COMMENTS :

- Each function should have a multi line comment describing it's purpose with parameters and return types

```
/**
 * This is         // And so          /* Or you can
 * okay.           // is this.        * even do this. */
 */
```

- Appropriate comments should be used in case of further development or optimization required in a piece of code with TODO tag
- Commenting single line of code can be done by // symbol, it should not be applied to multiple lines of code, for that use multi line commenting
- For every issue fix, one need not add the issue number in the comments, use the commit message instead. **ICAT-555 : <MESSAGE>**
- Doc comments are meant to describe the specification of the code,from an implementation-free perspective. /** ……. */ . Doc comments can be extracted to html using javadoc tools

BRACES :

- Braces are used with **if, else, for, do** and **while** statements, even when the body is empty or contains only a single statement.
- No line break before the opening brace.
- Line break after the opening brace.
- Line break before the closing brace.
- Line break after the closing brace, only if that brace terminates a statement or terminates the body of a method, constructor, or named class. For example, there is no line break after the brace if it is followed by **else** or a **comma**.

LINE BREAK :

- When a line is broken at a *non-assignment* operator the break comes *before* the symbol. Ex : the dot separator (.), the two colons of a method reference (::), an ampersand in a type bound (<T extends Foo & Bar>), a pipe in a catch block (catch (FooException | BarException e))
- When a line is broken at an *assignment* operator the break typically comes *after* the symbol, but either way is acceptable.
- A method or constructor name stays attached to the open parenthesis
- A comma (,) stays attached to the token that precedes it.
- Line of code should be broken before it reaches 80 char length due to other terminals wrapping criteria and plus long horizontal scroll on code is not welcome.

INDENTATION :

- When line-wrapping, each line after the first (each *continuation line*) is indented at least +4 from the original line.
- Code with characters exceeding the line limit should be wrapped appropriately as per the rule in the **line break section.**

SWITCH CASE :

- Each switch statement includes a **default** statement group, even if it contains no code.
- A switch statement for an enum type *may* omit the default statement group, *if* it includes explicit cases covering *all* possible values of that type. This enables IDEs or other static analysis tools to issue a warning if any cases were missed.

MODIFIERS :

- Class and member modifiers, when present, appear in the order recommended by the Java Language Specification.
  public protected private abstract default static final transient volatile synchronized native strictfp

NUMERIC LITERALS :

- **Long** -valued integer literals use an uppercase **L** suffix, never lowercase (to avoid confusion with the digit 1). For example, **3000000000L** rather than **3000000000l**.

NAMES :

- Package names are all lowercase, with consecutive words simply concatenated together (no underscores). For example, **com.example.deepspace**
- Class names are UpperCamelCase
- Method names are lowerCamelCase
- Constant names use CONSTANT_CASE: all uppercase letters, with each word separated from the next by a single underscore.
- Local variables and Parameters names are lowerCamelCase

- Variable names should not be used with non meaningful terms, names should identify its purpose, variables should be declared at the beginning of the scope of the block.

EXCEPTION HANDLING OPERATIONS :

- it is very rarely correct to do nothing in response to a caught exception. (Typical responses are to log it, or if it is considered "impossible", rethrow it as an AssertionError.)
- All operations which includes external resource or dependency of any kind should be included in a try catch block.

STATIC MEMBERS :

- When a reference to a static class member must be qualified, it is qualified with that class's name, not with a reference or expression of that class's type.

```
Foo aFoo = ...;
Foo.aStaticMethod(); // good
aFoo.aStaticMethod(); // bad
somethingThatYieldsAFoo().aStaticMethod(); // very bad
```

OPERATION PRACTICES :

- For making a long string by adding different small strings always use append method of **java.lang.StringBuffer / Builder** and never use ordinary **'+'** operator for adding up strings

- For easing out the work of java Garbage Collector always set all referenced variables used to 'null' explicitly thus dereferencing the object which is no more required by the application and allowing garbage Collector to swap away that variable thus realizing memory.

- Any session or driver occupied for the code must be released at the end of the operation.

- Logger use should be optimized as it requires disk I/O for every operation

- It is a good practice to use parentheses involving mixed operators to avoid operators preceding problems.
  Ex : if (a == b && c == d)   // AVOID!
         if ((a == b) && (c == d)) // RIGHT
- If an expression containing a binary operator appears before the ? in the ternary ?: operator, it should be parenthesized. Example:
  Ex : (x >= 0) ? x : -x;

- **Special Comments :** Use XXX in a comment to flag something that is bogus but works. Use FIXME to flag something that is bogus and broken.

- Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.

- Avoid string concatenation using '+' when you think it is going to be a big string in the end.

- Decide when you need a wrapper or a primitive

- Use Object Oriented feature of Superclass when multiple subclasses might be required to fit in.
  Ex : For example, java.lang.Number, could be considered as a bigger type than java.lang.Integer, since it can also contain java.lang.Double, java.lang.Float, and so on.

- Avoid returning null arrays or list from any api.

- Understand the difference between the time taken in different types of looping.
  Ex : For each which uses iterator internally and for loop which uses counter.
  Plus you can handler counters in various ways

- Code style of the newly added code block should be as consistent as possible with the project’s style.

- Allow one space between operator and operand

-

MEMORY LEAKS :

It is important to understand the meaning of different types of memory leaks as it comes most of the times in production env

Java Heap Space out of memory : It happens mostly when we have contiguous memory allocations with high indexes with high lifetime. Second scenario is when jvm has to manage unintentional managed references which should have been nullified.

PermGen Space out of memory : It happens when we have large number of class being loaded or String pool exhaustion.

ERRONEOUS SITUATIONS TO AVOID

1. We have classes which implements Comparable interface and overrides the compareTo. It is advisable to override the equals as well here because

   (x.compareTo(y)==0) == (x.equals(y))

   And this is not the case here because of missing equals override

   Ex : As we know that for sorting the collection of such class requires it to be comparable but for using the same in priorityqueue, the method like remove requires equals

2. Serializable request and response classes have non-transient non-serializable instance field.

3. User Exception class should extend the Exception class. Ex : MasterDataException

4. Class name starting with small character. Ex : clmParentProcess.java

5. In class ExcelColumnConstants, there are non final keywords starting with Caps

6. Streams not closed in various places.

7. String literals and objects comparison using == | != at various instances

8. File operations response are not checked at various locations, the response should be tracked and appropriate actions are required in case of any failure.
   Ex : f.delete() in CCFServiceImpl at line 2755

9. There are various fields in ApplicationConstant, Validaiton.ResponseCode and ExcelColumnConstant which needs to be finals

10. Null pointer exception scenarios handling.
    Ex : CCFServiceImpl.uploadFileToAWSAndSendLink() -> outStream.close();
    UploadMasterServiceImpl.uploadMasterFile -> File[] listOfFiles = folder.listFiles();
    if(listOfFiles.length > 0) // exception prone
    listFiles() returns null when folder abstract pathname does not denotes a directory

11. Null check on already dereferenced objects.
    Ex : SmartContractServiceImpl.searchCPMO() checks on smartContractNews after
        operating size attribute of the check which will throw null check exception
12. Multiple Switch case instances missing DEFAULT

13. Useless Objects being created on heap
    Ex : SmartContractNew.IfrsAnswer.mapIfrsAnswer creates ifrsAnswer
        ClmServiceImpl.createAdditionalJobs creates Set<String> itemsToBeAdded
        UploadMasterServiceImpl.uploadMasterFile creates indexReport object @ line 771