LiveData

1. Lifecycle
2. LiveData

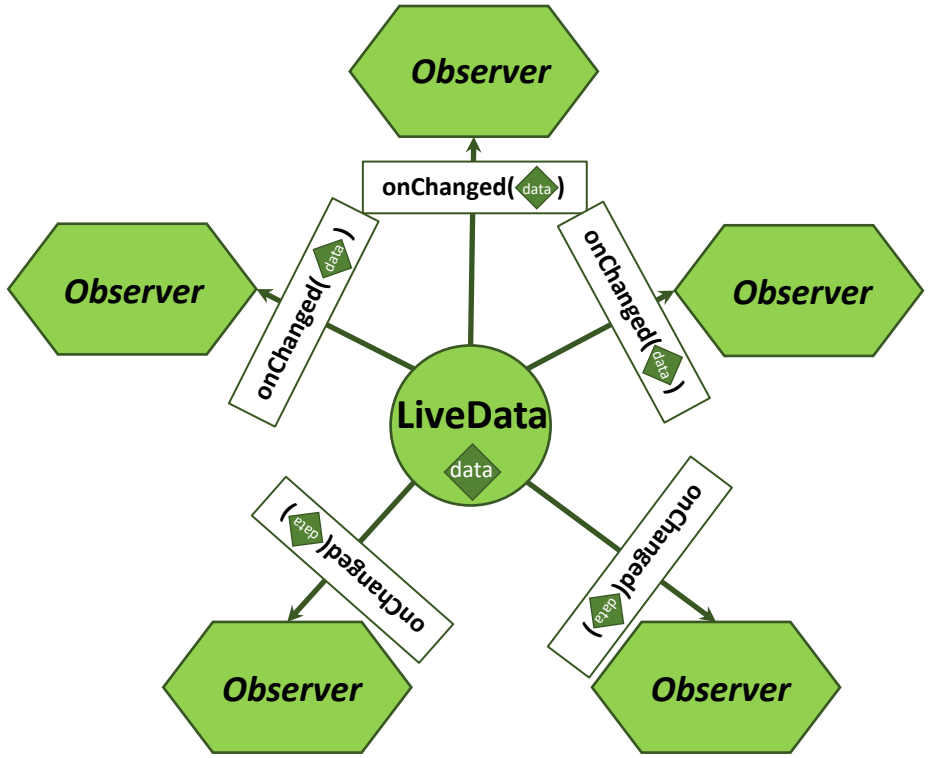
LiveData

Es un *data holder* (contiene de un dato) que facilita el desarrollo de ***Observer pattern***

LiveData

LiveData es un ***observable***, por tanto, permite que se suscriban a él cualquier número de ***observers***.

Los *observers* serán notificados cuando se cambie el dato que contiene LiveData.




```
package androidx.lifecycle.Observer

public interface Observer<T>
{
    public void onChanged(T data) ;
}
```

LiveData

Tipos de suscripciones

void LiveData#subscribeForever(Observer observer)

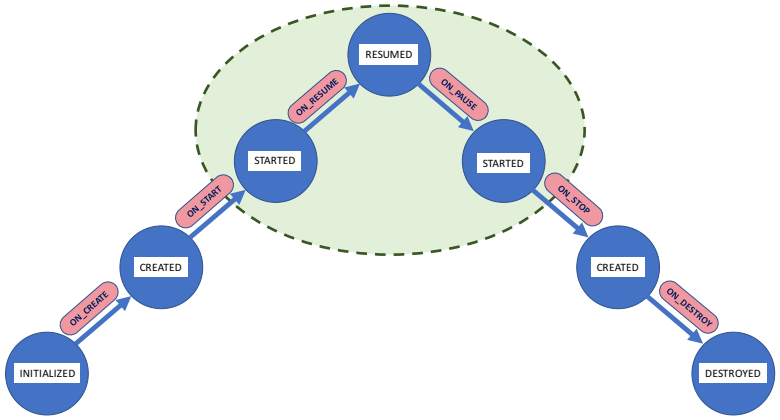
El objeto LiveData notificará al ***observer*** siempre que se cambie el dato que contiene

void LiveData#subscribe(LifeCycleOwner owner, Observer observer)

El ***observer*** será notificado siempre que se cumpla:

a. Se haya cambiado el dato que contiene el LiveData

b. El estado de ***owner*** debe ser al menos (State#isAtLeast(State state))
State.STARTED o **State.RESUMED**



LiveData

Tipos de suscripciones

void LiveData#subscribeForever(Observer observer)

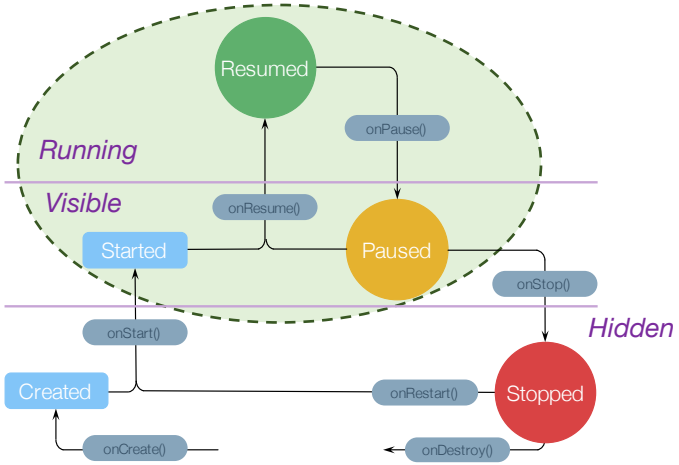
El objeto LiveData notificará al ***observer*** siempre que se cambie el dato que contiene

void LiveData#subscribe(LifeCycleOwner owner, Observer observer)

El ***observer*** será notificado siempre que se cumpla:

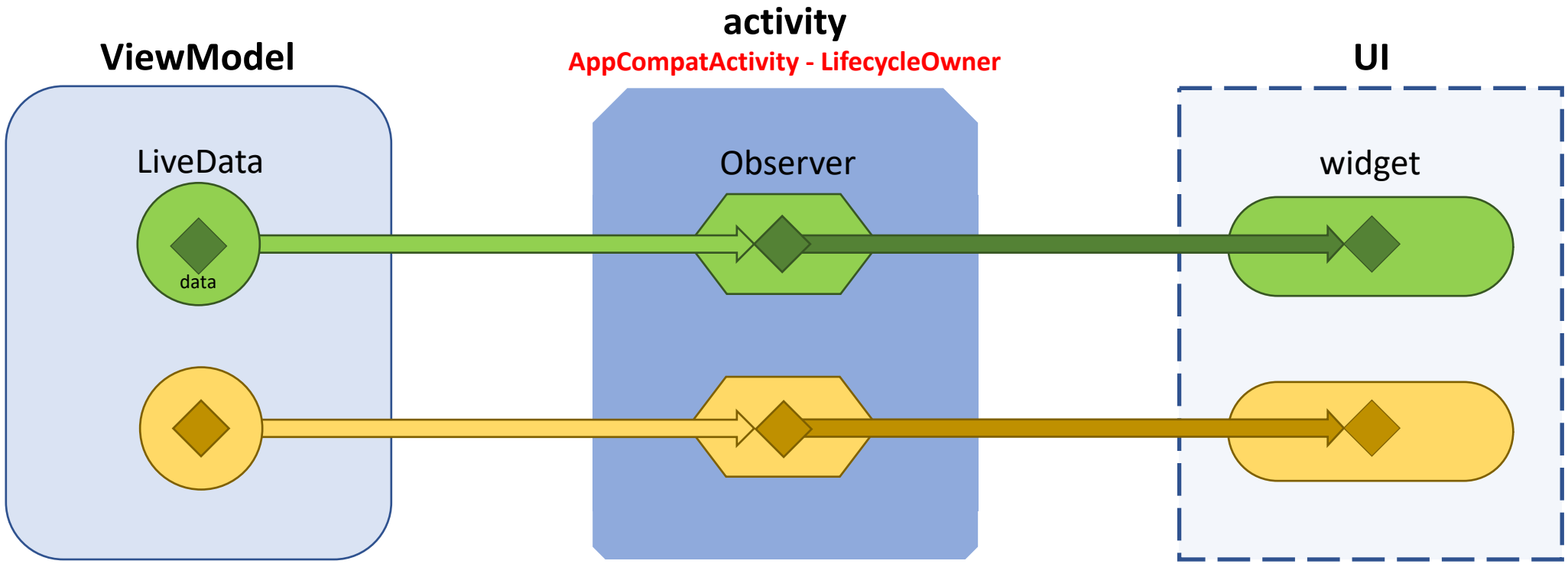
a. Se haya cambiado el dato que contiene el LiveData

b. El estado de ***owner*** debe ser al menos (State#isAtLeast(State state))
State.STARTED o **State.RESUMED**



LiveData

ViewModel



a. Los datos mostrados en la UI están contenidos en LiveData y residen en ViewModel

b. La *activity* se definen *observers* que se suscriben a los LiveData vínculándolos al ciclo de vida de la activity

c. Los observers modifican la UI a partir del dato notificado

LiveData

Implementaciones

MutableLiveData<T>

T getValue()

Obtiene el dato que contiene

void setValue(T value)

Modifica el dato que contiene notificándoselo a los *observers* que estén subscritos

void postValue(T value)

Envía una tarea al hilo principal (main thread - UI thread) para que modifique el dato que contiene y notifique de ello a los *observers* que tenga subscritos

LiveData

Implementaciones

MediatorLiveData<T>

extends MutableLiveData<T>

<S> void addSource(LiveData<S> source, Observer<S> onChanged)

Añade al MediatorLiveData<T> un LiveData<S> *source* asociándole el *observer onChanged*. El MediatorLiveData contiene un *observer* propio que se suscribirá a cada *source* añadida. Este *observer* invocará al *onChanged* asociado con aquel *source* que cambie de dato. Dicho *onChanged* podría modificar el dato del MediatorLiveData.

LiveData

Implementaciones

MediatorLiveData<T>

extends MutableLiveData<T>

```
MutableLiveData< Double > temperatura = new MutableLiveData<>();
MutableLiveData< Double > humedad = new MutableLiveData<>();
MediatorLiveData< Double > sensacionTermica = new MediatorLiveData<>();

temperatura.setValue(25);
humedad.setValue(0.5);

sensacionTermica.addSource(temperatura, grados -> sensacionTermica.setValue( grados + humedad.getValue() * 10) );
sensacionTermica.addSource(humedad,    percent-> sensacionTermica.setValue(temperatura.getValue() + 10*percent));
```

LiveData

Implementaciones - Transformations

static <X, Y> LiveData<Y> Transformations.map(LiveData<X> source, Function<X, Y> function)

Transformations.map() crea y devuelve un LiveData<Y>. Este LiveData<Y> a su vez es un *observer* del LiveData<X> que representa el parámetro *source*. Cuando *source* notifique a este *observer* que tiene un nuevo dato (x) el *observer* modificará el dato (y) de LiveData<Y> previa transformación del dato x en y. De esta trasformación se encarga la función que representa el parámetro *function*.

```
Function<Integer, String> intToString = new Function<Integer, String>()
                                        {
                                          @Override
                                          public String apply(Integer number)
                                          {
                                             return String.format("%d", number.intValue());
                                          }
                                        };

MutableLiveData<Integer> numero = new MutableLiveData<>();
Numero.setValue(33);

LiveData<String> numeroTexto = Transformations.map(numero, intToString);
```

LiveData

Implementaciones - Transformations

static <X,Y> LiveData<Y> Transformations.switchMap(LiveData<X> trigger, Function<X, LiveData<Y>> function)

Transformations.switchMap() crea y devuelve un LiveData<Y>. Este LiveData<Y> cambiará su valor cada vez que se modifique el valor del LiveData<X> *trigger*. Esto es debido a que LiveData<Y> contiene un *observer* que se suscribirá a *trigger*. Cuando *trigger* notifique a este *observer* que tiene un nuevo dato (x) el *observer* modificará el dato de LiveData<Y> previa transformación del valor (x). Esta transformación la realiza el parámetro *function.* Este parámetro representa una función a la que se pasa como parámetro el dato (x) y devuelve un LiveData denominado *backing*. El dato (y) contenido en este *backing* será el que utilice el *observer* para modificar el dato del LiveData<Y>.