Java Source Files

Each Java source file contains a single public class or interface. When private classes and interfaces are associated with a public class, you can put them in the same source file as the public class. The public class should be the first class or interface in the file.

Java source files have the following ordering:

- Beginning comments
- Package and Import statements
- Class and interface declarations

Line Length

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Wrapping Lines

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.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Simple Statements

Each line should contain at most one statement. Example:

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

Compound Statements

Compound statements are statements that contain lists of statements enclosed in braces "{ statements }". See the following sections for examples.

- The enclosed statements should be indented one more level than the compound statement.
- The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement.
- Braces are used around all statements, even single statements, when they are part of a control structure, such as a if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

return Statements

A return statement with a value should not use parentheses unless they make the return value more obvious in some way. Example:

```
return;

return myDisk.size();

return (size ? size : defaultSize);
```

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

The if-else class of statements should have the following form:

```
if (condition) {
    statements;
}

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

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

Note: if statements always use braces {}. Avoid the following error-prone form:

```
if (condition) //AVOID! THIS OMITS THE BRACES {}!
    statement;
```

for Statements

A for statement should have the following form:

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

An empty for statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:

```
for (initialization; condition; update);
```

When using the comma operator in the initialization or update clause of a for statement, avoid the complexity of using more than three variables. If needed, use separate statements before the for loop (for the initialization clause) or at the end of the loop (for the update clause).

while Statements

A while statement should have the following form:

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

An empty while statement should have the following form:

```
while (condition);
```

do-while Statements

A do-while statement should have the following form:

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

switch Statements

A switch statement should have the following form:

```
switch (condition) {
case ABC:
   statements;
   /* falls through */

case DEF:
   statements;
   break;

case XYZ:
   statements;
   break;

default:
        statements;
        break;
    }
```

Every time a case falls through (doesn't include a break statement), add a comment where the break statement would normally be. This is shown in the preceding code example with the /* falls through */ comment.

Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added.

try-catch Statements

A try-catch statement should have the following format:

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
}
```

A try-catch statement may also be followed by finally, which executes regardless of whether or not the try block has completed successfully.

```
try {
    statements;
} catch (ExceptionClass e) {
    statements;
} finally {
    statements;
}
```

Naming convention - PascalCase

The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized. Examples :

```
var UserName  =  “”;
var Password = “”;
void GoToHomePage(){
}
```

Do not use a lot of comments in codes, naming clearly, shortly instead,

Config:

Config setting parameters should be defined in .env file such as:

Browser

Browser size

Browser mode

Admin credential - Test user credential (These will be used in many case, would be different in another test environment (local - staging). We do not need to change the code if run test on cross environments.

Pages package:

All class in Pages package stand for each Page (screen) on website. Define a class should extends BasePOM class from Base package. Examples :

```
public class CommsPage extends BasePOM{
   public CommsPage(WebDriver driver) {
      super(driver);
   }
## Your code here
###########
##################
}
```

Steps package:

Each class in Steps package stand for steps of each Page. Define a class should extends BaseSteps class from Base package. Examples :

```
public class HomepageSteps extends BaseSteps{

   public HomepageSteps(BaseUtils base, DataStore data) {
      super(base, data);
   }
## Your code here
###########
##################
}
```

FuncModules class:

All common functions which able to share through many class, steps should be defined there.
All functions should be public.
Use defined functions from this class in any class extends BasePOM class.

Feature file:

A scenario should follow gherkin language:
Given: < precondition step >
And (if need - continue precondition step)
When: < Action for testing >
And (continue action for testing if need)
Then (Expect result after

Source code management:

Create each branch for each ticket
Commit frequently
Clean project before commit - push.