Include Files and Include Statements

The following is an amalgam of stack overflow and IBM research pages.

Header files must contain an include guard

```
#ifndef COM_COMPANY_MODULE_CLASSNAME_H
#define COM_COMPANY_MODULE_CLASSNAME_H
  :
#endif // COM_COMPANY_MODULE_CLASSNAME_H
```

The construction is to avoid compilation errors. The name convention resembles the location of the file inside the source tree and prevents naming conflicts.

Include statements should be sorted and grouped.

Sorted by their hierarchical position in the system with low level files included first. Leave an empty line between groups of include statements for ease of reading

```
#include <fstream>
#include <iomanip>

#include <qt/qbutton.h>
#include <qt/qtextfield.h>

#include "com/company/ui/PropertiesDialog.h"
#include "com/company/ui/MainWindow.h"
```

In addition to show the reader the individual include files, it also give an immediate clue about the modules that are involved.

Include statements should be located at the top of a file only.

Each source file should include All the header files it needs. Don't include headers through inclusion in other header files (If that file is removed you will get compilation errors).

Never include .cpp or source files

Do not declare non const variables in header files

non const variables are external *linkage means the symbol (function or global variable) is accessible throughout your program*

external linkage - Declare them in either cpp file or pass by reference in function calls. If you want them global to the project use extern. This is an advanced technique and should only be used for a valid reason (singleton for instance). You probably don't have a valid reason in this class.

If you don't use extern, When you declare a variable in a header file *every* source file that includes that header, either directly or indirectly, gets its own *separate copy* of the variable when compiled. Then when you go to link all the .o files together, the linker sees that the variable is instantiated in a bunch of .o files. You get a Multiple definition error.

What about const variables?

const variables are internal *linkage means that it's only accessible in one translation unit.*

Treated differently by linker, every single .o file still gets a copy, but they are considered internal to to the file. The linker allows it. The tradeoff, you use more memory. Each compilation unit will set aside the space for the same variable. Inefficient use of space but otherwise no problem since the variable is const and cannot change. You can also use extern here by declaring the variables as extern in the header file and then defining them in the cpp file like so;
extern const varname; in the .h file
const varname =value; in the cpp file

If you have circular dependencies

Use Forward Declarations.