# Standard coding Practice

## Coding standards for RAC (java)

1. Class names:

Class names should be meaningful & class name should be noun starting with uppercase letter. If it contains multiple word then every inner word should start with uppercase.

Eg. CreateItem, StringBuffer, CustomerAccount

1. Class should have comments at top level

Eg.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Name : Class name

Author : XXXXX

Created on : XX/XX/XXXX

Modified on :

Description : XXXXX.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Methods names:

A method should be readable enough so that it conveys the functionality that it actually encapsulates.

Method names should be written in camel case i.e the first letter of the first word is in lower case and the first letter of the remaining words in method names are capital.

Eg: print(), sleep(), setSalary()

1. Function comments should be present before method starting, below syntax should be used for method comment

/\*\*

    \* Method description.

    \* @param First parameter desctiption to method

    \* @param Second parameter description to method

    \* @return returnType description.

    \*/

Eg.

 /\*\*

    \* This method is used to find average of three integers.

    \* @param numA This is the first parameter to findAvg method

    \* @param numB  This is the second parameter to findAvg method

    \* @param numC  This is the second parameter to findAvg method

    \* @return int This returns average of numA, numB and numC.

    \*/

    public int findAvg(int numA, int numB, int numC)

    {

        return (numA + numB + numC)/3;

    }

1. Line Comments
2. Single – line comments.

Syntax: //Comments here

1. Multi-line Comments

To describe a complex snippet single line comments can be tedious to write, since we have to give ‘//’ at every line. So to overcome this multi line comments can be used.

Syntax:

/\*Comment starts

continues

continues

.

.

Commnent ends\*/

1. Variable names should be meaningful and It should be categorized according to data type. Variable name should start with lowercase letter. If it contains multiple words than every inner word should start with uppercase.

Eg. String sItemId = null / String sItemId = “”; //For string it should start with s

Integer iItemCount = 0; //For integer it should be start with

Double dItemCost =0.0 //For Double it should be start with d

1. Code should be modularized and divided into functions
2. Don’t put your code directly into main or starting function. Call separate function and write your code

Eg.

public class SampleHandler extends AbstractHandler {

/\*\*

\* The constructor.

\*/

public ProjectLevelHandler() {

}

/\*\*

\* the command has been executed, so extract extract the needed information

\* from the application context.

\*/

public Object execute(ExecutionEvent event) throws ExecutionException {

functionName funObject= new functionName(paramenters);

funObject.callUI();

}

}

1. Write wrapper function for redundant code.
2. Coding comments should be present before logic
3. Avoid multiple looping code into same function
4. Hardcode values strictly avoided (Which is very critical to change value in future). Create property or constant file same like existing plug-in jar
5. Code should be structured and avoid unwanted spaces.
6. Function size ( Function should not extend number of lines ( limit). If size extend beyond this then create separate functions
7. Error handling should be proper
8. Unit testing should be proper
9. Test cases should be there after unit testing

## Coding standards for ITK (c/c++)

1. Memory Management should be proper in ITK code (memory allocation and memory free should be proper).It will increase the system performance.
2. Class name should be meaningful
3. Class should have comments at top level

Eg.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class Name : Class name

Author : XXXXX

Created on : XX/XX/XXXX

Modified on :

Description : XXXXX.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Function comments should be present before function starting

Eg.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name : Function name

Author : XXXXX

Created on : XX/XX/XXXX

Modified on :

Description : XXXXX.

Inputs : XX,XX

Outputs : XX

Returns : XX

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Line Comments
2. Single – line comments.

Syntax: //Comments here

1. Multi-line Comments

To describe a complex snippet single line comments can be tedious to write, since we have to give ‘//’ at every line. So to overcome this multi line comments can be used.

Syntax:

/\*Comment starts

continues

continues

.

.

Commnent ends\*/

1. Variable Names should be meaningful and It should be categorized according to data type
2. Variable should be initialized before use

Eg.

tag\_t \*tAttachments = NULL //For tag\_t it should start with t

tag\_t tItemTag = NULLTAG

int ifail = ITK\_OK; //For Integer it should start with i

char \*cObjectTypeName; //For char\* it should start with c

string sOutstr, sFileName ; //For string it should start with s

1. Code should be modularized and divided into functions
2. Don’t put your code directly into main or starting function. Call separate function and write your code
3. Hard coded values strictly avoid, For hard coded values should use macro for same (#define)

Eg. #define ITEMREVISION = “ItemRevision”

1. Function should be declared in header file

/\* File foo. \*/

#ifndef FILE\_FOO\_SEEN

#define FILE\_FOO\_SEEN

the entire file

#endif /\* !FILE\_FOO\_SEEN \*/

1. Coding comments should be present before logic
2. Avoid multiple looping code into same function
3. Error should have trace call and macro return value should goes to syslog and return text so user can get idea if anything missed. Same as teamcenter error and syslog trace.
4. Unit testing should be proper
5. Test case creation after unit testing Document