**Generic coding style guide**

For all students within the

**School of Computing and Intelligent Systems**

**Faculty of Computing and Engineering**



**Table of contents**

[1. Introduction 2](#_Toc493858502)

[2. File structure and naming 2](#_Toc493858503)

[2.1 File naming 2](#_Toc493858504)

[2.2 Beginning comments 2](#_Toc493858505)

[2.3 Package and import statement 3](#_Toc493858506)

[3. Formatting conventions 4](#_Toc493858507)

[3.1 Indentation 4](#_Toc493858508)

[3.3 File length 4](#_Toc493858509)

[3.4 Braces 4](#_Toc493858510)

[3.5 Comments 4](#_Toc493858511)

[Block comments 4](#_Toc493858512)

[Single-line comments 5](#_Toc493858513)

[Trailing Comments 5](#_Toc493858514)

[End-Of-Line Comments 5](#_Toc493858515)

[4. Naming conventions 6](#_Toc493858516)

[4.1 Capitalisation rules 8](#_Toc493858517)

[5. Further reading 8](#_Toc493858518)

# Introduction

This document presents the core elements of a coding convention for all computing students within the School of Computing and Intelligent Systems. As part of your professional development you are expected to adopt a professional approach to writing code. Part of that includes using a consistent approach to authoring code. “*Why bother?*” I hear you ask, well:

* 80% of the lifetime cost of a piece of software goes to maintenance.
* Rarely is the maintenance undertaken by the original author. So many, many moons from now some random stranger may have to read your code and figure out just what you were thinking when you originally wrote it!
* Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
* It will make you more employable!
* Your employer will often require you to follow company specific coding conventions.

This document presents a coding convention for the main languages you will encounter within your studies. You should note that the style differs slightly between Java, C/C++ and C#.

# File structure and naming

## File naming

**Java**

Filename.java where Filename matches class name. This is a language requirement.

**C++**

Filename.cpp for the C++ source file and Filename.h for the header file. Where Filename matches the class name. The only exception here is when using C++ template classes all code is placed in the header file.

**C#**

Filename.cs where Filename matches class name. This is not a language requirement but should be done for consistency.

## Beginning comments

You should begin your source file with some comments that describe the file, This will normally take the format of:

/\*

\* Classname or filename)

\*

\* Version information

\* Author:

\* Date:

\*

\* Copyright notice

\*/

## Package and import statement

**Java**

The first non-comment line of a source file is the package statement. After that, import statements can follow:

package practical1;

import uulib.GUI;

**C#**

The first non-comment lines of a source file are the using (import) statements, followed by a namespace (package) statement

using System;

using System.Collections.Generic;

namespace Practical1.Models

{

// package content

}

**C++**

You should order #includes as follows:

Related headers

C Library

C++ Library

Other libraries’.h

Your project’s.h [ the relationship between and different treatment of these headers is unclear]

For Example:

#include "foo/server/fooserver.h"

#include <sys/types.h>

#include <unistd.h>

#include <hash\_map>

#include <vector>

#include "base/basictypes.h"

#include "base/commandlineflags.h"

#include "foo/server/bar.h"

# Formatting conventions

## 3.1 Indentation

Four spaces should be used as the unit of indentation. **Do not** use tabs to indent your code.

3.2 Line length

Avoid lines longer than 80 characters. Lines of 80 characters should be wrapped over multiple lines.

String s = "Hellooooooooooooooooo thisssssssssssssssssssssss isssssssssssssssss " +

" a verrrrrrrrrrrrrry long line";

## 3.3 File length

If you find that your source code file is great than 2,000 lines then it is a pretty good indication that your source code is overly complex and needs refactoring.

## 3.4 Braces

For Java and C/C++ you should keep the opening brace on the same line as the opening method as illustrated in the example1 below. In C# the convention is to have the opening brace on a new line as shown in example2.

|  |  |
| --- | --- |
| Java and C++ | C# |
| void example1() {  if (...) {  …  }  } | void Example2()  {  if (...)  {  …  }  } |

## 3.5 Comments

Use either the // or /\* \*/ syntax, as long as you are consistent.

### Block comments

Blocks of comments are used to provide descriptions of files, methods, data structures and algorithms. Blocks of comments may be 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 method should be indented to the same level as the code they describe.

A block comment should be preceded by a blank line to set it apart from the rest of the code.

/\*

\* Here is a block comment.

\*/

### 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. Here's an example of a single-line comment in Java code:

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 \*/

}

### 

### End-Of-Line Comments

The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code.

if (foo > 1) {

// Do a double-flip.

...

}

else {

return false; // Explain why here.

}

//if (bar > 1) {

//

// // Do a triple-flip.

// ...

//}

//else {

// return false;

//}

# Naming conventions

What you decide to call a function or variable is very important. It is one of the influential aids to understanding the logical flow of an application. A name should tell “*what*” rather than “*how*”.

**Do** choose easily readable identifiers

For example, a variable named flightDepartureDate is more readable than flghtDate

**Do** favour readability over brevity

The variable bool canScrollHorizontally is better than bool scrollX

**AVOID** using identifiers that conflict with keywords within the language

**DO NOT** use abbreviations or contractions as part of identifier names.

For example, use GetBalance rather than GetBal

|  |  |  |
| --- | --- | --- |
| **Identifier type** | **Rules** | **Examples** |
| Classes | Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. | class Raster;  class ImageSprite; |
| Interfaces | Interface names should be capitalised like class names | interface RasterDelegate;  interface Storing; |
| Functions / Methods | Functions / Methods should be verbs. For Java function names should be in camelCase, i.e. in mixed case with the first letter lowercase, with the first letter of each internal word capitalised.  For C++ and C# PascalCase should be used with function names starting with a capital letter.  Function names should not have underscores. | //Java  run();  runFast();  getBackground();  //C++ and C#  Run();  RunFast();  GetBackground();  //Java  int getCount();  void setCount(int count)  //C#  int GetCount();  void SetCount(int count)  //C++  int get\_count();  void set\_count(int count); |
| Class data members | C++  Data members should be mixed cases starting with a lower case and ending with an underscore ‘\_’  C# Class properties should be declared in mixed case with a lowercase first letter. e.g. userName  Using the property syntax, properties can be accessed like variables, yet still controlled via defined accessors and mutators (get and set functions) e.g. Length  Java  Class properties should be declared in mixed case with a lowercase first letter. | //C++  class SomeCase {  private: int length\_;  }  //C#  class SomeCase  {  private String userName;  public int Length {get; set;}  }  // Java  class SomeCase {  private int length;  private String userName;  } |
| Variables | Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.  Single underscores ‘\_’ may be used to separate words within a variable.  **Java**  Variable names should not start with underscore \_ or dollar sign $ characters, even though both are allowed. | int age;  char c;  float deskWidth;  bool ok\_button\_pressed; |
| Constants | The names of variables declared class constants should be all uppercase with words separated by underscores (“\_”). | static final int MIN\_WIDTH = 4;  static final int MAX\_WIDTH = 999;  static final int GET\_THE\_CPU = 1; |

## Capitalisation rules

To differentiate words in an identifier, capitalise the first letter of each word in the identifier. There are two appropriate ways to capitalise identifiers depending on the language and platform, camelCasing and PascalCasing.

|  |  |  |
| --- | --- | --- |
| **Language** | **Capitalisation Method** | **Example** |
| Java | camelCasing | int accountBalance; |
| C/C++ | camelCasing | int accountBalance; |
| C# | PascalCasing | int AccountBalance; |

Do Not give class names a prefix (e.g., “C”)

Consider ending the name of derived classes with the name of the base class. For example

ArgumentOutOfRangeException, which is a kind of Exception

Prefix interface names with the letter I, to indicate that the type is an interface.

For example, IComponent (descriptive noun), ICustomAttributeProvider (noun phase (, and IPersistable (adjective) are appropriate interface names.

# Further reading

The guideline presented here covers the most basic elements of a coding standard. You should note that the specific coding standard will vary between programming language, platform and company. A diverse range of programming style guides exist and some of the most established are listed here for reference:

**Java**

Oracle - Code conventions for the Java programming Language:

<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

Google - <https://google.github.io/styleguide/javaguide.html>

**Javascript**

Google - <https://google.github.io/styleguide/jsguide.html>

**C++**

Google: <https://google.github.io/styleguide/cppguide.html>

ISO coding standards: <https://isocpp.org/wiki/faq/coding-standards>

Bjarne Stroustrup: <http://stroustrup.com/JSF-AV-rules.pdf>

**C#**

**Microsoft -** <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>

Coding style conventions - <https://msdn.microsoft.com/en-us/library/windows/desktop/aa378932(v=vs.85).aspx>

Naming guidelines - <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines>

Coding techniques and programming practices - <https://msdn.microsoft.com/en-us/library/aa260844(VS.60).aspx>

**Apple**

Cocoa coding guidelines

<https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html#//apple_ref/doc/uid/10000146i>

**Swift**

<https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID309>

<https://swift.org/documentation/api-design-guidelines/>