**Coding Guide Lines**

**Java Source Files**

Files longer than 2000 lines are cumbersome and should be avoided

Java source files should have the following ordering:

1. Beginning comments
2. Package and Import statements
3. Class and interface declarations

Beginning Comments – All source files should begin with a comment that lists the copyright notice, as shown below.

/\*  
 \* Copyright 2020 ONPASSIVE AI Company  
 \*  
 \* All Rights Reserved.  
 \* No portions of this source code or the resulting compiled program may be

\* used without express written consent and licensing by ONPASSIVE  
 \*/

**Indentation**

Avoid lines longer than 120 Characters.

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.

Align the new line with the beginning of the expression at the same level on the previous line.

Following are examples :

someMethod(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

 var = someMethod1(longExpression1,

someMethod2(longExpression2, longExpression3));

**Comments**

*Every published class and interface should have a Javadoc comment.*

Every published static member variable should have a Javadoc comment.

Every published method should have a Javadoc comment, including @param, @return and @throws tags for all its parameters, its return value and its thrown exceptions, when applicable.

*For more information on writing JavaDoc comments can be found at below URL:*

[https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html#format](https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html)

**Statements**

Each line should contain at most one statement.

argv++; // Correct

argc++; // Correct

argv++; argc--; // AVOID!

Braces are used around all compound statements – even single statements or empty blocks – when they are part of a control structure, such as an if-else or for statement.

// AVOID: THIS OMITS THE BRACES {}

if (condition)

statement;

// AVOID: THIS OMITS AN EMPTY BLOCK

for (init; condition; update);

// Instead use the following :

if (condition) {

statement;

}

for (init; condition; update) {

}

if (condition) {

statements;

}

 if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else if (condition) {

statements;

} else {

statements;

}

for (initialization; condition; update) {

statements;

}

**White Spaces**

Blank lines improve readability by setting off sections of code that are logically related.

*One blank line should always be used between methods*

*One blank line should always be used between logical sections inside a method to improve readability*

*One blank line should always be used before a block or single-line comment.*

Blank Spaces example :

while (true) {  
 ...  
}

for (init; condition; update)

a += c + d;  
a = (a + b) / (c \* d);

**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.

Files and folder names should be in smaller case,

E.g., : image.gif, connect.jsp

*Java source files should follow the class name’s casing.*

E.g., : SomeInterface.java, SomeClass.java

Package Names should always be written in all lowercase.

Package prefix for all components should be com.onpassive.cp

Interface and class names should be self descriptive.

Constants should be all CAPS.

*Methods should be verbs, in camel-casing.*

*E.g.,* ***:*** run(); runFast(); getBackgroundColor();

**Programming Practices**

1. Providing access to instance and class variables.
2. Do not make any instance variable public.
3. Do not define same constant in multiple classes.
4. Use Enum types for sake of readability.
5. Do not implement Serializable unless necessary.
6. Do not throw Throwable, as the Throwable indicates a generically unrecoverable run-time error.
7. Do not throw Exception, use specific subclass instead.
8. Do not catch Exception generically, unless at the very highest level of the application, except in Junit.
9. Do not throw exceptions from inside finalize() method.
10. Only catch and re-throw exceptions when you want to specialize the exception.
11. *Use StringBuilder or String.Format for construction of strings*
12. *Do not cast types where a loss of precision is possible.*
13. Use **Log4j** for logging in the system.

**Software and Versions**

* Spring Boot – 2.3.3
* Java – 11
* MySQL – 8

**Definition of Done**

JUnit Code Coverage needs to be more than 80%