Homework 4

To: CSCI5801, All students

From: Mr. Deng, Mr. Mittal, Mr. Wendt

Date:11/18/2015

Re: Homework Assignment 4

The Problem

We have a design for the GRADS, the customer wants it, and we need to build it.

The Assignment

Revise (as necessary) your design draft. Then, implement the SCRS design you handed in for Assignment 3. The customer insisted on Java, so Java it is.

Code for the interface and external data stores are available on the assignment web page (make sure you've downloaded the most recent version and keep your eye out for possible updates).

Make sure you pay attention to these and the rest of the document. If you code does not compile, run, and provide a compatible interface, you will be severely penalized in the grading.

You are required to follow the coding standard provided by us in an appendix to this document.

Detailed expectations follow below.

SCRS Implementation Notes

You will be expected to develop in Java. If you use any external jars (such as Google GSON), make sure you put them in a directory labeled `lib/`. You source files should be in a directory labeled `src/`.

Your code must implement the provided `SCRS` interface (if you know C++, this is the Java equivalent to a pure virtual abstract class), and the class that implements this must be named something reasonable. It must reside in the `edu.umn.csci5801` package. In order to fully test your code before submission (you should never expect it to work just because it compiles), you will want to create the database as described in the documentation posted with Assignment 3. (Especially to test corner cases.) Please also give us your versions of the data files, stored in a directory labeled `resources/`.

Do not change the signatures of any methods in the provided interface. You will need to write your own test driver (class with a Main method). Do not turn this in with your code.

You are required to write your own exceptions. These will extend either `java.lang.Exception` or `java.lang.RuntimeException`. You cannot simply throw an `Exception`, or `RuntimeException`. You will want to wrap exceptions that occur with your exception (using the copy constructor `Exception(Throwable t)`). For more information about exceptions, please see: http://tutorials.jenkov.com/java-exception-handling/index.html

Do not assume we will run on one operating system or another, one IDE over another, or that a specific file system exists on the system.

Deliverables

You are responsible for delivering the following as a single zip via Moodle:

- Implementation of SCRS, incorporating all feedback provided on the design assignment (the implementation must be located in the src/ directory)
- Any required libraries for your implementation (in the lib/ directory)
- A description of how to compile your code – we will be using our own ant script, but provide any additional instructions that are necessary for code compilation (**you should not instruct us to compile and execute your code in Eclipse or any other IDE**).

Peer evaluations should also be submitted. See the peer evaluation form description for instructions.

Asking Questions

If you have questions about the systems, post the question for discussion on the class forum or come in to office hours. If we are slow to respond, email the TAs and Professor (using the csci5801-help@cs.umn.edu e-mail).

Due Dates

Assignment Due: Tuesday, December 8th at 11:55pm via Moodle.

Note that you NEED to start this early or it will not get done in time. Make sure you and your group have set expectations appropriately. We recommend documenting this in a manner that all members can refer to it.

Peer Evaluation Due: Thursday, December 10th at 11:55pm via Moodle.

Appendix A: Coding Standards

Below is a list of coding conventions that are adopted from Apache Commons Net (http://commons.apache.org/net/code-standards.html) everything else not specifically mentioned here should follow the official Sun Java Coding Conventions (http://www.oracle.com/technetwork/java/codeconvtoc-136057.html).

1. Variables and Class/Interface/Enum names should use CamelCase with variable names starting with a lower case letter and Class/Interface/Enum names starting with an upper case letter. Moreover names should be easily readable, using long names over abbreviations.

2. Brackets should begin on the same line as the opening code, and end on a new line and should exist even for one line statements. Examples:

```
if ( foo ){
    // code here
}

try{
    // code here
}catch (Exception bar){
    // code here
}finally{
    // code here
}

while ( true ){
    // code here
}
```

3. Though it's considered okay to include spaces inside parens, the preference is to not include them. Both of the following are okay:

```
if (foo)

or

if ( foo )
```

4. 4 space indent. **NO tabs**. Period. We understand that many developers like to use tabs, but the fact of the matter is that in a distributed development environment where diffs are sent to the mailing lists by both developers and the version control system (which sends commit log messages), the use of tabs makes it impossible to preserve legibility.

   In Emacs-speak, this translates to the following command:

```
(setq-default tab-width 4 indent-tabs-mode nil)
```

5. Native linefeeds (svn:eol-style native) for all .java source code and text files. Platform specific files should have the platform specific linefeeds.

6. Descriptive JavaDoc comments **MUST** exist for all methods and classes. JavaDocs on data members is preferred and encouraged, but a standard comment (called an implementation comment) describing the data member on the line before it is acceptable here. As a general rule, if your code modifications use an existing class/method/variable which lacks a JavaDoc, it is required that you add it. This will improve the project as a whole.

For more information on how to write JavaDocs, please see:
http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

7. Blocks of code should have inline implementation comments to help with finding code quickly during maintenance. These should be there in addition to the JavaDoc comments above the method/class/member. For example:

```
/**
 * JavaDoc description here.
 * @param param1  describe param1 here.
 */
public void myMethod(String param1)
{
   //Process parameter
   ...

   //Do something else non-trivial
   ...

   //Do yet another non-trivial thing
   ...
}
```

It is generally bad practice to include these comments as trailing comments as it makes the code harder to read.

8. We would like to encourage you to make your code available under an open source license such as the Apache Software License. If you choose to do this, the license header (found here: http://www.apache.org/legal/src-headers.html) **MUST** be placed at the top of each and every file.

9. Import statements must be fully qualified for clarity.

```
import java.util.ArrayList;
import java.util.Hashtable;

import org.apache.foo.Bar;
import org.apache.bar.Foo;
```

And not

```
import java.util.*;
import org.apache.foo.*;
```