General Instructions for Grading Programs
(instructions for the grader)

Unless instructed otherwise, grade programs based on:

- Correctness
- Process
- Style
- Documentation

*** For any points deducted, put a CONSIDER tag saying how many points deducted and why. ***

For *some* assignments (whatever your instructor announces), the student can "earn back" points, as follows. Within 3 days after received the graded assignment, the student can (for any error, as indicated by a CONSIDER comment):

- Correct the error
- Change the comment from CONSIDER to REGRADE

The grader will re-grade anything so marked and update the score appropriately.

Students: "earn back" is a privilege – don't abuse it. Put forth your "good faith" effort on the project and reserve earn-back for errors that you did not anticipate.

Correctness: For each item marked (per the grading instructions specific to the assignment):

- Full credit: Behaves as specifies
- 80%: One "very small" error
- 40 or 50%: One "small to medium-sized" error
- 0%: A "large" error or more than one error

Process: Students should use:

- Documented stubs for each method
- Unit tests written *before* implementing the method, when unit tests are used
- UML class diagrams written *before* implementing the classes, when UML is used.

Grader: You need not "grade" the students' use of process (since it is hard to do so), but we reserve the right to reduce a student's score if he is not using the above process.

Style: Deduct 10% of the assignment's total for each of the following (but limit the total deducted for style to a maximum of 40% of the assignment's total):

1. Control-Shift-F on the file causes any significant change. Deduct 10% for EACH such change.
   - Eclipse highlights in purple the line numbers of lines that change when you do Control-Shift-F. Grader, you can put a single CONSIDER that says something like "-30% for 3 errors exposed by Control-Shift-F" – you don't have to explain any further, since the student can do Control-Shift-F herself to see the changes.
2. Any TODO's are left (unless they are intentional).
3. Any warning messages are left (students should use @SuppressWarnings if the source of the warning is intentional).

- Any missing documentation will generate a warning message. Deduct as a style or documentation error (but not as both).

4. Variable, method or class names are chosen poorly.
5. A method is too long (i.e., it should have been broken up into sub-method calls).
6. A statement is too long (i.e., an intermediate variable should have been introduced).
7. A variable is introduced where the code is clearer without it. For example:

```
int answer = ... blah ...;
return answer;
```

   is clearer when written simply as:

```
return ... blah ...;
```

8. A variable has greater scope that necessary.
   - For example: a variable that could be local to a method is instead a field, or a *for* loop variable is not local to the *for* loop.
9. White space is used poorly (Control-Shift-F catches most of these errors).
10. Any other poor style that you notice. (Let me know what you take off per this "other poor style" bullet so that I can add it to the above list.)

Documentation: Deduct 10% of the assignment's total for each of the following (but limit the total deducted for documentation to a maximum of 40% of the assignment's total):

1. Class does not have a reasonable description.
2. Public or protected method or field does not have a reasonable description.
3. Missing or extra tags.
4. Tags without a reasonable description, e.g. @param with nothing after it.
   - Exception: if the name of the parameter is self-documenting, nothing is needed after the @param (e.g. @param ballToAdd is self-documenting).
5. Undocumented (or poorly documented) private methods whose name and parameter names do not make it clear what the method accomplishes.
   - Such comments should rarely be necessary, since method and parameter names should generally be chosen to make the method self-documenting.
6. Ditto for private fields.
7. Chunks of code which are especially long and/or obtuse, yet lack an appropriate in-line comment. (Such comments should rarely be necessary.)