Live Data

**Descripción general de LiveData Parte de** [**Android Jetpack**](https://developer.android.com/jetpack?hl=es)

[LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) es una clase de contenedor de datos observable. A diferencia de un observable regular, LiveData está optimizado para ciclos de vida, lo que significa que respeta el ciclo de vida de otros componentes de las apps, como actividades, fragmentos o servicios. Esta optimización garantiza que LiveData solo actualice observadores de componentes de apps que tienen un estado de ciclo de vida activo.

**Nota:** Para importar componentes de LiveData a tu proyecto de Android, consulta [Cómo agregar componentes a tu proyecto](https://developer.android.com/topic/libraries/architecture/adding-components?hl=es#lifecycle).

LiveData considera que un observador, que está representado por la clase [Observer](https://developer.android.com/reference/androidx/lifecycle/Observer?hl=es), está en estado activo si su ciclo de vida está en el estado [STARTED](https://developer.android.com/reference/androidx/lifecycle/Lifecycle.State?hl=es#STARTED) o [RESUMED](https://developer.android.com/reference/androidx/lifecycle/Lifecycle.State?hl=es#RESUMED). LiveData solo notifica a los observadores activos sobre las actualizaciones. Los observadores inactivos registrados para ver objetos [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) no reciben notificaciones sobre los cambios.

Puedes registrar un observador vinculado con un objeto que implemente la interfaz [LifecycleOwner](https://developer.android.com/reference/androidx/lifecycle/LifecycleOwner?hl=es). Esta relación permite quitar al observador cuando el estado del objeto [Lifecycle](https://developer.android.com/reference/androidx/lifecycle/Lifecycle?hl=es) correspondiente cambia a [DESTROYED](https://developer.android.com/reference/androidx/lifecycle/Lifecycle.State?hl=es#DESTROYED). Esto es especialmente útil para actividades y fragmentos, ya que pueden observar objetos [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) de forma segura y no preocuparse por las filtraciones: las actividades y los fragmentos se anulan instantáneamente cuando se destruyen sus ciclos de vida.

Para obtener más información sobre cómo usar LiveData, consulta [Cómo trabajar con objetos LiveData](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#work_livedata).

### **Ventajas de usar LiveData**

El uso de LiveData brinda las siguientes ventajas:

**Garantiza que la IU coincida con el estado de los datos**

LiveData sigue el patrón del observador. LiveData notifica a los objetos [Observer](https://developer.android.com/reference/androidx/lifecycle/Observer?hl=es) cuando cambian los datos subyacentes. Puedes consolidar tu código para actualizar la IU en esos objetos Observer. De esa manera, no necesitas actualizar la IU cada vez que cambian los datos de la app porque el observador lo hace por ti.

**Sin fugas de memoria**

Los observadores están vinculados a objetos [Lifecycle](https://developer.android.com/reference/androidx/lifecycle/Lifecycle?hl=es) y borran lo que crean cuando se destruye el ciclo de vida asociado.

**Actividades detenidas para evitar las fallas**

Si el ciclo de vida del observador está inactivo, como en el caso de una actividad de la pila de actividades, no recibe ningún evento de LiveData.

**No más control manual del ciclo de vida**

Los componentes de IU solo observan los datos relevantes y no detienen ni reanudan la observación. LiveData se ocupa automáticamente de todo esto, ya que está al tanto de los cambios de estado del ciclo de vida relevantes mientras lleva a cabo la observación.

**Datos siempre actualizados**

Si un ciclo de vida queda inactivo, recibe los datos más recientes después de quedar activo de nuevo. Por ejemplo, una actividad que estuvo en segundo plano recibe los datos más recientes inmediatamente después de volver al primer plano.

**Cambios de configuración apropiados**

Una actividad o un fragmento que se vuelve a crear debido a un cambio de configuración, como la rotación del dispositivo, recibe de inmediato los datos disponibles más recientes.

**Compartir recursos**

Puedes extender un objeto [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) con el patrón singleton para unir los servicios del sistema de modo que puedan compartirse en la app. El objeto LiveData se conecta al servicio del sistema una vez y, luego, cualquier observador que necesite el recurso puede simplemente mirar el objeto LiveData. Para obtener más información, consulta [Cómo extender LiveData](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#extend_livedata).

## **Cómo trabajar con objetos LiveData**

Sigue estos pasos para trabajar con objetos [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es):

1. Crea una instancia de LiveData para contener un tipo de datos determinado. Por lo general, esto se hace dentro de la clase [ViewModel](https://developer.android.com/reference/androidx/lifecycle/ViewModel?hl=es).
2. Crea un objeto [Observer](https://developer.android.com/reference/androidx/lifecycle/Observer?hl=es) que defina el método [onChanged()](https://developer.android.com/reference/androidx/lifecycle/Observer?hl=es#onChanged(T)), el cual controla lo que sucede cuando cambian los datos retenidos del objeto LiveData. Por lo general, debes crear un objeto Observer en un controlador de IU, como una actividad o un fragmento.
3. Conecta el objeto Observer al objeto LiveData con el método [observe()](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#observe(android.arch.lifecycle.LifecycleOwner,%0Aandroid.arch.lifecycle.Observer%3CT%3E)). El método observe() toma un objeto [LifecycleOwner](https://developer.android.com/reference/androidx/lifecycle/LifecycleOwner?hl=es). Esto suscribe el objeto Observer al objeto LiveData para que se le notifiquen los cambios. Normalmente, debes conectar el objeto Observer a un controlador de IU, como una actividad o un fragmento.  
   **Nota:** Puedes registrar un observador sin un objeto [**LifecycleOwner**](https://developer.android.com/reference/androidx/lifecycle/LifecycleOwner?hl=es) asociado con el método [**observeForever(Observer)**](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#observeForever(android.arch.lifecycle.Observer%3CT%3E)). En este caso, se considera que el observador siempre está activo y, por lo tanto, siempre recibe notificaciones sobre los cambios. Puedes quitar esos observadores llamando al método [**removeObserver(Observer)**](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#removeObserver(android.arch.lifecycle.Observer%3CT%3E)).

Cuando actualizas el valor almacenado en el objeto LiveData, activa todos los observadores registrados, siempre que el LifecycleOwner conectado esté en el estado activo.

LiveData permite que los observadores del controlador de IU se suscriban a actualizaciones. Cuando los datos retenidos por el objeto LiveData cambian, la IU se actualiza automáticamente en respuesta a eso.

### **Crea objetos LiveData**

LiveData es un wrapper que se puede usar con cualquier dato, incluidos los objetos que implementan [Collections](https://developer.android.com/reference/java/util/Collections?hl=es), como [List](https://developer.android.com/reference/java/util/List?hl=es). Por lo general, un objeto [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) se almacena dentro de un objeto [ViewModel](https://developer.android.com/reference/androidx/lifecycle/ViewModel?hl=es), y se puede acceder a este a través de un método get, como se muestra en el siguiente ejemplo:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

public class NameViewModel extends ViewModel {

// Create a LiveData with a String

private MutableLiveData<String> currentName;

public MutableLiveData<String> getCurrentName() {

if (currentName == null) {

currentName = new MutableLiveData<String>();

}

return currentName;

}

// Rest of the ViewModel...

}

Inicialmente, los datos de un objeto LiveData no están configurados.

**Nota:** Asegúrate de almacenar objetos **LiveData** que actualicen la IU en objetos **ViewModel**, en lugar de una actividad o un fragmento, por los siguientes motivos:

* Para evitar las actividades y fragmentos demasiado cargados: ahora estos controladores de IU son responsables de mostrar los datos, pero no retienen el estado de los datos
* Para desacoplar las instancias **LiveData** de instancias de actividades o fragmentos específicos y permitir que los objetos **LiveData** sobrevivan a los cambios de configuración

Puedes obtener más información sobre los beneficios y el uso de la clase ViewModel en la [Guía de ViewModel](https://developer.android.com/topic/libraries/architecture/viewmodel?hl=es).

### **Observa objetos LiveData**

En la mayoría de los casos, el método [onCreate()](https://developer.android.com/reference/android/app/Activity?hl=es#onCreate(android.os.Bundle)) de un componente de la app es el lugar adecuado para comenzar a observar un objeto [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) por los siguientes motivos:

* Para garantizar que el sistema no realice llamadas redundantes desde el método [onResume()](https://developer.android.com/reference/android/app/Activity?hl=es#onResume()) de una actividad o un fragmento
* Para garantizar que la actividad o el fragmento tenga datos que pueda mostrar tan pronto como quede activo. Tan pronto como un componente de la app se encuentra en el estado [STARTED](https://developer.android.com/reference/androidx/lifecycle/Lifecycle.State?hl=es#STARTED), recibe el valor más reciente de los objetos LiveData que está observando. Esto solo ocurre si se configuró el objeto LiveData a observar.

En general, LiveData solo brinda actualizaciones cuando los datos cambian, y solo a observadores activos. Una excepción a este comportamiento es que los observadores también reciben una actualización cuando cambian de un estado activo a un estado inactivo. Además, si el observador cambia de inactivo a activo por segunda vez, solo recibe una actualización si el valor cambió desde la última vez que estuvo activo.

En el siguiente código de ejemplo, se indica cómo comenzar a observar un objeto LiveData:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

public class NameActivity extends AppCompatActivity {

private NameViewModel model;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Other code to setup the activity...

// Get the ViewModel.

model = new ViewModelProvider(this).get(NameViewModel.class);

// Create the observer which updates the UI.

final Observer<String> nameObserver = new Observer<String>() {

@Override

public void onChanged(@Nullable final String newName) {

// Update the UI, in this case, a TextView.

nameTextView.setText(newName);

}

};

// Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.

model.getCurrentName().observe(this, nameObserver);

}

}

Después de llamar a [observe()](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#observe(android.arch.lifecycle.LifecycleOwner,%0Aandroid.arch.lifecycle.Observer%3CT%3E)) y pasar nameObserver como parámetro, [onChanged()](https://developer.android.com/reference/androidx/lifecycle/Observer?hl=es#onChanged(T)) se invoca de inmediato y proporciona el valor más reciente almacenado en mCurrentName. Si el objeto LiveData no estableció un valor en mCurrentName, no se llama a onChanged().

### **Actualiza objetos LiveData**

LiveData no tiene métodos disponibles públicamente para actualizar los datos almacenados. La clase [MutableLiveData](https://developer.android.com/reference/androidx/lifecycle/MutableLiveData?hl=es) expone los métodos [setValue(T)](https://developer.android.com/reference/androidx/lifecycle/MutableLiveData?hl=es#setValue(T)) y [postValue(T)](https://developer.android.com/reference/androidx/lifecycle/MutableLiveData?hl=es#postValue(T)) de forma pública; debes usarlos si necesitas editar el valor almacenado en un objeto [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es). Por lo general, MutableLiveData se usa en [ViewModel](https://developer.android.com/reference/androidx/lifecycle/ViewModel?hl=es) y, luego, ViewModel solo expone objetos LiveData inmutables a los observadores.

Después de configurar la relación del observador, puedes actualizar el valor del objeto LiveData, como se muestra en el siguiente ejemplo, que activa todos los observadores cuando el usuario presiona un botón:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

button.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

String anotherName = "John Doe";

model.getCurrentName().setValue(anotherName);

}

});

Llamar a setValue(T) en el ejemplo da como resultado que los observadores llamen a sus métodos [onChanged()](https://developer.android.com/reference/androidx/lifecycle/Observer?hl=es#onChanged(T)) con el valor John Doe. En el ejemplo, se presiona un botón, pero se puede llamar a setValue() o postValue() para actualizar mName por varias razones, incluso en respuesta a una solicitud de red o para completar la carga de una base de datos. En todos los casos, la llamada a setValue() o postValue() activa observadores y actualiza la IU.

**Nota:** Debes llamar al método [**setValue(T)**](https://developer.android.com/reference/androidx/lifecycle/MutableLiveData?hl=es#setValue(T)) para actualizar el objeto **LiveData** del subproceso principal. Si el código se ejecuta en un subproceso del trabajador, puedes usar el método [**postValue(T)**](https://developer.android.com/reference/androidx/lifecycle/MutableLiveData?hl=es#postValue(T)) en su lugar para actualizar el objeto **LiveData**.

### **Usa LiveData con Room**

La biblioteca de persistencias de [Room](https://developer.android.com/training/data-storage/room?hl=es) admite consultas observables, que muestran objetos [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es). Las consultas observables se escriben como parte del objeto de acceso a la base de datos (DAO).

Room genera todo el código necesario para actualizar el objeto LiveData cuando se actualiza una base de datos. El código generado ejecuta la consulta de manera asíncrona en un subproceso en segundo plano cuando es necesario. Este patrón es útil para mostrar los datos en una IU sincronizada con los datos almacenados en una base de datos. Puedes obtener más información sobre Room y DAO en la [guía sobre la biblioteca de persistencias Room](https://developer.android.com/topic/libraries/architecture/room?hl=es).

### **Usa corrutinas con LiveData**

LiveData incluye compatibilidad con corrutinas de Kotlin. Para obtener más información, consulta [Cómo usar corrutinas de Kotlin con componentes de la arquitectura de Android](https://developer.android.com/topic/libraries/architecture/coroutines?hl=es).

## **Extiende LiveData**

LiveData considera que un observador está en estado activo si su ciclo de vida está en el estado [STARTED](https://developer.android.com/reference/androidx/lifecycle/Lifecycle.State?hl=es#STARTED) o [RESUMED](https://developer.android.com/reference/androidx/lifecycle/Lifecycle.State?hl=es#RESUMED). En el siguiente código de muestra, se describe cómo extender la clase [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es):

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

public class StockLiveData extends LiveData<BigDecimal> {

private StockManager stockManager;

private SimplePriceListener listener = new SimplePriceListener() {

@Override

public void onPriceChanged(BigDecimal price) {

setValue(price);

}

};

public StockLiveData(String symbol) {

stockManager = new StockManager(symbol);

}

@Override

protected void onActive() {

stockManager.requestPriceUpdates(listener);

}

@Override

protected void onInactive() {

stockManager.removeUpdates(listener);

}

}

La implementación del oyente de precio en este ejemplo incluye los siguientes métodos importantes:

* El método [onActive()](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#onActive()) se llama cuando el objeto LiveData tiene un observador activo. Esto quiere decir que debes comenzar a observar las actualizaciones de cotización de las acciones a partir de este método.
* El método [onInactive()](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#onInactive()) se llama cuando el objeto LiveData no tiene ningún observador activo. Como ningún observador tiene implementado un objeto de escucha, no hay razón para mantenerse conectado al servicio StockManager.
* El método [setValue(T)](https://developer.android.com/reference/androidx/lifecycle/MutableLiveData?hl=es#setValue(T)) actualiza el valor de la instancia de LiveData y notifica a los observadores activos sobre el cambio.

Puedes usar la clase StockLiveData de la siguiente manera:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

public class MyFragment extends Fragment {

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

LiveData<BigDecimal> myPriceListener = ...;

myPriceListener.observe(getViewLifeycleOwner(), price -> {

// Update the UI.

});

}

}

El método [observe()](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es#observe(androidx.lifecycle.LifecycleOwner,%20androidx.lifecycle.Observer%3C?%20super%20T%3E)) pasa el [LifecycleOwner](https://developer.android.com/reference/androidx/lifecycle/LifecycleOwner?hl=es) asociado con la vista del fragmento como primer argumento. Hacer esto indica que este observador está vinculado al objeto [Lifecycle](https://developer.android.com/reference/androidx/lifecycle/Lifecycle?hl=es) asociado con el propietario, lo que significa lo siguiente:

* Si el objeto Lifecycle no está en estado activo, no se llama al observador, aunque cambie el valor.
* Después de que se destruye el objeto Lifecycle, el observador se quita automáticamente.

El hecho de que los objetos LiveData tengan en cuenta el ciclo de vida significa que puedes compartirlos entre varias actividades, fragmentos y servicios. Para que el ejemplo sea simple, puedes implementar la clase LiveData como un singleton de la siguiente manera:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

public class StockLiveData extends LiveData<BigDecimal> {

private static StockLiveData sInstance;

private StockManager stockManager;

private SimplePriceListener listener = new SimplePriceListener() {

@Override

public void onPriceChanged(BigDecimal price) {

setValue(price);

}

};

@MainThread

public static StockLiveData get(String symbol) {

if (sInstance == null) {

sInstance = new StockLiveData(symbol);

}

return sInstance;

}

private StockLiveData(String symbol) {

stockManager = new StockManager(symbol);

}

@Override

protected void onActive() {

stockManager.requestPriceUpdates(listener);

}

@Override

protected void onInactive() {

stockManager.removeUpdates(listener);

}

}

Además, puedes usarlo en el fragmento de la siguiente manera:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

public class MyFragment extends Fragment {

@Override

public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {

super.onViewCreated(view, savedInstanceState);

StockLiveData.get(symbol).observe(getViewLifecycleOwner(), price -> {

// Update the UI.

});

}

}

Múltiples fragmentos y actividades pueden observar la instancia MyPriceListener. LiveData solo se conecta al servicio del sistema si uno o más de ellos están visibles y activos.

## **Transforma LiveData**

Es posible que desees realizar cambios en el valor almacenado en un objeto [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) antes de despacharlo a los observadores, o tal vez debas mostrar una instancia de LiveData diferente según el valor de otro. El paquete [Lifecycle](https://developer.android.com/reference/android/arch/lifecycle/package-summary?hl=es) proporciona la clase [Transformations](https://developer.android.com/reference/androidx/lifecycle/Transformations?hl=es), que incluye métodos de ayuda que admiten estas situaciones.

[Transformations.map()](https://developer.android.com/reference/androidx/lifecycle/Transformations?hl=es#map(android.arch.lifecycle.LiveData%3CX%3E,%20android.arch.core.util.Function%3CX,%20Y%3E))

Aplica una función al valor almacenado en el objeto LiveData y propaga el resultado de manera descendente.

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

LiveData<User> userLiveData = ...;

LiveData<String> userName = Transformations.map(userLiveData, user -> {

user.name + " " + user.lastName

});

[Transformations.switchMap()](https://developer.android.com/reference/androidx/lifecycle/Transformations?hl=es#switchMap(android.arch.lifecycle.LiveData%3CX%3E,%20android.arch.core.util.Function%3CX,%20android.arch.lifecycle.LiveData%3CY%3E%3E))

De manera similar a map(), aplica una función al valor almacenado en el objeto LiveData, y separa y despacha el resultado en sentido descendente. La función que se pasa a switchMap() debe mostrar un objeto LiveData, como se indica en el siguiente ejemplo:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

private LiveData<User> getUser(String id) {

...;

}

LiveData<String> userId = ...;

LiveData<User> user = Transformations.switchMap(userId, id -> getUser(id) );

Puedes usar métodos de transformación para transportar información durante el ciclo de vida del observador. Las transformaciones no se calculan a menos que un observador esté viendo el objeto LiveData que se mostró. Debido a que las transformaciones se calculan lentamente, el comportamiento relacionado con el ciclo de vida se traslada de manera implícita sin requerir llamadas o dependencias explícitas adicionales.

Si crees que necesitas un objeto Lifecycle dentro de un objeto [ViewModel](https://developer.android.com/reference/androidx/lifecycle/ViewModel?hl=es), una transformación probablemente sea una mejor solución. Por ejemplo, imagina que tienes un componente de IU que acepta una dirección y devuelve el código postal de esa dirección. Puedes implementar el ViewModel básico para este componente como se indica en el siguiente código de muestra:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

class MyViewModel extends ViewModel {

private final PostalCodeRepository repository;

public MyViewModel(PostalCodeRepository repository) {

this.repository = repository;

}

private LiveData<String> getPostalCode(String address) {

// DON'T DO THIS

return repository.getPostCode(address);

}

}

Luego, el componente de la IU debe anular el registro del objeto LiveData anterior y registrarse en la nueva instancia cada vez que llama a getPostalCode(). Además, si se vuelve a crear el componente de la IU, se activa otra llamada al método repository.getPostCode() en lugar de usar el resultado de la llamada anterior.

En su lugar, puedes implementar la búsqueda del código postal como una transformación de la entrada de dirección. Eso se muestra en el siguiente ejemplo:

[Kotlin](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#kotlin)

[Java](https://developer.android.com/topic/libraries/architecture/livedata?hl=es#java)

class MyViewModel extends ViewModel {

private final PostalCodeRepository repository;

private final MutableLiveData<String> addressInput = new MutableLiveData();

public final LiveData<String> postalCode =

Transformations.switchMap(addressInput, (address) -> {

return repository.getPostCode(address);

});

public MyViewModel(PostalCodeRepository repository) {

this.repository = repository

}

private void setInput(String address) {

addressInput.setValue(address);

}

}

En este caso, el campo postalCode se define como una transformación de addressInput. Siempre y cuando la app tenga un observador activo asociado con el campo postalCode, el valor del campo se recalcula y se recupera cuando cambia addressInput.

Este mecanismo permite que los niveles inferiores de la app creen objetos LiveData que se calculan de forma diferida a pedido. Un objeto ViewModel puede obtener fácilmente referencias a objetos LiveData y, luego, definir reglas de transformación sobre ellos.

### **Crea transformaciones nuevas**

Hay decenas de transformaciones específicas que pueden ser útiles para tu app, pero no se proporcionan de manera predeterminada. Para implementar tu propia transformación, puedes usar la clase [MediatorLiveData](https://developer.android.com/reference/androidx/lifecycle/MediatorLiveData?hl=es), que implementa objetos de escucha para otros objetos [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) y procesa los eventos emitidos por estos. MediatorLiveData propaga correctamente su estado al objeto LiveData de origen. Para obtener más información sobre este patrón, consulta la documentación de referencia de la clase [Transformations](https://developer.android.com/reference/androidx/lifecycle/Transformations?hl=es).

## **Fusiona varias fuentes de LiveData**

[MediatorLiveData](https://developer.android.com/reference/androidx/lifecycle/MediatorLiveData?hl=es) es una subclase de [LiveData](https://developer.android.com/reference/androidx/lifecycle/LiveData?hl=es) que te permite combinar varias fuentes de LiveData. Los observadores de los objetos MediatorLiveData se activan cada vez que cambia alguno de los objetos de origen originales de LiveData.

Por ejemplo, si tienes un objeto LiveData en la IU que se puede actualizar desde una base de datos local o una red, puedes agregar las siguientes fuentes al objeto MediatorLiveData:

* Un objeto LiveData asociado con los datos almacenados en la base de datos.
* Un objeto LiveData asociado con los datos a los que se accede desde la red.

Tu actividad solo necesita observar el objeto MediatorLiveData para recibir actualizaciones de ambas fuentes. Si deseas ver un ejemplo detallado, consulta la sección [Apéndice: Cómo exponer el estado de la red](https://developer.android.com/topic/libraries/architecture/guide?hl=es#addendum) de la [Guía de arquitectura de apps](https://developer.android.com/topic/libraries/architecture/guide?hl=es).