Coding Standards for Triple Threat

Shiner, Raymond
raymondshiner@gmail.coml

Hoxie, Daylyn
dhoxie@eagles.ewu.edu

Curley , Spencer
scurley2@eagles.ewu.edu

2019-9-27

Contents

1 Intro and Purpose

Because Tom told us to
Otherwise - to unify the way in which TripleThreat writes software as a team. This document, although it may be changed, serves to function as the set of guidelines under which each member of TripleThreat will work on projects that represent the team. We will all try our best to adhere to these guidelines, to better function as a unit and to write better software.

2 Naming Conventions

2.1 Interfaces

Interfaces will be named in camel case always beginning with a lower case I and then the intended use of the interface. For example, a CarComparator interface would be named "iCarComparator"

2.2 For Loop Variables

in the case of for loops, counting variables shall be denoted as i (and then j and k for any nested loops). In appropriate situations where more complicated logic or iteration occurs, author may utilize more meaningful variable names such as cur, row, etc.
Example
(for(int i = 0; i < someNumber; i++))

2.3 Variable Names

Variables should named with meaningful names. To make the code easier for others to read.

2.4 Casing Styles

Camel Case will be use in almost all situations, as is according to recommended casing by the Java coding standards. (need to cite here). (Method Signatures, varibles, etc.)

2.5 class conventions

Pascale case will be used for Class names

3 Commenting Conventions

3.1 Files

At the top of each source file, a comment will list the name of the file, author, any revision authors, and a brief class description

3.2 Methods

If a constructor, method, or field's purpose would not be clear without an explanation, a brief comment should be included providing a description
At authors discretion, One line comments may be placed above any line or block of code that the author deems unclear to help clear confusion. (An example may be a manual insertion into a 2D array in which the context is not immediately clear based on the variable and method Names).

4 Spacing Conventions

4.1 Indentations

Indenting code is critical to make the code human readable.

For all indenting purposes a tab of size 4 will be used.

Code blocks will be indented as such, with the open brace on the same line as the start of the block, and the code inside the block beginning one line down and one tab indented. All code within this block will begin indendted one tab past code block declaration. (Example) The end brace for the code block will be placed on its own line at the end of the block, at the same tab indentation as the original block declaration.
EXAMPLE

```
while(a < 10){
        a++;
        a++;
}
```

4.2 Variable Declarations

variable type and name, name and assignment operator, operator and assignment value, will all be separated by a single space. (Example – "int a = 1;")

4.3 Method Calls

method names will not be separated from opening parentheses. Inside parentheses will be each variable, if any, separated by a comma and a single space with no spaces on either end. Example ("methodCall(var1, var2);")

4.4 Operators

all operands and operators will be separated by a single space, with the exception of unary operators (i++, I–);

4.5 Loops

spacing in all kinds of loops will be the same as method calls, with the loop declaration and any semi colons replacing the method name and commas in this situation. As for operations within any loop declaration, standard operation/operand spacing applies (one space between them).
(Example - "for(int cur = 0; cur < something; cur++) ——— while(a < 10)"

4.6 Semi-Colons

Whenever a semi colon is used to end a statement, it will go right after the last character in the given statement, no whitespace between them. After the end of the statement, a new line will always be used.
EXAMPLE – int I = 0;

5 Testing Conventions

5.1 Classes

Each production project located at src/projectName/projectName.csproj will have a corresponding unit test project located at test/projectName.Tests/projectName.Tests.csproj. In this example the C# Solution that links all relevant projects in the application both testing and not is located on the same package level as the src and test directories. Inside of each projects corresponding unit test project will be the same number of

C# classes as the production project, with each one being named identical followed by the word “tests” and example of such a structure is below

5.2 Methods

Unit Test Methods shall use the following system to be named, “TheMethodThatIsBeingTested_TheConditionsOfTheTest_Exp
purpose of this naming convention is to provide users of the unit test with as much information as possible so that when they are running/testing their code, they know exactly what each test is accomplishing without having to look at the source code for the tests. Example – A unit test that tests CarCompator’s compare method may be named as such. (in the file CarComparatorTests) “Compare_TwoIdenticalCars_Returns0” The Unit test of course would do just this, pass two identical cars into the comparator compare Method and verify that they in fact return the expected result.

EXCEPTION - When applicable to use data rows for reutilization of test logic with different test cases, the naming convention will be changed to TheMethodBeingTested_WhatGeneralTypeOfInput_ExpectedResult - The most common use case of this will be Valid or Bad Input (Bad Input used instead of Invalid because Invalid and Valid look too similar to be immediately distinct from each other). Input for methods with data rows shall be specified as input, num, or some other general name specifying what type of input is coming into the test method. An example of this may be the following. Example – A unit test that tests CarCompator’s compare method may be named as such. (in the file CarComparatorTests)
“Compare_CompareTwoIdenticalCars_Returns0”
The Unit test of course would do just this, pass two identical cars into the comparator compare Method and verify that they in fact return the expected result.

5.3 Code

The actual code of the unit test should adhere to the coding standards we have set for all of our other source code, this rule is subject to change in the future.

6 Git Conventions

The Main project will have its own dedicated repository and the current updated project will be located on the master branch. Whenever one or more persons decides to work on a given feature, they will create a branch from master and begin work on said feature, when said feature is done, they will update their current branch with any changes from master that may have occured, and then submit a pull request (PR) back into master. Any changes made to master must be approved by at least 2 out of the 3 team members, and will always be done using a PR.. In short, any changes to the master branch must be submitted via aPR, and one team member other than the PR submitter must approve said PR before the changes are added to master.

7 Code Style Conventions

7.1 Conditional

Conditional statements and any loops will always follow the indentation rules as described above, even when code braces are not necessary.
EXAMPLE

```
if(a < 10)
        a++;
```

∧∧THIS IS WRONG

```
if(a < 10 ){
        a++;
}
```