Java Coding Style Guidelines and Standards

Introduction:

IIC has started a new division "**Innovation Center (IC)**" to provide GIS based solutions and products such as displaying GIS Charts/Maps, analyzing GIS data etc. to its various clients. IC intend to follow the coding standards while developing GIS applications so that the development team and management teams will be in sync.

Why coding standards?

Successful software tends to live a long time:

- Bugs are fixed;
- New features are added;
- New platforms supported;
- Software adapted to new markets.

That is, successful software development is a long term activity. If you plan to be successful, you should therefore plan for your code base to be maintained by a succession of many different programmers over a period of many years. Not planning for that is planning to fail.

This is the primary reason why you want a company coding standard: to make long term code maintenance easier.

The Importance of coding standard are:

- Improved code maintainability.
- Improved code quality. In particular: testability, supportability, portability, security, efficiency, robustness, high cohesion/loose coupling.
- Improved development speed. Your developers don't need to make all decisions starting from first principles.
- Better teamwork. Don't waste time with needless debates. Spend your creative energy on things that matter.
- Make it quicker and easier for a developer to move to a different project or team.
- Better performance under pressure. Under stress, people fall back on what they've been trained to do.
- Fewer bugs. Good standards should minimize common coding mistakes.

Acknowledgements

This document reflects the Java language coding standards presented in the Java Language Specification , from Sun Microsystems, Inc.

Few Naming and Coding conventions are as follows

1. **Java Packages :**
   Naming for java packages should be in all lower cases
   Ex :
   ```
   package com.iic.hdc.model
   package org.iic.shoalanalysis.controller
   package org.springframework.web.servlet
   ```

   Here the first the initial package name respresents domain name, company name, project name, classes of types

2. **Classes:**
   Naming for java classes must be nouns written in MixedCase starting with upper case
   Ex : `ShoalAnalysis, GeometricInformation, Employee`

3. **Variables :**
   Naming for variables must be nouns and must be mixedCase starting with lower case
   Ex : `ShoalAnalysis shoalAnalysis;` // ClassName variableName

   ```
   Integer emoployeeId; //ClassName variableName

   String analysisName;

   double piValue;

   float salary;

   Date startDate;
   ```
   In the above all examples, first one represents Class Name and next one is variable names.

4. **Final Variables (Constants) or Static Variables:**
   Naming for Final Variables or Static Variables must be all UPPER_CASE using underscore to separate words.
   Ex : `private final String CLASS_PATH=”C://sample.txt”;`
   ```
   private final int PI_VALUE =3.414;


   // static variable declaration
   Public static final String FILE_NAME = “login.properties”;
   Public static final double MAX_VALUE= 99.09;
   ```

5. **Methods:**
   Naming for methods must be verbs and written in mixed case starting with lower case
   Ex : `public void calculateSalary();`
   `Public double getSalary();`

```
Pubic double computeTotalWidth();
```

6. **Getter/Setter methods:**
   The terms get/set must be used where an attribute is accessed directly.
   Ex:
   ```
   Employee employee = new Employee();

   employee.setName(String name); // setter
   employee.getName(); // getter
   ```

7. **Boolean variables :**
   'is' prefix should be used for Boolean variables and methods
   Ex:
   ```
   boolean isSet = true;
   boolean isVisible = false;
   boolean isFound = true;
   ```

   some time few alternatives to 'is' that fits better in some situations.

   ```
   boolean hasNext();
   boolean canEvaluate();
   boolean shouldStart();
   ```

8. **Collection variables names:**
   Plural form should be used on names representing a collection of objects.
   Ex:
   ```
   String name; // for single object
   int year; // for single object

   Collection<String> names; // can be used
   Collection<String> namesCollection; // can be used

   String[] names;
   Int[] years;

   List<String> namesList;
   Set<String> nameSet;
   Map<String, String> namesMap;
   ```

9. **Iterator variables**: namely, for, while, do while
   Iterator variables should be called i, j, k

   Ex :
   ```
   for(int i=0; i < = 10; i++){
         // business logic
   }

   int i = 0;

   while(i<10){
   ```

```
        system.out.println(i);

        i++;

}
```

10. No abbreviation Names:

Abbrevation names for the variables should be avoided.

cmd // should not be used. Use 'command' instead
pt // should not be used. Use 'point' instead
ex // should not be used. Use 'exception' instead

11. Exception classes : Exception classes should be suffixed with Exception

Ex :
```
        ValueUnknownExcetion extends Exception{

        }

        InvalidReportException extends Exception{

        }
```

12. If-else statements : The if else statements should have the following form.

```
if(condition){
        statements ;
}

If(condition){
        Statements;
}else{
        Statements;
}

If(condition){
        Statements;
}else if(condition){
        Statemenets;
}else{
        Statements;
}


Swithch(int i){
        Case 1:
        Statement;
        Break;
        Case 2:
        Statement;
```

```
        Break;
        Default :
        Statement;
}
```

13. Comments :

Java doc comments should have the following form. Java programs can have two kinds of comments:

i) implementation comments :

Block Comments:

Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments may be used at the beginning of each file and before each method.

```
/*
* Here is a block comment.
*/
```

Single-Line Comments:

Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format.

```
if (condition) {
        /* Handle the condition. */
}
```

End-Of-Line Comments:

The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments;

```
if (foo > 1) {
        // Do a double-flip.
}else{
        return false; // Explain why here.
}


//if (bar > 1) {
//
// // Do a triple-flip.
// ...
//}
//else{
// return false;
//}
```

ii) Documentation Comments:

Doc comments describe Java classes, interfaces, constructors, methods, and fields. Each doc comment is set inside the comment delimiters /**...*/, with one comment per class, interface, or member. This comment should appear just before the declaration:

```
/**
 * Returns lateral location of the specified position.
 * If the position is unset, NaN is returned.
 *
 * @param x    X coordinate of position.
 * @param y    Y coordinate of position.
 * @param zone Zone of position.
 * @return     Lateral location.
 * @throws IllegalArgumentException If zone is <= 0.
 */
public double computeLocation(double x, double y, int zone)
throws IllegalArgumentException
{
      // statements ;
}
```

14. Oreder of the AccessModifiers in java:

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class. There are 4 types of java access modifiers and the order of the access modifiers are as follows

```
private int employeeId;
protected String address;
default String employeeName;
public void getEmployeeName();
```

variable fields should not be declared as public, because, there may be some implementation with that field with in the class. To access that fields, use the setters and getter method instead. The variable can be declared as public if it is STATIC or CONSTANT variable.

15. Try to reduce the method or function implementation to a small as much as possible :

Class should be implemented or created in such a way that, it should contain single responsible. single responsibility principle states that every class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.

Example :

```
/**
  *  email class is responsible for only sending the mails to the email addresses
  *  @author iictechnologies
  *  @version 1.0
  *  @Date xxxx-xx-xx
  */
Public class Email {
      /**
      * config the initial parameters to send the email to the users
      *
      * @param fromEmail
      *            to send from which email id
      * @param fromEmailPassword
      *            refers to password of the sender
      * @param smtpHost
      *            refers to host of the server configaration SMTP,TCP, etc
      */
      public void config(String fromEmail, String fromEmailPassword, String
      smtpHost){
            // implementation goes here
      }

      /**
      * send the email to the to email id addresses
      * @param mailTo
      * @param subject
      * @param body
      * @return the status of the email
      */
      public    void    send(String    toEmailID,    String    subject,    String
      messageContent){
            // implementation here
      }
}
```

In the above Email class, it only has the email related implementation. i.e. it is only responsible for sending the email. This is called the single responsibility.

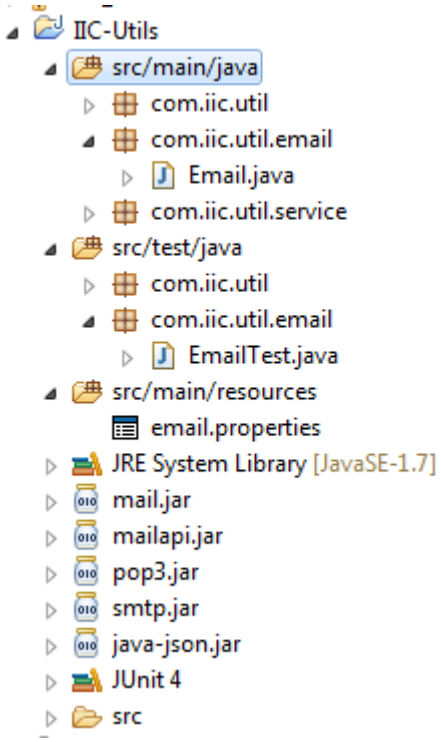
Suggested Prefixes for Controls:

Control type	Prefix	Example
Prompt	pmt	pmtName
Combo Box	cbo	cboList

Button	btn	btnOK
Radio Button	rbtn	rbtnMale
Text Box	txt	txtAddress
Group Box	gbx	gbxGroup
Table	tbl	tblEmp
List	lst	lstItems
Check Box	chk	chkReadOnly
Tree	tre	treOrganization
File Open Dialog	fdlg	fdlgSource
Menu Bar	mnubar	mnubarMenu
Tool Bar	tbar	tbarTolls
Status Bar	sbar	sbarStatus
Dialog	dlg	dlgDialog
Line Edit	ledt	ledtEdit
Text Edit	txt	txtEdit
Text Box	txt	txtBox
Push Button	pbtn	pbtnButton
Label	lbl	lbl

16. A typical Java project structure as follows.

The java project structure can be the following.



Here,

src/main/java folder contains the packages, which includes the business logic implanted classes.

src/test/java folder contains the same packages, which includes the test classes for the business classes that are there in the main folder. Which also includes the junit test classes.

src/main/resources folder contains the resources file like, properties files or xml files.