GENERAL INSTRUCTIONS AND REGULATIONS FOR ASSIGNMENTS

COMP-202A, Fall 2007, All Sections

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 specification.

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.

LATENESS POLICY

Unless otherwise specified, a lateness penalty of 5% per day late or fraction thereof will be applied to all late submissions, up to a maximum of two full days past the assignment deadline, after which submissions for the assignment in question will **NOT** be accepted and therefore receive a grade of 0.

SUBMISSION PROCEDURE

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

- Answers to the mandatory questions of an assignment **MUST** be sent for grading using the main WebCT submission box for that assignment.
- You **MUST NOT** send anything for grading using the WebCT problematic submissions box for any assignment unless you have been specifically told to do so by your instructor or the course coordinator. Graders will not even look at submissions sent to the WebCT problematic submissions box that have not been authorized by an instructor or the course coordinator.

ASSIGNMENTS NOT SUBMITTED THROUGH WEBCT WILL NOT BE GRADED

GENERAL SUBMISSION GUIDELINES

Each student **MUST** submit:

- For each question that requires writing or modifying a program: the file(s) containing the source code of your program (that is, the files with extension `.java`) and/or your modifications. You **MUST NOT** submit files containing source code provided to you by the instructors unless you made modifications to these source code files.

  Note that unless otherwise specified, you **MUST NOT** submit the executable files generated by compiling your program (that is, files with extension `.class`).

  All your programs must conform to the programming standards specified below.

- For each question that requires a written answer: an **ASCII TEXT** file containing your answer. Graders will have the discretion to penalize students who submit files in any other format as they deem appropriate, and even to refuse to look at the file and give a grade of 0 for the corresponding question.

You **MUST** follow the instructions given in the specification of each assignment regarding the names to assign to the files you submit, including case-sensitivity. For example, if the assignment specification states that a file you are asked to submit **MUST** be named `HelloWorld.java`, you **MUST** name it `HelloWorld.java`; you **MUST NOT** name it `helloworld.java`, `HELLOWORLD.JAVA`, or assign it any other name.

You **MUST NOT** submit any files other than those required by the specification of each assignment, with one exception: you **MAY** submit an **ASCII TEXT** file containing notes for the grader if you judge that submitting such a file is relevant. If you do submit such a file, you **MUST NOT** submit more than one such file and this file **SHOULD** be called `readme.txt`.

PLAGIARISM AND ACADEMIC FRAUD

Assignments must be done **INDIVIDUALLY**; you **MUST NOT** work in groups, you **MUST NOT** copy your submission from another student or any part thereof, and you **MUST NOT** allow another student to copy your submission or any part thereof. Performing any of the preceding actions constitutes academic fraud, and will not be tolerated.

Graders will be randomly checking submissions for suspicious similarities. Additionally, instructors will use automated plagiarism 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 academic fraud based solely on the output of these tools.

Students caught committing academic fraud on any assignment question will receive a grade of 0 for that question. Cases of repeat offenders (students caught committing academic fraud more than once) will be referred to the appropriate university officials.

IDENTIFICATION

You **MUST** identify yourself clearly in every file you submit. Your identification information **MUST** include:

- your name
- your McGill ID
- the course number (COMP-202A)
- the section you are registered in (MWF 11:30 - 12:30: section 1; TR: 13:00 - 14:30: section 2; MWF 12:30 - 13:30: section 3)

- your instructor's name (section 1: M. Petitpas; section 2: Professor C. Verbrugge; section 3: L. A. Mahabadi)
- the assignment number
- the question number

This information **MUST** appear in all of the following places:

- At the beginning of **EACH** file containing source code (that is, files with extension `.java`) in comments
- At the beginning of **EACH** file containing written answers to an assignment question
- At the beginning of your `readme.txt` file if you submit one.

RESTRICTIONS

Every program you submit **MUST AT LEAST** compile and run using the Sun JDK 1.5 installed on the PCs found on the third floor of Trottier engineering building.

You **MUST NOT** use any Java construct, feature, or library method or class that has not been introduced in the lectures at the time the assignment is due, unless the assignment specification explicitly allows the use of this construct, feature, or library method or class. For example, you **MUST NOT** use Java arrays before they have been covered in the lectures, unless the assignment specification explicitly states that you are allowed to use them.

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

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 (also called program listing) 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 values from the keyboard, it **MUST** display a prompt describing what the user is required to enter, which values are acceptable, and/or which values are illegal.

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 should 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 should 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 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 to enable graders to use automated testing programs to grade 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 labs 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 and classes have been covered in the lectures).

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 specifica-

tion) **MUST** be `private`. Do not forget that not 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 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 NOT** be used for variables that are not part of the state of an object.
- Graders will penalize students who violate the four preceding regulations by deducting marks from those allocated to the question, not from the marks allocated to instructions and regulations.

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, or before each significant code fragment in the method. These comments should be meaningful and **BRIEF**.

COMPILATION ERRORS

- Starting with assignment 2, students who submit source code files for a given question which contains 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.

Last update: 2007-09-13, 17:00 EST
Update log:

- 2007-10-08, 18:10: Clarified grader's penalty options on forbidden file formats.
- 2007-09-13, 17:00: Corrected a few errors and formatting problems.
- 2007-09-10, 18:20: Initial publication.