Project 2: Grading Rubric

REMINDER: No Credit for…

- Non submitted assignments, Non compiling assignments, Non-independent work, "Hard coded" solutions, Code that would win an obfuscated code competition with the rest of CS310 students, etc.
- Assignments late by more than 48 hours (*without exceptions*)
- Code that violations the "you may not" restrictions ← Reminder: this means array usage will automatically result in a score of 0!

How will my assignment be graded?

- Grading will be divided into two portions:
  - Manual/Automatic Testing (85%): To assess the correctness of programs.
  - Manual Inspection (15%): 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. 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.
- "Off the top" points refer to items that will lose you points rather than earn you points
- Extra credit for early submissions:
  - 1% extra credit rewarded for every 24 hours your submission made before the due time
  - Up to 5% extra credit will be rewarded
  - Your latest submission before the due time will be used for grading and extra credit checking. You CANNOT choose which one counts.

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 and the comments in the code templates. A breakdown of the point allocations is given below:

5 pts	Customer
20 pts	ThreeTenList
20 pts	ThreeTenQueue
20 pts	TarmartLine
15 pts	ThreeTenPriorityQueue
5pts	Tarmart main() produces ***exact*** output

Manual Code Inspection Rubric

"Off the top" points are items that will lose you points rather than earn you points.

Inspection Point	Points	High (all points)	Med (1/2 points)	Low (no points)
Submission Format (Folder Structure)	5pts ("off the top")	Code is in a folder which in turn is in 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.
Coding Conventions and JavaDocs	5pts (in ADDITION to the automatic check)	The entire code base is well documented with meaningful comments in JavaDoc format. Occasional in-method comments used for clarity. [AND] Code has good, meaningful variable, method, and class names.	The code base has some comments, but is lacking comments on some classes/methods/ fields or the comments given are mostly "translating" the code. [AND/OR] Names are mostly meaningful, but a few are unclear or ambiguous (to a human reader)	The only documentation is what was in the template and/or documentation is missing from the code (e.g. taken out). [AND/OR] Names often have single letter identifiers and/or incorrect/meaningless identifiers. (Note: i/j/k acceptable for indexes.)
Big-O Requirements	10pts	The Big-O requirements for all (or almost all) 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.
Code Reuse	5pts	The ThreeTenQueue, TarmartLine, and ThreeTenPriorityQueue implementations are correctly reusing code from the parent class and not re-implementing the parent code.	The ThreeTenQueue, TarmartLine, and ThreeTenPriorityQueue implementations are rewriting one or two methods rather than using the parent class implementations.	Little or no code reuse is evident in ThreeTenQueue, TarmartLine, or ThreeTenPriorityQueue