**CSC-2591 Java Programming Standards**

The following standards are to be followed in writing Java programs. They are important enough to be part of the grading criteria for all programs. Any constructs not covered in this standard are to be implemented in a consistent and readable fashion.

**PURPOSE**

The purpose of these rules are (1) to introduce you to a representative set of coding standards that are typical in professional programming and (2) to help you develop good style as a habit, something you will find necessary to successfully complete large programs. There is no doubt that you can write Java programs to solve the small problems in this course without attention to style. But, you must keep in mind that you are building professional skills, not merely finding the answers to problems handed out as exercises.

|  |
| --- |
| **[Assignment Submission](#ASSIGNMENT_SUBMISSION)** |
| [**Classes**](#Classes) |
| [**Code Style**](#CODE_STYLE) |
| [**Comments**](#COMMENTS) |
| [**Constants**](#CONSTANTS) |
| [**Declarations**](#DECLARATIONS) |
| [**File Names And Java Compiler**](#FILE_NAMES_AND_JAVA_COMPILER) |
| [**Homework Template Submission**](#Homework_Template_Submission) |
| [**Output**](#OUTPUT) |
| [**Program Design**](#PROGRAM_DESIGN) |
| [**Program Header**](#PROGRAM_HEADER) |
| [**The “this” Construct**](#The_this_Construct) |
| [**The toString Method**](#The_toString_Method) |

**ASSIGNMENT SUBMISSION**

**Please name your document submission based on the assignment instructions**

**Please see each individual JAVA problem for naming requirements.**

**The deliverables that are to be uploaded to the assignment are:**

**1) This completed document. (PLEASE DO NOT INCLUDE IN THE ZIPPED FILE).**

**2) All Java source code in a zipped file.**

***Make sure you:***

**Attach your completed template in Microsoft Word format (.doc or docx) or a PDF format. If the word or pdf file that is submitted does not open in myLearning so I may grade it, the assignment will receive a zero.**

**Make sure all screen shots are complete and legible. See directions on assignment.**

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**CODE STYLE**

Use doubles for all floating point calculations.

**Use int for all integer calculations**

**Use integer division only when you are willing to lose accuracy.**

Each constant and variable declaration should be on a separate line that includes a comment above it that explains what each is. For example:

// voltage in the circuit

double volts = 12.3;

// amperage in the circuit

int amps = 1;

// name of the circuit

String circuitName = “”;

**Variables should be initialized where they are declared. You shall always initialize variables.**

Line Length - Avoid lines longer than 80 characters, since they’re not handled well by many terminals and tools.

Include line numbers in all programs.

Be descriptive in your prompts to the user.

When you notice you are repeating blocks of code, look for a better approach.

Alignment and indentation of lines of Java statements add much to the readability of programs. The following guidelines are general so that you may develop your own programming style:

* Use indentation and blank lines as you see fit to increase the readability of your programs.
* Use whitespace (blank lines and spaces on lines) liberally!
* Use parentheses liberally!!
* Before you submit your program use the Eclipse Format function to format your final submission.

The body of a control structure, such as an if, a switch, a while, a do, or a for statement, should be indented. The following examples illustrate the only formatting style that is to be used in this class. Each level of indenting is 4 spaces. (All of the below examples should have 4 spaces for indentation, but not all browsers display it correctly.)

**if statement**

**if (expression)**

**{**

**statements**

**}**

**if-else statement**

**if (expression)**

**{**

**statements**

**}**

**else**

**{**

**statements**

**}**

**switch statement**

**switch (selector variable)**

**{**

**case case-1-value:**

**case 1 statements;**

**break;**

**case case-2-value:**

**case 2 statements;**

**break;**

**...**

**case case-n-value:**

**case n statements;**

**break;**

**default:**

**default statements;**

**break;**

**}**

**while statement**

**while (expression)**

**{**

**statements**

**}**

**do-while statement**

**do**

**{**

**statements**

**}**

**while (expression);**

**for statement**

**for (initialization; condition; increment)**

**{**

**statements**

**}**

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

### CLASSES

All classes should be implemented in separate files.

Any potential for **stale data** should be avoided. When designing a class you should limit the fields that you have in the class to those that are only necessary for input to the computations.

All data that is calculated should not be return through fields in the class but should be returned through methods that return the results of the calculation.

For example, if you have a class Rectangle that has fields length and width, there should not be a field area; a method such as claculateArea() should return the area of the rectangle.

Getters and setters should be included for all instance variables.

Setters should use data validation checking to insure that all data passed to the class is valid.

Multiple constructors should be implemented including the default constructor.

The default constructor **must** be included in every class.

When **overriding a class** make sure the first line before the override is:

***@Override***

**For example:**

**@Override**

**public String toString() {**

**String str;**

**...**

**return str; }**

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**COMMENTS**

Please include comments throughout your program where applicable.

**Have your comments above the statements they are commenting; not to the side of them. It makes for better readability.**

Each constant and variable declaration should be on a separate line that includes a comment above it that explains what each is. For example:

// voltage in the circuit

double volts = 12.3;

Before you submit your program use the Eclipse Format function to format your final submission. This will make a better fit for your comments on the page.

*Each method* ***MUST*** *be identified by a Javadoc comment describing its purpose, parameter(s), and the method(s) it calls. See Chapter 02.*

The format to be used is as follows:

**/\*\***

**\* calculates and returns the charge for shipping cargo**

**\* between two planets.**

**\***

**\* @param distance distance in miles between two planets**

**\* @param weight weight in pounds of item being shipped**

**\* @return A double representing the cost to ship the item in space**

**\*/**

**public double findSpaceCost(long distance, float weight)**

**{**

**// implement here**

**}**

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**CONSTANTS**

Use named constants to make your program more readable and maintainable.

Constant name should be in upper case and use underscores between words.

Use the modifier final for constants.

Do not use literals in your calculations; use named constants.

Use 5 digits of accuracy for all constants such as PI.

Constant values used in your program will be given in final declarations. Constant identifiers should always appear in all-capital letters; underscore characters delimit multiple words. For example:

Examples:

**// Units kg/m3**

**final double DENSITY\_OF\_LEAD = 11340.1;**

**// PI to 5 places**

**final double PI = 3.14159;**

**// signals word continued on next line**

**final char HYPHEN = '-';**

**// names can be 20 characters**

**final int NAME\_LENGTH = 20;**

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**DECLARATIONS**

Use mnemonic names for all identifiers. Be descriptive of what the variable represents when you name variables. (Use speed instead of s or spd). Minimize the use of abbreviations in naming variables.

Use camel casing for all variable names. Variable identifiers should be in all lower-case letters, except for capital letters used to delimit multiple words in a variable identifier. For example, shippingRates, idEntry, numberOfItems, and minScoreRequired.

When dealing with floating point numbers, use doubles and when dealing with integers, use type int.

**One declaration per line is required. In other words,**

// indentation level

int level = 3;

// size of table

int size = 4;

Should be used instead of:

int level = 3, size = 4;

Use separate variables for everything you are to compute. Do not perform calculations in output statements. Associate all calculations with variable names.

**Classes** should begin with a capital letter, and multiple words should be indicated by additional capital letters (for example, Date, VehicleNumber, SoftwareEngineer, BankAccount).

**Methods** should begin with a lowercase letter, and have any multiple words indicated by capital letters (for example, isValidNumber(), area(), setWidth(), etc). Note: Constructors are methods which cannot follow this convention, because their names must exactly match the class name.

Each identifier declaration (including method declarations page 79 of the text) must be accompanied by a comment that explains its use.

Avoid declaring globally-visible variables. Globally-visible constants, enumerated types, classes, and file names are acceptable, but do not use global variables unless the project assignment specifically tells you to do so.

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**[FILE NAMES AND JAVA COMPILER](#TOP_OF_PAGE)**

[In this course each Java file name should clearly represent what it is and each should have the “.java'' suffix. Files names must match the class name. For example, Project 1 should be Project1.java. We will be using the Eclipse IDE exclusively for all program development.](#TOP_OF_PAGE)

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**INPUT DATA VALIDATION**

Once chapter 3 has been completed, all input data should be checked for validity. Some data should fall in a specific range, for example if you were to input a person's age, it would be realistic to check for values between 0 and some upper limit, perhaps 125 years. If you were inputting length, you would expect a value greater than or equal to zero. In some cases you will have data such as temperature with negative values.

If you get data that falls outside your expected ranges, notify the program user and loop until the user inputs data within the acceptable range. In general, a do-while statement tends to be an efficient structure to perform these checks.

**PROGRAM HEADER**

At the beginning of each Java class there should be a **PROGRAM HEADER using a**

JavaDoc block (Chapter 02 of the text) that contains the following: 

1. The assignment name, your name email and telephone number, the date built, the file name and the project name.
2. A brief description (at least 2 lines) of the program that describes the method or algorithm used to obtain the solution, the major data structures (if any) in the program, and other general comments including extra features.
3. The Completed Honor Code.

The format to be used is as follows:

/\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

'Project: Assignment #

'Programmer: FULL NAME

'Company Info: EMAIL

'Date: DD MM YYYY

'Description: Problem Number #.

'

' LINE 1 AT LEAST 3 LINES OF PROGRAM DESCRIPTION

' LINE 2 AT LEAST 3 LINES OF PROGRAM DESCRIPTION

' LINE 3 AT LEAST 3 LINES OF PROGRAM DESCRIPTION

'

' --------------------------------------------------------------------------

' HONOR CODE:

' I pledge that this program represents my own program code, I have received

' help from no one and I have given help to no one.

'

' OR

'

' I received help from NAME OR NO ONE in designing and debugging my program.

' I given help to NAME OR NO ONE in designing and debugging my program.

'-----------------------------------------------------------------------------

'

' LINE LENGTH - AVOID LINES LONGER THAN 80 CHARACTERS

' SCALE BELOW IS TO CALIBRATE SCREENSHOTS

' DO NOT HAVE YOUR CODE OR SCREENSHOT EXTEND BEYOND THE SCALE

0........1.........2.........3.........4.........5.........6.........7.........8

12345678901234567890123456789012345678901234567890123456789012345678901234567890

\*/

*This you tube video is excellent and describes exactly how and why to use JAVADOC COMMENTS.*

**[Play Now](http://www.youtube.com/watch?v=6XoVf4x-tag)**

[**GO TO TOP OF DOCUMENT**](#TOP_OF_PAGE)

**PROGRAM DESIGN**

Use stepwise refinement. The programming assignments given are ones that can be solved in a step-by-step manner. The method of solution used in your program should exhibit this kind of structure.

Modularize your program. When a problem solution readily splits into distinct tasks, your program should handle each task in a separate function and in a straightforward manner.

Make your program efficient. Your algorithm should be direct and not unnecessarily complicated. Proper use of data structures will often simplify an algorithm.

Use parameters appropriately. All variables used in a method should either be declared as parameters or be declared locally within the method.

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**OUTPUT**

All input values should be ***echoed*** in a labeled readable manner unless otherwise indicated. This means when data is input via the keyboard it should be output to the console in one place, just before the results of any computations. This assures that the person who input the data will see what she input.

**For all work use printf or String.format for formatting of all output. Do NOT use DecimalFormat.**

**(After completing Chapter 02)**

The program output should be readable and clearly labeled.

Headings should be used when outputting data in tabular format.

Use 2 decimal places with all numeric output unless specifically indicated otherwise. (After completing Chapter 02)

Use separate variables for everything you are to compute. **Do NOT perform calculations in output statements**.

Use correct spelling throughout your program.

Space and format your output correctly.

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**HOMEWORK TEMPLATE SUBMISSION**

**Please name Homework Template Submission XXX\_ Assignment0N where XXX are your initials and N is the assignment number.**

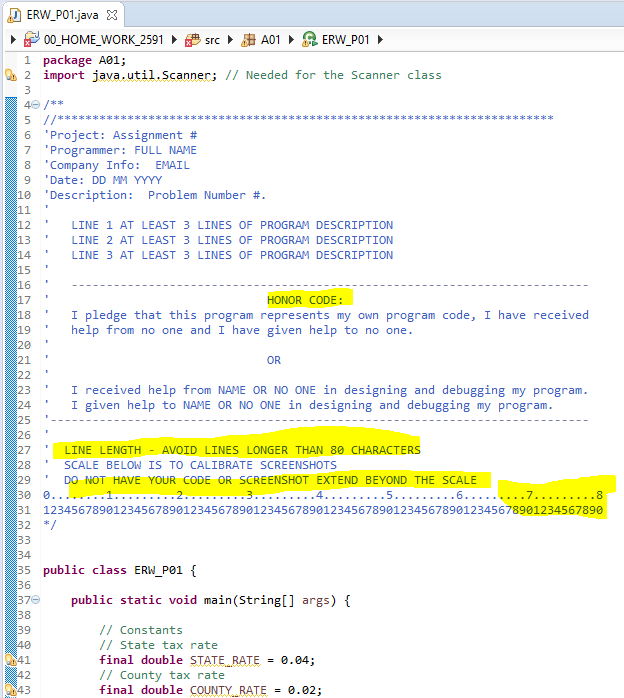
**Where your assignment asks for:**

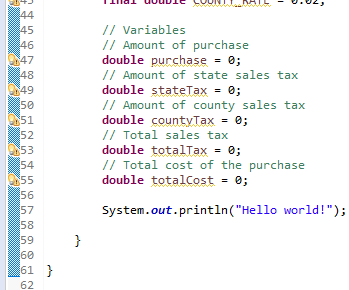
**"PLACE SCREEN SHOT OF PROGRAM CODE and CONSOLE OUTPUT HERE."**

Please note that I used the windows snipping tool for the screen shot, had the appropriate comments in the program, had the program number as it was in the assignment in the program, highlighted the program name and project comment in yellow using the snipping tool. A yellow highlight is also used with the output and the date and time the program was run is shown. In addition, all code from the program meets all standards in this document, is numbered, visible and legible. **Use a white background to show screen shots of programs, as seen below. Your console output may have a black background.** ***Screen shots of programs with a black background are quite difficult to read and will receive a grade of zero for the problem.*** Your console output may have a black background.

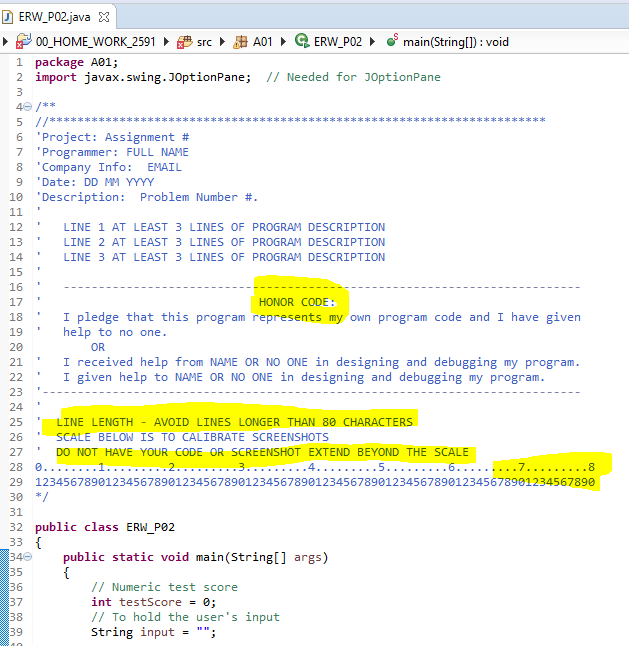
**Line Length - Avoid lines longer than 80 characters, since they’re not handled well by many terminals and tools.**

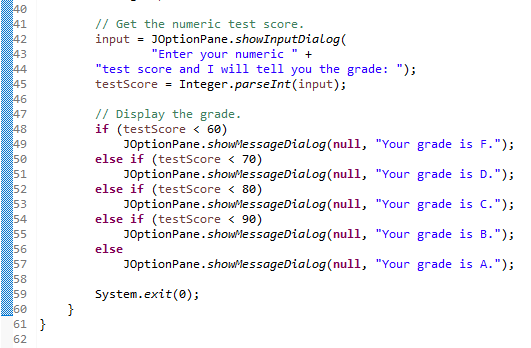
**THIS IS WHAT IS EXPECTED FROM THE SCREEN SHOT:**

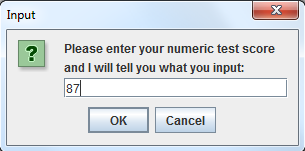


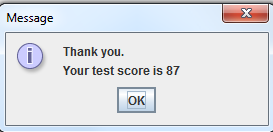


If you are using JOptionPane Show Screen shots of all messages and I/O for all cases, for example:









**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**The toString Method**

The toString method should be implemented for all classes in **Homework 03** and forward. ***See Chapter 06 videos.***

Use Stringbuilder in developing toString Methods for **Homework 05**.

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

**The “this” Construct**

The “this” construct should be exclusively used in setting instance variables for all classes in **Homework 05*. See Chapter 08.***

**[GO TO TOP OF DOCUMENT](#TOP_OF_PAGE)**

Finally, of course, your program should produce correct results!

Some of this material was paraphrased from a Purdue University document.