GENERAL INSTRUCTIONS AND REGULATIONS FOR ASSIGNMENTS

COMP-202B, Summer 2011

PREAMBLE

All assignments that students will submit as part of this course are subject to the instructions and regulations specified in this document.

Assignments specifications **MAY** impose additional instructions and regulations, or overrule any of the following instructions or regulations; all such cases will be explicitly mentioned in the relevant assignment specifications.

In cases where an assignment specification contains instructions or regulations which conflict with those listed in this document, the instructions or regulations contained in the assignment specification shall have precedence.

SUBMISSION PROCEDURE

All assignment submissions **MUST** be sent to the graders via the appropriate myCourses submission box. In particular:

Assignments not submitted through mycourses, including those sent to instructors or TAs by email, will not be graded unless otherwise agreed upon with the instructor beforehand.

GENERAL SUBMISSION GUIDELINES

- Make sure you always submit the .java file rather than the .class file. If you submit the .class file, the TA will be unable to grade your assignment because he/she can not see what you have written.
- Any question with a written answer should be in a plain text (i.e. txt) file. No word documents or pdfs.
- You must follow the case-sensitivity requirements given on the assignment. For example, if the assignment says to write a class `HelloWorld`, you need to write the class `HelloWorld` and not `HELLOWORLD`.

ILLEGAL COLLABORATION AND USE OF SOURCE CODE

Assignments **MUST** be done **INDIVIDUALLY**; you **MUST NOT** work in groups, you **MUST NOT** copy any student's submission or any part thereof, and you **MUST NOT** allow another student to copy your submission or any part thereof.

In addition, unless otherwise stated, all source code you submit **MUST** have been written **ENTIRELY** by you; you **MUST NOT** submit source code which has been written in whole or in part by any other party, even if this source code is in the public domain or you have permission from its author to use and/or modify it in your work (obviously, this prohibition does not apply to source code supplied by instructors explicitly for this purpose).

Performing any of the preceding actions constitutes illegal collaboration and/or use of source code, which is a serious violation of these instructions and regulations and will not be tolerated.

Graders will randomly check submissions for suspicious similarities. Additionally, instructors will use automated software similarity detection tools to compare each submission to every other submission. However, note that these tools will be used solely to determine which submissions should be manually compared for similarity; instructors will **NOT** accuse anyone of illegal collaboration based solely on the output of these tools. Finally, instructors can request at any time that you explain the workings of any source code you submit for an assignment.

Students caught collaborating illegally and/or using source code they did not write themselves illegally on any assignment question will be formally accused of academic fraud and their cases will be referred to the appropriate university officials for disciplinary action.

RESTRICTIONS

Every program you submit **MUST AT LEAST** compile and run using the Sun/Oracle JDK 6 installed on the PCs found on the third floor of Trottier building.

There are some cases where it is OK to experiment with more advanced features of Java. There are other cases where it will not be. Generally, it is best to stick to what was presented so far in class, unless your instructor has specifically given permission for it. If you have any doubts about whether what you are doing is allowed for the assignment, please consult Dan before the assignment is due.

When provided source code, you **MUST NOT** change any part which you have not been explicitly permitted to change by the assignment specification.

PROGRAMMING STANDARDS

The following are general guidelines that you should follow when coding in Java. At the beginning, we won't enforce them as much, but will have higher expectations as the semester progresses.

INPUT/OUTPUT

- Output **MUST** be nicely spaced and easy to understand. In particular, the user of your program **MUST** be able to understand the output **WITHOUT** looking at the source code of your program. This implies that for each value you display, you **MUST** display a short message which explains briefly the meaning of this value.
- Before your program reads a value from the keyboard, it **MUST** display a prompt describing what the user is required to enter, the meaning of this value, which values are acceptable, and/or which values are illegal.
- If a sample session is provided in the assignment specification, your program must match the output format of the sample session **EXACTLY**.

IDENTIFIERS: VARIABLE, METHOD, AND CLASS NAMES

- Identifiers for variables and helper methods you write (that is, those whose implementation is not required by the assignment specification) **MUST** be as meaningful as possible and follow the standard upper-case/lower-case conventions. That is:
  - Variable names **MUST** be entirely in lower-case, except for the first letter of each word in the variable name other than the first word; those letters **MUST** be upper-case letters. There **MUST NOT** be any characters between the last letter of a word within a variable name and the first letter of the next word within the same variable name. Examples: `counter`, `myNumber`, `myOtherNumber`.
  - Method names **MUST** follow the same convention as variable names. In addition, the first word in a method name **SHOULD** be a verb. Examples: `execute()`, `isCalculationComplete()`, `getThisVariable()`.
  - Class names **MUST** be entirely in lower-case, except for the first letter of each word in the class name, including the first word, which **MUST** be upper-case letters. There **MUST NOT** be any characters between the last letter of a word within a class name and the first letter of the next word within the same class name. Examples: `Example`, `MyClass`, `MyOtherClass`.
  - Constant names **MUST** be entirely in upper-case. There **MUST** be **ONE** underscore (_) between the last letter of a word within a constant name and the first letter of the next word within that constant name. Examples: `CONSTANT`, `ANOTHER_CONSTANT`, `YET_ANOTHER_CONSTANT`.
- You **MUST** follow the method signature contracts described in the assignment specification for each required method's access modifier, return type, name (including case-sensitivity) and order and types of the parameters it accepts; in other words, if the assignment specification asks you to write the body of a method with signature `public void myMethod(int i, double d)`, then the method you write **MUST** be `public`, return `void`, be called `myMethod` (`mymethod`, `MYMETHOD`, or any other name is **NOT** acceptable), and accept as parameters an `int` and a `double` in that order. This requirement is imposed in order to allow instructors and graders to develop programs which call methods that you write, such as automated testing programs used to assist in the grading of student submissions.
- Likewise, and for the same reasons, you **MUST** follow the class name contracts described in the assignment specification, including case-sensitivity. For example, if the assignment specification asks you to write a class called `MyClass`, the class you write **MUST** be named `MyClass`; it **MUST NOT** be named `myclass`, `MYCLASS`, or any other name.
- Only declare variables your program actually uses.
- Do **NOT** use the same variable for different purposes. In particular, do **NOT** overwrite the value of input variables (whether they are parameters to a method or variables in which your program stores the values it reads from the keyboard).

PROGRAM STRUCTURE

- Good structure is important. You **SHOULD** decompose your methods into meaningful sub-methods whenever this improves the clarity of your program. Also, you **MUST** avoid copying and pasting code fragments if it is possible to turn them into a helper method.
- Code you submit **MUST** be indented in a systematic way that reflects how its statements are nested. You will be taught in the lectures and/or in the tutorials how to properly indent your programs. You can also consult the reference program for an example of how programs you submit **SHOULD** be indented.
- Your programs **MUST** strictly separate user interface code (the code which handles input and output) from application code (the code which actually performs the computations required by the assignment specification). In particular, if user interface code and application code are part of the same method

(`main()`, for example), you **MUST NOT** start computing **ANY** of the results required by the assignment specification before **ALL** necessary inputs have been entered by the user. Additionally, you **MUST NOT** display **ANY** of the results required by the assignment specification before **ALL** required results have been computed.

- Ideally, application code **SHOULD NOT** be placed in the same method or class as user interface code (however, note that this recommendation does not apply until methods have been covered in the lectures).
- Loops **MUST** terminate as soon as possible; for example, a loop which verifies whether an array contains a given value **MUST** stop as soon as it encounters that value, without looking at any further values in the array.
- Nothwithstanding the previous point, the use of reserved words `break` and `continue` to control the execution of loops is **STRONGLY DISCOURAGED**, as it makes the stopping condition of loops unclear.

ACCESS MODIFIERS AND SCOPE

- Unless otherwise specified, all instance variables **MUST** be `private`, all required methods **MUST** be `public`, and all helper methods (methods you write but are not required by the assignment specification) **MUST** be `private`. Do not forget that not explicitly specifying an access modifier defaults to an access modifier which is neither `public` nor `private`.
- All variables **MUST** be declared in the most restrictive scope possible. In particular, a variable which is used to store intermediate values within a method and whose value is no longer needed once a method returns **MUST** be declared as a local variable, and **NOT** as an instance or class variable.
- Variables local to a method **SHOULD** be declared at the beginning of the method in which they are declared, although loop index variables for `for` loops **MAY** be declared in the initialization clause of the `for` loop.
- Instance variables **MUST** be used **ONLY** to store part of the state of an object.

DOCUMENTATION

- Each of your methods **MUST** be documented in such a way that a person who reads your code can easily understand what a method does and how it does it (if a method is very simple, explaining the algorithm behind it is not necessary). These explanations should take the form of comments inserted either **BEFORE** the method, and/or **BEFORE** each significant code fragment in the method. **AVOID** writing these comments next to the line to which they pertain, as it makes the code harder to read on narrow displays. These comments should be meaningful and **BRIEF**.

COMPILATION ERRORS

- Starting with Assignment 1, students who submit source code files for a given question which contain compilation errors will get at most 25% of the value of that question. In other words, if, for a question that is worth 20 marks, you submit a program which does not compile, your grade for that question will be at most 5 marks. The reason this penalty is so harsh is that the computer tells you exactly what the problem is, so you should be able to fix it yourse.f If you are having trouble getting your code to compile, please consult Dan or one of the TAs.

Last update: 2011-05-08, 10:30 EDT