# Project 1: Grading Rubric

***No Credit***

- Non submitted assignments
- Assignments late by more than 24 hours (*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 and the comments in the code templates. A breakdown of the point allocations is given below:

| | |
|---|---|
| 3 pts | StringTimer |
| 3 pts | SubstringCounter |
| 22 pts | DynamicArray |
| 18 pts | DynamicGrid |
| 29 pts | Table |

***Manual Code Inspection Rubric***

"Off the top" points (i.e. items that will lose you points rather than earn you points):

- Submission format (-5pts or -2.5pts, see below)
- Code formatting/layout (-5pts or -2.5pts, see below)

| 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. |
| Code Formatting | 5pts ("off the top") | Code has a set indentation and formatting style which is kept consistent throughout and code looks "well laid out". | Code has a mostly consistent indentation and formatting style, but one or more parts do not match. | Code indentation and formatting style changes throughout the code and/or the code looks "messy". |
| 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.<br><br>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)<br>[and/or]<br>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.)<br>[and/or]<br>Many or all fields and methods are public or package default. |
| Code Reuse (especially in DynamicGrid and Table) | 5pts | Code is reused rather than copied. | There is some code reuse, but other code is copied and pasted which could have been reused. | There is no code reuse apparent. |
| Big-O Requirements | 10pts | The Big-O requirements for 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. |