Project 3: Grading Rubric

No Credit

- Non submitted assignments
- Late assignments after 48 hours of original due time (with or without tokens)
- Non compiling assignments
- Non-independent work
- "Hard coded" solutions
- Code that would win an obfuscated code competition with the rest of CS310 students.

How will my assignment be graded?

- Grading will be divided into two portions:
  - Manual/Automatic Testing (75%): To assess the correctness of programs.
  - Manual Inspection (25%): A checklist of features your programs should exhibit. These comprise things that cannot be easily checked via unit tests such as good variable name selection, proper decomposition of a problem into multiple functions or cooperating objects, overall design elegance, and proper asymptotic complexity. These features will be checked by graders and assigned credit based on level of compliance. See the remainder of this document for more information.
- You CANNOT get points (even style/manual-inspection points) for code that doesn't compile or for submitting just the files given to you with the assignment. You CAN get manual inspection points for code that (a) compiles and (b) is an "honest attempt" at the assignment, but does not pass any unit tests.

Manual/Automated Testing Rubric

For this assignment a portion of the automated testing will be based on JUnit tests and a manual run of your program. The JUnit tests used for grading will not be provided for you (you need to test your own programs!), but the tests will be based on what has been specified in the project description. A breakdown of the point allocations is given below:

Overall Points Breakdown

55 pts	FCNS Tree
10 pts	Stack
10 pts	Queue

FCNS Tree Points Tentative Breakdown (subject to change)

8 pts	Build Tree
8 pts	Build Binary Tree
8 pts	Evaluate (Recursive)
5 pts	Evaluate (Iterative)
18 pts	PreFix, PrettyInFix, PostFix, and level-order traversals
8 pts	Other miscellaneous methods

Note: You CANNOT get points (even if you pass the JUnit tester) if

- Your solution is hard-coded. For example, always return true without any checking.
- Your solution is not following the requirement. For example, a recursive implementation is provided when a non-recursive implementation is required.

Manual Code Inspection Rubric

Inspection Point	Points	High (all points)	Med (1/2 points)	Low (no points)
Submission Format (Folder Structure)	10pts	Code is in a folder which in turn is compressed as a zip file. Folder is correctly named.	Code is not directly in user folder, but in a sub-folder. Folder name is correct or close to correct.	Code is directly in the zip file (no folder) and/or folder name is incorrect.
JavaDocs	5pts	The entire code base is well documented with meaningful comments in JavaDoc format. Each class, method, and field has a comment describing its purpose. Occasional in-method comments used for clarity.	The code base has some comments, but is lacking comments on some classes/methods/fields or the comments given are mostly "translating" the code.	The only documentation is what was in the template and/or documentation is missing from the code (e.g. taken out).
Coding conventions	5pts	Code has good, meaningful variable, method, and class names. All (or almost all) added fields and methods are properly encapsulated. For variables, only class constants are public.	Names are mostly meaningful, but a few are unclear or ambiguous (to a human reader) [and/or] Not all fields and methods are properly encapsulated.	Names often have single letter identifiers and/or incorrect/meaningless identifiers. [Note: i/j/k acceptable for indexes.] [and/or] Many or all fields and methods are public or package default.
Big-O Requirements	5pts	The Big-O requirements for all listed methods are met and/or exceeded. Code is very efficient.	The Big-O requirements are sometimes met and sometimes not. Code could be more efficient.	The Big-O requirements are never (or almost never) met. Code is extremely inefficient.