Project 0: Grading Rubric

For each project you will get a file like this that outlines (1) very specific things that receive "no credit", (2) details on how your assignment will be graded, (3) a point breakdown, and (4) any manual grading that your GTA will perform after your submission. This file is an example of what you might see.

No Credit

- Non submitted assignments
- Non compiling assignments
- Non-independent work
- "Hard coded" solutions

How will my assignment be graded?

This project is a little different than other projects in this course. For one thing, it will be using an automated script to grade and the GTAs will not be manually checking anything. Other assignments will have manual grading components (an example is shown on page two of this document). This assignment can also be submitted multiple times after the deadline for regrading, but this is not true for any other project in this course. You also cannot use emergency tokens for this project since you can submit again after the deadline without penalty. One last time: ***this is not true for any other project in this course, other projects have strict deadlines which must be adhered to for credit***.

Automated Testing Rubric for Project 0

Normally this section contains information about how many points are in each section of the project (e.g. 5pts for Part 1, 10pts for Part 2), but for this project you need to do all of the following to flip a single boolean flag. As mentioned in the syllabus, this "flag" allows you to get credit for other projects this semester. If you don't flip the flag the final resubmission date (see the schedule), all your other project grades will be set to 0!

ALL of the following need to be true to flip the boolean flag for this project:

1. You must submit a zip file following the naming conventions exactly as described on Page 1 of the project description.
2. Your zip file must contain a folder named exactly as described on Page 1 of the project description (your user folder).
3. Your user folder must contain your java files directly in the folder as shown on Page 1 of the project description.
4. All the java files in your user directory need to compile with the command `javac *.java` from within your user directory *without errors or warnings*.
5. All the java files in your user directory need to pass the `checkstyle` tests for code style.
6. All the java files in your user directory need to pass the `checkstyle` tests for JavaDocs.
7. Your program cannot do anything from the "You may NOT" part of the assignment description.
8. All the java files in your user directory need to compile with the unit tests (this should not be an issue if you follow the assignment instructions).
9. `OneItemBag` must pass our automated unit tests. These will be ***similar*** to the code shown in `BagUsageDemo`.
10. `BananaCoconut` must pass our automated unit tests. These will be ***similar*** to the tests in `BananaCoconutTest`.

EXAMPLE Manual Code Inspection Rubric

Most projects will have something that looks like the rubric below for the GTAs to grade with for the manual grading components. "Off the top" points are items you "lose points for" instead of earning points for. In the below example, submission format and code style can "take off" up to five points each, whereas doing JavaDocs correctly "earns" ten points. This means that if your project is 100%, working but you don't put it in the folder correctly, you'll only get a 95.

EXAMPLE Inspection Point	Points	High (all points)	Med (1/2 points)	Low (no points)
EXAMPLE 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.
EXAMPLE Code Style Basics	5pts ("off the top")	Code passes all checks for cs310code.xml [and] Code has a set indentation and formatting style which is kept consistent throughout and code looks "well laid out".) [and] Code has good, meaningful variable, method, and class names.	Code passes all checks for cs310code.xml [but] Code looks "messy" and/or names often have single letter identifiers and/or incorrect/meaningless identifiers. (Note: i/j/k acceptable for indexes.)	Code does not pass cs310code.xml checks
EXAMPLE JavaDocs	10pts	Code passes all checks for cs310comments.xml [and] 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.	Code passes all checks for cs310comments.xml [and] The code base has some comments, but is lacking comments on some classes/methods/fields or the comments given are mostly "translating" the code.	Code does not passes all checks for cs310comments.xml [and/or] The only documentation is what was in the template and/or documentation is missing from the code (e.g. taken out).