# CSCE 247 - Project 4 - Final Design & Implementation

**Due Date:** Sunday, April 21, 11:59 PM (via the course website)

**Overview**

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

You have three tasks in this assignment:

- The first is to take the feedback from your draft design and make any changes needed to address that feedback.
- Second, add sequence diagrams to solidify the dynamic design of your system.
- The third task is to take that design and implement it using Java. The customer insisted on Java, so Java it is.

Sample data is available on the assignment web page to assist in system development (make sure you've downloaded the most recent version and keep your eye out for possible updates).

Make sure you pay attention to the sample data 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 also required to follow the coding standard provided by us in an appendix to this document.

Detailed expectations follow below.

**GRADS Implementation Notes**

You will be expected to develop in Java (we will compile and run your code using Java 8). If you use any external jars, make sure you include them in a directory labeled `lib/`. You source files should be in a directory labeled `src/`. You should use a package structure mirroring that implied by the provided interface.

In order to fully test your code before submission (you should never expect it to work just because it compiles), you will want to add more data to the sample data and input files provided (Especially to test corner cases). Please also give us your versions of the data files, stored in a directory within the `src/` folder.

You are required to write your own exceptions, except in cases where an existing exception is appropriate (such as a `NullPointerException`). 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 (instead of hardcoding file paths, we recommend using the Java classloader to pull resource files from the classpath - http://www.mkyong.com/java/java-read-a-file-from-resources-folder/).

**Deliverables**

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

- Updated static structural design.
- Dynamic models (UML sequence diagrams). This section should contain sequence diagrams illustrating three of the major use cases of the system. One of those three must correspond to the generation of a progress summary. You may include more than three diagrams if you choose.
- Implementation of GRADS, incorporating all feedback provided on the design assignment (the implementation must be located in the src/ directory).
- Any required libraries for running your implementation (in the lib/ directory).
- A description of how to compile your code – **you should not instruct us to compile and execute your code in any particular IDE**. We recommend writing your own build script or generating one in an IDE (ant, mvn, etc.)

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

**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. Names should be descriptive and easily readable, using long names over abbreviations.

2. Bracketing style should be **consistent**. Brackets should either always begin on the same line as the opening code, and end on a new line (preferred Java syntax) OR begin and end on new lines (preferred C syntax). Brackets 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
}
```

4. 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

5. 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.

6. We would like to encourage you to make your code available under an open source license such as the Apache Software License or Mozilla Public License. If you choose to do this, all proper licensing standard must be followed.

7. 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.*;
import org.apache.bar.*;
```