

{desafío}
latam_

LiveData and UI _

Parte I



Patrón Observador y Ciclos de vida de una actividad y fragmento

El patrón de diseño Observador

El Patrón observador está calificado como un patrón de comportamiento, este tipo de patrón se encarga principalmente en la comunicación entre clases y objetos en un programa.

El patrón observer está recomendado para cuando necesitamos reaccionar a los cambios que ocurren a un objeto, pero sin la necesidad de que estemos constantemente preguntando por el estado de ese objeto. El objeto se encarga de avisar cuando ocurre un cambio en su estado, notificando a todos los que están observando esos cambios.

Un problema que resuelve el patrón observador:

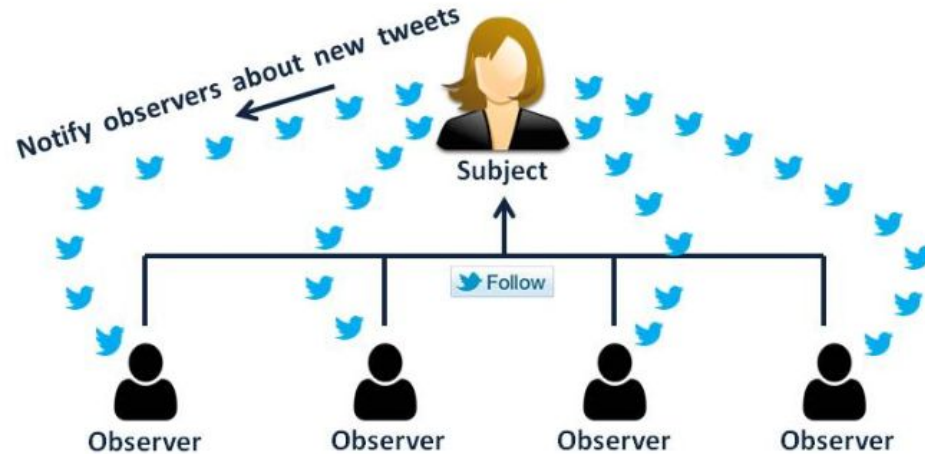
Interfaces gráficas

1. La interfaz reacciona a los cambios, no pregunta por ellos
2. El objeto que cambia debe avisar a los interesados cada vez que cambia
3. Cada vez que los interesados son notificados de los cambios, se procede a actualizar la interfaz que esperaba esos cambios.
4. La relación entre un Sujeto y sus observadores es 1 - N, es de dice uno a muchos.

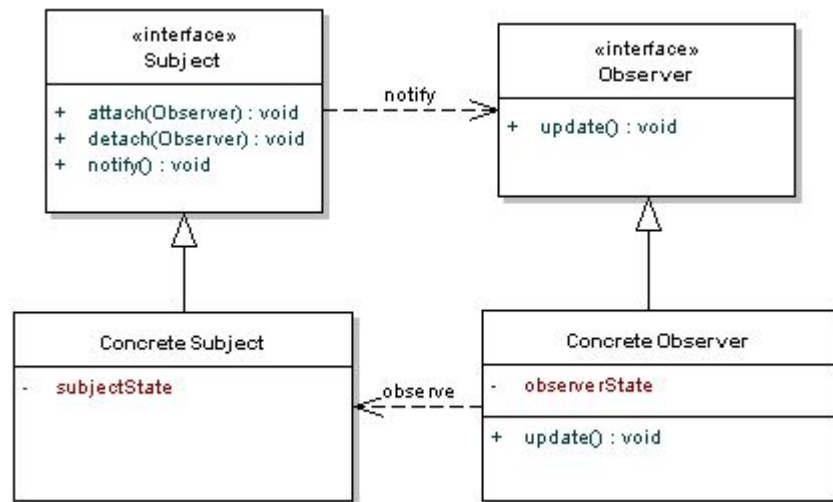
Un problema que resuelve el patrón observador:

Interfaces gráficas

Observer Design Pattern

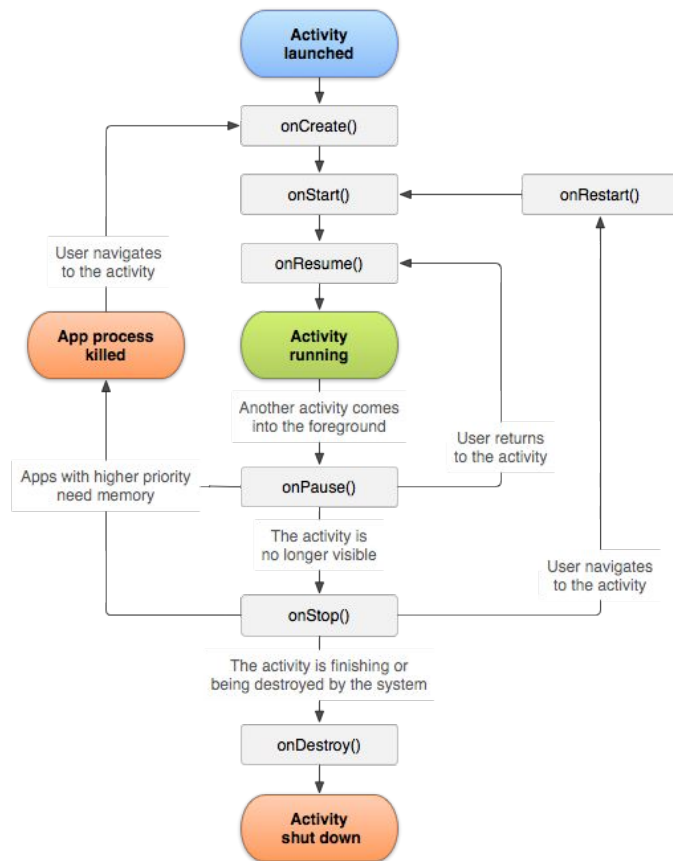


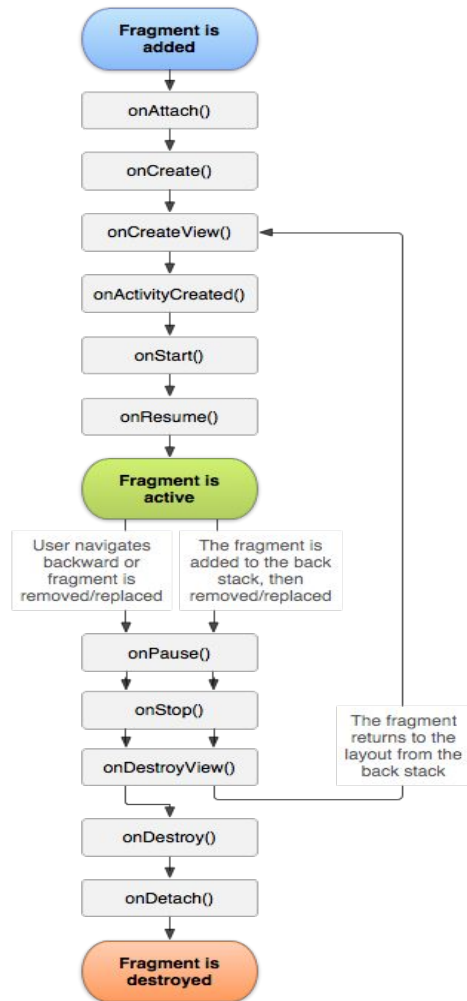
Diagrama



Ciclo de vida de una Actividad y Fragmento.

Sabemos que una actividad es un punto de partida en una aplicación Android, también es asociada con una representación de una vista cuando se asocia a un Layout. También podemos afirmar que una actividad tiene distintos estados y siempre está asociada a un ciclo de vida el cual llama a diferentes métodos cuando va cambiado de uno a otro, nosotros incluso podemos aprovechar estos métodos para ejecutar código que beneficie las funciones de nuestra aplicación.





LifeCicle-Aware components

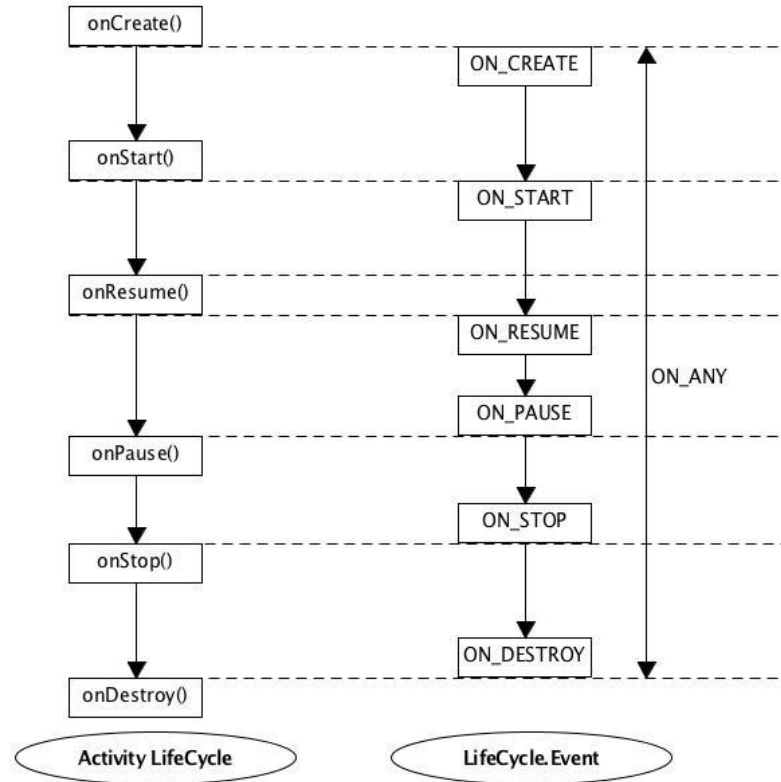
Gracias a los nuevos componentes de arquitectura ya no tenemos que preocuparnos del manejo de los ciclos de vida como hace algunos años, debido a que elementos como LifeCycle classes se crearon para que los componentes puedan manejar el ciclo de vida en el cual están inmersos y sobrevivir a los cambios de configuración sin tener que realizar tareas extras por nuestra parte.

Lifecycle

Es una clase que mantiene información acerca del estado del ciclo de vida de un componente, una actividad o un fragmento. Esta clase ocupa dos enumeraciones principales para mantener un registro de estado del ciclo de vida del componente asociado a la clase.

- **Event:** Los eventos del ciclo de vida que son despachados desde el framework de Android y la clase Lifecycle. Estos eventos mapean los callbacks del ciclo de vida de las actividades y los fragments.
- **State:** El estado actual del componente que es trackeado por el objeto Lifecycle.

Diagrama Lifecycle



Lifecycle Observer y Lifecycle Registry

Un Lifecycle owner es cualquier componente que implementa la interfaz LifecycleOwner, esta interfaz indica que tiene un ciclo de vida en Android. Los Fragmentos y las Actividades implementan esta interfaz desde la librería de soporte 26.1.0.

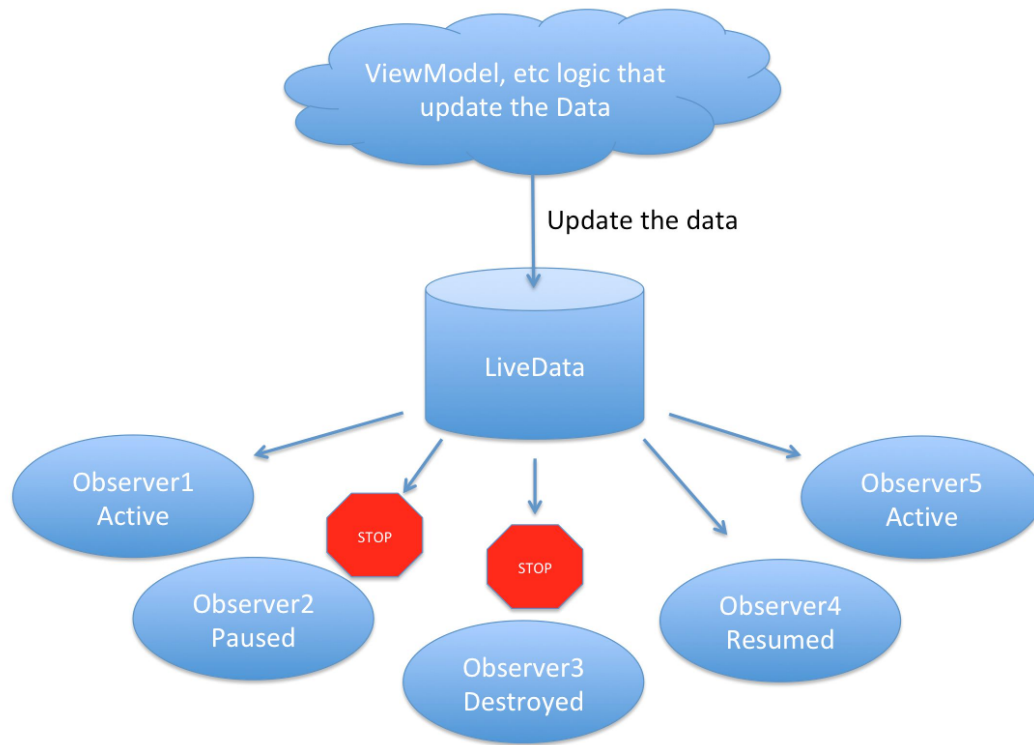
Se pueden crear Lifecycle Owners custom, implementando la interfaz LifecycleRegistry que le permite a el componente custom manejar múltiples observers.

Un Lifecycle Observer es un componente que observa los diferentes estados asociados a un LifecycleOwner, reaccionando a los diferentes cambios.

LiveData y ViewModel

LiveData

Según la documentación oficial [LiveData](#) es una clase contenedora de datos observables. Además indica que LiveData al contrario de otros observables es consciente de los ciclos de vida, lo que quiere decir que respeta los ciclos de vida de otros componentes como por ejemplo Actividades, Fragmentos o servicios.

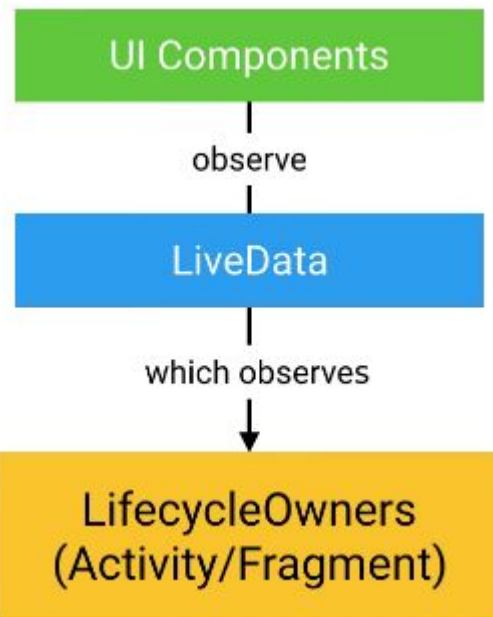


Ventajas de Usar Live Data

1. Asegura que la Interfaz de Usuario, UI, esté siempre actualizada.
2. No más memory leaks.
3. No más caídas debido a actividades que son detenidas.
4. No hay necesidad de manejar los ciclos de vida a mano.
5. Datos siempre actualizados.
6. Manejo de cambios de configuración.

Utilizar LiveData objects

1. Crea una instancia de LiveData para mantener algún tipo de dato, regularmente esto se realizará al interior de una clase ViewModel.
2. Crea un objeto Observador que defina cuando se ejecuta un método `onChange()`, el cual controla que ocurre con el objeto LiveData cuando se produce un cambio. Regularmente el objeto observador lo crearemos en algún elemento de UI, ya sea actividad o fragmento.
3. Luego debes unir al objeto observador con el de LiveData usando el método **`observe()`**, Esto realiza la subscripción del objeto observador al objeto LiveData para que los cambios puedan ser notificados. Regularmente el objeto observador será atado a una actividad o fragmento.



Crear Objetos Live Data

```
class NameViewModel : ViewModel() {  
    // Create a LiveData with a String  
    val currentName: MutableLiveData<String> by lazy {  
        MutableLiveData<String>()  
    }  
  
    // Rest of the ViewModel...  
}
```

Observar Objetos Live Data

```
class NameActivity : AppCompatActivity() {  
  
    private lateinit var model: NameViewModel  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        // Other code to setup the activity...  
  
        // Get the ViewModel.  
        model = ViewModelProviders.of(this).get(NameViewModel::class.java)  
  
        // Create the observer which updates the UI.  
        val nameObserver = Observer<String> { newName ->  
            // Update the UI, in this case, a TextView.  
            nameTextView.text = newName  
        }  
  
        // Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.  
        model.currentName.observe(this, nameObserver)  
    }  
}
```

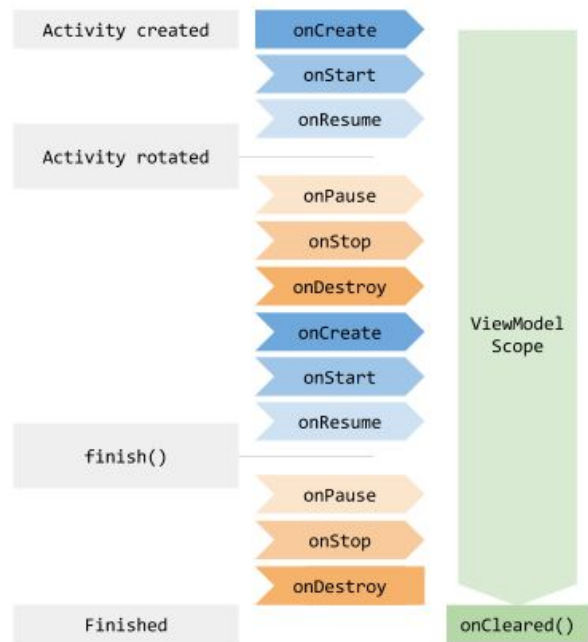
Actualizar Objetos Live Data (Mutable Live Data)

LiveData no tiene métodos públicos para actualizar los datos almacenados. Para esto, la clase MutableLiveData tiene a nuestra disposición los métodos setValue(T) y postValue(T), por lo tanto debemos usarla si necesitamos cambiar los datos almacenados en un objeto LiveData.

Usualmente MutableLiveData es usado en el ViewModel y el ViewModel solo expone objetos inmutables Live Data a los observadores. Para esto se utiliza la buena práctica de exponer estos objetos con un getter hacia la vista, esto ayudará a limitar la modificación de ese objeto. Esto ocurre de esta forma porque Live Data no tiene implementado los métodos setValue y postValue. Entonces si necesitáramos que se modifiquen, es mejor añadir una función que lo actualice

ViewModel

[ViewModel](#) también es un nuevo elemento presente en los componentes de arquitectura, se define como una clase llamada ViewModel, la cual es la responsable de preparar la data para ser mostrada en la UI o vista, esta clase está optimizada para respetar los ciclos de vida. Esto quiere decir que no importa que se produzca alguna rotación del dispositivo o cambio de configuración, los datos estarán disponibles para la actividad o fragmento objetivo.



Pasar datos entre fragmentos con ViewModel

En el desarrollo de aplicaciones, existen varias formas de pasar datos entre Fragmentos la más usual es a través de una Interface. Gracias a la creación de **ViewModel** existe una nueva forma de hacerlo.

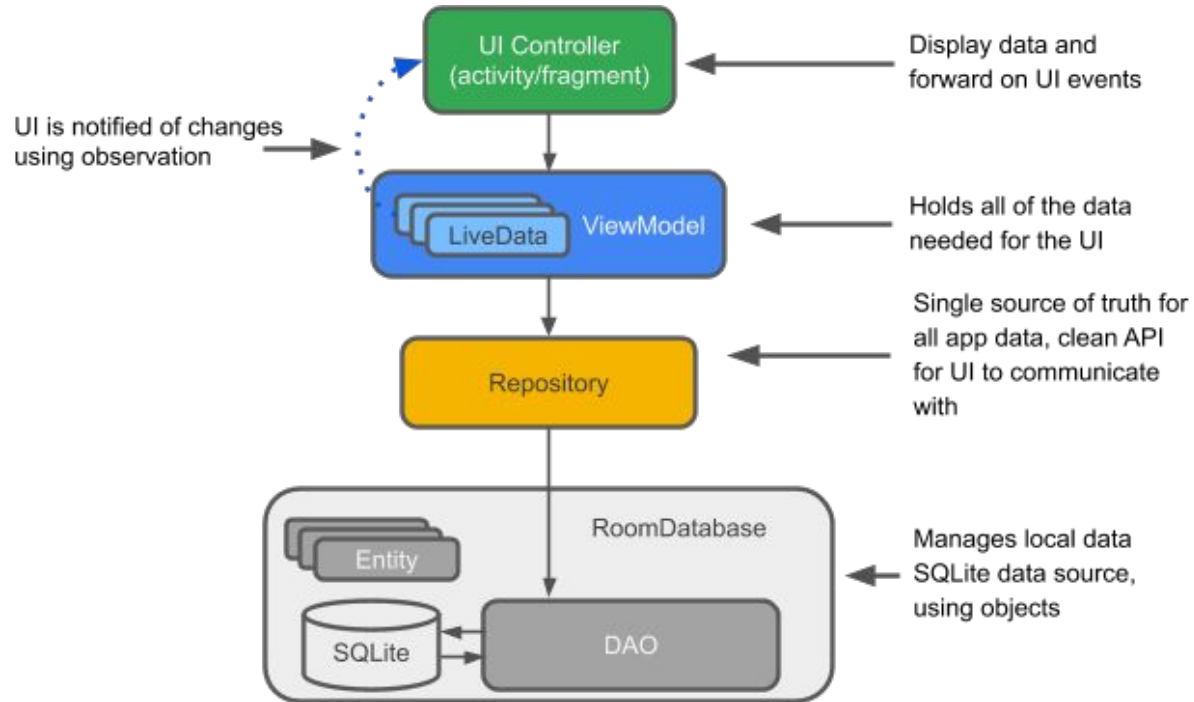
Esta forma nos trae todas las ventajas como permitirnos no tener que traer data múltiples veces si existe un cambio de configuración o rotación del dispositivo. Y esto ocurre gracias a que el viewModel está asociado al ciclo de vida de la actividad.

Los pasos son simples, necesitamos crear un ViewModel con el scope de la actividad de los dos fragmentos, luego inicializar el objeto ViewModel y pasar los valores a través de un objeto LiveData.

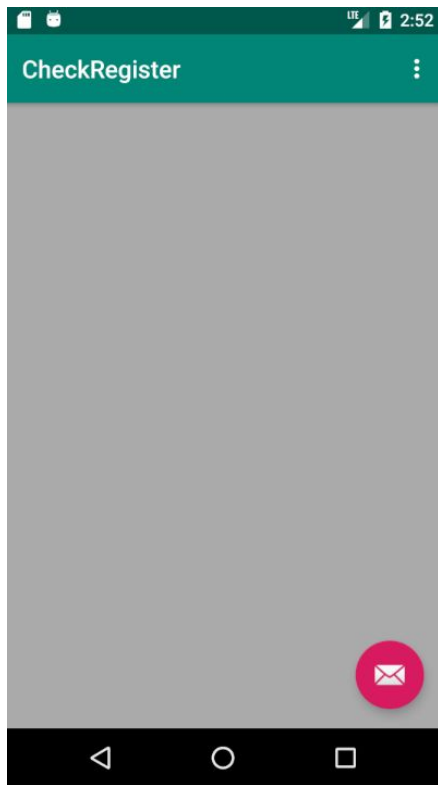
Después de esto el otro fragmento puede observar el objeto Livedata y obtener los valores para ser mostrados en la UI

LiveData y ViewModel en un proyecto

Esquema del proyecto




Creando el proyecto CheckRegister



Creando el proyecto CheckRegister

Configure your project



Basic Activity

Creates a new basic activity with an app bar.

Name
CheckRegister

Package name
cl.desafiolatam.checkregister

Save location
/Users/user/AndroidStudioProjects/CheckRegister

Language
Kotlin

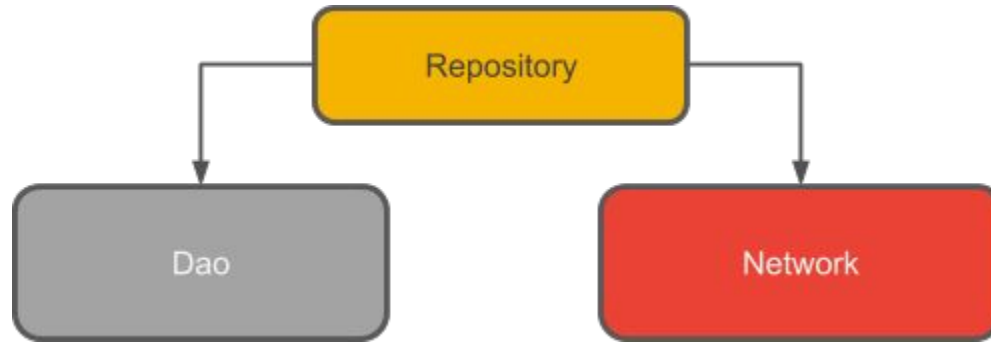
Minimum API level
API 23: Android 6.0 (Marshmallow)

i Your app will run on approximately **62.6%** of devices.
[Help me choose](#)

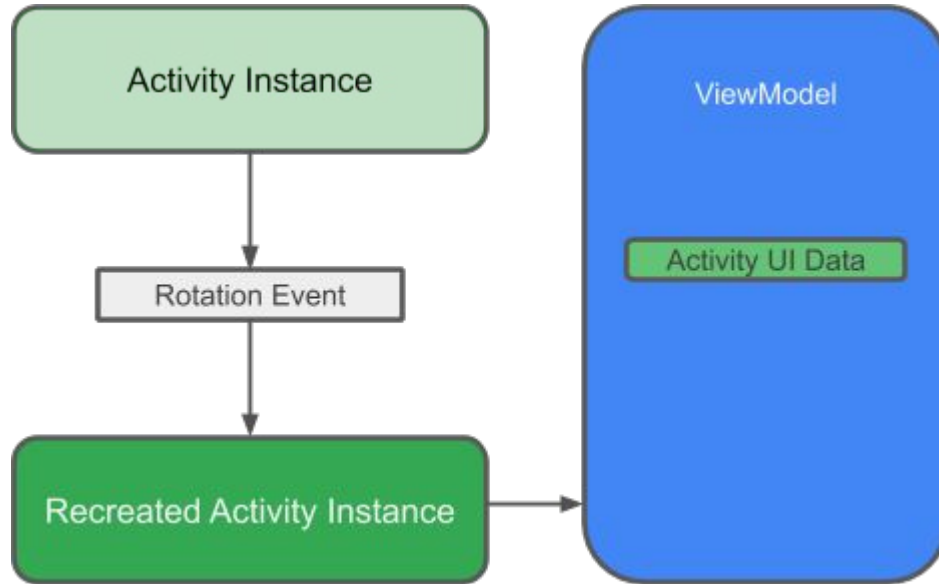
☐ This project will support instant apps

☒ Use androidx.* artifacts

Creando el Repositorio



Creación de ViewModel



{desafío}
latam_

*Academia de
talentos digitales*

www.desafiolatam.com