Software Development Ground Rules for CS 2430

General Rules

- All programs must be done using NetBeans 8.0
- Some programming assignments will be done individually, some will be done in groups. For those that are to be done individually, it's okay to discuss isolated details with others and to give and receive debugging help, but only *in moderation*. You *must* not borrow other people's work or compare more than a few lines of your work (either printed or on the screen) with others. Excessive collaboration will result in a 0 on the assignment for *all* students involved. According to Chapter UWS 14 Student Academic Disciplinary Procedures, 14.06 Section (3), a summary of the excessive-collaboration-incident will be sent to the Dean of Students.
- All programs must adhere to the style and documentation standards given below. All test documents and class diagrams must adhere to the standards given below. Maximum deductions for not following the standards are listed for each category, near the end of this document.
- Some programs must be submitted using the Grader program (available on the share drive) on or before the grace date/time. If your program does not compile or run under the Grader by the grace date/time, you automatically get 0 points for that program. However, you still must complete the program to pass the course. The Grader checks that your program compiles and runs. It also checks to see how closely your output matches the required output. When submitting to the Grader, only submit the Java files unless the program description instructs you to do otherwise.
- Some programs may need to be demonstrated in my office. The grade for these programs will be based on the demonstration as well as the Grader output and style, documentation, decomposition, test document, etc.

Documentation Standards

Files must be documented according to the javadoc standards. A javadoc comment is made up of two parts - a description followed by zero or more tags.

```
/**
This is the one sentence, descriptive summary, part of a doc comment.
There can be more lines after the first one.
....
@tag1    Comment for the tag1
@tag2    Comment for the tag2
...
*/
```

The first line is indented to line up with the code below the comment, and starts with /** followed by a return. The last line begins with */ followed by a return. The comment for a code entity (class or method) must be immediately before the code entity. The first sentence of each doc comment should be a summary sentence, containing a concise but complete description of the code entity. It is important to write crisp and informative initial sentences that can stand on their own. This sentence ends at the first period that is followed by a blank, tab, or line terminator, or at the first tag. Any tags come at the end.

Minimally, we require the following documentation:

- Comment block at the top of each class

```
/**
First, a single, very descriptive sentence describing the class.
Then, a couple more sentences of description to elaborate.
@author  "your name goes here, but not in quotes"
*/
```

Since we usually have one class per file, this usually serves as the "file" comment block.

- Every method and constructor must start with a comment block which describes what the method (or constructor) does (points lost otherwise). The first sentence must be a very descriptive summary of the method (or constructor). The following lines, if necessary, elaborate and/or give any extra information the user should know. All parameters must be listed using the @param tag. If there is a return value, it is listed with the @return tag. For example:

```
/**
Deletes the person with the given name from the list.
Does nothing if name doesn't appear in the list.
@param name  of the person to delete
@return  true is person was deleted, false otherwise
*/
public boolean deletePerson(String name)
```

- You can comment sections of code within methods. Use the // comments when you do. But don't overdo it! Excessive comments can be distracting, and comments that add nothing to the understanding of the code are particularly distracting. For example,

```
count = count + 1;      // add one to count
```

is a useless comment. For the most part, you shouldn't need more than one line of comments within methods for every few lines of code. If you feel you need to write a comment to make a section of code clear, then you probably should break that section out into a separate method!

Names

- **Use descriptive names!** You'll find that this not only improves your grade, but it also makes your programs easier to write and debug. If you're tempted to use a poor name for something, then you probably don't completely understand the problem you're trying to solve yet! Figure that out first before trying to go on.

- Variable and data members:
  - These should generally be nouns or noun phrases such as `grade` and `gradeForStudent`. The exception is `for` loop counters; this is the only place where it is *sometimes* acceptable to use a one-letter name such as `i`.
  - These names must start with a small letter and each subsequent word in a multi-word name must be capitalized. Use lower case for the remaining letters.

- Method names:
  - Names for methods with a return type of `void` should generally be verb phrases such as `printOrders()`.
  - Names for methods with other return types should generally be nouns or noun phrases such as `monthlySalary()`.
  - Method names shall start with a small letter and each subsequent word in a multi-word name shall be capitalized. Use lower case for the remaining letters.

- Class names:
  - Use meaningful common nouns.

- Start each class name with an upper-case letter, and capitalize each "word" in a multi-word name. Use lower case for the remaining letters.

- Names for constants (final variables):
  - Use meaningful nouns or noun phrases. For example, the name `TEN` in

    ```
    public final int TEN = 10; // useless!
    ```

    doesn't add to the understanding of the program at all.
  - Use upper case for the letters, with underscores to separate words.
  - In general, any value other than 0 or 1 should be given a name in a constant declaration.

Formatting

- Indent your programs! You must indent 3 spaces
  - inside all brace pairs, and
  - for simple statements following `if`, `while`, `for`, `switch`, and `do`.

  You should be sure your editor is set up to indent each line by 3 spaces and that it does *not* insert tab characters in the source code.

- Statements that are spread over multiple lines must be indented to make it obvious which lines are continuations. For example,

  ```
  System.out.print("This is a message that's broken into two"
                   + " parts for no good reason.");
  ```

- When a line gets too long (more than 78 columns), break it at a reasonable place.

- Each brace (`{` and `}`) must appear on its own line. Line up the braces to make it clear how they are matched. For example,

  ```
  if ( radius > 0 )
  {
     area = PI * radius * radius;
  }
  ```

- Each line must contain at most one statement, though a single statement may be spread over multiple lines.

- There must be a space before and after each operator. Use one space after a comma.

- Skip lines between different sections of the program and between different methods.

Output

- You must follow the output format exactly. **Be sure to look at your output!** Does it make sense to you? Just because your program produces output doesn't mean it works!

Unit Tests

- For some programs, unit tests with JUnit and/or test-bed mains will be required
- The tests for the units must be thorough
- Details will be specified in the program description

Test Specification

- For some programs, a test specification will be required.
- The test specification must be done in MSWord and turned in by the specified date/time.
- The test specification must have tests to cover the cases implied by the program description
- The test specification must list each case, with a description for the case, the precise input required, and the result expected. Each case must be separate and stand-alone. There are different ways to do this. A table is preferred. But it could be something as simple as:

```
Test Cases for RPN Program

    1. Valid expression, multiple operators
         Input:  3 5 4 - 6 + *
         Expected Results:  The value is 21

    2. Too many operands
         Input:  4 5 6 +
         Expected Results: Error message
             "Invalid Postfix Expression - Too many operands"

    3. Divide by 0
         Input:  4 0 /
         Expected Results: Error message
             "Invalid Postfix Expression - Division by 0"

    etc.
```

Class Diagrams

- Some programs will have a class diagram required
- The class diagram must use the UML notation discussed in class
- The diagram must show the classes and the relationships between the classes. Only the class names need to be specified; attributes or operations for the classes don't need to be included.
- Specify all the classes, including classes such as Vector, Button, List, etc.
- You can draw these by hand or use some package, such as MSWord or Rational Rose.

Program Grading

The programs you turn in should be considered "polished works of art!" There are many factors that will determine your grade on a program. Certainly it is expected that your program will compile, run, and produce the correct output. But merely satisfying these criteria does not guarantee you will receive any points! It is expected that you will apply the object-oriented techniques and data structure concepts covered in this course. If not, there would be no reason for you to be taking this course! Also, it is expected that you will adhere to the style and documentation guidelines. Furthermore, most programs will require a test specification and a class diagram, which will figure in to your program grade. Sometimes there will be other requirements, such as JUnit tests and/or "test-bed mains" for certain classes, that will affect your program grade. The maximum points that can be lost for each of these categories is listed below. These assume a 25 point program, but will be appropriately scaled if the number of points is not 25. Note that although these numbers total more than the 25 points, the lowest grade you can receive for a program is 0 (no negative grades). But please recall that all programs must be completed to pass the course.

Maximum Point Losses

- Doesn't Compile: 25 (this is also the minimum off)
- Run-time Error: 25 (this is also the minimum off)
- Output Errors: 20
- OO or Data Structure infractions: 20
- Style & Documentation: 7 (further broken down below)

- Test Specification: 5
- Unit Tests: 5 per required class
- Class Diagram: 4
- Extra/unnecessary files submitted to grader: 1 point per file

Below are further breakdowns for style and documentation:

Guideline Violated	each offense	max off
Class Comment	1	2
Method/Constructor comment block	0.5	3
Braces lined up	0.5	2
Naming Conventions	0.5	3
Indentation	0.5	2
Magic Numbers	0.5	2
Space between Operators	0.5	1