Programowanie reaktywne na przykładzie platformy Android z użyciem biblioteki rxjava

Reactive programming on the example of the Android platform using the rxjava library

# Wstęp

Nie sposób zaprzeczyć, iż jednym z najszybciej rozwijających się obszarów w świecie IT jest obszar budowania interfejsów aplikacji. Podejście programistów zarówno do interfejsów webowych, jak i mobilnych cały czas ewoluuje, a coraz to nowsze biblioteki i frameworki wymuszają ciągłą potrzebę poznawania nowych rozwiązań. Jednym z kluczowych aspektów budowania interfejsu użytkownika jest możliwość zapanowania nad zdarzeniami, które wywołuje użytkownik aplikacji. Z pomocą przychodzi nam podejście reaktywne, które łączy w sobie paradygmat programowania funkcyjnego i wzorzec projektowy obserwatora. Poniższy rozdział opisuje podstawy programowania reaktywnego zrealizowanego za pomocą biblioteki rxJava i rxAndroid. Następujący rozdział opisuje proces budowania interfejsu użytkownika w platformie Android, a następnie opisuje, w jaki sposób można udoskonalić działanie aplikacji posługując się programowaniem reaktywnym.

## Interfejs użytkownika w platformie Android

Jednym z podstawowych komponentów, który używany jest w procesie budowania aplikacji to Activity. Jest to klasa, której zadaniem jest umożliwienie interakcji użytkownika z aplikacją. Za jej pomocą możemy utworzyć okno, w którym umieszony zostanie widok interfejsu użytkownika. Posługując się przykładem aplikacji, która pełni rolę klienta poczty elektronicznej, możemy spodziewać się, że lista z wiadomościami będzie prezentowana za pomocą jednej instancji klasy Activity, a natomiast inna instancja posłuży nam do utworzenia wiadomości. Istnieje możliwość grupowania kilku Activity w grupy, ale zazwyczaj należy utożsamiać widoczne okno aplikacji z jedną instancją klasy Activity. (1) Jednym z kolejnych podstawowych elementów UI w systemie Android jest klasa Fragment. Można używać ją w wieloraki sposób, lecz najczęściej jest ona ściśle połączona z Activity. W odróżnieniu od Activity klasa Fragment ma raczej za zadanie odpowiadać za pewną mniejszą część interfejsu użytkownika. Warto zaznaczyć, iż zachowanie instancji Fragmentu jest ściśle powiązane z zachowaniem jej rodzica, czyli Activity. W przypadku zakończenia działania danego Activity, wszystkie instancje klasy Fragment, które utworzone zostały w danym Activity również zakończą swoje działanie. (2) Aby lepiej zrozumieć zachowanie się interfejsu użytkownika w systemie Android należy zapoznać się z cyklem życia klas Activity i Fragment.

## Cykl życia komponentów

Podczas używania aplikacji użytkownik może wprowadzać Activity w różnego rodzaju stany. Sam system operacyjny także może oddziaływać na stan, w jakim znajduje się nasza aplikacja. Taką sytuacją może być przykładowo wyświetlenie natywnego ekranu z przychodzącym połączeniem. W takiej sytuacji programista musi posiadać narzędzia, które umożliwią mu reagowanie na zmianę stanu interfejsu użytkownika. Za pomocą odpowiednich metod zdefiniowanych w klasie Activity programista jest w stanie określić jak zachowa się aplikacja w danym stanie zdefiniowanym w cyklu życia. Aby zapanować nad zmianami, które zachodzą w obrębie Activity zdefiniowanych zostało siedem zapytań zwrotnych (ang. Callbacks) za pomocą, których możemy reagować na zachodzące zmiany. Zostały one wyszczególnione i opisane szerzej poniżej.

## Cykl życia klasy Activity

* OnCreate

Jest to obligatoryjny callback, który zostaje wywołany podczas utworzenia danego Activity. W tej metodzie powinien znajdować się taki kod programu, który zostanie wywołany tylko raz w ramach danego cyklu życia Activity. To w obrębie tej metody należy zdefiniować referencję do wszelkiego rodzaju przycisków, pól tekstowych i innych elementów interfejsu użytkownika. Po zakończeniu wykonywania tej metody system wykonuje następny callback OnStart. (3)

* OnRestart

Wywoływana po zatrzymaniu Activity w celu przygotowania go do restartu. Zawsze następuje przed metodą onStart.

* OnStart

Ta metoda skutkuje pojawieniem się interfejsu aplikacji na ekranie urządzenia. Jest ona zawsze wywoływana po metodach OnCreate lub OnRestart. (4)

* OnResume

Metoda ta zostaje wywołana, gdy użytkownik powraca do aplikacji na przykład po odebraniu połączenia lub po ponownym włączeniu ekranu telefonu. W tym miejscu należy umieść kod odpowiedzialny za odświeżenie widoku lub pobranie nowych danych. Po zakończeniu działania tej metody następuje aktywny czas życia aplikacji (ang. The Active Lifetime). (5)

* OnStop

Metoda ta zostaje wywołana, kiedy aplikacja nie jest już widoczna dla użytkownika. W tym miejscu umieszczony może zostać kod programu odpowiedzialny za zatrzymanie wszelkiego rodzaju animacji, serwisów i innych bytów, które związane są z interfejsem użytkownika. (5)

* OnPause

Wywołanie tej metody może nastąpić, kiedy użytkownik zacznie opuszczać Activity lub nastąpi przysłonięcie go przez inne Activity lub Dialog. Aplikacja nadal może być widoczna na ekranie, dlatego nie zaleca się spowalnia wywoływania OnPause różnego rodzaju operacjami zapisu danych. Po tej metodzie mogą zostać wywołane metody: OnResume lub OnStop. (6)

* OnDestroy

Metoda ta zostaje wywołana przed zakończeniem działania Activity. W tym miejscu aplikacji powinna zadbać o zwolnienie zasobów, z których korzysta. Może to być na przykład zamknięcie połączenia z bazą danych.



Rysunek 1 - cykl życia klasy Activity

## Layout jak narzędzie do definiowania widoku

Podstawowym narzędziem, które umożliwia nam zdefiniowane interfejsu użytkownika jest Layout. Wszystkie elementy używane podczas tworzenia UI budowane są za pomocą hierarchii klas View i ViewGroup. Klasa View odpowiedzialna jest za renderowanie bytów, z którymi użytkownik może wchodzić w interakcję. Natomiast klasa ViewGroup to rodzaj niewidzialnego kontenera, który odpowiedzialny jest za definicje struktury interfejsu użytkownika.



Rysunek 2 - Definicja UI za pomocą klas View i ViewGroup

Deklarowanie UI może zostać zrealizowane na dwa sposoby:

* Deklaracja elementów UI bezpośrednio w pliku XML.
* Tworzenie poszczególnych elementów UI podczas działania programu.

Zazwyczaj stosuje się pierwsze podejście w celu odseparowania widoku od logiki aplikacji. Łatwiejsze jest też, zdefiniowanie różnego rodzaju szablonów w zależności od szerokości ekranu lub jego orientacji. (7)

## Format pliku xml

Każdy plik zawierający definicje szablonu UI musi być zdefiniowany w formacie xml. Ponadto musi zawierać dokładnie jeden element nadrzędny, który jest typu View lub ViewGroup. Poniżej zaprezentowany został przykład pliku definiującego UI w systemie Android zawierający pole tekstowe i przycisk.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
              android:layout\_width="match\_parent"  
              android:layout\_height="match\_parent"  
              android:orientation="vertical" >  
    <TextView android:id="@+id/text"  
              android:layout\_width="wrap\_content"  
              android:layout\_height="wrap\_content"  
              android:text="Hello, I am a TextView" />  
    <Button android:id="@+id/button"  
            android:layout\_width="wrap\_content"  
            android:layout\_height="wrap\_content"  
            android:text="Hello, I am a Button" />  
</LinearLayout>

Rysunek 3 - przykład szablonu interfejsu użytkownika

W celu stworzenia referencji do danego elementu zdefiniowanego w szablonie należy przypisać do każdego elementu unikalne ID. Następnie, aby stworzyć instancję danego elementu należy wyszukać go za pomocą zdefiniowanego wcześniej identyfikatora (Zazwyczaj odbywa się to w metodzie OnCreate()). Poniżej zaprezentowany został przykładowy kod w języku Java, który odpowiada za utworzenia instancji przycisku zdefiniowanego w szablonie.

Button myButton = (Button) findViewById(R.id.my\_button);

Rysunek 4 - stworzenie referencji do elementu zdefiniowanego w szablonie

## Istota paradygmatu reaktywnego

Zanim zapoznamy się z problemami, którymi musimy stawić czoła podczas budowania interfejsu aplikacji w systemie Android warto uzmysłowić sobie, co należy rozumieć pod pojęciem programowania reaktywnego. Raczej każdy programista rozpoczyna swoją przygodę z programowaniem od podejścia imperatywnego, w którym kroki w programie są realizowane w sposób sekwencyjny. Taki sposób programowania mocno odbiega od podejścia reaktywnego, które zdobywa coraz większe uznanie wśród programistów. Samo pojęcie nie jest jednak niczym nowym i istnieje już od pewnego czasu w świecie IT. Istota programowania reaktywnego to próba odzwierciedlenia otaczającego nas świata, który zbudowany jest z asynchronicznych zdarzeń. Takie same asynchroniczne zdarzenia zachodzą także w interfejsie użytkownika jak i w całej aplikacji. Takimi zdarzeniami w aplikacji mogą być: pobranie danych z pliku/serwera, wypełnienie pola tekstowego, naciśnięciu przycisku. Podejście reaktywne pozwala nam zapanować nad asynchronicznymi zdarzeniami za pomocą paradygmatu funkcyjnego i wzorca projektowego obserwator. Funkcyjne operacje na danych zapewniają nam zwięzłość w sposobie manipulacji nimi a wzorzec obserwatora zapewnia wygodny sposób reagowania na zdarzenia występujące w aplikacji.

* 1. Elementy paradygmatu funkcyjnego w programowaniu reaktywnym

Najważniejszym komponentem, z którego zbudowany jest paradygmat funkcyjny są funkcje. W podejściu tym nie występują pętlę ani zmienne. Zamiast tego posługiwać musimy się zdefiniowanymi stałymi i rekurencją. Jednymi z największych zalet tego paradygmatu są funkcje wyższego rzędu, strumienie oraz funktory. W rezultacie otrzymany kod zawiera mniej efektów ubocznych i jest bardziej przewidywalny w działaniu. To z tej dziedziny pochodzą takie pojęcia jak czysta funkcja (ang. Pure Function) lub niezmienne dane (ang. Immutable Data). Programując funkcyjnie odpowiadamy raczej na pytanie: „Co chcemy osiągnąć?”, a nie „W jaki sposób chcemy to osiągnąć?” (To pytanie zadajemy sobie programując w paradygmacie imperatywnym). Poznając programowanie reaktywne bardzo często spotykamy się z takimi funkcjami wyższego rzędu jak: map, filter czy fold, które znajdują swoje odzwierciedlenie w programowaniu reaktywnym.

* + 1. Wzorzec obserwator

Drugim składnikiem po paradygmacie funkcyjnym w programowaniu reaktywnym jest wzorzec obserwator (ang. Observer Pattern). Jest to wzorzec typu behawioralnego. Łatwo wydedukować, iż jest to wzorzec, który odpowiada za nasłuchiwanie czy też obserwacje jakiegoś zdarzenia. Wzorzec ten odpowiada, za powiadomienie wszystkich obiektów, które nasłuchiwały na dany rodzaj zdarzenia. Mamy, więc do czynienia z podstawową relacją jeden do wielu.

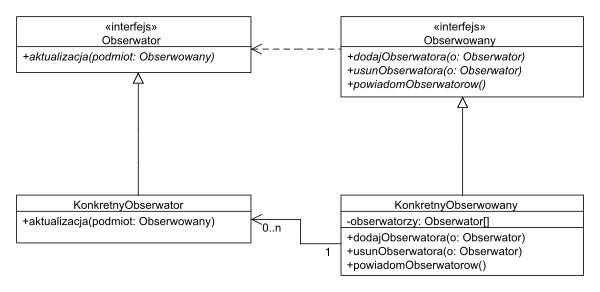


Diagram klas wzorca obserwator (https://tomasz-tomczykiewicz.blog/2017/03/28/wzorzec-projektowy-obserwator/)

Contents

[1. Wstęp 2](#_Toc532424995)

[1. Interfejs użytkownika w platformie Android 2](#_Toc532424996)

[1.1. Cykl życia komponentów 3](#_Toc532424997)

[1.1.1. Cykl życia klasy Activity 3](#_Toc532424998)

[1.1.1. Layout jak narzędzie do definiowania widoku 6](#_Toc532424999)

[1.1.1. Format pliku xml 6](#_Toc532425000)

[1. Istota paradygmatu reaktywnego 8](#_Toc532425001)

[1.1. Elementy paradygmatu funkcyjnego w programowaniu reaktywnym 8](#_Toc532425002)

[1.1.1. Wzorzec obserwator 9](#_Toc532425003)

[Bibliografia 10](#_Toc532425004)

[Rysunek 1 - cykl życia klasy Activity 5](#_Toc532422555)

# Bibliografia

1. https://developer.android.com/guide/components/fundamentals. [Online]

2. https://developer.android.com/reference/android/app/Fragment. [Online]