



Curso de

Patrones de Diseño en Android

Cristian Jiovany Villamil

Cristian Villamil

@jiovanyvillamil

- Desarrollador Android
- Experiencia con apps de uso masivo
- Coach de certificaciones Android
- Associate Android Developer



Ganador Hackathones



Ganador Hackathones





Prerrequisitos del curso

- Programación orientada a objetos
- Creación de vistas
- Android Studio
- Kotlin
- Manejo de asincronía

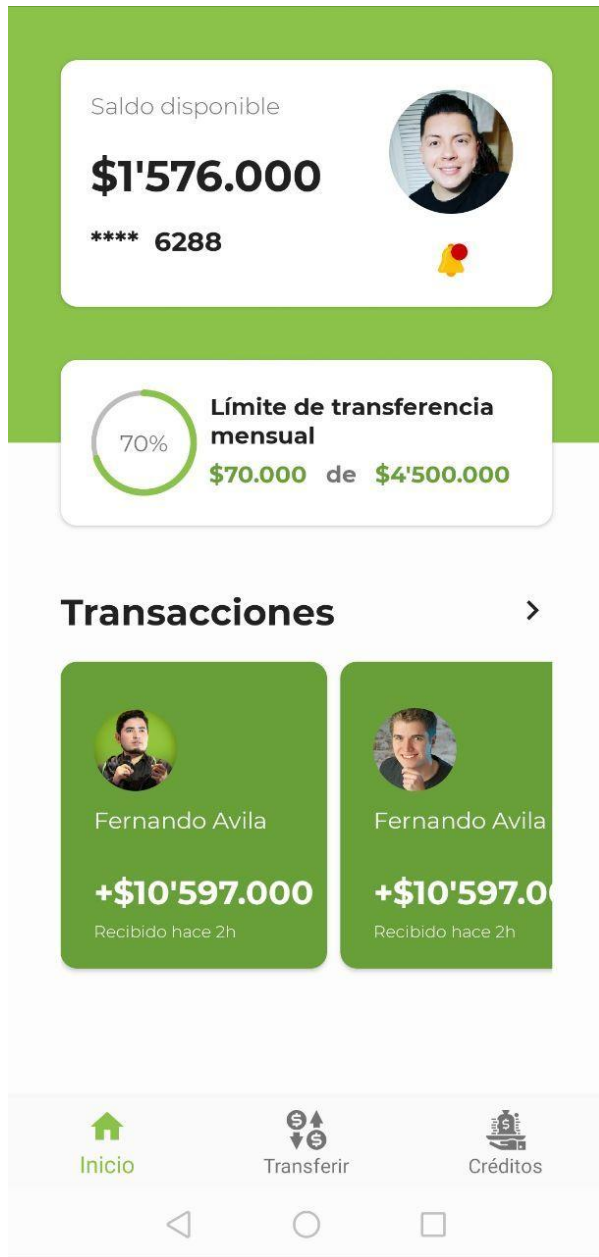


Lo que aprenderemos en este curso

- Identificar problemas donde podamos aplicar los patrones de diseño.
- Repaso por la arquitectura de aplicaciones móviles.
- Introducción a Architecture Components.

PlatziWallet

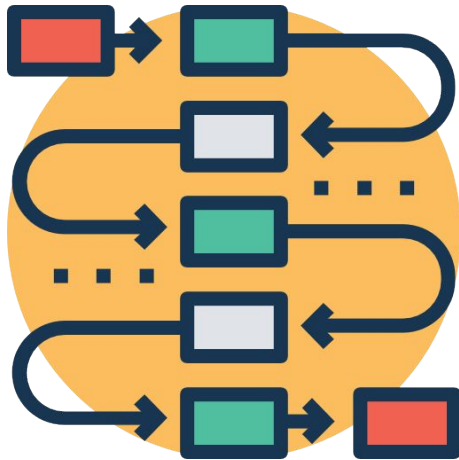
- App para transferir dinero
- Involucra sistemas complejos y niveles de seguridad.
- IMPORTANTE: los patrones de diseño no involucran cambios de vistas sino cambios estructurales.



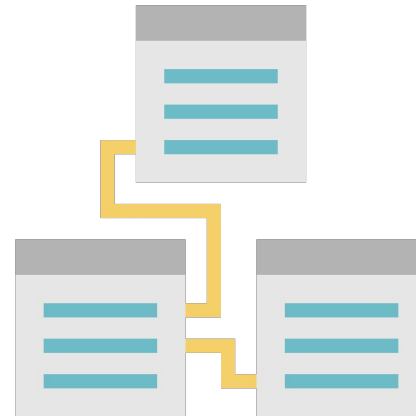
¿Qué es Arquitectura?

Arquitectura

La definimos como



**Organización y definición
de componentes**



Comunicación

Como en la vida real

La arquitectura nos brinda formas de crear cosas cada vez más estables y escalables como los edificios.





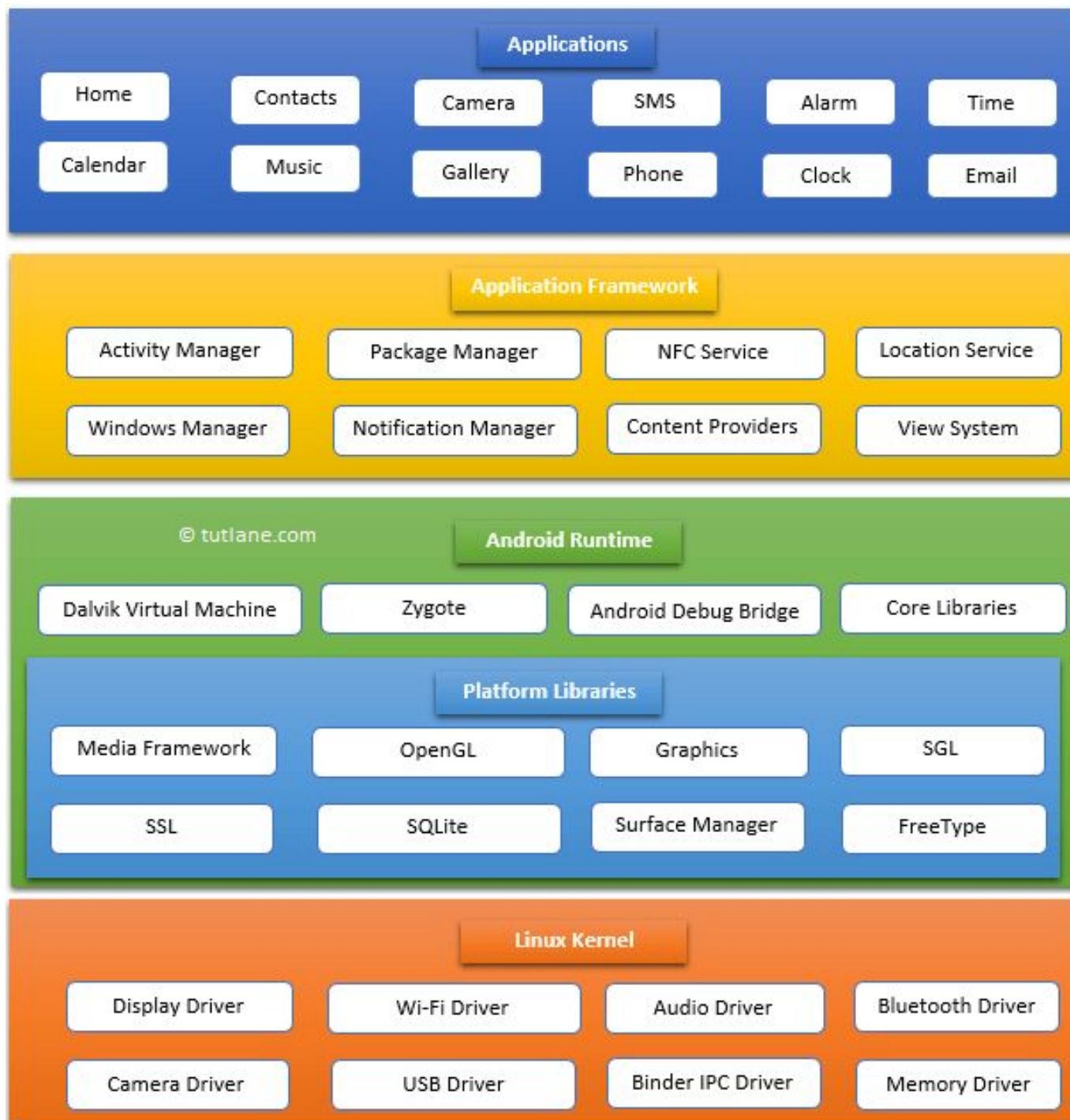
A tener en cuenta...

Cada decisión que se toma definiendo la arquitectura debe ser orientada al tipo de negocio en el que trabajemos.

Si tomamos una mala decisión, va a repercutir en todo el desarrollo de la aplicación.

Arquitectura Android





Tipos de Arquitectura en Android

Arquitectura

¿Qué es un Patrón de Arquitectura?

Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto.

Los patrones arquitectónicos son similares al patrón de diseño de software pero tienen un alcance más general.

Nos ayudan a:



Desacoplar las diferentes capas; vista, lógica de negocio, conexión.



Hacer pruebas aisladas de cada componente



Arquitecturas en Android

- **MVP**

Modelo - Vista - Presentador

- **MVVM**

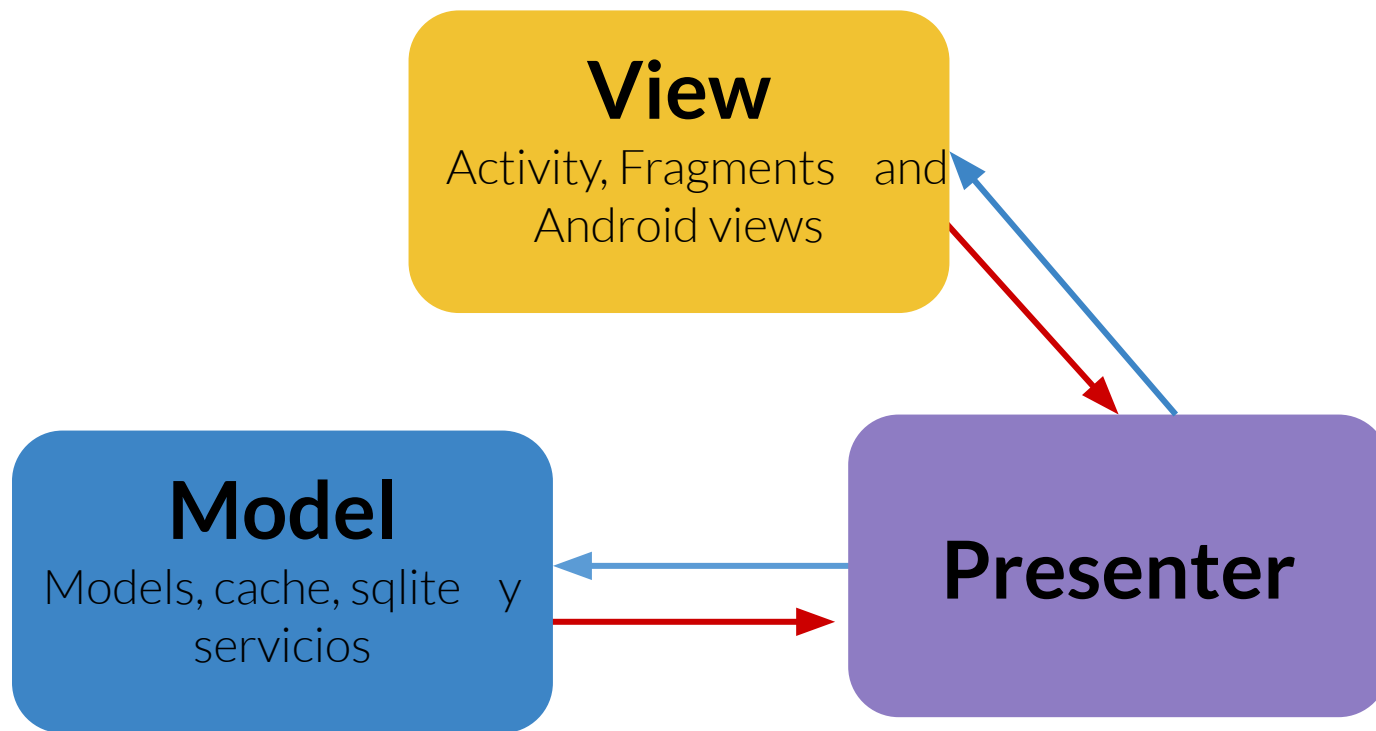
Modelo - Vista - Vista Modelo




Model View Presenter

Arquitectura

MVP





View (Vista)

- Su responsabilidad es simplemente mostrar componentes visuales y comportamientos visuales.
- Todos los eventos de la vista se comunican al presentador a través de un contrato (interfaz).

Vamos a crear el contrato de la vista

```
public interface LoansView {  
    fun showLoader()  
    fun hideLoader()  
    fun showLoansList(loansList : List<Loan>)  
    fun showTotalLoansProgress(progress: Int)  
}
```

```
public LoansFragment : Fragment, LoansView {  
    @Override  
    fun showLoader() {  
        ...  
    }  
    .  
    .  
    .  
}
```



Presenter (Presentador)

- Se encarga de ser el intermediario entre la vista y las capas de comunicación (red, BD).
- Su responsabilidad se basa en modelar los datos y enviar eventos.
- No tiene contexto de la vista.

Vamos a crear el contrato del presentador

```
public interface LoansPresenter {  
    fun getLoans()  
    .  
    .  
    .  
}
```

```
public LoansPresenter : Presenter {  
    @Override  
    fun getLoans() {  
        ...  
    }  
    .  
    .  
    .  
}
```



Model (Interactor)

- Tiene toda la comunicación de la capa de red y de base de datos.
- Tiene los modelos de request y response.
- No tiene contexto de la vista.
- Se comunica a través de Callbacks.



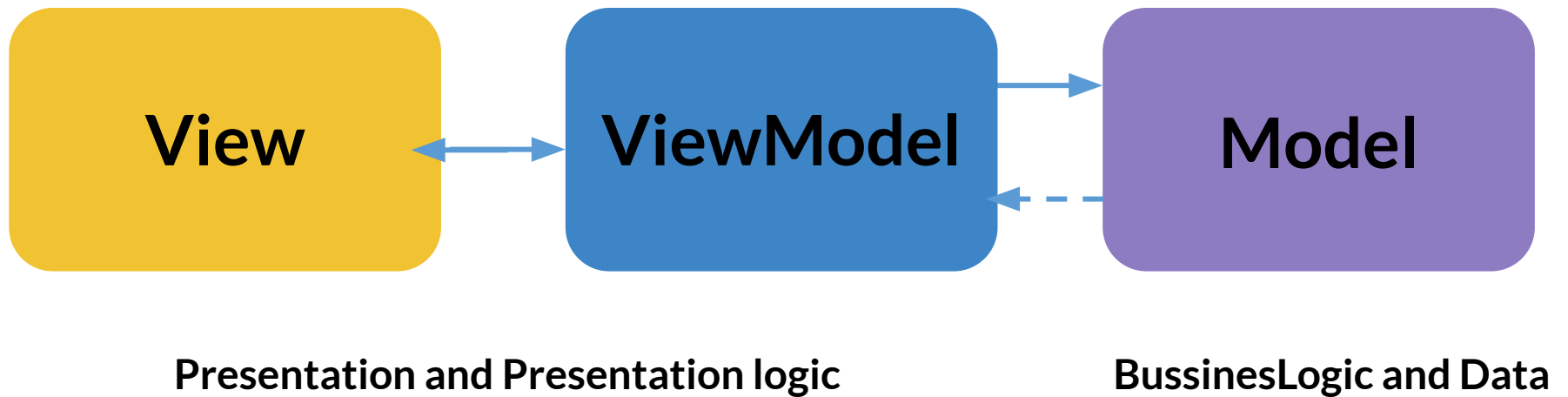
Let's code



Model View ViewModel

Arquitectura

MVVM





Diferencias con MVP

- La vista se suscribe a los cambios que genera el view model, por lo tanto, no necesita contratos.
- Se comunica a la vista por medio de data binding o LiveData.



Patrones de Diseño

Patrones de Diseño

¿Qué son los Patrones de Diseño?

Son plantillas para soluciones a problemas comunes en el desarrollo de software que se pueden usar en diferentes contextos.

Nos dan soluciones fáciles a problemas complejos sin importar el lenguaje que estemos usando.

Patrones de Diseño Creacionales

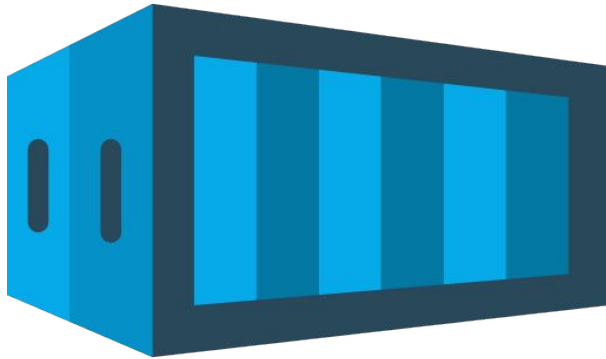
Patrones de Diseño



Patrones Creacionales

Nos ayudan a delegar la responsabilidad de creación de los objetos, creándolos para las situaciones necesarias.

Tienen dos pilares



Encapsular el
conocimiento de las
clases



Ocultar cómo se crean y
se instancian



¿Cuáles son?

- Singleton
- Builder
- Factory



Patrones Estructurales

Patrones de Diseño



Patrones Estructurales

Se enfocan en cómo las clases y objetos se componen para formar estructuras mayores.

- Adapter
- Proxy
- Facade

Patrones de Comportamiento

Patrones de Diseño

Patrones de Comportamiento

Nos indica cómo es la comunicación entre objetos y cómo lo podemos desacoplar y delegar responsabilidades.

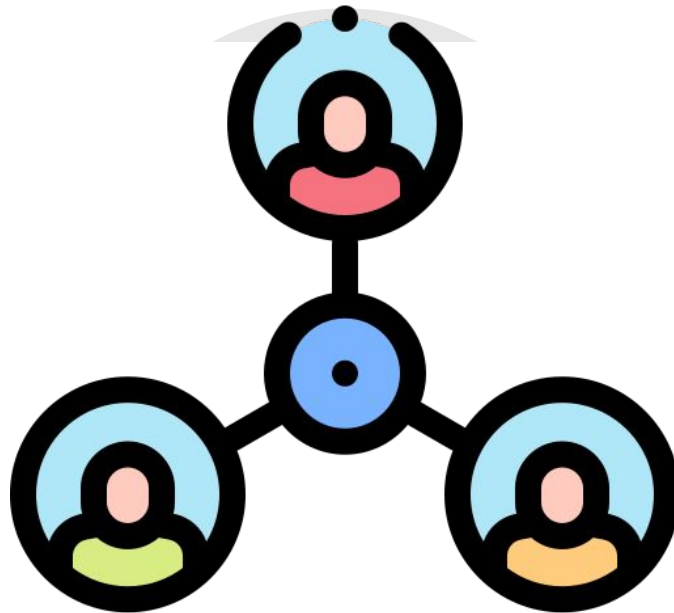
- Observer
- Command



Singleton

Patrones Creacionales

Singleton



Una única instancia para toda la aplicación



¿Cuándo usarlo?

- Cuando queremos datos transversales a toda la aplicación.
- Datos que son útiles para múltiples flujos.
- Para tener acceso global a los datos.



Beneficios

- Una única instancia de los datos transversales.
- Acceso desde cualquier lugar.
- Se crea la instancia solamente cuando se va a usar.



Let's code



Builder

Patrón de Diseño Creacional

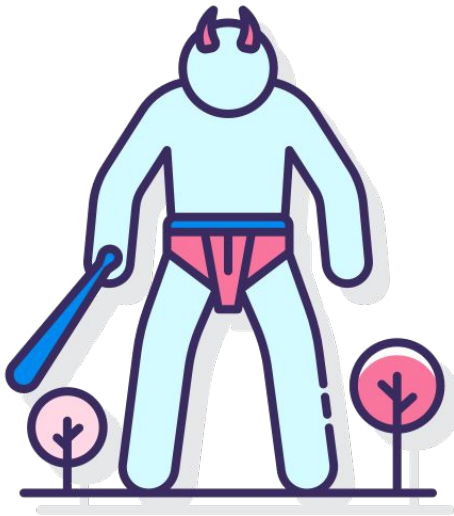


Builder

Patrón de Diseño Creacional que nos ayuda a crear objetos complejos de una manera sencilla, legible y escalable.

Con el mismo proceso de construcción de una clase podremos crear diferentes representaciones de nuestra clase.

Beneficios



Evita constructores gigantes



Podemos tener parámetros opcionales

El AlertDialog es un ejemplo claro

AlertDialog usa el **patrón Builder** debido a que tenemos muchos parámetros y además unos son opcionales.

Hola Builder

Aquí mostramos el mensaje del dialog.

NO GRACIAS

OK

Aquí un ejemplo de como crear un AlertDialog desde un Activity por medio de un **Builder**.

```
AlertDialog.Builder(this)
    .setTitle("Hola Builder")
    .setMessage("Aquí mostramos el mensaje del dialog.")
    .setNegativeButton("No Gracias", { dialogInterface, i ->
        // Línea de código para el click del botón negativo
    })
    .setPositiveButton("OK", { dialogInterface, i ->
        // Línea de código para el click del botón positivo
    })
    .show()
```




Let's code



Factory

Patrón de Diseño Creacional



Factory

Nos ayuda a tener instancias de un objeto dado el tipo.

Tenemos la ventaja de que la responsabilidad de creación de las clases se delega al Factory y lo usamos a base de abstracciones (interfaces).

Beneficios



- Sabemos qué solicitar pero no los detalles de su implementación.
- Si hay un nuevo tipo, es fácil escalar en la factoría.



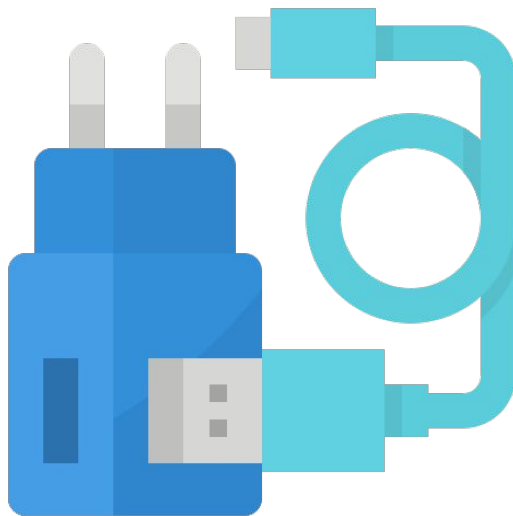
Let's code



Adapter

Patrones Estructurales

Adapter



Cuando dos clases no se entienden, el adapter es el mediador y adapta una clase para que la otra la entienda.



¿Cuándo usarlo?

Nuestro sistema se debe comunicar de varias formas con entes externos, entonces, la comunicación con los entes debe hablarse en términos de **adaptador**. Con eso, si el sistema cambia, el adaptador modifica el contenido para que siempre lo entendamos.



Let's code

Proxy

Patrones Estructurales

Proxy



Es un elemento que se encarga de introducir un nivel de acceso a una clase. Ese nivel de acceso puede ser por seguridad o por complejidad.



¿Cuándo usarlo?

Si necesitamos que una clase tenga un acceso restringido, o simplemente tenga unas validaciones previas, con el proxy podemos exponer los métodos necesarios y hacer las validaciones antes de que llegue al objeto final.

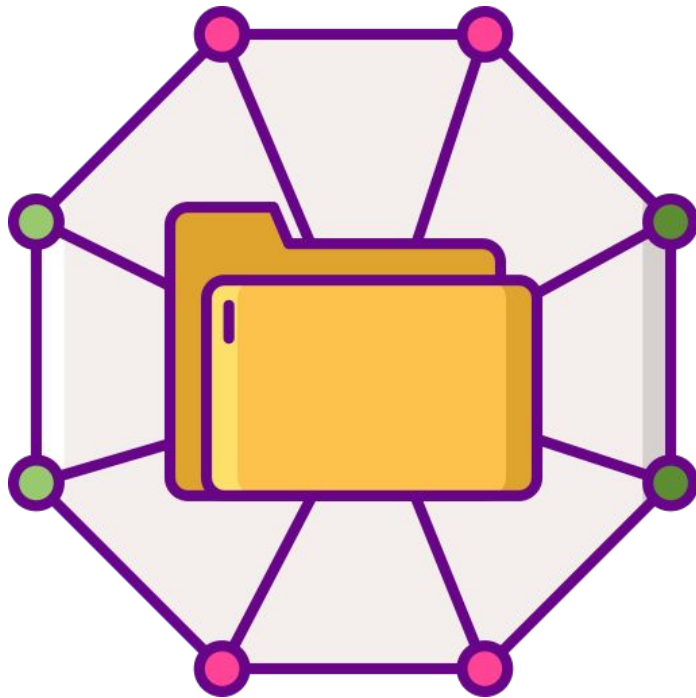


Let's code

Facade

Patrones Estructurales

Facade



Es una clase que encapsula la complejidad de varios subsistemas y los proporciona de una manera sencilla.



¿Cuándo usarlo?

En el caso que tengamos que hacer una operación que involucra varios sistemas, le delegamos esa responsabilidad a un **facade**, y él se encarga de manejar los subsistemas y exponernos un método sencillo para usarlos.



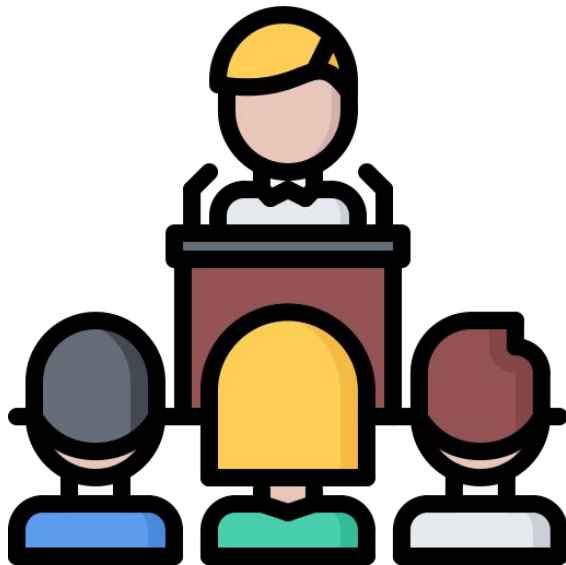
Let's code



Observer

Patrones de Comportamiento

Observer



Es un Patrón de Diseño que nos sirve para subscribirnos a unos eventos de una clase y obtener notificaciones.



¿Cuándo usarlo?

Cuando varias vistas dependen de un dato, podríamos subscribirnos a los eventos de ese dato, y cuando cambie, actualizamos todas las vistas de manera reactiva.



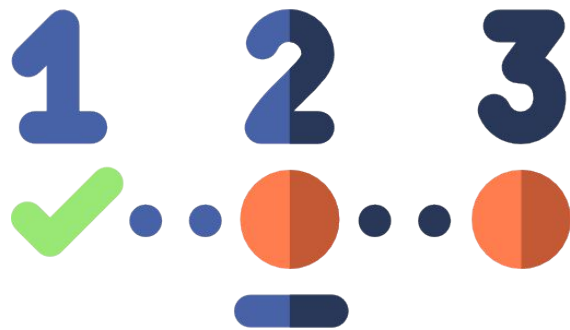
Let's code



Command

Patrones de Comportamiento

Command



Separa acciones que pueden ser ejecutadas desde varios puntos diferentes de la aplicación a través de una interfaz sencilla.

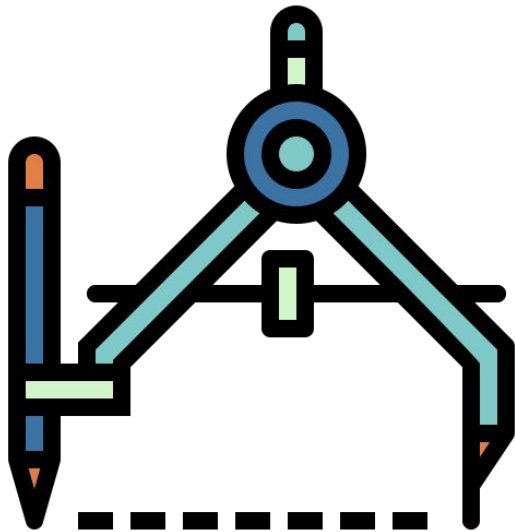
¿Cuándo usarlo?

Si hay fragmentos de código que se ejecutan, o son partes de una serie de ejecuciones y se repiten, los encapsulamos en un comando y van a poder ser ejecutados o cambiados sin impactar el código.

Introducción a Architecture Components

Architecture Components

Architecture Components



Son un conjunto de herramientas y librerías que nos ayudan a construir aplicaciones escalables y robustas solucionando los principales problemas.



¿Cuáles son?

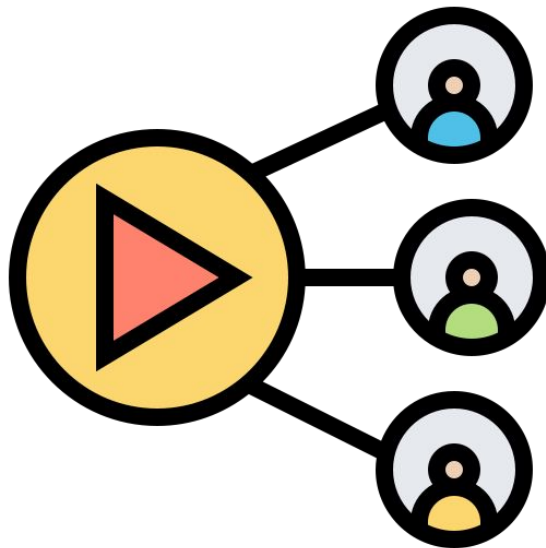
- ViewModel
- Room
- LiveData
- Navigation
- Paging
- WorkManager



LiveData

Architecture Components

LiveData



Es un Observable que nos ayuda con la solución del problema del ciclo de vida de las activities.



Room

Architecture Components

Room



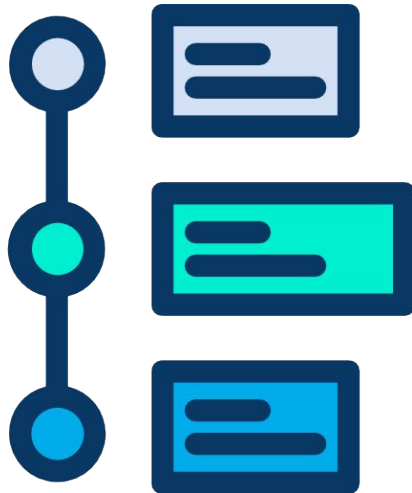
Es un ORM(Object Relational Mapping), que utiliza el motor de SQLite. Se integra perfectamente con LiveData.



ViewModel

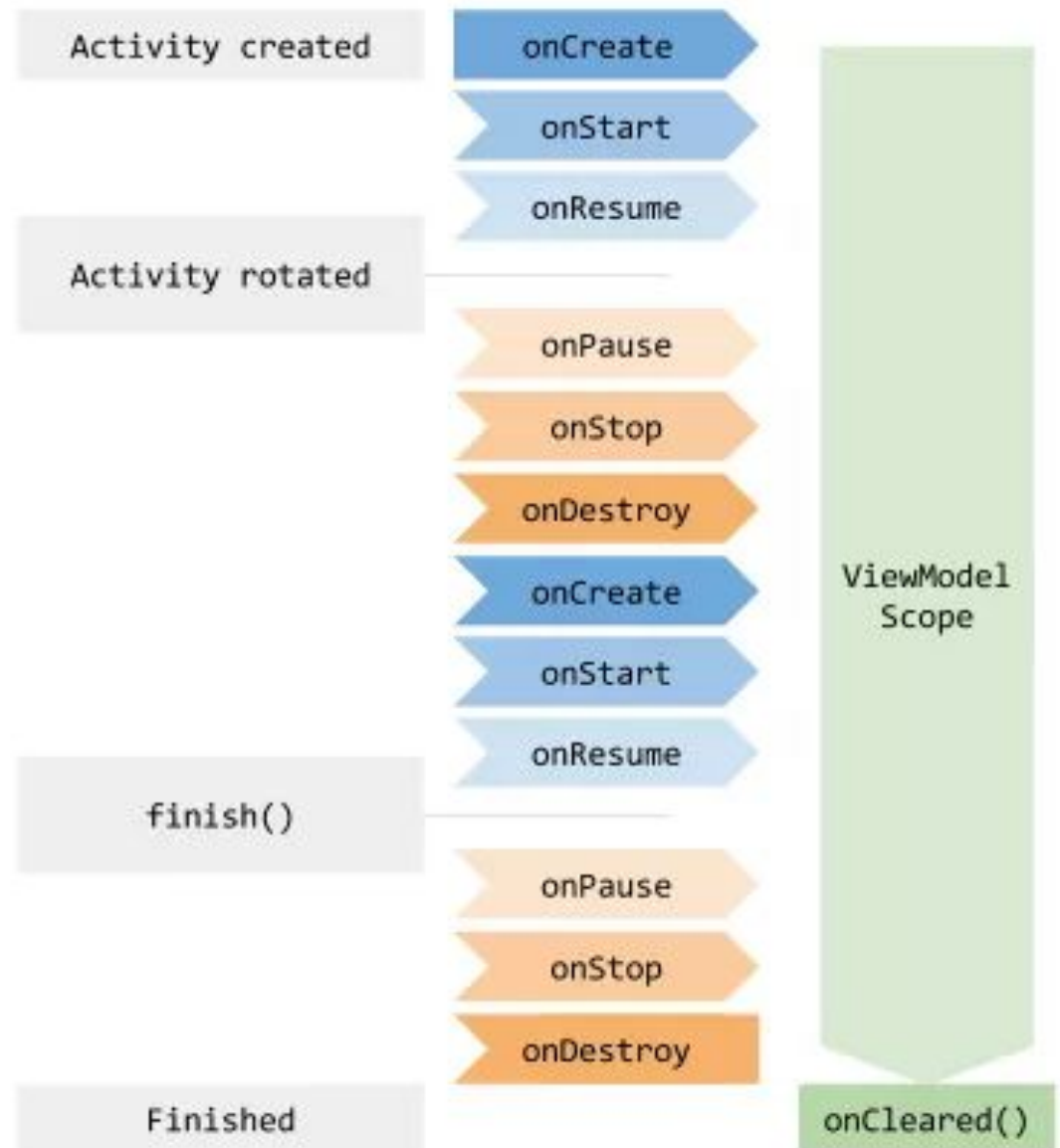
Architecture Components

ViewModel

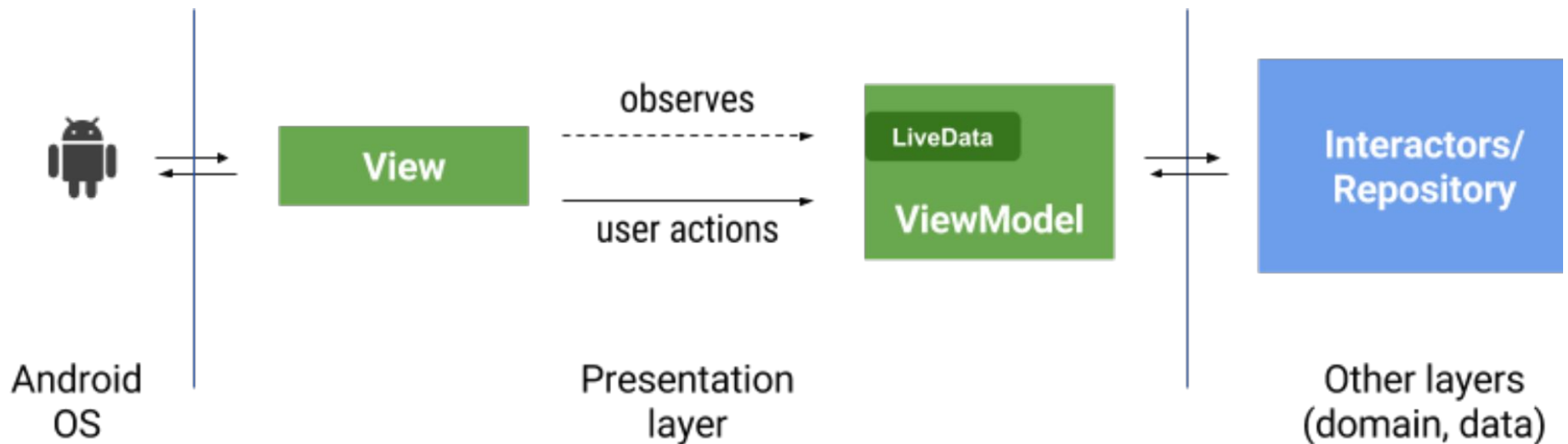


Es una clase que nos ayuda a manejar el ciclo de vida de un contexto, dándonos sólo un método (`onCleared()`) donde deberíamos liberar recursos.

ViewModel



ViewModel

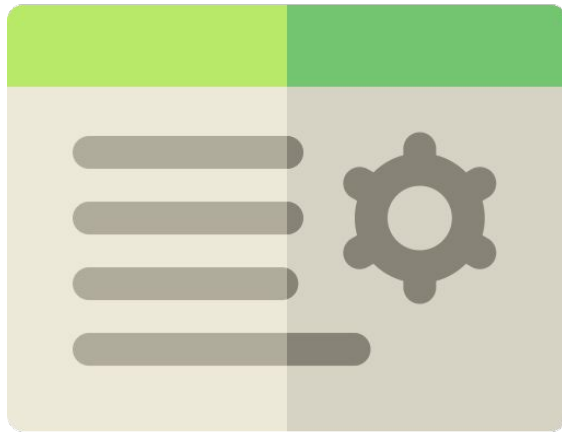




CustomViews

Bonus

Custom Views



Heredamos de un View y le damos un comportamiento personalizado en base a nuestros requerimientos visuales.



¡Felicitaciones!

Conclusiones



Aprendiste

- Los patrones de diseño más usados en Android
- Los aplicaste en el proyecto PlatziWallet
- Comenzaste con Architecture Components, el complemento perfecto de los patrones de diseño.



Tips para seguir aprendiendo

- Mira constantemente el canal de Youtube Android Developers
- Seguir en Twitter a Jeroen Mols y Jake Wharton
- Seguir en Medium a Android Pub
- Escuchar el Podcast Android Dev
- Busca la comunidad de Android más cercana.

**¡Nunca pares de
Aprender!**