha
    val numeroAdultos: LiveData<Int> = _numeroAdultos
    val numeroNinos: LiveData<Int> = _numeroNinos
    val precio: LiveData<Double> = _precio
}
```

También añadiremos unos métodos set para establecer los valores:

```
fun setTipoReserva(tipo: String)
{
    _tipoReserva.value = tipo
}

fun setFecha(fecha: Calendar)
{
    _fecha.value = fecha
    actualizarPrecio()
}

fun setNumeroAdultos(numero: Int)
{
    _numeroAdultos.value = numero
    actualizarPrecio()
}

fun setNumeroNinos(numero: Int)
{
    _numeroNinos.value = numero
    actualizarPrecio()
}
```



Hoja_UT2_11

Para usar el viewModel en FragmentInicio, hay que inicializar el elemento ReservaZooViewModel con **activityViewModels()** en lugar de la clase delegada viewModel():

- viewModel() ofrece la instancia ViewModel con alcance para el fragmento actual. Será diferente para los distintos fragmentos.
- activityViewModels() ofrece la instancia ViewModel con alcance para la actividad actual. Por lo tanto, **la instancia permanecerá igual en varios fragmentos en la misma actividad**.

De este modo, en todos los Fragment añadiremos el viewModel compartido:

```
private val viewModelCompartido: ReservaZooViewModel by activityViewModels()
```

La clase **FragmentInicio** quedará así:

```
class FragmentInicio : Fragment()
{
    private lateinit var binding: FragmentInicioBinding
    private val viewModelCompartido: ReservaZooViewModel by activityViewModels()

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?)
    {
        super.onViewCreated(view, savedInstanceState)

        binding.apply {
            botonZoologico.setOnClickListener { tipoVisita("Zoológico") }
            botonReptario.setOnClickListener { tipoVisita("Reptario") }
            botonVisitaGuiada.setOnClickListener { tipoVisita("Visita guiada") }
        }
    }

    fun tipoVisita(tipo: String)
    {
        viewModelCompartido.setTipoReserva(tipo)
        findNavController().navigate(R.id.action_fragmentInicio_to_fragmentPersonas)
    }
}
```

Hay que fijarse que en cada botón se establece el evento onClick, que llamará a una función que establecerá el tipo de reserva del viewModel, justo antes de navegar hacia el siguiente fragment.



Hoja_UT2_11

En **FragmentPersonas** lo único que haremos en el método `onViewCreated` es establecer el número de adultos y de niños cuando se lance el evento `onValueChanged` de cada `numberPicker` (adultos y niños).

El código para el `numberPickerAdultos` es el siguiente:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
    super.onViewCreated(view, savedInstanceState)

    //Establecemos el viewModel definido en el layout (ver explicación)
    binding.viewModel = viewModelCompartido
    binding.lifecycleOwner = viewLifecycleOwner

    binding.numberPickerAdultos.apply {
        minValue = 0
        maxValue = 5
        value = 1
        wrapSelectorWheel = false
        setOnValueChangedListener { _, _, nuevo
            -> viewModelCompartido.setNumeroAdultos(nuevo)
        }
    }

    // ...

    //Realizar lo mismo con numberPickerNinos
```

Para realizar correctamente la vinculación de datos, hay que crear una variable en cada layout. Podemos llamar `viewModel`. Por ejemplo en **fragment_personas.xml** sería:

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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"
    tools:context=".FragmentPersonas">

    <data>
        <variable
            name="viewModel"
            type="es.ivanlorenzo.app_ut2_06.modelo.ReservaZooViewModel" />
        </data>

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

Hay que indicar el nombre del paquete en el atributo `type`. Debe coincidir exactamente con el `ViewModel`.



Hoja_UT2_11

Ahora podemos establecer que este viewModel será nuestro viewModelCompartido (como se ve en el método onViewCreated de la clase FragmentPersonas)

De este modo, ahora se puede establecer directamente en el layout cualquier dato almacenado en nuestro ReservaZooViewModel.

En **FragmentFecha** simplemente modificaremos el Toast que mostrábamos al cambiar de fecha por una llamada al método setFecha del viewModel:

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
    super.onViewCreated(view, savedInstanceState)

    binding.viewModel = viewModelCompartido
    binding.lifecycleOwner = viewLifecycleOwner

    val hoy = Calendar.getInstance()
    binding.datePicker.init(hoy.get(Calendar.YEAR), hoy.get(Calendar.MONTH),
        hoy.get(Calendar.DAY_OF_MONTH))
    { _, year, month, day ->
        var fecha = Calendar.getInstance()
        fecha.set(year, month, day)
        viewModelCompartido.setFecha(fecha)
    }

    binding.botonSiguiente.setOnClickListener {
        findNavController().navigate(R.id.action_fragmentFecha_to_fragmentResumen)
    }
}
```

En el **FragmentResumen** únicamente estableceremos nuestro viewModel definido en el layout (al igual que hacíamos con los anteriores fragments):

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
    super.onViewCreated(view, savedInstanceState)

    binding.viewModel = viewModelCompartido
    //No hace falta ya que no se modifica ningún dato en la UI
    //binding.lifecycleOwner = viewLifecycleOwner

    binding.botonReservar.setOnClickListener{
        Toast.makeText(context, "Se ha realizado la reserva", Toast.LENGTH_SHORT).show()
        findNavController().navigate(R.id.action_fragmentResumen_to_fragmentInicio)
    }
}
```

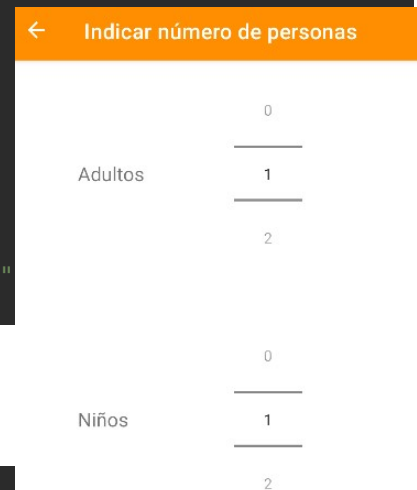


Hoja_UT2_11

VINCULACIÓN DE DATOS

Como ya hemos realizado la vinculación de datos podemos llamar a viewModel desde los layout. Así para mostrar el precio en la ventana de selección de personas podemos hacer lo siguiente:

```
<TextView
    android:id="@+id/total"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    android:text="@{@string/precio_total(viewModel.precio)}"
    android:textAppearance="?attr/textAppearanceHeadline6"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/numberPickerNinos"
/>
```



En el fichero strings.xml hemos establecido el texto precio_total:

```
<string name="precio_total">Total: %s</string>
```

Ahora bien, en ningún lugar hemos dicho cuánto cuesta cada entrada.

Para ello, vamos a crear un método **actualizarPrecio** en la clase ReservaZooViewModel:

```
private fun actualizarPrecio()
{
    var precioCalculado = (numeroAdultos.value ?: 0) * PRECIO_ADULTO + (numeroNinos.value ?: 0) *
    PRECIO_NINO
    //Fecha sábado o domingo -> 1€ más por persona
    if((fecha.value?.get(Calendar.DAY_OF_WEEK) ?: -1) == Calendar.SATURDAY
        || (fecha.value?.get(Calendar.DAY_OF_WEEK) ?: -1) == Calendar.SUNDAY)
    {
        precioCalculado += ((numeroAdultos.value ?: 0) + ( numeroNinos.value ?: 0))
    }
    _precio.value = precioCalculado
}
```

El precio de un adulto es 5€ y el de niño 2€. Si es fin de semana se añade 1€ a cada persona.

Como se puede ver en el código anterior hay dos constantes. Podemos definir una constante antes de la declaración de la clase así:

```
private const val PRECIO_ADULTO = 5.0
private const val PRECIO_NINO = 2.0

class ReservaZooViewModel: ViewModel()
{
```



Hoja_UT2_11

Este método actualizarPrecio será llamado desde el método setFecha, setNumeroAdultos y setNumeroNinos, ya que en ese momento el precio variará.

También podemos crear un método **resetearReserva**:

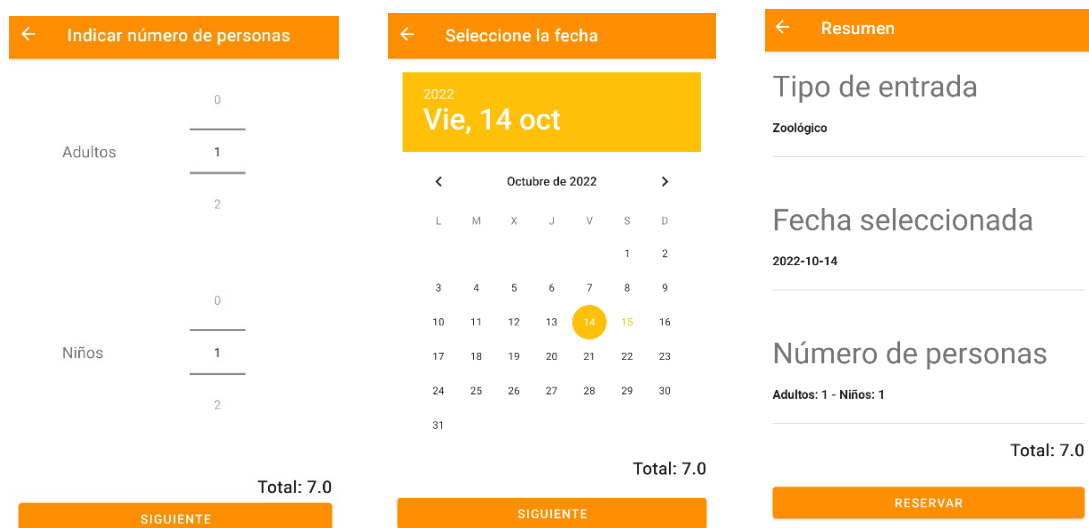
```
fun resetearReserva()
{
    _fecha.value = Calendar.getInstance()
    _numeroNinos.value = 0
    _numeroAdultos.value = 1
    _tipoReserva.value = ""
    actualizarPrecio()
}
```

Lo llamaremos al crear el viewModel con un bloque init:

```
init
{
    resetearReserva()
}
```

Y también lo llamaremos desde el evento onClick de FragmentResumen, justo cuando se realiza la reserva y se muestra el Toast.

Completar el resto de vistas para obtener la funcionalidad adecuada y los valores adecuados en la UI.



The image shows three sequential screenshots of an Android application for booking a zoo entry.

- Screen 1: Indicar número de personas** (Indicate number of people). It features two spinners: 'Adultos' (Adults) set to 1 and 'Niños' (Children) set to 1. The total price shown is 7.0. A 'SIGUIENTE' (Next) button is at the bottom.
- Screen 2: Seleccione la fecha** (Select the date). It shows a calendar for October 2022 with the 14th (Friday) selected. The total price shown is 7.0. A 'SIGUIENTE' (Next) button is at the bottom.
- Screen 3: Resumen** (Summary). It displays the booking details: 'Tipo de entrada' (Entry type) as 'Zoológico', 'Fecha seleccionada' (Selected date) as '2022-10-14', and 'Número de personas' (Number of people) as 'Adultos: 1 - Niños: 1'. The total price is 7.0. A 'RESERVAR' (Book) button is at the bottom.

NOTA: para que se modifiquen los datos en la interfaz de usuario (UI) al llamarse un evento se debe asociar **binding.lifecycleOwner** con **viewLifecycleOwner** (tal y como aparece en FragmentPersonas y FragmentFecha).

LifecycleOwner es una clase que representa el ciclo de vida de Android, por ejemplo, una actividad o un fragmento. Un observador LiveData ve los cambios en los datos de la app sólo si el propietario del ciclo de vida está en estado activo (STARTED o RESUMED).