Introduction to Object Oriented Programming

There are three separate concepts that are being used in this week's programming assignment: creation of classes, partitioning a project into multiple files, and creating a project in Code::Blocks.

Classes and Object

This material is presented in the textbook, but just as a brief summary.

The idea of using object oriented programming is to simplify the organization of a program by encapsulating the data and associating it with methods or functions that can be used to manipulate it.

You can consider a class as a blueprint for how an object should be built, similar to a design for how to build a car or a house. Once you have the design, you can create multiple objects from it, just as you would create multiple cars from the same design. Some of the cars could be white or red or have leather interiors or different options. Similarly, each object has its own internal values for the variables that are defined in the class. The methods are the same though.

As an example, consider the following class (this is taken from the rectangle class which can be found in Moodle under Resources/Examples: simple classes/Rectangle.cpp).

```
class Rectangle
{
    private:
        int length;
        int width;

    public:
        int getLength() {return length;}
        int getWidth() {return width;}
        void setLength(int l) {length = l;}
        void setWidth(int w) {width = w;}
        int calcArea();
};


int Rectangle::calcArea()
{
    return length * width;
}
```

The name of the class is Rectangle. It has private variables for the length and width. These store the unique values that each rectangle will have for these variables. When you create a new object of this type, it will space in memory to store its copies of these variables. Since they are private, only the object can access them, they are not available to any external function or program.
The class has several public methods. These are the ways that the objects are accessed or used.

- There is no constructor (method with the same name as the class that is used to create it), so the compiler will create a default one that will allocate space for the two variables, but not initialize them.
- There are two accessor or getter methods: getLengh() and getWidth(). These each return the current value of the appropriate object variable.

- There are two mutator or setter methods: setLength(int l) and setWidth(int w). These each update or set the value of the appropriate object variable.
- There is another method: calcArea(). This is declared in the class definition, but the code is external to the class. Note that it is necessary to specify the class name as part of the method definition. If you tried to define a method with just the name, `int calcArea()`, the compiler would not know that it was to be associated with the class Rectangle.

This class is used in the associated main program:

```
int main()
{
    Rectangle myRectangle;
    myRectangle.setLength(10);
    myRectangle.setWidth(5);
    cout << myRectangle.calcArea() << endl;

    Rectangle rec;
    rec.setLength(15);
    rec.setWidth(2);
    cout << rec.calcArea() << endl;
    cout << myRectangle.calcArea() << endl;

    return 0;
}
```

An instance of the class, or an object of type rectangle, is created and named `myRectangle`. Methods from this object are then used to set its length and its width and the method `calcArea()` is used to calculated the resulting area.

A second rectangle is then created with the name `rec` and its length and width are set and its area computed.

Note that creating the second rectangle and modifying its values has no effect on the first rectangle. This is exactly the same as if you had the code:

```
int x;
x = 5;
int y;
y = 6;
```

In this case, creating the second integer variable `y` and setting its value does not affect the first variable `x`.

So, in summary, a class is a definition of a way to associate some collection of variables or data with a set of methods or functions. When you create an instance of the class, you are creating an object. This object will have a name and a location in memory. You can then use its methods to access its unique values.

Breaking a program into separate files

In programming with C++, we normally break a class into two parts. One is called a header file and contains the class declaration. The other contains the implementations or definitions of the methods for the class.

If you look at the zip file RectangleProject in the same folder Resources/Examples of simple classes, you will find a Code::Blocks project. This has three C++ files and a Code::Blocks definition file. The file Rectangle.h is the header file, Rectangle.cpp contains the implementation of the methods for this class, and main.cpp is the calling program that uses the class. Lets look at each one separately.

Rectangle.h

```
    #ifndef RECTANGLE_H_INCLUDED
    #define RECTANGLE_H_INCLUDED

    // This class models a rectangle
    class Rectangle
    {
        private:
        int length;
        int width;

        public:
        // Constructors
        Rectangle() {length = 0; width = 0;}
        Rectangle(int l, int w) {length = l; width = w;}
        Rectangle(int s) {length = s; width = s;}

        // Accessors and mutators
        int getLength() {return length;}
        int getWidth() {return width;}
        void setLength(int l) {length = l;}
        void setWidth(int w) {width = w;}

        // Processing
        int calcArea();
        bool isSquare();
    };

    #endif // RECTANGLE_H_INCLUDED
```

The first two lines and the last line:

```
      #ifndef RECTANGLE_H_INCLUDED
      #define RECTANGLE_H_INCLUDED

      #ENDIF // RECTANGLE_H_INCLUDED
```

are used to make sure that only one copy of the header file will be read by the compiler. If you happened to do #include “Rectangle.h” more than one time, the first time the compiler would see that the symbol RECTANGLE_H_INCLUDED was not defined and would include all of the code in the file. The next time, the symbol would already be defined and the compiler would skip down to the #ENDIF and not include any of the intermediate code.

Notice that this version of the class has three constructors. The first one takes no arguments and initializes the width and length both to zero. This is the equivalent of the default constructor that we had in the first example. The second constructor takes two arguments and sets the width and length from them. The third constructor is for creating a square and has a single argument that it uses to set both the width and length so that they are equal.

We have the same accessors and mutators as before and they do the exact same thing – setting and retrieving the values for the class variables.

There is also the method `calcArea()` as before. In addition, there is now a new method `isSquare()` that returns true if the rectangle is a square and false if it is not. Notice that no code is present in this file that defines how these methods work.

Rectangle.cpp

```
    #include "Rectangle.h"

    // Reuturn the area of the recatangle
    int Rectangle::calcArea()
    {
        return length * width;
    }

    // Return true only if the rectangle is a square
    bool Rectangle::isSquare()
    {
        return length == width;
    }
```

The next file we will consider is Rectangle.cpp. This is the file that contains the C++ code that defines how the two methods `calcArea()` and `isSquare()` will be implemented.

First, note that it is necessary to include the file "Rectangle.h" as part of this file. This is so that the compiler will have the class declaration to refer to when compiling this file.

Next, you will see that the names of the methods include the class name. This is identical to the earlier example when it was all in one file and is for the same purpose. If we had multiple functions with the name `calcArea()`, this would identify for the compiler that this one is to be associated with the class Rectangle.

The calcArea() method returns the width times the length, exactly as we would expect. The isSquare() method returns the result of the boolean expression length == width. If they are equal, then it will return true, otherwise it will return false.

main.cpp

```
    #include <iostream>
    #include "Rectangle.h"
    using namespace std;

    // Author: Brian Bird, 4/6/12
    // This program uses a class that models a rectangle as
    // a demonstartion of how to declare and use a simple class.
    // It also is an example of a multi-file project in Code::Blocks.
```

```
int main()
{
    // This main consists only of test drivers

    Rectangle myRectangle;
    cout << "Default constructor test. Expect length and width = 0" << endl;
    cout << "Length: " << myRectangle.getLength() << endl;
    cout << "Width: " << myRectangle.getWidth() << endl;

    cout << "\nOne parameter overloaded constructor test. Expect length = 10,
width = 10" << endl;
    Rectangle square(10);
    cout << "Length: " << square.getLength() << endl;
    cout << "Width: " << square.getWidth() << endl;

    cout << "\nTwo parameter overloaded constructor test. Expect length = 2,
width = 15" << endl;
    Rectangle rec(2,15);
    cout << "Length: " << rec.getLength() << endl;
    cout << "Width: " << rec.getWidth() << endl;

    cout << "\nSetter and getter tests. Expect length = 3, width = 5" << endl;
    myRectangle.setLength(3);
    myRectangle.setWidth(5);
    cout << "Length: " << myRectangle.getLength() << endl;
    cout << "Width: " << myRectangle.getWidth() << endl;

    cout << "\ncalcArea test. Expect 15" << endl;
    cout << "Area: " << myRectangle.calcArea() << endl;

    cout << "\nisSquare test with sides equal. Expect 1" << endl;
    cout << "isSquare: " << square.isSquare() << endl;
    cout << "\nisSquare test with sides not equal. Expect 0" << endl;
    cout << "isSquare: " << rec.isSquare() << endl;

    return 0;
}
```

Finally we have the main program that uses the class. Note that we have the instruction
#include "Rectangle.h"
This is so that the program has access to the class declaration and knows what methods the class will implement and what parameters and return values they have.

This program first creates a rectangle myRectangle using the default constructor and sets its width and length with the setter methods. It then creates a second rectangle using the single parameter constructor and calls it square. Finally, it creates a third triangle using the two parameter constructor and names it rec.

You can load and test this project by opening it in Code::Blocks or by double clicking on the file Rectangle.cbp.

Creating a project in Code::Blocks

You can look at how the project Rectangle appears when opened in Code::Blocks. Notice that there are two separate categories of files – source and header. The two files ending in .cpp are in the source category and the one file ending in .h is in the header category.

Here is a link showing how you can create a project on your own:

http://wiki.codeblocks.org/index.php?title=Creating_a_new_project