# (2.3.1.1)Coding Standards

## Commenting Style

As the coding is to be a collaborative effort, developers must adhere to the commenting style standards defined here. This is so other developers who may need to read, update or remove code will be able to understand with ease the functionality of methods, algorithms or data structures in the code. Comments must not impact the formatting conventions, for example, comments must not make altering blocks of code unnecessarily task heavy.

### Conventions:

* Javadoc must include Author, version and java version.
* Javadoc all functions and methods.
* Variable declarations are to be grouped with a comment above.
* Comments at the side of variables to describe their use. However this must not conflict with formatting conventions described above.
* Comments must be descriptive without being too large and avoiding ‘walls of text’.
* Comments covering few lines (1-2 lines) may use the standard // feature.
* Larger comments which cover more than a few lines must be grouped with the /\* \*/ feature for code clarity and neatness.

## Variable Naming Style

Variables must be named in accordance to the standards defined below. Again, the collaborative nature of the project, there must be a standard to variable declaration used by all developers for the ease of altering and reading others’ code. Variables must avoid being ambiguous for ease of referencing, passing and casting.

### Conventions:

* Declaring variables ‘on the fly’ must be avoided, with the exception of for loops, and declared at the top of classes and functions.
* All variables declared must use title case, as seen in the examples below
* Underscores must be avoided in variable declaration.
* Underscores must be used to denote parameter variables.
* The exception to these standards is for loops, which may use single variable names, for example; i, j, k etc.
* Arrays and data structures must be declared as such: arrValues. Followed by a comment which describes what data types are in the array.
* Variables must follow the 3 (Boolean types break this exception) Hungarian notation.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Correct Notation** | **Parameter** |
| Integer | intIntegerVar | \_intIntegerVar |
| Double | dblDoubleVar | \_dblDoubleVar |
| Boolean | boolBooleanVar | \_boolBooleanVar |
| String | strStringVar | \_strStringVar |
| Char | chrCharVar | \_chrCharVar |
| Long | lngLongVar | \_lngLongVar |
| Float | fltFloatVar | \_fltFloatVar |
| Short | shtShortVar | \_shtShortVar |
| Byte | bytByteVar | \_bytByteVar |
| Object | objObjectVar | \_objObjectVar |

## Function and Method Naming Style

Again, correctly named functions and methods are essential for code clarity and ease of modification for other developers. The names must give a hint as to what the method or function does, this is for ease of calls to the method or function in code.

### Conventions:

* As with variables, names must use title case with the first word in lowercase regardless of the circumstance.
* Underscores must be avoided, unless in the case of parameter variables.

## Brace and Indentation Formatting

For code clarity and neatness, all developers must adhere to the following conventions. Code clarity is essential when working on a collaborative project and poor formatting is inexcusable, as all IDE’s have customisable formatting options.

### Indentation Conventions:

* Declarations within class bodies must be indented.
* Statements within methods/constructors, blocks, switch and case statements must be indented.
* Vertical lines must be adhered to in the case of line breaks and comments occurring after variables.

### Brace Conventions:

* Braces must follow the declaration on the same line.
* Catch/finally blocks must begin on a new line after the brace.
* Else if blocks must also being on a new line after the brace.

## Example:

*/\*\**

*\* @author Jeff Jones*

*\* @version 1.01*

*\* @since 1.6*

*\*/*

class MyClass {

*// Variable Declaration*

int intHouseNumber; *// Number of the house*

double dblGrams; *// Weight in grams*

*/\**

*Constructor for MyClass*

*I will make*

*this comment span*

*several lines.*

*\*/*

public MyClass() {

super.MySuperClass;

}

*// switch statement for something*

switch (intHouseNumber) {

case 1:

return true;

break;

case 2:

return false;

break;

default:

}

*/\*\**

*\* @param \_intHouseNum passed house number*

*\*/*

public void setHouseNum(int \_intHouseNum) {

intHouseNumber = \_intHouseNum;

}

}