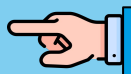
# Arquitectura en Android

## Patrones de arquitectura en Android (Parte II)

{desafío}
latam_

***Utilizar patrones de arquitectura escalables para la construcción de una aplicación Android de acuerdo a los requerimientos***

- Unidad 1: Kotlin para el desarrollo de aplicaciones.
- Unidad 2: Ciclo de vida de componentes.
- Unidad 3: Arquitectura en Android.
- Unidad 4: Programación asíncrona en Android.



***Te encuentras aquí***

{desafío} latam_

# ¿Qué aprenderás en esta sesión?

- *Implementar un aplicativo utilizando el patrón MVVM de acuerdo a las buenas prácticas recomendadas.*

# ¿Recuerdas cuáles son los patrones de diseño sugeridos para Android? Nombra al menos uno

{desafío} latam_

# /* Patrón de diseño recomendado MVVM */

# MVVM

En MVVM, la parte principal es **ViewModel**, que maneja su lógica de negocio utilizando datos en vivo y no se requiere más interfaz u otra dependencia. Usando todas las características anteriores podemos lograr los siguientes resultados:

- Facilita las pruebas unitarias
- La estructura de código limpio de la aplicación.
- Flexibilidad de desarrollo
- Separación de la lógica

# Cómo funciona MVVM

Para esto vamos a analizar una app muy simple, la cual hace uso de un ViewModel para pasar datos entre fragments. Descarga el proyecto SurveyDonkey.zip y ábrelo.

Navega hasta `StartFragment()` y veamos cómo se implementa:

```
private val viewModel: MainViewModel by activityViewModels()
```

- **viewModel:** es el nombre de la variable
- **MainViewModel:** es el ViewModel una clase que extiende de ViewModel
- by ***`activityViewModels()`***: es la delegación de propiedad de Kotlin

**TIP:** para poder *by activityViewModels()* o *by viewModels()*, agrega a build.gradle la siguiente línea de código: `implementation "androidx.fragment:fragment-ktx:XXX"`

# Funciones MVVM

Si vemos las funciones que tiene él en su interior, podemos ver que son bastante simples, por ejemplo:

```
fun addFirstAnswer(value: String): List<String>
{
    firstAnswer.add(value)
    return
firstAnswer.distinct().sorted().toList()
}
```

La funcion es muy simple, recibe un `String` y lo agrega a una lista de `Strings`, luego retorna la lista de `Strings` sin repetir elementos y ordenados alfabéticamente

Si queremos que las funciones sean visibles desde Fragments o Activities, entonces **NO** debemos declararlas `private`

# Cómo podemos hacer uso de las funciones que definimos en el ViewModel

Tomemos la función del ejemplo anterior y veamos en qué parte del `FirstQuestionFragment()` se ocupa:

```
answer11.setOnCheckedChangeListener { _, checked ->
    if (checked) viewModel.addFirstAnswer(answer11.text.toString())
    else viewModel.removeFirstAnswer(answer11.text.toString())
}
```

Si el checkbox **`answer11`** es seleccionado, entonces, toma el valor String del checkbox, es decir el nombre, y lo pasa a la función **`addFirstAnswer`**

# Ciclo de Vida del ViewModel

El ciclo de vida de un ViewModel está ligado directamente a su alcance o scope.

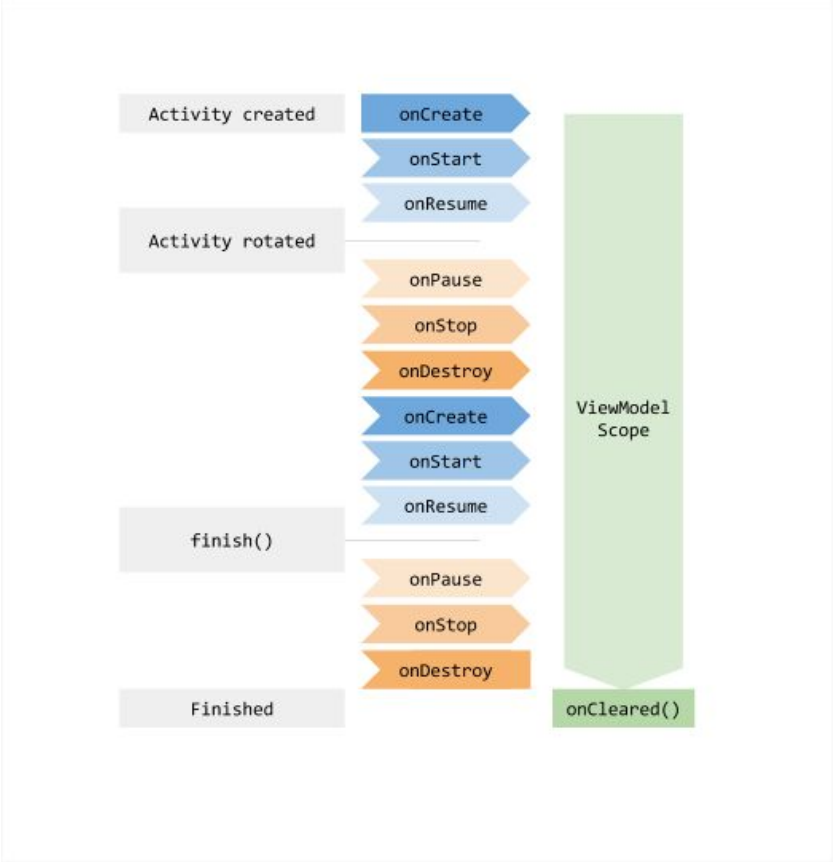
Un ViewModel permanece en la memoria hasta que el `ViewModelStoreOwner` al que pertenece desaparece. Esto puede ocurrir en los siguientes contextos:

- En el caso de una **actividad**, cuando finaliza.
- En el caso de un **fragmento**, cuando se desprende.
- En el caso de una entrada de **Navegación**, cuando se elimina de la pila posterior.

Esto convierte a ViewModels en una excelente solución para almacenar datos que sobreviven a los cambios de configuración.

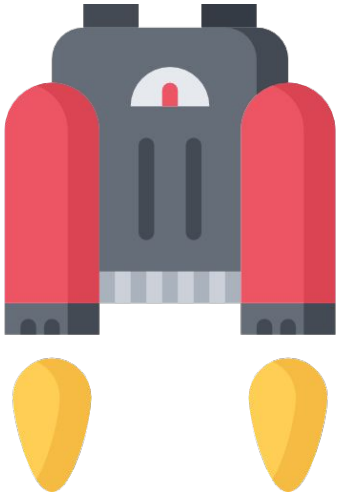
# Ciclo de Vida del ViewModel

La siguiente figura ilustra los diversos estados del ciclo de vida de un Activity a medida que pasa por una rotación y luego finaliza.
La ilustración también muestra la vida útil de ViewModel junto al ciclo de vida de la actividad asociada. Este diagrama particular ilustra los estados de una actividad. Los mismos estados básicos se aplican al ciclo de vida de un fragment.



# /* Jetpack (all) */

# Jetpack (all)



- Jetpack es un conjunto de bibliotecas para ayudar a los desarrolladores a seguir las mejores prácticas, reducir el código repetitivo y escribir código que funcione de manera uniforme en todas las versiones y dispositivos de Android para que los desarrolladores puedan concentrarse en el código que les interesa.

- La guía de arquitectura de aplicaciones de Jetpack brinda una descripción general de las mejores prácticas y la arquitectura recomendada para tener en cuenta al crear su aplicación de Android.

- Las siguientes secciones cubren cómo puede comenzar con los componentes de Jetpack.

# Jetpack (all)

| Nombre | Función |
|---|---|
| **Activity** | Accede a las APIs que admiten composición compiladas sobre Activity. |
| **Appcompat** | Permite el acceso a nuevas API en versiones de API anteriores de la plataforma (muchas con Material Design). |
| **Appsearch** | Ofrece a los usuarios búsquedas personalizadas en la app. |
| **Camera** | Compila apps para cámaras móviles. |
| **Compose** | Define tu IU de manera programática con funciones de componibilidad y que describan su forma y sus dependencias de datos. |
| **Databinding** | Usa un formato declarativo para vincular los componentes de la IU en tus diseños con las fuentes de datos de tu app. |
| **Fragment** | Segmenta tu app en varias pantallas independientes alojadas en un objeto Activity. |

# Jetpack (all)

| Nombre | Función |
|---|---|
| **Hilt** | Extiende la funcionalidad de Dagger Hilt para habilitar la inserción de dependencias de ciertas clases de las bibliotecas de AndroidX. |
| **Lifecycle** | Compila componentes optimizados para ciclos de vida que puedan ajustar el comportamiento según el estado actual del ciclo de vida de una actividad o un fragmento. |
| **Material Design Components** | Son componentes de IU de Material Design modulares y personalizables para Android. |
| **Media3** | Admite bibliotecas para casos de uso de contenido multimedia. |
| **Navigation** | Compila y estructura la IU integrada en la app, controla los vínculos directos y navega entre pantallas. |
| **Paging** | Carga datos en páginas y los presenta en un RecyclerView. |

# Jetpack (all)

| Nombre | Función |
|---|---|
| **Room** | Crea, almacena y administra datos persistentes con copia de seguridad en una base de datos SQLite. |
| **Test** | Pruebas en Android |
| **Viewpager2** | Muestra objetos de Views o Fragments en formato deslizable. |
| **Webkit** | Trabaja con las API modernas de WebView en Android 5 y versiones posteriores. |
| **Work** | Programa y ejecuta tareas en segundo plano diferibles y basadas en restricciones |

# /* Jetpack (Compose) */

# Jetpack (Compose)

Jetpack Compose es un conjunto de herramientas modernas que nos permite construir nuestras vistas con un enfoque **declarativo** escribiendo menos código.

Antes de Jetpack Compose, usábamos diseños XML para crear la interfaz de usuario nativa. Teníamos una estructura muy dependiente cuando construimos las vistas con XML. Además, los fragmentos son componentes muy pesados para la interfaz de usuario.

# Jetpack (Compose)

- Jetpack Compose es el conjunto de herramientas moderno de Android para crear una interfaz de usuario nativa.
- Simplifica y acelera el desarrollo de la interfaz de usuario en Android, dando vida a sus aplicaciones con menos código, herramientas potentes y API de Kotlin intuitivas.
- Hace que la creación de la interfaz de usuario de Android sea más rápida y sencilla.

# Ventajas del uso de Jetpack (Compose)

**Menos código**

Escribir menos código afecta todas las etapas del desarrollo: como autor, puede concentrarse en el problema en cuestión, con menos para probar y depurar y con menos posibilidades de errores; como revisor o mantenedor, tiene menos código para leer, comprender, revisar y mantener.

Compose le permite hacer más con menos código, en comparación con el uso del sistema Android View: Botones, listas o animaciones: lo que sea que necesite crear, ahora hay menos código para escribir.

**Intuitivo**

Compose usa una **API declarativa**, lo que significa que todo lo que necesita hacer es describir su interfaz de usuario y Compose se encarga del resto. Las API son intuitivas, fáciles de descubrir y usa

Con Compose, crea componentes pequeños sin estado que no están vinculados a una actividad o fragmento específico. Eso los hace fáciles de reutilizar y probar

El estado es explícito y se pasa al composable. De esa manera, hay una única fuente de verdad para el estado, haciéndolo encapsulado y desacoplado. Luego, a medida que cambia el estado de la aplicación, su interfaz de usuario se actualiza automáticamente.

**Acelerar el desarrollo**

Compose es compatible con todo su código existente: puedes llamar al código Compose desde Vistas y Vistas desde Compose. Las bibliotecas más comunes, como las corrutinas Navigation, ViewModel y Kotlin, funcionan con Compose, por lo que puede comenzar a adoptar cuando y donde quiera.

Con el soporte completo de Android Studio, con características como vistas previas en vivo, puede iterar y enviar el código más rápido

**Poderoso**

Compose le permite crear hermosas aplicaciones con acceso directo a las API de la plataforma Android y soporte integrado para Material Design, tema oscuro, animaciones y más.

Con Compose, dar movimiento y vida a sus aplicaciones a través de animaciones es rápido y fácil de implementar.

Ya sea que construyas con Material Design o con su propio sistema de diseño, Compose le brinda la flexibilidad de implementar el diseño que desee.

# RESUMEN

- **Jetpack** es un conjunto de librerías desarrolladas especialmente para Android por Google, recuerda consultar estas librerías como primera opción cuando andes en busca de librerías.
- **Jetpack Compose** ayuda a desarrollar código de forma concisa y rápida, recuerda seguir las prácticas recomendadas para así poder aprovechar todo su potencial.
- Se diseñó la clase **ViewModel** a fin de almacenar y administrar datos relacionados con la IU de manera optimizada para los ciclos de vida.
- La clase **ViewModel** permite que se conserven los datos luego de cambios de configuración, como las rotaciones de pantallas.

# Próxima sesión...

- *Continuaremos implementando un aplicativo utilizando el patrón MVVM de acuerdo a las buenas prácticas recomendadas.*

{desafío}
latam_

Academia de
talentos digitales