CS 278

Code Review

Name: Sean Story

Review of the solution to Asgn1 implemented by Seth Friedman

Git Repo: https://github.com/sethfri/CS278

Path: /sethfri/CS278/tree/dev/impl/Asgn1/SodaCloudSMSExampleClient/src/com/example/sodacloudsmsexampleclient

This project is made of two pieces, a client and server, through which an android mobile device may connect to a web-hosted application, which will listen for and record SMS messages. The Soda framework is what both applications are built upon, allowing interfaces for easy websocket connections and SMS protocols. A Module interface is used to create mappings from interfaces to implementations, which will allow for eased dependency injection.

1. Refactoring 1:
   1. The code assumes that the reader knows how the application runs and functions.
   2. There is very little commenting in the code, either block comments, or inline. This leads to the reader being unsure of the workflow, or purpose of any files, without reading every line of every file. In a situation where the fellow student/grader/teammate was not familiar with the code, lots of time will be lost trying to generate understanding from the source code, rather than summary comments.
   3. I propose that you add comments at the head of each file, detailing the purpose of the class/interface in the file. You should also have brief header comments for each function for which the purpose is not immediately obvious. Finally, inline comments would not be amiss for steps in larger functions, or particularly long lines of code.
   4. 1) Provide header comments for all files

2) Provide header comments for all non-trivial functions

3) Provide inline comments for tricky parts of longer functions

3) provide inline comments for long/complex lines of code

1. Refactoring 2:
   1. There is duplicate code between the MainActivity and SMSBridgeActivity classes. Namely:

configuration\_ **=** **new** ModuleImpl**()**

configuration\_**.**setComponent**(**SMSManager**.**class**,** **new** SMSManagerImpl**(this));**

configuration\_**.**setComponent**(**ObjRefExtractor**.**class**,** **new** QRCodeObjRefExtractor**());**

* 1. This code duplication could lead to uncaught/unremembered problems if the strategy is changed, and one file changes, but the other does not. This could lead to inconsistent patterns in the code.
  2. Creating a factory to generate these common types of Modules would be optimal. This will allow you to simply call something like:

configuration\_ = FactoryClass.generateSMSandQRModule();

This would lead to only 1 line of duplicate code (which is trivial) and

would allow the project to be well set up to grow, should any more common Modules be needed.

* + 1. Create a new class for the Factory pattern
    2. Include a static function for creating the SMSManager and QRCodeObjRefExtractor ModuleImpl
    3. Replace the repeated 3 lines with the call to the new function.

1. Refactoring 3:
   1. The style for referencing private instance variables within a class should be consistent.
   2. There are inconsistencies in the reference to private instance variables in the code. Sometimes the variables are myvar\_, sometimes this.myvar\_, and sometimes they are this.myvar.

Examples are:

ExternalObjRefImpl: **return** **this.**objRef\_**;**

MainActivity: configuration\_ **=** **new** ModuleImpl**();**

ModuleImpl: **this.**classMap**.**put**(**type**,** component**);**

This inconsistent style can be confusing, as it might confuse the reader into thinking that different types of variables are being referred to.

* 1. By choosing a consistent style, your reader will never be uncertain as to when you are accessing or modifying private class variables. Since this is the goal, the best option would be to be as clear as possible at all times, and use only the “this” keyword. That way you are literally expressing that the variable belongs to a specific instance.
  2. .
     1. Replace all instances of <var-name>\_ with this.<var-name>
     2. Remove all underscores from references to this.<var-name>\_