Programowanie reaktywne na przykładzie platformy Android z użyciem biblioteki rxjava

Reactive programming on the example of the Android platform using the rajasa 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, których używa się w procesie budowania aplikacji jest 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 utworzeni wiadomości. Istnieje możliwość grupowania kilku Activity w grupy, ale zazwyczaj możemy utożsamiać aktualne 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 cześć 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 również zakończą swoje działanie. (2) Aby lepiej zrozumieć zachowanie się interfejsu użytkownika należy zapoznać się z cyklem życia klasy Activity i Fragment.

## Cykl życia klas Activity i Fragment

# 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. 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 sob 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.

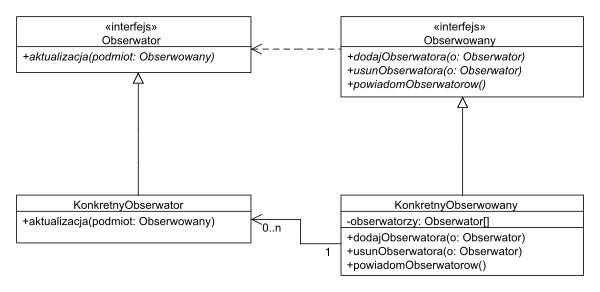


Diagram klas wzorca obserwator (https://tomasz-tomczykiewicz.blog/2017/03/28/wzorzec-projektowy-obserwator/)

Contents

[1. Wstęp 2](#_Toc532155159)

[1. Interfejs użytkownika w platformie Android 2](#_Toc532155160)

[1.1. Cykl życia klas Activity i Fragment 3](#_Toc532155161)

[1. Istota paradygmatu reaktywnego 4](#_Toc532155162)

[1.1. Elementy paradygmatu funkcyjnego w programowaniu reaktywnym 4](#_Toc532155163)

[1.1.1. Wzorzec obserwator 5](#_Toc532155164)

[Bibliografia 6](#_Toc532155165)

# Bibliografia

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

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