IT 179 Program Grading Standards

Program grading in this course is holistic, considering all aspects of the program in assigning the grade rather than allocating points for various program elements. The goal is that the grade accurately reflects the value of the program from a real-world perspective and that you as a student recognize the importance of the various aspects of a program. Note that this is a fully detailed document that will apply throughout the semester as we learn new things. You currently don't know what an ADT is – you will learn that before you write a program where it matters.

The program grading criteria will take into account the following:

- Functionality: Does the program do what it is supposed to do and do it exactly as specified?
- Algorithm/Data Structure Accuracy: Are the algorithms and/or data structures you are required to implement correct implementations of the specified algorithms and/or data structures following all assignment instructions?
- Program/Class Design: Does the program reflect principles of good program and class design?
- Code Quality: Is the program code clear and easy to read? Does it reflect best practices?
- Efficiency: Does the program avoid obvious inefficiencies such as unnecessarily running code multiple times, storing large chunks data, duplicating data, reading files more than once, or using recursion where a simple loop will do?
- Commenting: Does the program have appropriate commenting?
- Special program instructions: Does the program follow all instructions for the assignment that do not fall under the categories above (if any)?

The following table shows how programs are assigned to letter grades based on performance in the various categories. In general, the letter grade is determined by the lowest category ranking, while position within the letter grade range is determined by looking at the performance in other categories. For example, an otherwise perfect program with no comments would receive a high C, while a program that also had issues with incorrect output on sample input would receive a low C. There are two exceptions to the rule that the program receives the letter grade reflective of the lowest category: 1) a program may receive an A (but not a 100) with minor issues in up to 2 categories other than Functionality and Algorithm/Data Structure Accuracy and 2) a program that falls to the B level in at least 4 categories including one of the first two will receive a C, and similarly one that falls to the C level in at least 4 categories including one of the first two will receive a D.

	A*	**B**	**C**	**D**	**F**
Functionality	Program works correctly with all reasonable input as well as all provided test cases.	Works correctly on any sample data provided, but may have one execution error on other data.	May have errors on sample data or have multiple errors, but the program does run and produce output that is recognizable as closely related to the intended functionality. Does not produce runtime errors on sample data.	Program runs and clearly has some functional relationship to what is required but has a variety of errors. Runtime errors may occur, even on sample data, but something needs to work	Has immediate run-time errors or has compile time errors or bears little relationship to the intended functionality.**

	A*	**B**	**C**	**D**	**F**
Algorithm and Data Structure Accuracy	All algorithms/data structures are implemented correctly and follow any specific program guidelines.	Required algorithms and/or data structures are recognizable in implementation, but may have one minor variation from the requirement	Required algorithms and/or data structures are recognizable but may have issues such as: data structure class includes functionality that is not part of the ADT or is missing operations that are standard parts of the ADT; algorithms may have more than one issue.	Required algorithms/data structures are incorrectly implemented, but do bear some resemblance to the requirement	Required algorithms/data structures are not present or bear little to no resemblance to the requirement
Design	Program and any classes are well-designed.	Program has 1 or more of: • some unnecessary duplicate code • public methods that should be private • inappropriate instance variables • methods that should be broken up further • methods that don’t make sense as units of code • classes that don’t represent a program entity with data	Program has public instance variables or does not break the program logic into functions or has a lot of duplicate code or has several of the issues from the B category in design.	N/A	N/A
Code Quality	Program code is clear and uses best practices.	Program has 2 or more of: • multiple poorly named variables, methods, or classes • break out of a loop • use of continue • return out of loops • return out of deeply nested ifs • very poorly formatted code	Has many of the issues from the B category so that code is very difficult to follow.	N/A	N/A
Efficiency	There are no obvious inefficiencies in the program.	Program has at least one of the listed obvious inefficiencies	Program has multiple efficiency issues that keep it from running in a reasonable time.	N/A	N/A

	A*	B	C	D	F
Commenting	Program has appropriate Javadoc comments on all public elements as well as useful comments regarding code implementation.	Program is missing significant commenting or has consistent issues with commenting (such as missing @param or @return elements on many methods that need them).	Program has no comments.	N/A	N/A

*This column contains summaries only. A full description of the expectations for excellent (A) performance in each category is provided below.
**The first program submission with compile-time errors will receive an automatic F (a 50 if there is reasonable substance to the submission). Any subsequent program submissions with compile-time errors will receive an automatic zero.

Full Description of Excellent Performance in Each Category

<u>Functionality</u>

- Program works perfectly on all reasonable data.
- If sample output is provided, it matches perfectly, including formatting.

<u>Algorithm/Data Structure Accuracy</u>

- Any required algorithms are implemented correctly according to the approach specified in class and any specific requirements specified in the assignment.
- Any data structures you are required to implement from scratch are implemented in appropriate classes that correctly reflect the ADT.
    - Only the appropriate data structure operations are public
    - The ADT does not include data or functionality specific to the application.
    - All program instructions regarding implementation are followed.

<u>Program/Class Design</u>

- Classes follow standard principles of good object-oriented design
    - Only instance variables necessary to represent the state of an object are present. Any other variables in a class will be local to the method(s) where they are used.
    - Instance and class variables are private unless they need to be protected.
    - Methods that exist to support other methods of the same class (helper methods) are private unless they need to be protected.
- Methods do one thing, and are appropriately broken down if that one thing can be divided into subtasks that would work well as methods.
- Methods are broken into units that make sense and depend on one another as little as possible.
- Code is not duplicated unnecessarily.
- Variables are generally declared within the smallest possible scope.

Code Quality

- Names of classes, methods, parameters, variables, constants and any other identifiers are meaningful and clearly describe their purpose.
  - Exceptions are conventional names such as `i` and `j` used for integer counters.
  - Names are capitalized in accord with a standard convention: specifically all caps for constants; Pascal case for classes, camel case for other identifiers.
- Code files are appropriately named reflecting their purpose. For example, QueueSimulation.java or Customer.java. Generic meaningless file names like Main.java or Test.java or Driver.java are not used, and names of files containing main reflect the purpose of the program (which is typically not to test a class).
- Code is written as clearly as possible, minimizing the need for explanatory comments.
- Loops are designed to make code clear and easy to understand.
  - They are written with a single entrance and exit.
  - No returns or breaks out of loops.
  - Loops do not use `continue`.
  - When writing loops, use `for` when processing collections (arrays, lists, etc.) and expecting to go through the entire collection in the usual case, and use `while` when you expect to stop before getting to the end of the collection.
- Functions typically have no more than one return statement.
  - Exceptions are functions that consist of a linear nested if else with a return from each branch (as with simple recursive functions).
  - Functions do not return out of multiple layers of control structures, such as nested ifs or switches inside ifs or loops.
- Well-written code avoids cumbersome expressions, particularly when they reduce code clarity. Examples:
  - Excellent code uses `if (isValid)` or `while(!done)` rather than `if (isValid == true)` or `while(done == false)`.
  - Excellent code uses `return numLines % 10 == 0;` rather than `if (numLines % 10 == 0) return true; else return false;`
- Code is well-formatted, using correct indentation and employing blank lines to break code into sections, separate methods, clarify what code comments belong with, and so forth.

Efficiency

- Code does not have unnecessary inefficiencies.
- Examples of issues would be unnecessarily:
  - reading files more than once
  - storing large amounts of data
  - storing multiple copies of data
  - executing functions extra times
  - looping through arrays or other collections extra times
- Using recursion in place of simple loops would also be an issue.
- Note that programs should not go to great lengths to be efficient at the cost of clarity. I am simply looking for program to avoid inappropriate inefficiencies.

Commenting

- All classes have correct class comments including at least a brief description of the class (data or program) and an @author clause with the author’s full name.
- All public methods have correct Javadoc style comments with @param and @return clauses as appropriate.
- Private and protected methods have non-Javadoc comments with brief descriptions of the method
- In addition, minimal commenting is included where helpful to increase code clarity.