Отчет за период 16.03-25.03.
Запланировано задач к текущему дню: 12
Выполнено: 12
Всего выполнено: 12/36.
Общий процент выполнения плана: 33%
Остается недель до окончания: 5

К текущему дню познакомился с применением технологии реактивного программирования в других компаниях. Разобрал базовые типы RxJava, поработал с основными операторами, узнал больше о том, как работает смены потоков, что именно делают subscribeOn/observeOn и поработал с операторами обработки ошибок. Со следующей недели приступаю к углубленному изучение операторов преобразования. Также завел проект на GitHub, в котором пока изучаю на примерах Rx. https://github.com/Marche1os/RxPlay/

Отчет по дням:

День: 15.03.

Прочитал Intro to Rx (http://introtorx.com/Content/v1.0.10621.0/00_Foreword.html). Это оказалось полноценным курсом для .NET платформы, пришлось делать выжимку из обще-реактивных вещей. Не сказал бы, что прим большая польза от этого источника. Скорей, получилась обзорная статья по базовым классам (Observer/Observable), как работает Life management (Disposable). Всегда стоит отписываться от подписки, чтобы избежать утечек памяти. Также прочитал интересную мысль: “Garbage collector был создан для не того, чтобы управлять ресурсами, а чтобы управлять памятью”. Многие заблуждаются, думая, что можно не заботиться об освобождении ресурсов, т.к. GC все соберет. Но это не так.

День 16.03

Изучение статьи **https://en.wikipedia.org/wiki/Reactive_programming**. Почитал про философию реактивного программирования. Узнал о подходах, программных моделях.
Узнал также о следующих подходах:

- Императивный
- Объектно-ориентированный
- Функциональный
- Основанный на акторах
- Основанный на правилах

Просмотрел доклад от Netflix (https://www.infoq.com/presentations/netflix-functional-rx/)

Проектирование API в реактивном стиле предполагает большую гибкость в плане асинхронности. Можно возвращать необходимый источник данных и клиент сам решит, на каком потоке вызывать этот метод и на каком потоке обрабатывать результат.

День 17.03

Начал прохождение курса на StartAndroid. Прошел первый урок. Собственно была дана очень краткая информация о паттерне Наблюдать. Observable, observer. Пока что ничего нового для меня.

День 18.03

Приступил к прохождение второго урока. В нем приведены различные операторы, разбитые по категориям (операторы преобразования, объединения и т.д.), создал проект на гитхабе для изучения их работы на практике и для создания различных комбинаций операторов.
Ссылка на проект: https://github.com/Marche1os/RxPlay/

Прочитал статью про использование технологии в Netflix. Интересно, что у них своя расширенная реализация библиотеки, но в целом статья дублирует видео доклад, уже просмотренный.

День 20.03

Прошел третий урок. Узнал про метод CompositeDisposable, позволяющий собрать в список все подписки и разом их отменить. А так урок был коротким, больше чего-то нового не узнал.

День 21.03

Прошел четвертый урок. Поработал с операторами, создающие Cold/Hot observables. Попрактиковался с операторами replay, publish для создания горячих потоков данных. Дополнил проект на андроиде примерами этих операторов.

День 22.03

Прошел пятый урок. Впервые поработал с различными реализациями Subject, нашел их полезными для решения определенных задач. Дополнил проект примерами работ Subjects.

День 23.03

Прошел шестой урок. Улучшил понимание в операторах смены потока - **subscribeOn**, **observeOn**. Оператор **subscribeOn** задает поток, на котором выполнится создание цепочки => может находиться в любой позиции в цепочке вызовов. А **observeOn** переключает поток на тот, что передается в параметре. Важно использовать **observeOn** в правильном месте в цепочке вызовов, потому что после переключения потока все последующие операторы будут выполнены на

том потоке, на который переключились. Можно использовать этот оператор несколько раз в цепочке, суть его работы не поменяется -> все последующие операции будут выполнены на новом потоке (на который переключились). Это становится важным, когда у нас в цепочке есть запрос в сеть, после чего преобразование данных. Поход в сеть это IO операция, не сильно нагружающая процессор. А преобразование данных это уже вычисления, они сильнее нагружают процессор. Поэтому разумно будет использовать шедулер **IO** для похода в сеть, а преобразование данных переключить на шедулер **computation**.

День 24.03

Прошел седьмой урок. Познакомился с новыми для себя операторами обработки ошибок, улучшил понимание ретрая. Узнал при интересный оператор retryWhen, который принимает Observable с ошибкой и возвращает Observable ретрая, который в зависимости от вызова метода (onNext, onCompleted, onError), продолжает работу, завершает ее или прокидывает ошибку.

Также прочитал статью о том, как избегать утечек памяти в андроиде на Rx. В целом ничего нового и полезного.