Dataflow mediator for LiveData reference

FEMA Studios S.r.l.s.

Version 1, revision 1

Table of Contents

Dataflow mediator for LiveData

This mediator library provides an easy conversion between `Field` and `LiveData` objects.

Getting started

Setup

To add this extension all you have to do is add the following dependency:

Gradle

```
implementation 'com.femastudios:dataflow-mediator-livedata:1.0.0'
```

Further instructions on how to configure you Android™ project can be found here.

Usage

LiveData to Field conversion

When using Android™, you may need to handle `LiveData` objects: they may come from an external library such as Room, where you don't have control over which type of data is returned.

To convert them to fields this mediator offers `toField()`, a very easy and intuitive extension function to the `LiveData` class:

Kotlin

```kotlin
val liveData : LiveData<String> = getLiveData() //Obtain a LiveData object from somewhere
val fld : Field<String?> = liveData.toField() //Converts the LiveData to a Field
```

Java

```java
LiveData<String> liveData = getLiveData(); //Obtain a LiveData object from somewhere
Field<String> fld = DataflowLiveDataMediation.toField(liveData) //Converts the LiveData to
a Field
```

The returned field will `observe()` the value changes of the `LiveData` and will update the `Field` value accordingly.

The Android `Lifecycle` used to observe the `LiveData` has the following behavior:

- When the `Field` is first created, `ON_CREATE` is dispatched
- When the field becomes active (has attached listeners), `ON_START` will be dispatched
- When field becomes inactive (has NOT attached listeners), `ON_STOP` will be dispatched

Notice how the returned `Field` may contain `null`: this is necessary because the `LiveData` can always contain a `null` pointer.

Mutable variant

Similarly we can convert a `MutableLiveData` to a `MutableField` with the `toMutableField()` function:

Kotlin

```kotlin
val mutableLiveData : MutableLiveData<String> = getMutableLiveData() //Obtain a
MutableLiveData object from somewhere
val mutableField : MutableField<String?> = mutableLiveData.toMutableField() //Converts the
MutableLiveData to a MutableField
```

Java

```java
MutableLiveData<String> liveData = getMutableLiveData(); //Obtain a MutableLiveData object
from somewhere
MutableField<String> field = DataflowLiveDataMediation.toMutableField(mutableLiveData)
//Converts the MutableLiveData to a MutableField
```

Whenever the returned `MutableField` value is changed, the original `MutableLiveData` value will be updated accordingly using the `postValue()` function.

Field to LiveData conversion

Sometimes it may also arise the need to convert a `Field` to a `LiveData` object, if some external library requires the latter class.

As for the opposite conversion, we have an extension function just for that, which is called `toLiveData()`:

Kotlin

```kotlin
val fld : Field<String> = getField() //Obtain a Field object from somewhere
val liveData : LiveData<String> = field.toLiveData() //Converts the Field to a LiveData
```

Java

```java
Field<String> fld = getField(); //Obtain a Field object from somewhere
LiveData<String> liveData = DataflowLiveDataMediation.toLiveData(field) //Converts the
Field to a LiveData
```

The returned `LiveData` instance will listen to the `Field` changes as long as it is active.

Mutable variant

Once again, the function to convert a `MutableField` to a `MutableLiveData` is pretty idomatic: `toMutableLiveData()`.

Here's an example usage:

Kotlin

```kotlin
val mutableField : MutableField<String> = getMutableField() //Obtain a MutableField object
from somewhere
val mutableLiveData : MutableLiveData<String> = mutableField.toMutableLiveData() //Converts
the MutableField to a MutableLiveData
```

Java

```java
MutableField<String> mutableField = getMutableField(); //Obtain a ableField object from
somewhere
MutableLiveData<String> mutableLiveData =
DataflowLiveDataMediation.toMutableLiveData(mutableField) //Converts the MutableField to a
MutableLiveData
```

Whenever the returned `MutableLiveData` value is changed, the original `MutableField` will be updated accordingly.

Limitations

Thread safety

Unfortunately `LiveData`s can only be changed from the main (UI) thread. In order to achieve a full compatibility, you must ensure that all changes to fields that cause a `LiveData` to change happen on the main thread.

This is necessary because otherwise, after changing the `Field` from a background thread, the value of the `LiveData` won't be synchronized with the one in the `Field`.

As a workaround, you can call `postValue()` or `setOrPostValue()` to ensure that the value will be changed on the main thread.