**Coding Standards**

**for**

**Port Authority North Shore Extension**

**Version 1.0 approved**

**Prepared by Sarah Bunke, Nicholas Schnur, Meyling Taing, Keith Payne, Brandon Bock, Catherine Nalesnik**

**On Track Trainwreck**

**9/11/13**

# 1. File Names

## 1.1 File Sufﬁxes

The following file suffixes will be used:

Java source .java

Java bytecode .class

## 1.2 Common File Names

We will use these file names:

Make file Makefile

Readme README

# 2. File Organization

## 2.1 Java Source Files

Each Java source ﬁle contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source ﬁle as the public class. The public class should be the ﬁrst class or interface in the ﬁle.

Layout of Java source files

* Beginning Comments – Javadoc style

/\*\*

\* Classname

\*

\* Version info

\*

\* Copyright notice

\*/

* Package and Import Statements
* Class and Interface Declarations – Include these parts in this order
  1. Class/interface documentation comment in Javadoc style
     + /\*\*...\*/
  2. class or interface statement
  3. Class/interface implementation
  4. Class (static) variables
     + First the public class variables, then the protected, and then the private.
  5. Instance variables
     + First public, then protected, and then private.
  6. Constructors
  7. Methods
     + Grouped by functionality

# 4. Indentation

Use tabs, not spaces.

## 4.1 Line Length

In general, keep lines less than 80 characters, but this is not strict.

## 4.2 Wrapping Lines

When an expression will not ﬁt on a single line, break it according to these general principles:

• Break after a comma.

• Break before an operator.

• Align the new line with the beginning of the expression at the same level on the previous line.

• If the above rules lead to confusing code or to code that’s squished up against the right margin, just indent once instead.

# 5. Comments

## 5.1 Documentation Comments

These will be in Javadoc style and will appear at the beginning of the source files, and before class and method declarations. All other comments are implementation comments.

## 5.2 Implementation Comment Formats

5.2.1 Block Comments will be used to provide descriptions of ﬁles, methods, data structures and algorithms. They will also be used for comments that span multiple lines.

/\*

\* Here is a block comment with

\* some stuff in it.

\*/

5.2.2 Single-Line Comments will use this format. They can appear before the code it describes or on the same line, right after the code.

// This is a comment

# 6. Declarations

## 6.1 Number Per Line

We will not mix different data types on the same line when declaring variables. Variables of the same type may be declared on the same line or in separate lines. We will use one space between the type and the identifier.

int variableName;

int variable1, variable2, variable;

## 6.2 Placement

Variable declarations will always appear at the beginning of the blocks they are used in.

## 6.3 Class and Interface Declarations

When coding Java classes and interfaces, the following formatting rules should be followed:

• No space between a method name and the parenthesis “(“ starting its parameter list

• Open brace “{” appears on its own on the line following the declaration statement

• Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately after the “{“

• Methods are separated by a blank line

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

# 7. Statements

## 7.1 Simple Statements

Each line should contain at most one statement.

## 7.2 return Statements

A return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

return;

return myDisk.size();

return (size ? size : defaultSize);

## 7.3 If statements

The if-else class of statements should have the following form:

if (condition)

{

statements;

}

if (condition)

{

statements;

} else

{

statements;

}

if (condition)

{

statements;

} else if (condition)

{

statements;

} else if (condition)

{

statements;

}

An if statements always use braces, even if it only contains one line.

## 7.4 for Statements

A for statement should have the following form:

for (initialization; condition; update)

{

statements;

}

## 7.5 while Statements

A while statement should have the following form:

while (condition)

{

statements;

}

## 7.6 do-while Statements

A do-while statement should have the following form:

do

{

statements;

} while (condition);

Avoid empty loops.

## 7.8 switch Statements

A switch statement should have the following form:

switch (condition)

{

case ABC:

statements;

/\* falls through \*/

case DEF:

statements;

break;

case XYZ:

statements;

break;

default:

statements;

break;

}

Every switch statement should include a default case.

## 7.9 try-catch Statements

A try-catch statement should have the following format:

try   
{

statements;

} catch (ExceptionClass e)

{

statements;

}

# 8. White Space

## 8.1 Blank lines

Blank lines will be included to group sections of code that are logically related.

## 8.2 Blank Spaces

Blank spaces should be used in the following circumstances:

• A keyword followed by a parenthesis should be separated by a space. Example:

while (true)

{

...

}

Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.

• A blank space should appear after commas in argument lists.

• All binary operators should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment (“++”), and decrement (“--”) from their operands. Example:

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++)

{

n++;

}

println("size is " + foo + "\n");

• The expressions in a for statement should be separated by blank spaces. Example:

for (expr1; expr2; expr3)

• Casts should be followed by a blank. Examples:

myMethod((byte) aNum, (Object) x);

myFunc((int) (cp + 5), ((int) (i + 3)) + 1);

# 9. Naming Conventions

* Each “word” in a class or interface name should be capitalized.
  + Class MyClass;
* Method and variable names should start with a lowercase letter and use camel casing. These names should be descriptive and meaningful.
  + doSomething();
  + int myVariable;
* One character variable names should be avoided, except for temporary variables such as loop counters.
* Constants should be all uppercase and separated by underscores.
  + int MAX\_WIDTH = 1000;

# 10. Programming Practices

## 10.1 Don’t make any instance or class variable public without good reason

## 10.2 Constants

Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

## 10.3 Make everything as clear as possible.

## 10.4 Special Comments

Use TODO: to label code that needs to be worked on or added in the future.