Contents

[Project Guidelines 1](#_Toc438649232)

[Global 1](#_Toc438649233)

[Variable Names 2](#_Toc438649234)

[Method Names 3](#_Toc438649235)

[Class Names 3](#_Toc438649236)

[Interface Names 4](#_Toc438649237)

[Package Name 4](#_Toc438649238)

[Client Application 4](#_Toc438649239)

[Latin abbreviations 5](#_Toc438649240)

[Structure 7](#_Toc438649241)

[Explanation 8](#_Toc438649242)

# Project Guidelines

## Global

##### Length:

* 1 char for loop counters,
* 1 word for condition/loop variables,
* 1-2 words for methods,
* 2-3 words for classes,
* 3-4 words for globals.

##### Characters consistently

Don't use lowercase/uppercase characters inconsistently: e.g. userName, UserName, USER\_NAME, m\_userName, username, ...:

* use Camel Case (aka Upper Camel Case) for classes: VelocityResponseWriter
* use Lower Case for packages: com.company.project.ui
* use Mixed Case (aka Lower Camel Case) for variables: studentName
* use Upper Case for constants : MAX\_PARAMETER\_COUNT = 100
* use Camel Case for enum class names and Upper Case for enum values.
* don't use '\_' anywhere except constants and enum values (which are constants).

##### Reuse variable

Don't reuse same variable name in the same class in different contexts: e.g. in method, constructor and class.

Don't use same variable for different purposes in a method, conditional etc.

## Variable Names

* Use camel case in variable names as well e.g. price, quantity, totalAmount etc.
* Use all caps for constants e.g. MAX\_QUANTITY, MAX\_PRICE etc.
* Avoid using non ASCII characters and words from local language! Using any character other than ASCII character, especially non English is worst practice. English is a universal language for programming and stick with it.
* All names start with a **letter.**
* If you have to represent a collection or plural, prefer something like listOfEmployees, bunchOfEmployees over employees.

##### Avoid Pointless Names:

No: abc, temp, data

##### Avoid Similar Names:

No: employee and employees

##### Avoid Clutters:

**No:** \_, m\_, o\_, simply \_, obj\_

##### Avoid Hungarian notation:

No: bExit for boolean variable, iMax for integer variables

Make good use of common verb e.g. is, has, can or do  
  
Yes: isAlive(), hasNext(), canExecute()…

Example:

if(isRaining){

bringUmbrella();

}

##### Use specific names for variables:

No: "value", "equals", "data"…

Follow Classical Programming Convention:  
  
i and j as loop counter in for loop

E.g.: for(int i=0; i<10; i++){ // your code }

##### Spaces Around Operators

Always put spaces around operators ( = + - \* / )

Yes: Integer x = 42; var x = y + z; var values = ["Volvo", "Saab", "Fiat"];  
No: Integer x=42;

## Method Names

* Start name of method from small character and follow camel case e.g. getEmployee(), getPayDate() etc.
* Instead of having three different methods destroy(), kill(), or finish() at different modules, prefer one of them e.g. destroy().
* Method names should start with verb e.g. get, set, do, invoke etc.
* Prefer shorter name over longer one, if it reveal intent clearly: getPayDate() is better than retreivePaymentDate().
* Prefer shorter name if and only if it reveal intent completely, otherwise choose longer and descriptive name: getLiquidityIndicator() is better than getLInd().
* If variable name is payDate then getter method name must be getPayDate()and setter method must be setPayDate().

##### Give Meaningful Names:

Yes: getPayDate()

No: getPD()

## Class Names

Start name of class as capital letter e.g. Employee, Student or Thread.  
  
Class name should be noun and should tell what does this class represent e.g. Employee, Thread, String etc.

## Interface Names

Interface name should describe ability or CAN DO part e.g.

[Runnable - can run](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html), Callable - can be called etc. 

## Package Name

Package name should follow standard company structure e.g. com.company.project.module.

## File Names

Always use lower case file names (if possible).

# Client Application

## JavaScript

##### Variables

* Variable and function names written as camelCase
* Global variables written in UPPERCASE (We don't, but it's quite common)
* Constants (like PI) written in UPPERCASE
* Don't start names with a $ sign. It will put you in conflict with many JavaScript library names.

##### Avoid Global Variables

* Minimize the use of global variables.
* This includes all data types, objects, and functions.
* Global variables and functions can be overwritten by other scripts.
* Use local variables instead.

##### Initialize Variables

It is a good coding practice to initialize variables when you declare them.

This will:

* Give cleaner code
* Provide a single place to initialize variables
* Avoid undefined values
* Initializing variables provides an idea of the intended use (and intended data type).

Example:

// Declare and initiate at the beginning  
var firstName = "",  
    lastName = "",  
    price = 0,  
    discount = 0,  
    fullPrice = 0,  
    myArray = [],  
    myObject = {};

##### Never Declare Number, String, or Boolean Objects

Declaring these types as objects, slows down execution speed, and produces nasty side effects

Yes: var x = "John";

No: var y = new String("John");

* Use {} instead of new Object()
* Use "" instead of new String()
* Use 0 instead of new Number()
* Use false instead of new Boolean()
* Use [] instead of new Array()
* Use /()/ instead of new RegExp()
* Use function (){} instead of new function()

##### Brackets

* Open-brace ("{") characters on the same line as the statement that opens the block.
* Use one space before the opening bracket
* Put the closing bracket on a new line, without leading spaces

if (condition) {

/\* handle the condition \*/

} else {

/\* handle the "else" case \*/

}

##### Code Indentation

Always use 4 spaces for indentation of code blocks:

function toCelsius(fahrenheit) {  
    return (5 / 9) \* (fahrenheit - 32);  
}

##### Line Length < 80 & Line breaks

* Long lines shouldn't be allowed - break at natural breaking points.
* For readability, avoid lines longer than 80 characters.
* If a JavaScript statement does not fit on one line, the best place to break it, is after an operator or a comma

Example 1:

if (class.CONDITION || class.OTHER\_CONDITION || class.SOME\_OTHER\_CONDITION

|| class.YET\_ANOTHER\_CONDITION ) {

/\* something \*/

}

var toolkitProfileService = Components.classes["@mozilla.org/toolkit/profile-service;1"]

.createInstance(Components.interfaces.nsIToolkitProfileService);

Example 2:

document.getElementById("demo").innerHTML =

"Hello Dolly.";

##### Object Rules

* Place the opening bracket on the same line as the object name.
* Use colon plus one space between each property and its value.
* Use quotes around string values, not around numeric values.
* Do not add a comma after the last property-value pair.
* Place the closing bracket on a new line, without leading spaces.
* Always end an object definition with a semicolon.

Example:

var person = {  
    firstName: "John",  
    lastName: "Doe",  
    age: 50,  
    eyeColor: "blue"  
};

## Typescript

## HTML

#### HTML elements

* **Correct**: the [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span) element
* **Incorrect**: the span tag

#### Comments on closing tags

After every major thing of HTML, for example, the end of a carousel, or the end of the content <div>, place a closing-comment.

Example:

...

</div><!-- /carousel -->

...

</div><!-- /content -->

#### Namespaced fragment identifiers

Nice way to add a little more meaning to your fragment identifiers and give a little bit more of a clue as to what they actually link to.

Example:

<a href=#section:fragment-identifiers>Fragment identifiers</a>

...

<div id=section:fragment-identifiers>...</div>

## CSS

#### Declaration order

##### /\* Positioning \*/

position: absolute;

z-index: 10;

top: 0;

right: 0;

bottom: 0;

left: 0;

##### /\* Display & Box Model \*/

display: inline-block;

overflow: hidden;

box-sizing: border-box;

width: 100px;

height: 100px;

padding: 10px;

border: 10px solid #333;

margin: 10px;

vertical-align: top;

white-space: nowrap;

##### /\* Other \*/

background: #000;

color: #fff;

font-family: sans-serif;

font-size: 16px;

text-align: right;

#### No IDs

This is more a technical thing, but I have a blanket-ban on IDs in CSS. There is literally no point in them, and they only ever cause harm. Everything that needs styling is done so without using IDs.

## Text

##### Title and heading capitalization

Use sentence-style capitalization (only capitalize the first word and proper nouns):

* **Correct**: "A new method for creating JavaScript rollovers"
* **Incorrect**: "A New Method for Creating JavaScript Rollovers"

### Latin abbreviations

#### In notes and parentheses

Common Latin abbreviations (etc., i.e., e.g.) may be used in parenthetical expressions and notes. Use periods in these abbreviations.

* Correct: Web browsers (e.g. Firefox) can be used ...
* Incorrect: Web browsers e.g. Firefox can be used ...
* Incorrect: Web browsers, e.g. Firefox, can be used ...
* Incorrect: Web browsers, (eg: Firefox) can be used ...

#### In running text

In regular text (i.e. text outside of notes or parentheses), use the English equivalent of the abbreviation.

* + **Correct**: ... web browsers, and so on.
  + **Incorrect**: ... web browsers, etc.
  + **Correct**: Web browsers such as Firefox can be used ...
  + **Incorrect**: Web browsers e.g. Firefox can be used ...

#### Meanings and English equivalents of Latin abbreviations

|  |  |  |
| --- | --- | --- |
| **Abbrev** | **Latin** | **English** |
| cf. | confer | compare |
| e.g. | exempli gratia | for example |
| et al. | et alii | and others |
| etc. | et cetera | and so forth, and so on |
| i.e. | id est | that is, in other words |
| N.B. | nota bene | note well |
| P.S. | post scriptum | postscript |

#### Plurals of acronyms and abbreviations

For plurals of acronyms or abbreviations, add s. Don't use an apostrophe. Ever. Please.

* **Correct**: CD-ROMs
* **Incorrect**: CD-ROM's

#### Hyphenation

Hyphenate compounds when the last letter of the prefix is a vowel and is the same as the first letter of the root.

* **Correct**: email, re-elect, co-op
* **Incorrect**: e-mail, reelect, coop

#### Dates

For dates (not including dates in code samples) use the format "January 1, 1990":

* **Correct**: February 24, 2006
* **Incorrect**: February 24th, 2006; 24 February, 2006; 24/02/2006

Alternately, you can use the YYYY/MM/DD format:

* **Correct**: 2006/02/24
* **Incorrect**: 02/24/2006; 24/02/2006; 02/24/06

#### Decades

For decades, use the format "1990s". Don't use an apostrophe.

* **Correct**: 1990s
* **Incorrect**: 1990's

#### Plurals of numerals

For plurals of numerals add "s". Don't use an apostrophe.

* **Correct**: 486s
* **Incorrect**: 486's

#### Commas

In running text, use commas only in five-digit and larger numbers.

* **Correct**: 4000; 54,000
* **Incorrect**: 4,000; 54000

#### Serial comma

**Use the serial comma**. The serial (also known as "Oxford") comma is the comma that appears before the conjunction in a series of three or more items.

* **Correct**: I will travel on trains, planes, and automobiles.
* **Incorrect**: I will travel on trains, planes and automobiles.

## Structure

ASP.NET Structure <- AngularJS & Typescript

ASP.NET Structure

AngularJS

Typescript

* Solution *Name*
* ***Name*Application**
* Properties
* References
* App\_Data
* Config
* Content
* Fonts
  + - Images
    - Scripts
      * Angular*Name*Project
* module.ts
* routes.ts
* Controllers
* ts*Name*Controller.ts

…

* Interfaces
* interfaces.ts

…

* Services
  + *name*Services.ts

…

* Views
  + partial.html

…

* + - * includeScript
* typings
* angularjs
* jquery

…

* Styles
  + less
* style.less

…

* + css
* style.min.css

…

* + includeStyles
* Controller
* *ProjectName*Controller.cs
* Services
* Models
* Views
  + - ProjectName
  + Index.cshtml
* Shared
  + \_Layout.cshtml
* \_ViewStart.cshtml
* web.config
* Global.asax
* packages.config.
* Web.config

# Explanation

Reasons for using a naming convention

* to reduce the effort needed to read and understand source code;[[1]](https://en.wikipedia.org/wiki/Naming_convention_(programming)#cite_note-1)
* to enable code reviews to focus on more important issues than arguing over syntax and naming standards.
* to enable code quality review tools to focus their reporting mainly on significant issues other than syntax and style preferences.
* to enhance source code appearance (for example, by disallowing overlong names or unclear abbreviations).

Documentation allows you to transfer the why behind code. Much in the same way code comments explain the why, and not the how, documentation serves the same purpose.

If people don’t know why your project exists,

they won’t use it.

If people can’t figure out how to install your code,

they won’t use it.

If people can’t figure out how to use your code,

they won’t use it.

You only get contributions after you have put in a lot of work.

You only get contributions after you have users.

You only get contributions after you have documentation.

Technical writing is an art that doesn’t come naturally. Writing documentation will start you down the road to being a better technical writer, which is a useful skill to have as a programmer.

### Documentation of Project:

What problem your project solves

Installation instructions

# References

*Code Magazin*. (n.d.). Retrieved from http://www.codemag.com/article/1405071

*DZone*. (n.d.). Retrieved from https://dzone.com/articles/best-practices-variable-and

*Javarevisited*. (n.d.). Retrieved from http://javarevisited.blogspot.rs/2014/10/10-java-best-practices-to-name-variables-methods-classes-packages.html

*W3School*. (n.d.). Retrieved from http://www.w3schools.com/js/js\_conventions.asp

*Wikipedia*. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Naming\_convention\_(programming)