Coding Standards for Java

Version 1.0

PACIFIC UNIVERSITY
1849
O R E G O N

Why have coding standards?

It is a known fact that 80% of the lifetime cost of a piece of software goes to maintenance. Therefore it makes sense for all programs within an organization to be as consistent as possible. Code conventions also improve the readability of the software.

This document specifies the coding standards for all computer science courses at Pacific University. It is important for you to adhere to these standards in order to receive full credit on your assignments.

The document is divided into four main sections:
- Naming Conventions
- Formatting
- Comments
- Printing

Naming Conventions

Constants

A constant is to be mnemonically defined using all capital letters such as MAXNAMECHARS. Further, your program is to contain no "magic constants." That is, all magic constants must be defined as constants to make program modification easier. In the case below, 100 is a magic constant and if used in several places throughout a program, can create problems if 100 is to be modified for any reason.

Poor Program Style

```
.....
for (indx = 0; indx < 100; indx++)
{
  .....
}
```

Correct Program Style

```
final int MAXGRADESCORES = 100;
.....
for (indx = 0; indx < MAXGRADESCORES; indx++)
{
  .....
}
```

Variable Names

1) A variable name is defined in all lowercase letters unless the variable name contains multiple names such as readStudentRecord. After the first word, each subsequent word has the first letter capitalized with the remainder of the word made up of lowercase letters.

2) Variable names are to be mnemonic unless the variable is being used in a for loop in which case the names i, j, k, l, m, n are acceptable names to be used. If however the nested loop is being used in conjunction with a two-dimensional array, then the names row and column should be used.

Class Names

Class definitions will follow the regular variable naming conventions except the first letter of the class must be capitalized.

Method Name - methods are named using the standard naming convention described for variables where the first word begins with a lowercase letter and each subsequent word has the beginning letter capitalized.

Methods should be documented as follows:

```
//*****************************************************
// Method:      equal
//
// Description: Compares two objects of Rational returning
//              a value of true if the numerators and
//              denominators of both fractions are the
//              same.
//
// Parameters:  fraction - object of type Rational.
//
// Returned:    true if objects are equal; else, false
//*****************************************************
```

Formatting

Indentation

Two spaces must be used as the unit of indentation per tab. Every IDE (Integrated Development Environment) such as CodeWarrior or Visual C++ includes an option for changing the number of spaces in a tab. These can usually be found in the preferences section.

Line Length

Lines must be no longer than 80 characters. Anything longer than that is normally not handled well in many terminals and tools.

Wrapping Lines

If an expression cannot fit on a single line then break it:

- After a comma
- Before an operator

Make sure that the new line is aligned with the beginning of the expression at the same level on the previous line.

Spaces

All arithmetic and logical operators must have one space before and after the operator. The only exceptions are:

- Unary operators
- The period
- No spaces before the comma and only one space after the comma

Blank Lines

Use blank lines to separate distinct pieces of code. For example, separating the `imports` from the rest of the program and breaking up long sections of code into logical units. The important thing to remember is that blank lines must be used consistently.

Braces

Any curly braces that you use in your program (e.g. surrounding classes, functions) must appear on their own lines. Any code within the braces must be indented relative to the braces.

```
Class User
{
  private BufferedReader input;
}
```

Comments

Comments should be used to explain the purpose of the code fragment they are grouped with. Comments should state what the code is doing, while the code itself shows how you are doing it.

Use comments sparingly and only comment code segments that are not obvious. Giving your variables meaningful names will improve the readability of your code and reduce the need for comments.

File Header

The main purpose of a file header is to explain the purpose of the program as briefly as possible. You must include the following sections in your program header:

- File name
- Your name
- Date
- Class and Assignment Title
- Purpose
- Program input

```
//******************************************************
// File name:  myprogram.cpp
// Author:     Joe Bloggs
// Date:       02/20/2004
// Class:      CS250
// Assignment: Morse code
// Purpose:    This program encodes an English message
//             into Morse code
//******************************************************
```

Declaration Comments

Variables should be declared as one per line. Each variable should have a sidebar comment to the right of it indicating the variable's purpose. Do not put any blank lines between the variables being declared. You should also group together variables that are related.

```
int seconds;                  // Range (0-59)
int minutes;                  // Range (0-59)
int hours ;                   // Range (0-23)
String firstName;             // First name of employee
String lastName;              // Last name of employee
```

Method Header

In the same way that a program header is used to describe the purpose of the program, the function header should be used to describe the purpose of the function. All your function headers must include the following:

- Method name
- Description

- Parameters
- Returned

```
//***************************************************
//
// Method:      setValues
//
// Description: Changing the values of the numerator and
//              denominator of the data members.
//
// Parameters:  numerator   - numerator of the fraction
//             denominator - denominator of the fraction
//                           whose value is nonzero
//
// Returned:   None
//***************************************************
```

Sidebar and In-line Comments

A sidebar comment appears on the same line as the single statement it is describing. The comment must be brief and not exceed that line.

```
value <<= 1;   // multiply value by 2
```

In-line comments appear on their own lines and precede the segment of code they describe. You should use in-line comments to describe complex code that is not limited to a single statement. You should use blank lines to separate the comments from the segments of code they are describing.

```
// read message from BufferedReader

try
{
  message = input.readLine();
}
```

Although using comments helps in describing your code, you must always make sure that your variables have meaningful names to make the code more understandable.

Printing

When printing your code you must use a fixed width font. Courier and Courier New are examples of fixed width fonts. You must also make sure that your lines do not wrap nor do they get cut off when printing. All printing is to be done in Portrait.