## Payment Tracker

Basic information:

Implemented program fulfills the task of tracking payments in different currencies as well as accepting new payments and displaying summary of those payments.

PaymentTracker was implemented as a Java Maven project using multiple threads. Program runs in two threads with modified run() method. First of them controls continual input entering and validates every input string. Second thread serves to display net amounts of each currency in regular intervals. Payments can be added by typing directly into the console or by providing an input file for the program. Either way, format of every new payment is strictly limited to 3 capital letters followed by space and a Integer number. Payment and List of given payments are represented in individual classes with specified parameters and methods. Data are kept in memory in different types of objects, not using any database engine.

When the program starts, user can type as many payments to the console as necessary. Once per minute (or if the timer is set differently), program will display sum of every valid currency grouped by their code. Only currencies where the sum value is different from 0 are displayed. Program can be easily modified to display currencies in any value limitations. Running the program from console, user can specify an argument (path to a input file), where more payments can be pre-specified in required format.

There were few assumptions and possibilities made to customize running of the program, specifically described in readme.txt file included to the project.

In addition to the optional bonus question, I managed to create a file with mostly used currencies in the world, specified with their conversion rate to USD. In the beginning, data from this file are loaded and later used in displaying sum of currencies to show conversion of each valid currency to USD (USD excluded) as described in bonus question. This list of currencies is also possibly used in validating currency code from input.

Program is also catching different exceptions in case of incorrect input format or invalid currency code. User is informed about incorrect input or other exceptions through a short eloquent error message. Despite the fact that we are expecting a correctly formatted input file, program is also catching incorrectly specified payments in input file and pointing them out for the user.

After typing string “quit” to the console, program will stop running immediately.

Tests:

Project contains 3 short and simple test cases for input of a payment. Two of those cases are trying to input a payment in incorrect format of either currency code or value of the payment. Third case is testing correct input of a payment. Each of these tests check if a payment in given format was added to the list of payments and therefore if payment format matched requirements.

How to run:

Considering user is running java program from console, in general it is necessary to locate the main source file and compile it using (in this case) “javac PaymentTracker.java” command. Runnable class will be created upon which user can run “java PaymentTracker” command to start a program. At this point, user can also specify an input file as program argument as requested in the assignment using “java PaymentTracker \*path\_to\_input\_file\*”. If the input file format was correct, pre-specified payments will be immediately displayed. With or without input file, user can start typing new payments right after program started to run.