Coding Standards Document

Software Engineering Project 2015

Project ID: WD-SEP-008

Submitted by:

1. IT 13026776 – K.D.K. Shamantha
2. IT 13014100 – K.S. Amaraweera
3. IT 13007348 – K.M.C.B. Kendaragama
4. IT 13092702 – H.M.I. Thilakarathna

Submitted to:

……………………………..

Ms. Dimanthinie de Silva

Date of submission – 05/09/2015

Table of Contents

1. Introduction

1.1 Importance of Coding Standards

Coding standards give a good programming practice for programmers and it provides a uniform appearance to the codes written by different programmers. It is simply a set of rules and guidelines for the formatting of source code. A coding standard enhances the understandability, readability and maintainability of a program. So as programming students we should have a good practice of following the coding standards.

1.2 Common Aspects of Coding Standards

There are many common aspects involved in coding standards. Following are some of the most common aspects of coding standards.

- Naming conventions
- File naming and organization
- Comments and documentation
- Classes, Functions and Interfaces
- Testing

1.3 Benefits of Coding Standards

- Code integration
- Maintenance
- Improved code maintainability
- Minimizes communication
- Uniform problem solving
- Fewer bugs. Good standards should minimize common coding mistakes
- Better teamwork

2. Indentation

Proper and consistent indentation is important in producing easy to read and maintainable programs. Indentation should be used to:

- Identify the body of a control statement such as a loop or a select statement
- Identify the body of a conditional statement
- Identify a new scope block

A minimum of 3 spaces or 1 tab space shall be used to indent. Once the programmer chooses the number of spaces to indent by, then it is important that this indentation amount be consistently applied throughout the program.

```
//Indentation used in a for loop construct. Four spaces are used for indentation.
for(int i=0;i<10;i++)
{
    Log.e(“count”,String.valueOf(i));
}
// Indentation used in the body of a method
public void print(String name)
{
    Log.e(“Name : ”,name);
}
// Indentation used in a conditional statement.
if(i==10)
{
    i=0;
}
else
{
    i++;
}
```

3. Naming Convention for Identifiers

The most important consideration in naming a variable is that the name accurately describe the entity or action that the structure represents. Clear, complete, and meaningful names makes the code more readable and minimizes the need for comments.

Pascal Casing - First character of all words are Upper Case and other characters are lower case.

Camel Casing - First character of all words, except the first word are Upper Case and other characters are lower case.

- Use Pascal casing for Class names

```
public class GetAllStaffDetails
{
   //
}
```

- Use Pascal casing for Method names

```
public void SetListViewAdapter(JSONArray jsonArray)
{
   //
}
```

- Use Camel casing for variables and method parameters

```
String firstName = 0;
public void SetListViewAdapter(JSONArray jsonArray)
{
   //
}
```

- Variable declarations that span multiple lines should always be preceded by a type.

```
int price , score ; // Acceptable

int price ;
int score ; // Acceptable

int price ,
score ; // Not Acceptable
```

- constants should be all uppercase with words separated by under-scores (“ _”)

```
String TAG_SUCCESS=”success”;
```

4. Program Statements

Program statements should be limited to one per line. Also, nested statements should be avoided when possible.

```
//Bad
int  lengthOfName= name.length() ; button = (Button) this.findViewById();

//Better
int  lengthOfName= name.length() ;
button = (Button) this.findViewById();
```

```
//Bad
return password.length() >5;

//Better
int passwordLength=password.length();
return passwordLength > 5;//
```

5. Use of Parentheses

It is better to use parentheses liberally. Even in cases where operator precedence unambiguously dictates the order of evaluation of an expression, often it is beneficial from a readability point of view to include parentheses anyway.

```
//Acceptable
if ( a == b && c == d )

//Better
if ( ( a == b ) && ( c == d ) )
```

6. Meaningful Error Messages

Error handling is an important aspect of computer programming. This not only includes adding the necessary logic to test for and handle errors but also involves making error messages meaningful. They should indicate what the problem is, where the problem occurred, and when the problem occurred. A useful Java exception handling feature is the option to show a stack trace, which shows the sequence of method calls which led up to the exception.
Error messages are stored in separate file and use Toast to display error message.

7. Use of Braces

Braces are used to delimit the bodies of conditional statements, control constructs, and blocks of scope. Programmers shall use either of the following bracing styles.

```
for (int j = 0 ; j < max_iterations ; ++j)
{
   //
}
//The Kernighan and Ritchie style
for ( int j = 0 ; j < max_iterations ; ++j ) {
   //
}
```

Braces shall be used even when there is only one statement in the control block

```
if (j == 0)
   Log.e(“TAG”,”msg”);

if (j == 0)
{
    Log.e(“TAG”,”msg”);//Better
}
```

8. Line Length

It is considered good practice to keep the lengths of source code lines at or below 80 characters. Lines longer than this may not be displayed properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

9. Spacing

The proper use of spaces within a line of code can enhance readability.

- A keyword followed by a parenthesis should be separated by a space
- A blank space should appear after each comma in an argument list.
- All binary operators except "." 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.
- Casts should be made followed by a blank space.

10. Wrapping Line

When an expression will not fit on a single line, break it according to these following principles Break after a comma or operator and align the new line with the beginning of the expression at the same level on the previous line. Prefer higher-level breaks to lower-level breaks.

```
Toast.makeText(getApplicationContext(),"Please Enter First Name",
Toast.LENGTH_SHORT).show();    //Break after comma

 String url = "http://mystyle.3eeweb.com/updateStaffDetails.php?id="+staffID+
                "&fname="+newFname+"&lname="+newLname;  //Break after operator

double salary = (basicSalary * 108)/100 + (hourlyRate * noOfHours

                    – epf + bonus) //Avoid. Because break occur inside the parenthesis

double salary =  (hourlyRate * noOfHours  – epf + bonus) +
                    (basicSalary * 108)/100 //better.
Because break occurs outside the parenthesis
```

11. Inline Comments

Inline comments promote program readability. They allow a person not familiar with the code to more quickly understand it.
Inline comments appear in the body of the source code itself. They explain the logic or parts of the algorithm which are not readily apparent from the code itself. Inline comments can also be used to describe the task being performed by a block of code. Inline comments should be used to make the code clearer to a programmer trying to read and understand it. Do not use overuse inline comments to explain program details which are readily obvious to an intermediately skilled programmer.
A rule of thumb is that inline comments should make up 20% of the total lines of code in a program, excluding the header documentation blocks.

12. Reasonably Sized Functions and Methods

Software modules and methods should not contain an excessively large number of lines of code. They should be written to perform one specific task. If they become too long, then chances are the task being performed can be broken down into subtasks which can be handled by new routines or methods.

A reasonable number of lines of code for routine or a method is 200. This does not include documentation, either in the function/method header or inline comments.

This documentation block is located at the beginning of the file.

```
/**********************************************************************
* Original Author:
* File Creation Date:
* Development Group:
* Description:
*
**********************************************************************/
```

13. Statements

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

The if-else class of statements should have the following format.

```
if (condition) {
   //statements;
}
```

```
if (condition) {
   //statements;
}
else {
   //statements;
}

if (condition){
   //statements;
}
 else if (condition) {
   //statements;
} else if (condition){
   //statements;
}
```

- **for statement**

A for loop should have the following format.

```
for (initialization; condition; update)
{
   //statements;
}
```

- **while statement**

A while statement should have the following format

```
while (condition)
{
   //statements;
}
```

- **do while statement**

```
do {
   statements;
} while (condition);
```

- **switch case statement**

```
switch (condition)
{
         case abc:
         //
         break;
```

```
            case def
            //
            break;
            default;
            //
            break;
}
```

- **try-catch statement**

```
try{
    //statement
}catch(ExceptionClass e){
     e.printstacktrace();
}
```