Programming Assignment - Twitter Messaging Engine

(Very rough draft and example of incomplete requirements.)

## Overview/Motivation

This assignment is aimed at getting you acquainted with the real world of software engineering. You will find it necessary to use Slack to clarify any doubts you may have about the assignment. That is, if you don’t communicate with the class and instructor about the assignment, you will likely find it difficult to complete the assignment.

You have one week to complete this assignment; however, this deadline should be thought of as your “first draft”, which you will improve for week two after getting some potentially difficult feedback. Everyone is expected to do the actual programming for this assignment on their own.

In addition, you are expected to work on it every day (not just the night before it’s due) by using the git version control system (Bitbucket or a private GitHub repository shared with gkthiruvathukal). I will be discussing git commits and the importance of committing all work you do, no matter how big or small.

## Details

Your job is to develop a set of classes for processing Twitter-like messages and writing unit tests to show that your code works.

Everyone knows at least *something* about Twitter, unless you have been completely off grid for years, which likely doesn’t apply to most of us. For example, you can mention someone using the @ syntax. For example, @*person* would be a way of mentioning someone with a Twitter account. The account may or not exist, but this is unimportant for our assignment.

You can also mention a topic using the # syntax. For example, #topic is a way to mention the topic. The distinction between @ and # is not so important right now, but @ usually refers to actual Twitter users, whereas # can refer to any topic of cultural interest.

You can also include a URL using any proper URL. For example, <http://luc.edu>.

Twitter messages are limited to a certain number of characters.

As you can see, there are several things that we need to know about Twitter messages.

Your first job is to figure out the basic syntax of a Twitter message. You may use the Slack channel to discuss the syntax with your classmates. For example, one question you might have already is what can appear after @ and #? This is not specified here.

Once you have figured out how a Twitter message is formatted and what is permitted to appear within it, the next step is to write the code. Your code must make use of object-oriented design (i.e. classes). You will also likely need to use collections.

To keep this first assignment fairly focused, we are only interested in processing one message at a time. For example, suppose we had this message:

@franky goes to #hollywood. See <http://cnn.com>.

Your message parser must be able to support interfaces for being able to extract the mentions and hashtags used in the message. In this example, you would only find a mention of franky and a hashtag hollywood. There is also a URI <http://cnn.com>. Of course, this is a simple example. It can be more complicated, because there can be more than one of any of these.

Using the above, here are some tests I might perform:

* is mentioned “franky” => true
* is topic “hollywood” => true
* is referenced URL “<http://cnn.com>” => true
* is mentioned “george” => false
* number of mentions (regardless of who) is 1 => true
* number of topics is 1 => true
* number of URLs is 1 => true

Your job will therefore be to think of what interfaces your messaging class(es) must support. You may find a need to use collections interfaces (e.g. list of something, map of something).

Now for the hard part: You need to learn about unit testing. Your code is not going to be an application. It won’t even have a main method. It is just a set of model classes and a set of testing classes. We will talk about testing in greater detail as we go forward, but you will want to read my lecture on unit tests, which I give to COMP 170 students. See <http://introcs.courses.thiruvathukal.com/html/testing.html>.

Grading Criteria

This is a quick checklist of how you’re going to be graded for this assignment.

* significant number of commits for your project, preferably daily
* well-organized code
* appropriate documentation, especially to clarify your understanding of the requirements
* more unit tests than actual code
* good use of classes and objects and library classes (instead of reinventing your own), especially when it comes to core data structures (lists, tables/dictionaries, string handling, etc.)