WEEK 2 DAY 1

LUIS EDUARDO ENRIQUEZ PEREZ

ASYNCTASK

The *AsyncTask* class allows to run instructions in the background and to synchronize again with the main thread. It also reporting progress of the running tasks. *AsyncTasks* should be used for short background operations which need to update the user interface.

To use *AsyncTask* you must subclass it. The parameters are the following *AsyncTask <TypeOfVarArgParams, ProgressValue, ResultValue>*.

An AsyncTask is started via the *execute()* method. This *execute()* method calls the *doInBackground()* and the *onPostExecute()* method.

TypeOfVarArgParams is passed into the *doInBackground()* method as input. *ProgressValue* is used for progress information and *ResultValue* must be returned from *doInBackground()* method. This parameter is passed to *onPostExecute()* as a parameter.

The *doInBackground()* method contains the coding instruction which should be performed in a background thread. This method runs automatically in a separate *Thread*.

The *onPostExecute()* method synchronizes itself again with the user interface thread and allows it to be updated. This method is called by the framework once the *doInBackground()* method finishes.

RXJAVA

RxJava is Java implementation of Reactive Extension. Basically it's a library that composes asynchronous events by following Observer Pattern. You can create asynchronous data stream on any thread, transform the data and consumed it by an Observer on any thread. The library offers wide range of amazing operators like map, combine, merge, filter and lot more that can be applied onto data stream.

RxAndroid is specific to Android Platform with few added classes on top of RxJava. More specifically, Schedulers are introduced in RxAndroid (AndroidSchedulers.mainThread()) which plays major role in supporting **multithreading** concept in android applications. Schedulers basically decides the thread on which a particular code runs whether on background thread or main thread. Apart from it everything we use is from RxJava library only.

Even through there are lot of Schedulers available, **Schedulers.io()** and **AndroidSchedulers.mainThread()** are extensively used in android programming. Below are the list of schedulers available and their brief introduction.

- **Schedulers.io()** – This is used to perform non CPU-intensive operations like making network calls, reading disc / files, database operations etc., This maintains pool of threads.
- **AndroidSchedulers.mainThread()** – This provides access to android Main Thread / UI Thread. Usually operations like updating UI, user interactions happens on this thread. We shouldn't perform any intensive operations on this thread as it makes the app glitchy or ANR dialog can be thrown.
- **Schedulers.newThread()** – Using this, a new thread will be created each time a task is scheduled. It's usually suggested not to use schedular unless there is a very long running operation. The threads created via newThread() won't be reused.
- **Schedulers.computation()** – This schedular can be used to perform CPU-intensive operations like processing huge data, bitmap processing etc., The number of threads created using this scheduler completely depends on number CPU cores available.
- **Schedulers.single()** – This scheduler will execute all the tasks in sequential order they are added. This can be used when there is necessity of sequential execution is required.
- **Schedulers.immediate()** – This scheduler executes the the task immediately in synchronous way by blocking the main thread.
- **Schedulers.trampoline()** – It executes the tasks in First In – First Out manner. All the scheduled tasks will be executed one by one by limiting the number of background threads to one.
- **Schedulers.from()** – This allows us to create a scheduler from an executor by limiting number of threads to be created. When thread pool is occupied, tasks will be queued.

RxJava is all about two key components: **Observable** and **Observer**. In addition to these, there are other things like **Schedulers**, **Operators** and **Subscription**.

Observable: Observable is a data stream that do some work and emits data.

Observer: Observer is the counter part of Observable. It receives the data emitted by Observable.

Subscription: The bonding between Observable and Observer is called as Subscription. There can be multiple Observers subscribed to a single Observable.

Operator / Transformation: Operators modifies the data emitted by Observable before an observer receives them.

Schedulers: Schedulers decides the thread on which Observable should emit the data and on which Observer should receives the data i.e. background thread, main thread etc.,