JAVA / SELENIUM Coding Standards

Version 1.0

DXC.technology | American Airlines

Purpose of the document

The purpose of this document is to describe standards to be followed when designing and developing java/selenium code. This document will help ensure consistency across the code, resulting in increased usability and maintainability of the developed code.

Scope

The scope of this document is to provide standards for designing and developing java/selenium code.

Overview

This document provides standard guidelines for:

- Whitespace and Indentation
- Methods
- Control Statements
- Comments
- Variables
- Classes

White Spaces

Blank spaces should be used in the following circumstances:

- A keyword followed by a parenthesis should be separated by a space. Example:

```
//Avoid
while(true) {
   . . . .
}
```

```
//OK
while (true) {
   . . . .
}
```

Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.

- A blank space should appear after commas in argument lists.
- The expressions in a **`for`** statement should be separated by blank spaces. Example:

```
//Avoid
for(expr1;expr2;expr3)
```

```
//OK
for (expr1; expr2; expr3)
```

Braces

It is recommended to use { } on control statements. With Java's control statements such as if and for, the { } braces are technically optional if the body of the control statement contains only a single line. Regardless of this, always use the { } braces.

```
//Avoid
if (isPalindrome())
    break();
```

```
//OK
if (isPalindrome()) {
    break();
}
```

Line Breaks

Place a line break after every {

```
//Avoid
if (isPalindrome()) { return true; }
```

```
//OK
if (isPalindrome()) {
    return true;
}
```

Statements per line

Do not place more than one statement on the same line.

```
//Avoid
click(); selectItem(); close();
```

```
//OK
Click();
selectItem();
Close();
```

Methods

Naming

Give methods descriptive names, such as **`clickElement`** or **`editTextBox`**. Avoid one-letter names or non-descriptive names.

Capitalization

Name methods with camel-casing. Example, **`isPalindrome()`**, **`isElementVisible()`**.

Minimize Redundant Code

Avoid repetition of the same code block two or more times, find a way to remove the redundant code so that it appears only once. For example, you can place it into a method that is called from both places.

Long Methods

Each method should perform a single, clear, coherent task. No one method should do too large a share of the overall work. If you have a single method that is very long, break it apart into smaller sub-methods.

Returning booleans

If you have an if/else statement that returns a boolean value based on a test, just directly return the test's result instead.

```
//Avoid
if (score1 == score2) {
    return true;
} else {
    return false;
}
```

```
//OK
return score1 == score2;
```

Control statements

Avoid Empty if /else.

When using if/else statements, you shouldn't have an if or else branch that is blank. Rephrase your condition to avoid this.

```
//Avoid
if (isElementDisabled) {
   //do nothing
} else {
   clickElement();
}
```

```
//OK
if (isElementEnabled) {
    clickElement();
}
```

For vs **while**

Use a **for** loop when the number of repetitions is known; use a **while** loop when the number of repetitions is unknown.

```
// for: prints items of the list
for(int i=0; i < mylist.size(); i++) {
   print(mylist.get(i));
}
```

```
//while:
While(isElementVisible()) {
   clickNext();
}
```

Control statements

If/else patterns

When using if/else statements, properly choose between various if and else patterns depending on whether the conditions are related to each other. Avoid redundant or unnecessary if tests

```
//Avoid
if (points >= 90) {
    println("You got Gold!");
}
if (points >= 70 && points < 90) {
    println("You got Silver!");
}
if (points >= 50 && points < 70) {
    println("You got Bronze!");
}
...
```

```
//OK
if (points >= 90) {
    println("You got Gold!");
} else if (points >= 70) {
    println("You got Silver!");
} else if (points >= 50) {
    println("You got Bronze!");
}
...
```

Testing Booleans

Don't test whether a boolean value is == or != to true or false. It's not necessary! All booleans (expressions and variables) evaluate to true or false on their own.

```
//Avoid
if (x == true) {
    ...
} else if (x != true) {
    ...
}
```

```
//OK
if (x) {
    ...
} else {
    ...
}
```

Comments

Single line Comments

Short comments can appear on a single line indented to the level of the code that follows. If a comment can't be written in a single line, it should follow the block comment format. A single-line comment should be preceded by a blank line

```
if (condition) {

    /* Handle the condition. */
    ...
}
```

Trailing Comments

Very short comments can appear on the same line as the code they describe, but should be shifted far 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.

```
if (a == 2) {
    return TRUE;            /* special case */
} else {
    return isPrime(a);      /* works only for odd a */
}
```

Comments

Method Comments

Place a comment heading on each method. The heading should describe the method's behavior. If your method makes any assumptions (preconditions), mention this in your comments. If it leaves the program in a certain state when it is done (postcondition), mention this as well. Below is an example of a good method header comment:

```
/*
 * Function Name: readConfigurationFile
 * Author:
 * Date of Creation:
 * Description: This function reads the Configuration file and returns the value of the corresponding
                key.
 * Input Parameters: Key
 * Date Modified:
 * Reviewed By:
 *
 */
```

Trivial Comments

Do not include obvious or redundant comments about the meaning of statements

```
//Avoid
clickElement();  //clicks on element
editTextFiled(); //enters value in text filed
```

Comments

TODO’s

You should remove any // TODO: comments from your programs before turning them in.

Commented-Out Code

It is considered bad style to turn in a program with chunks of code "commented out". It's fine to comment out code as you are working on a program, but if the program is done and such code is not needed, just remove it.

```
//Avoid
public class Apples {
   public void working() {
/*
    System.out.println(“Program started executing”);
    System.out.println(“**************************”);
*/
    startRun();
    stop();
    startRun();
    startWalking();
  }
}
```

Variables

Declaration/Initialization

Never declare or initialize more than one variable in a single statement.

```
//Avoid
int a = 17; b = 8; c = 26;
```

```
//OK
int a = 17;
int b = 8;
int c = 26;
```

Constants

If a particular constant value is used frequently in your code, declare it as a constant, and always refer to the constant in the rest of your code rather than referring to the corresponding value. Name constants in uppercase with underscores between words.

```
//OK
public static final int DAYS_IN_WEEK = 7;
public static final int MONTHS_IN_YEAR = 12;
```

Classes

Private and instance variables

Properly encapsulate your objects by making any instance variables in your class private

```
public class person {
   private int age;
   . . . .
}
```

Initialize instance variables in Constructors

It is not recommended to initialize instance variables where they are declared; instead, initialize them in the constructor.

```
//Avoid
public class person {
   private int age = 26;
}
```

```
//OK
public class person {
   private int age;
//Constructor
public person() {
    age = 26;
  }
}
```

Classes

Instance vs Static

A static variable in a class is a single occurrence shared by that class and all of its objects. An instance variable is one that each object has its own copy of. When designing your class, any data that should be unique to each object should be an instance variable. This still holds even if the assignment only happens to construct one instance of your class

```
//Avoid
public class person {
//all persons share same age.
  private static int age;
}
```

```
//OK
public class person {
//each person has their own age.
  private int age;
public person(int a) {
      age = a;
   }
}
```

Thank you.