Studio BlueBox Coding Standards

Naming Conventions

- Functions
  - Lower camel case - start with lower case letter and capitalize the first letter of each word after

    e.g. `functionName();`

  - Functions should have a meaningful name; they should be named relative to what they do
  - Use `getVar()` and `setVar()` for class accessors
- Variables
  - Lower camel case - start with lower case letter and capitalize the first letter of each word after
  - Variables should have meaningful names

    e.g. `int characterLives;` or `string characterName;`

  - i, j, k will be the variables used for for loops (declare inside for loop)
- Classes
  - Lower camel case - start with lower case letter and capitalize the first letter of each word after
  - Classes and class objects should have meaningful names

    e.g
    ```
    class sampleClass{
        Public int characterCoinCount;
    }
    ```
- Files
  - Upper Camel Case – the first letter of every word in the file name should be capitalized
  - Name of file followed by single hyphen and initials if it belongs to just you, or group if it is worked on by everyone

    e.g. `FileName-IH.cs` or `FileName-Group.cs`

Function Layout

- Opening curly brace is on the same line as the function or class definition
- Always one line (at most) of whitespace between "sections" of the function
  - i.e., if you separate variable declarations, loops, if statements, etc.
- One space between a variable initialization and value (yellow highlight)
- Nothing on the same level as the function definition
  - Continue tabbing (4 spaces) for each level of scope (blue highlight)

    e.g.
    ```
    void functionName(){
        Int characterLives = 0;
    ```

```
    String characterName = "Smith";


    If(characterLives == 0){
        // do something...
     }
}
```

Indentation

- 1 tab for every level of scope (4 spaces, need to change this in visual studio code or preferred platform)

  e.g. // pretend we are in the function:

```
for loop(){
   for loop(){
       if(){
       }
   }
}
```

Comments

Multi-line comments:

- Place a multi-line comment to describe what the function does before the function definition
- Multi-line comments are to be formatted like this:

  e.g.

```
/*
This function does this...
*/
Void functionName(){
}
```

- If it is a group file, put your initials in the comment sections of code that are yours
- Each program file should have a header comment that contains the file name and a short summary of what the program file does

e.g.
```
/*****************************
    fileName.cs
     This file does this …
    *****************************/
```

In-line comments:

- If in-line comments are necessary, place them 1 space after the end of line

e.g.
```
void functionName(){
        printf("Hello world!"); // this line prints words
}
```

- If the line is too long, the comment can be placed above the line

e.g.
```
void functionName(){
    // the line below this is too long
    printf("This line will print out a lot of things");
}
```

Error Handling

- Use C# exception handling (try, catch, finally)

e.g
```
try{
    // statements causing exception
}
catch(ExceptionName e_1){
    // error handling code
}
catch(ExceptionName e_2){
    // error handling code
}
catch(ExceptionName e_n){
   // error handling code
}
finally{
   // statements to be executed
}
```