General Rules

-Duplicate code should be removed/reduced when refactoring.

-Use descriptive, unique, and easily-understood names.

-Long chunks of code should have comments that explain what is happening.

-Of course, no errors or compiler warnings should be displayed in any pushed or published code.

-Each method (including helpers) within a .cpp file should have a comment that generalizes what that method does.

Good Code is:

-Easily understood by any reader of the code

-Simple, but concise and to the point.

-Consistent

-Easy to navigate through

Comments

Comments should be used whenever a large portion of code that may be seen as complex needs to be explained. Methods should always have a descriptive comment above telling what that method does (can be one line). For self-explanatory code sections do not need comments (with the exception of the aforementioned method comments).

Files

-Header files should take the same name as their respective .cpp files. The latter should also use camel case naming (Ex: thisIsASampleFile.cpp).

-Avoid nested classes as much as possible (this makes it difficult to read)

-When refactoring, make sure to put most of the "#include" libraries and classes in the header file and not the .cpp file.

Format Standards

-Accessing methods should have a get or set in front of them.

-Use camel case for names of methods and variables.

-Indenting is vital to make code sections legible. Anytime curly brackets are used, everything within them should be indented (code within a method should be indented from the method signature and code within a for, while, or if section need to be indented from those aforementioned lines). Also remember to indent the last curly bracket to align with the signature or starting line of a code block (this allows for easy understanding as to when a method, if, for, or while block ends).

-Use an empty line to separate methods or important parts of code, but use these empty lines sparingly.

Naming

-Needs to use camel case and should avoid using any non-alphabetic character as much as possible.

-Names need to be descriptive, unique, and easy enough to understand that their purpose and meaning could be interpreted by name alone.

-Almost all, if not all, variables need to be private or protected (use accessor methods if needed)