Table of Contents:

1. Introduction
   1. Purpose
   2. Glossary
   3. Scope
   4. Requirements
2. System Overview
   1. Classes & Purposes
   2. Systems
3. Design Rules
   1. Naming
   2. Coding Style
4. System Interactions
5. Time Estimates

Introduction:

1. Purpose:   
   The purpose of “Exchange Tracker” is to track the current conversion rates of foreign coins (for example from GBP to USD). If the user has requested it, then the application will also notify the user via push notification that a conversion rate has reached a point that the user wants to cash it in.
2. Glossary:
   * **“Exchange Tracker”** – the application.
   * **“Fixer**.**io” –** the API used to get the conversion rates for a list of currencies.
   * **“Service”** – a part of an application that can run in the background without any user interface.
   * **“Rates”** – all currencies that are available in the application.
   * **“System”** – any entity (class, service, thread) that has a distinct role in the application design. For example if class A and B both handle displaying part of the rates to the user, they are both under the same system. On the other hand, if A is used to update the UI about changes in the rates, and B is used to fetch the rates; they are in two different systems (UI system, and Background system, subsystem network operations).
   * **“Subsystem” –** any entity within a system, which has a distinct role within that specific system. For example, if class A and B both exist in the Background system, and both handle downloading rates from Fixer.io, then they are both found in the network operations sub-system. On the other hand, if A is used to download the rates from Fixer.io, but B is used to broadcast the rates to all whom are interested, A is found in the network operation sub-system, and B is found in the broadcasting sub-system, both in the Background System.
3. Scope:   
   Exchange Tracker is meant to provide people who have sums of money in foreign currency to be able to monitor the rates of conversions, and get notified on the rates, and in turn help them find the best time to give back their money, to get the highest payback. **Note:** Exchange tracker does not provide local rate tracking, meaning that the prices and rates found in the application and through Fixer.io, aren’t a true representation of the amount that you will get for a sum of money, but the sum you will get assuming you trade directly with the banks (without fees).
4. Requirements:
   * Android API – API version 19 (kit-Kat) and above.
   * Internet – either Wi-Fi or cellular doesn’t matter.

System Overview:

1. Classes & Purposes:
   * **“TrackerActivity”** – The main activity, overall purpose is to provide the user with a responsive UI, the only activity in the application.
   * **“TrackerService”** – The service for the tracker, overall purpose is to provide a way to collect information from Fixer.io, and pass it on to the activity, assuming the activity is visible.
     + **“TrackerThread”** – A HandlerThread that runs by itself, it sleeps every 30 minutes and upon waiting up, it downloads rates according to available trackers. This thread is supposed to die once there are no trackers, and come alive when there are trackers.
   * **“TrackerInfo”** – An instance of a currency tracker, its purpose is to hold all the currencies that the user wants to convert to from a base currency, there can only be, the number of currencies available on Fixer.io, TrackerInfo objects in the application.
2. Systems:
   * UI – Contains “TrackerActivity”, its only purpose is to display information about trackers to the user.
   * Background – Contains all classes regarding background activity, downloading information from the internet, broadcasting results if needed, updating the UI via broadcasts.
     + Network Operations – “TrackerService -> TrackerThread” is the main entity in this subsystem. This subsystem is used to download information from the internet, specifically from Fixer.io, and other locations in the future if needed.
     + Broadcasting – A subsystem that comes alive once the UI is visible and not hidden. This subsystem is used to broadcast results to the UI, which in turn will display to the user the current conversion result. **Note:** once this system is not alive, it is converted to the Updates subsystem, which follows.
     + Updates – A subsystem that is alive as long as the UI is not visible, it is used to check if the user needs to be notified about the rate changes, and if that’s the case, this subsystem is used to send out a push notification.

Design Rules:

1. Naming:  
   All objects in the project must follow the following naming conventions;
   * Activities – (Name of activity)Activity (i.e. TrackerActivity).
   * Services – (Name of service)Service (i.e. TrackerService).
   * Threads – (Name of thread)Thread (i.e. TrackerThread).
   * Fields – m(Name of field) (i.e. mTracker).
   * Variables – (Name of variable) (i.e. tracker).
   * Methods – Name methods as descriptively as possible, so a method that converts an array of bytes into a TrackerInfo Object will not be named:   
     `convert(byte[] b)`, but a better and more descriptive name will be: `convertBytesToInfo(byte[] b)`.
2. Coding Style:
   * Brackets will be on the same line, for example:

public static void main(String[] args){  
 //code here…  
}

* + Method parameter names are to be as simple as possible, for example:

private void calculate(int a, int b){  
 //code here…  
}

* + Final fields must be all caps, for example:

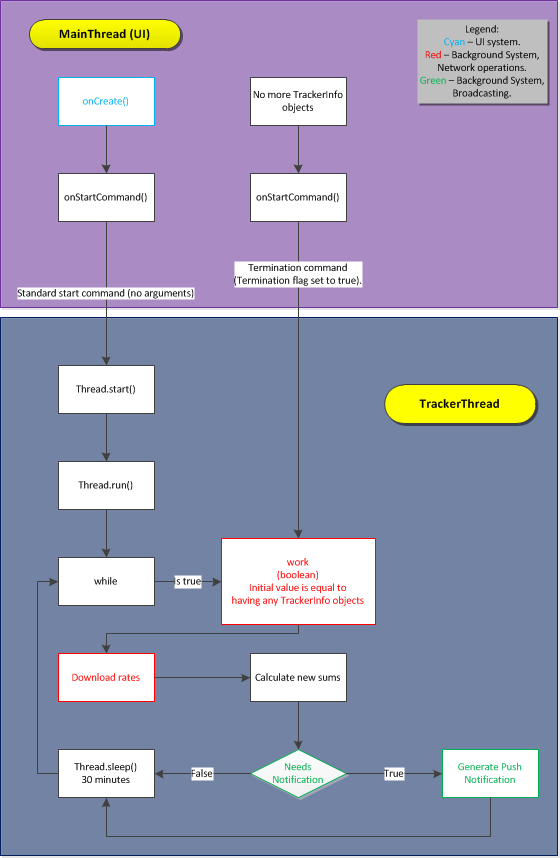
public static final String EXAMPLE = “This is a test string.”;

* + Use static variables as little as possible.
  + All constants must be in their own class (aka “Constants”).
  + Emphasize Modularity, extract as many methods as you can into other methods, for example:

private void processCommand(String cmd){  
 Object[] variables = parseVariables(cmd);  
 Object originator = getOriginator(cmd);  
 process(originator, variables);  
}

* + Document all methods and variables, even if it’s just one line of explanation, it’s better than trying to figure it out by myself. Additionally, there is no need to document getters and setters.

System Interaction:



Time Estimates:

­The time given to complete this application is:

96 Hours (4 days) of consecutive work, meaning that one hour can be spread over 2 days, each day having half an hour of work. The time frame in which the application should be completed is two weeks, meaning if the development is passed two weeks, it must be halted, no matter the progress.  
The starting date for development is 21/6/2017.