**Shockwave 4488 Java Coding Conventions**

# **Eclipse Settings**

· Use the default Eclipse curly bracket structure. This means Egyptian Style braces, which braces on the same line as their declaration.

# **Project, File, and Folder Naming**

· Main robot project should be named FRCxxxx, where xxxx is the current competition season

· Main robot class should be called FRCxxxx.

· Note most of the ones above will be correct if you use the extension to create the project.

· Files should be {nameofclass,interface,orstruct}.java.

· Nested folders are allowed as long as their name does not have any spaces in it. Any files contained in a nested folder will use the package {root}.{foldername}

# **Class, Interface, and Enum Naming**

· Class, and enum names should be in PascalCase, with the name matching the file name. Only one class per file. Files can contain multiple enums or static classes that pertain to the class.

· If multiple classes use the same static class or enum, split those into their own files. File can include multiple static classes or enums, and the file name should be suffixed with Enum or Struct (We will call static classes structs to match C#).

· Interfaces should be named in PascalCase,

· Abstract classes should be suffixed with “Base”.

# **Method, Field, and Property Naming and Visibility**

· Methods should be in camelCase, and can have any accessibility.

· No fields should be public or internal. Protected and private fields are ok. If they need to be accessed outside the class, make getters and setters to access the field instead.

· Instance fields should be in camelCase, with an “m\_” prefix.

· Static fields should be in camelCase, with an “s\_” prefix.

· Constants should be in kPascalCase.

· Parameter names should be in camelCase.

# **Class Ordering**

· Files should start with any required Enums, then any required Static Classes, and then the main class or interface.

· Members should be organized in the following way

o Fields

o Properties

o Constructors

o Methods

# **Singleton Pattern**

· If a class should only be instantiated once, but can be used from multiple classes, please use the Singleton pattern.

· The singleton pattern is a private variable of the class type with the name s\_instance. Then there is a public static getter called getInstance, with the following signature, with DriverStation being replaced by the class name.

o public static DriverStation getInstance { s\_instance ?? (s\_instance = new DriverStation()); }

· After this, the constructor is made private. This will make sure only 1 instance can created.

· You can now access the class from any other class.

· In the robotInit() method in the main robot class, all Instance properties should be grabbed in order to initialize all the systems.