Java Code Style Guidelines

Changes

2014/02/01	reorganized to put project related items at the top, added clarification to #1, added new item #12
2014/02/02	Clarified zip file naming requirement
2014/02/08	Variable, method, class naming conventions must follow Java conventions, see #13
2014/02/12	Spelling errors are a dingable offense in output to the user
2014/04/06	Use 4 spaces for indent rather than tabs
2015/01/18	Do not use abbreviations, spell everything out
2015/08/18	Corrected project name and zip file requirements
2016/01/16	Changed project name and zip file requirements (again)
2016/02/06	Comments not at end of line
2017/02/10	Package name in lower case, add email in file header
2017/08/21	Changed project name and zip file requirements (again)
2018/01/18	General update
2018/02/02	Use of flag variables
2018/02/11	Filenames are Case Sensitive

When you enter the real world as a programmer, you will most likely be given a set of coding or “style” guidelines that you will be expected to follow. You may or may not agree with these guidelines, but you will be expected to adapt your style to that of your employer.

For the purposes of this course, the following code style requirements are in effect.

Some of these guidelines exist to make it easier for me to grade your projects. Others of them exist because over my 30+ years of programming, I have found them to reduce the probability of errors creeping into my code.

These will variously cost you between 1 and 3 points, depending on how the mood strikes me. (i.e., I’ll go easy on you initially, then ding you more as you fail to do things my way.)

Key words in this document are as defined by RFC 2119: http://www.ietf.org/rfc/rfc2119.txt

1. All project names shall be of the form

   Ax_Lastname_Firstname

   where Ax is the assignment number.

   The purpose of this is two-fold:

Java Code Style Guidelines

1) to make it easy to find in my downloads folder
2) to make it easy to know whose assignment it is

Note also that I will probably NOT follow this particular requirement in class. I do not write assignments in class, therefore, this rule is not applicable.

2. All projects shall be submitted in ZIP file format only.

The name of the zip file shall follow the project name. If you submit a project that is named incorrectly or that is in any other format than a zip file, I will not look at the project and will record a 0 for that assignment.

3. When creating a new project, the "package name" shall be as follows:

edu.srjc.lastname.firstname.Ax{.package_name}

"X" in the above is the 1 or 2 digit assignment ID. "firstname" is your first name. "lastname" is your last name. This will be the top level package name for your project and it corresponds to the physical path on your system where the project files (java source, compiled output, IDE detritus, etc.) reside.

The annotation {.package_name} is an optional package that can be more descriptive of what the project actually is.

4. Each open bracket - { - shall be on the line following the statement which it opens. For example

```
if (x == 0)
{
}

while (true)
{
}
```

If you are clever, you can find a way to have the IDE do this for you automatically.

5. Each control construct shall have open/close bracket, regardless of whether one is strictly required. For example,

```
while (true) do_something();
```

is valid Java, but it is not acceptable for this class. The proper way to write this is

```
while (true)
{
        do_something();
}
```

Java Code Style Guidelines

The same is true for **any** construct:

```
if (true)
{
        ...
}
```

6. **Each and every file** that you write shall have a file header that will contain, at a minimum, the following information contained in a comment:

your name
your email address
date
project name
course identifier
description of the file

7. Comments shall be provided that explain sections of code that are **not obvious**.

Comments are seldom needed in well written code. More often than not, they only add noise.

If you need to write a comment, use it to describe **what** you are doing and **why** you are doing it. Don't bother telling me **how** you are doing it - the code will tell me this (unless you write really crappy code.)

Noisy comments are **Bad**.

Comments shall appear on a line by themselves. They shall never be at the end of a line of code.

```
// this is a valid comment
int x = 0;

int y = 3; // this is an invalid comment
```

Comments may be of the single line variety

```
//
```

or the multi-line variety

```
/*
*/
```

If you use the multi-line comment style, the close comment symbol must be on a separate line all by itself.

Java Code Style Guidelines

8. Single character identifier names are forbidden except in obvious cases such as for loop indexes. See 10 below.

9. No declarations without initializations, no single line declarations/initializations.

```
int x, y, z; <= forbidden, single line
int x,    <= forbidden, multi line but not initialized
    y,
    z;
```

Acceptable

```
int x = 0,
    y = Integer.MIN_VALUE,
    z = Integer.MAX_VALUE;
```

or better

```
int x = 0;
int y = Integer.MIN_VALUE;
int z = Integer.MAX_VALUE;
```

10. Meaningful identifier names

```
angleOfAttack
timeOfDay
```

Single letter identifiers are forbidden UNLESS

they are universally recognized as “part of the problem”

Pythagorean theorem: a ^ 2 + b ^ 2 = c ^ 2

(Note that this is not valid Java – that would be
c = Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));

they are used in “simple” loop counters

```
for (int i = 0; i < limit; i++)
{
…
}
```

If this loop has more than one or two statements in the loop block, you SHOULD consider using a counter variable that imparts meaning (I.e., you may get dinged.)

```
for (int studentCount; studentCount < TOTAL_STUDENTS;  studentCount++)
```

Java Code Style Guidelines

11. Order of Evaluation

    Parentheses shall be used to explicitly show the order of evaluation. Do NOT rely on operator precedence.

12. Unless otherwise specified, input for console based programs shall be taken from a Scanner object vs as a parameter from the command line.

13. All identifiers shall follow Java naming conventions in regards to case.

    variables = camelCase
    classes = ProperCase
    methods = camelCase

14. Every effort shall be made to ensure that spelling in user prompts and program output conforms to standard English spelling and grammar rules.

    For example, it may (arguably) acceptable to use the letter “u” as a shortcut for the word “you” in a text message, it is absolutely unacceptable in the output of a program or in code.

    This rule will not be used to penalize non-native English language speakers, provided a reasonable effort is made to ensure proper spelling and grammar are used. However, in the “real world”, you will be judged on your ability to clearly and correctly communicate in the English language, and you therefore must all know when it is OK to use shorthand and when it is not.

15. Configure the IDE editor (or any other editor that you use) to indent 4 spaces each time the tab key is pressed. Do not use TAB characters.

16. Do not abbreviate identifiers – spell out all names.

    celcius NOT C
    farenheit NOT F

17. Use of ‘flag’ variables

    Flag variables are frequently used to control program flow. Flag variables must be named according to their use. The following is unacceptable

    ```
    int flag = 0;
    while (flag==0)
            …
    ```

    Instead use

    ```
    boolean done = false;
    while (!done) …
    ```

Flag variables are also sometimes used to manage the state of the program as it's running. Once again, using int variables as flag variables is forbidden. It is better to define constants

```
private final static int USER_LOGGED_OUT = 1;
private final static int USER_LOGGED_IN = 2;
```

and a variable

```
int loginState = USER_LOGGED_OUT;
if (loginState == USER_LOGGED_OUT) …
```

Note that the constants and variable MUST be named appropriately so as to describe what they are doing in the context of the program.

Better yet, enumerations are the preferred mechanism for this.

```
public enum LoginState
{
        USER_LOGGED_OUT,
        USER_LOGGED_IN
}

LoginState usersLogginState = LoginState.USER_LOGGED_OUT;
if (usersLogginState = LoginState.USER_LOGGED_OUT) …
```

18. Filenames are Case Sensitive

Filenames on a Unix system are Case Sensitive. Thus

```
filename.txt
```

and

```
FileName.TXT
```

are different. Windows systems do not care about the case of a filename. Linux (and other) systems do. Therefore, since your programs are tested on my linux system, you must assume that filenames are case sensitive and you must match the case in code.