**GUIDELINES DOCUMENTATION FOR Find Interest Amount for Current Year (emicalculator.net)**

# Submitted to

## Cognizant Technology Solutions

**Submitted by**

**Team-HACKATHON XPRESS**

Shaik Riyaz Ahemed (899563)

Priya Gahloth(899623)

Sayed Sirajunnisa (899512)

Shaik Rubeena(899583)

Nagendla Sarath (899510)

For more information please visit on website:

## https://emicalculator.net

**Date of Submission**

19,April2021



**TEAM HACKTHON EXPRESS**

**April2021**

**Version 2.0.0**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **PREPARED BY** | **REVIEWED BY** | **APPROVED BY** |
| Name | Hackathon Express Team |  |  |
| Signature |  |  |  |
| Date |  |  |  |

TABLE OF CONTENTS

[1.0 INTRODUCTION](#_Toc506753065) 4

[1.1 Purpose](#_Toc506753066) 4

[1.2 Scope](#_Toc506753068) 4

[1.3 Intended Audience](#_Toc506753066) 4

[2.0 NAMING COVENTIONS](#_Toc506753070) 5

[2.1 Project Elements](#_Toc506753071) 5

[2.2 Variables](#_Toc506753072) 5

[2.3 Functions & Methods](#_Toc506753073) 6

[3.0 CODE ORGANISATION AND BEST PRACTICES](#_Toc506753079) 7

[3.1 Code Commenting Conventions](#_Toc506753080) 7

[3.2 Structured Coding Coventions](#_Toc506753085) 7

[3.3 Parameter Usage Guidelines](#_Toc506753089) 7

[4.0 COMMENTING STANDARDS](#_Toc506753090) 8

[4.1 Types of comments](#_Toc506753090) 8

[4.2 Style](#_Toc506753090) 8

[5.0 INDENTION](#_Toc506753090) 11

[6.0 AUTOMATION SCRIPTS](#_Toc506753090) 12

[6.1 Generic.fecth data](#_Toc506753090) 12

[6.2 Generic.global](#_Toc506753090) 12

[6.3 Page Factory](#_Toc506753090) 12

[6.4 Test Data](#_Toc506753090) 13

[6.5 Test Results](#_Toc506753090) 13

[6.6 Test Scripts](#_Toc506753090) 13

[7.0 GENERAL GUIDLINES](#_Toc506753090) 14

**1.0. INTRODUCTION**

# 1.1. Purpose

This document consists a collection of standards, conventions, and guidelines for designing and developing framework/scripts for java code in selenium automation. It will help to ensure consistency across the code, resulting in increased usability and maintainability of the developed code.

**1.2. Scope of this Document**

This Document is going to give the complete details regarding the project. This includes Testing Approaches, Testing Techniques, Coding Standards, Testing Standards and Communication channels etc.

This document will be reviewed by –

* Trainer
* Academy Coach
* Business Unit (BU) Person
* Project Team

**1.3. Intended Audience**

|  |  |
| --- | --- |
| **BUSINESS IDEA** | **TARGET AUDIENCE** |
| Car Loan | Customers who are intended to take Car Loans. |
| Home Loan | Customers who are intended to take Home Loans. |
| Loan Calculator | Customer who intended to take Loans |

# 2.0. NAMING CONVENTIONS

# 2.1. Project Elements

|  |  |  |
| --- | --- | --- |
| **S.NO** | **PROJECT ELEMENT** | **NAMING CONVENTIONS** |
| 1 | Classes | Descriptive name of entity |
| Browser - For opening the browser page |
| Read– Reading the data from Excel sheet. |
| Write –Writing the output to Excel Sheet. |
| Screenshot - Taking the screenshot of results |
| Testcases - Checking the functional and UI Testcases |
| Smokesuite - Checking the Smokesuite Testcases |
| Regressionsuite - Checking the Regressionsuite Testcases |
| 2 | Interfaces | TakeScreenshot, WebDriver, FindsByXPath |
| 3 | Methods | Read and Write Details |
| Taking Screenshot |
| Providing the result reports |
| 4 | Property | Providing the application properties for selection of respective browsers of given application |

# 2.2. Variables

# 

* + Variable names should be defined with data type abbreviation followed by the actual variable name (English descriptors that accurately describe the variable/field/class/interface)

Example:

* + String strMethodName
  + object objObjectName
  + Domain specific terminologies must be used.
  + Mixed case should be used to make names readable.

# 

# 2.3. Functions / Methods

# **Some functions and methods are**

# 1. Open URL in different browsers (Chrome, Firefox) 2. Do mouse-hovers 3. Taking snapshots for analyzing the failures 4. Getting report results 5. Handling a drop-down

**3.0. CODE ORGANIZATION AND BEST PRACTICES**

**3.1 Code Commenting Conventions**

Code conventions are important to programmers for a number

Of reasons:

* 80% of the lifetime cost of a piece of software goes to maintenance and hardly any software is maintained for its whole life by the original author.
* Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

**3.2. Structured Commenting Conventions**

* Indent standard nested blocks one tab space.
* Indent the highest level statements that follow the overview comments one tab space, with each nested block indented an additional one tab space.

**3.3. Parameter Usage Guidelines**

* Check for valid parameter arguments. Perform argument validation for every public or protected method and property set accessor.
* The actual checking does not necessarily have to happen in the public or protected method itself. It could happen at a lower level in private routines. The main point is that the entire surface area that is exposed to the developer checks for valid arguments.

# 4.0. COMMENTING STANDARDS

Comments are a vital part of any program. They help the person reading the code (other project members) better understand the intent and functionality of the program.

## 4.1. Type of Comments

There are two types of comments that are used in programs:

## 1. Documentation comments

Documentation comments describe the interpretation of a class, field, or method. These are an integral part of programming and are mandatory in this class.

## 2. Implementation comments

In contrast, implementation comments are used to explain how a particular piece of code operates or to comment out a particular piece of code. Implementation comments are written when first necessary.

# 4.2. Style

Programs can have four styles of implementation of comments: block, single-line, trailing and end-of-line.

## a. Block Comments

* + - Block comments are used to provide descriptions of files, methods, data structures and algorithms.
    - They are used at the beginning of each file and before each method. They can also be used in other places, such as within methods.
    - Block comments inside a function or are indented to the same level as the code they describe.
    - A block comment is preceded by a blank line to set it apart from the rest of the code.
    - Block comments have an asterisk “\*” at the beginning of each line except the first.

## Example:

## /\*

\* Here is the block comment.

\*/

## b. Single-Line Comments

* + - Short comments appear on a single line indented to the level of the code that follows.
    - A single-line comment should be preceded by a blank line.

**Example**

if (condition)

{

/\* Handle the condition.

}

## c. Trailing Comments

* + - Very short comments generally appear on the same line as the code they describe, but are placed at sufficient distance, enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.
    - Commenting every line of executable code with a trailing comment should be avoided.

**Example**: if (condition)

{

return TRUE; /\* special case \*/

}

## d. End-Of-Line Comments

The // comment delimiter begins a comment that continues to the newline. It can comment out a complete line or only a partial line.

## Example:

// Instance of a class Class c = new Class ();

**5.0. INDENTATION**

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 space.

When an expression does not fit on a single line, it can be broken 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.

## 6.0. AUTOMATION SCRIPTS

Folder/package structure should be created based on the functionality/purpose of the scripts/files present in the respective folder.

**6.1. Generic. fetch data:**

This package should contain all the java files related to fetching test data from the excel sheet for data driven test automation framework.

**6.2. Generic. global:**

This package should contain all java files related to global functionalities

**Example**

* **runScriptWriteExce.java ->**This java file should contain all methods to write the status of TestCases to excel sheet.
* **UtilFunctions.java ->**This java file should contain all methods which are common across the application**.**

# 6.3. Page Factory

## This package should consists of all java files specific to the web element’s Ids present in the web page and domain specific functions

## Example: carLoan.java

# 6.4. Test Data

This package consists of all the excel sheets containing the test data and environment specific data if any. The test data for each module is placed in a separate excel document.

# 6.5. Test Result

This folder will contain all the log files as a resultant of test case execution. The log files are named as per the time of execution which makes the tracking of results easier. The Test results should be maintained in the form of log files which are generated runtime during every execution. These are saved as text documents for easy understanding. The messages provided within the scripts should be saved in the log files.

# 6.6. Test Scripts

This package consists of the test scripts for all module and setup script for the test execution. Test scripts names are decided based on its functionality and the test script should contain group of test cases related to that functionality.

# 7.0. GENERAL GUIDELINES

* Objects should be set to nothing for cleaning the memory.
* Only one variable should be declared in a line and all variable should be initialized as null/0/’ ’ while declaring them.
* There should not be more than 80 characters per line.
* The code should be properly indented.
* Variables should be declared using appropriate data types.
* Success/ failure can be logged inside sub-methods, instead of re-writing in all called places.
* Finite number of loops should be defined when we use “While” loops.
* If the first line of the method is failed, then control should move to catch block. It should not try executing second line of the method.
* Try-Catch-Finally blocks should be used for all methods.
* If similar kind of logic is used in more than one place, then reusable components should be used. Redundancy should be avoided.
* Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values.