**CHAPTER 9: IMPLEMENTATION PLANNING**

**9.1 Coding Standards**

1.**File Names**

File Suffixes Aurdino Software uses the following file suffixes:

File Type Suffix

Aurdino Executable File .ino

**2.Include Statements**

The first non-comment line of most Java source files is a include statement, which is used to include the pre-defined libraries in our code and provide access to its methods.

For Example:

#include<LiquidCrystal.h>

**3.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.
* Prefer higher-level breaks to lower-level breaks.
* 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 8 spaces instead.

Example:

someMethod(longExpression1, longExpression2, longExpression3, longExpression4, longExpression5);

var = someMethod1(longExpression1, someMethod2(longExpression2, longExpression3));

**4.Number Per Line**

One declaration per line is recommended since it encourages commenting.

**5.Initialization**

Try to initialize local variables where they’re declared. The only reason not to initialize a variable where it’s declared is if the initial value depends on some computation occurring first

**6.Method Declarations**

When coding Methods, the following formatting rules should be followed: • No space between a method name and the parenthesis “(“ starting its parameter list • Open brace “{” appears at the end of the same line as 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 “{“

Example:

void setup() {

// put your setup code here, to run once:

}

void loop() {

// put your main code here, to run repeatedly:

}

**7.Statements**

**7.1 Simple Statements**

Each line should contain at most one statement. Example: argv++; // Correct

**7.2 Compound Statements**

The enclosed statements should be indented one more level than the compound statement.

* The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.
* Braces are used around all statements, even single statements, when they are part of a control structure, such as a if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

**7.3 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();

**7.4 if, if-else, if else-if else Statements**

The if-else class of statements should have the following form: if (condition) { statements;

}

**7.5 for Statements**

A for statement should have the following form: for (initialization; condition; update) { statements; }

**7.6 while Statements**

A while statement should have the following form: while (condition) { statements; }

**7.7 do-while Statements**

A do-while statement should have the following form: do { statements; } while (condition);

**8.Naming Conventions**

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier—for example, whether it’s a constant, package, or class—which can be helpful in understanding the code.

**9.Methods**

Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

Example:

runFast();

getBackground();

**10.Variables**

Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.

Example: int i; char c;

**11.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.

**12. Variable Assignments**

Avoid assigning several variables to the same value in a single statement. It is hard to read. Example: fooBar.fChar = barFoo.lchar = 'c'; // AVOID!

Do not use the assignment operator in a place where it can be easily confused with the equality operator.

Example: if (c++ = d++) { // AVOID!

(Java disallows) ... }should be written as if ((c++ = d++) != 0) { ... }

Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler.

Example: d = (a = b + c) + r; // AVOID! should be written as

a = b + c; d = a + r;

**Coding Guidelines**

* **Do not use a coding style that is too clever or too difficult to understand:**
* Code should be easy to understand.
* Many inexperienced engineers actually take pride in writing code ambiguous and beyond your understanding.
* Clever coding can doubtful meaning of the code and hamper understanding.
* It also makes maintenance difficult.
* **Do not use an identifier for multiple purposes:** Programmers often use the sameidentifier to denote several temporary entities.
* Each variable should be given a descriptive name indicating its purpose.
* **The code should be well-documented:** As a rule of thumb, there must be at least onecomment line on the average for every three-source line.
* **The length of any function should not exceed 10 source lines:**
* A function that is very lengthy is usually very difficult to understand.
* **Do not use goto statements:** Use of goto statements makes a program unstructuredand makes it very difficult to understand.