

PROGRAMMING STANDARDS

An important aspect in measuring your understanding of data structures is measuring your ability to implement and use them. In general, a major or minor in computer science must demonstrate the ability to successfully complete programming assignments of moderate complexity. The programming assignments of this course are designed to confirm this skill as well as measure your mastery of the subject matter. A well-written program should meet the following standards (note that a supplementary description is given on pages 100-104 of your text)

1. PROPER VARIABLE DECLARATION

The only variables to be declared outside of a function/method definition are instance variables that represent general properties of objects. Otherwise, use parameter passing to transmit data between functions. In addition, instance variables should be declared as *private* and define accessor and mutator functions as needed.

2. EXPLICIT INITIALIZATION OF VARIABLES

Every variable should be given a value before it is used. You can give a variable a value either through an assignment statement or an input statement.

3. NO UNUSED PARAMETERS

Only pass needed parameters to a function.

4. NO UNUSED VARIABLES

Every declared variable should have a specific purpose.

5. GOOD PROGRAMMING STYLE

The goal in programming is to write clear, readable programs which are as efficient as possible. In a case where efficiency and clarity are at odds, clarity should win out. In general, a program should be written so that a person who had not seen the problem before should be able to read and understand the code. The following are general style guidelines for writing Java programs.

- Comment extensively
 - Every program should have a comment block at the very top giving the author's name, class and section number, program number, date, and a description of the program. The program description should include a detailed description of the problem being solved, a high level description of the solution algorithm, and a description of the I/O format, error handling, and program assumptions.
 - Each function should also have a comment block describing what it does. This comment block should also describe the purpose of each parameter.
 - Every constant or variable declaration should be commented, specifying what that memory cell represents.
 - Any long or difficult to read block of code should also be commented. For example, every loop and every IF-THEN-ELSE branch should be commented, as well as any other "tricky" pieces of code.
- Use indentation to make the structure of the code more apparent. Indentation style varies, but be consistent. Be sure to use at least two spaces to separate levels of indentation.
- Always use meaningful variable names and symbolic constant names that reflect the use of the memory cell. Capital letters should be used to clarify multi-word identifiers, such as **nameOfStudent**. **But be careful!** Java on UNIX-systems is case sensitive.
- Use functions to break up the code into meaningful, easy to read chunks. Try to avoid making a function longer than one page.

6. TEST PLANS

When you turn in a program you must also turn in a 1 or 2 page written description of the tests you used to determine the correctness of your program. Your description should explain the purpose of each set of input values you used in testing your program. This description will count for 5% of your grade on each program.

7. EXTRA CREDIT

I reserve the right to award students bonus points on any program for any work that "goes the extra mile." Possible examples of such work includes

- Extra error handling
- Extra-informative presentation of the output
- Extra processing options

The number of bonus points depends on the correctness and the degree of innovation and independent thinking shown. To claim extra credit, it should be described in detail in the header documentation at the beginning of the program, headed by **EXTRA CREDIT CLAIM**.

8. LINUX SYSTEM RULES

- Program Submission: for each program you must submit

1. A written test plan submitted either on paper or electronically.
2. A paper copy of your program source file(s) (the **.java** file(s)).
3. An electronic submission of the source file from our departmental machines. You will use the following command for submitting your programs.

```
submit <assignment_number> <file_name>
```

where <assignment_number> will be an integer representing which assignment it is, and <file_name> is the name of your source file. After you type in this command and hit RETURN, you should see the following message:

```
File '<file_name>' was copied successfully.
```

In the case where you submit a file more than once, you will see a prompt asking if you want to overwrite the earlier submission. You may respond with *y* or *n* as appropriate.

4. In order for the above submit command to work, you will need to add [/home/INTRA/smithron/3310/bin](#) to your path or else prepend that text to your submit command.

- Help Submission: You may sometimes run into a problem in working on a program where you might try to get help from me when it is not my office hours. To do this, you should use the *submit* command with an assignment number of 0. Use comment lines at the top of your source file to describe the problem. Be sure to put the word **HELP** on the first line. In addition, you should send me an email at rws@cs.ecu.edu indicating you have submitted a query. Another common courtesy is to send me an email saying you've solved the problem on your own if you have not yet heard from me. This keeps from wasting my time looking at a problem you've already solved. Depending on when you submit your request it may be several hours before I am able to examine it.