The title is Objects in Class, that is not a typo, we will get to that. For most of my lessons there will be a bunch of code at the beginning and a lecture at the end, for this one because of the subject we will have to reverse course and lecture first. That is because our subject is bordering on the esoteric, Object orientation is more of a philosophy and mindset than it is any particular set of programming rules. So I am gonna show slides with a bunch of sudo-code, this is not real code that will work in the computer, it is just there to demonstrate an idea, when I start coding I will let you know. You can write this stuff in a notebook or text file if you want, just dont expect it to run until I start writing real code.

So the lecture begins.

We often when programming think much more about function than we do about the actual structure of our code or the data we are processing.

**doStuff(Data);**

For me, when I am programming an operation all I care about is doing it once, because I only have to program whatever operation once, right the computer will run it over and over again. That approach is fine for simple programs that run on the command line and very simple games. Once however you begin to manage memory and work with structures in the computer memory, things may move past your ability to manage them with a list of global variables.

**Int x, y, z;**

**Float f, g, h;**

**Bool v, i, d;**

**//imagine remembering what the value of all these functions during operation**

**//debugging nightmare**

**Etc…**

The solution is to create objects in the computer memory with functional and data members, then use them with the member access operators.

**class object{**

**Data;**

**doStuffWithData();**

**};**

Comparing an object to a function seems very esoteric at first, but once you see the paradigm and understand it, you will not be able to un-see it. It actually helps to learn about objects and containers at the same time, and I will explain with an easy example.

To make a real world example, you might try to call a **toolbox** an object, and you might be right, you could say that **wrenching, screwdriving** and **hammering** are all functional members of the object **toolbox**, this works as a mental exercise, but kinda breaks down when you have say, an adjustable wrench. Now should you make a functional member called wrench up, adjust wrench down, close and open wrench, and stick all these functions next to each other in a huge long list. What about the hammer, do we have a list hammerPullNails(), hammerHammerNail(), etc… now we are right where we were with the long list of global variables.

**Toolbox** is an object, I can pick **toolbox** up and I can throw **toolbox** around, and if **toolbox** is well made it won't break into its constituent parts and the objects inside wont come out. Take note of the last idea, **toolbox** has constituent parts, and objects inside it, that reveals our mistake. **Toolbox** isn’t just any old object, it is a container. It is an object that contains other objects, so those objects **hammer**, **wrench**, **screwdriver**, should have the functions, **hammerNail(), openWrench(),** and **tightenScrew(),** respectively. **Toolbox** should have functional members that allow it to retrieve the **wrench**, **hammer** and **screwdriver**. You might write one for each tool, like **Toolbox->getWrench(),** or if you only had three or so you might just say **Toolbox->getTool(x),** and pass the number of the tool you want as a parameter. I have seen both ideas implemented (in a more refined mannar). In this way we gain better use of our ‘tools’(our functional members), and the materials(our data).

Screwdrivers and hammers are well understood by carpenters and mechanics, but what about how this applies to computers? As an example I will use a videogame.

So we have our data, all the operations we need for our data, and we call that stuff together an object. Imagine you have a goblin in a computer game, you could store in that goblin, his health, his location, and a boolean to say he is sleeping or awake, then have a function called move, attack, die.

**class goblin{**

**//data members**

**Int health; // heathbar**

**float x,y,z, w; // cartesian coordinates for location**

**bool sleeping; // true or false, goblin is sleeping**

**//functional members**

**bool isAsleep(); // tells us if is sleeping is true or false**

**float\* move(float\* ); // moves goblin, returns new location**

**void die(); // if this runs, the goblin ‘dies’ (deletes itself)**

**void attack(); // this is how the goblin attacks**

**};**

If your goblin is sleeping he can’t attack or move, but he can be killed by the player, so you might program that by saying something like:

**if (goblin->isAsleep()){**

**//do not move or attack**

**}**

**Else{**

**//you can move and attack**

**}**

Now this is a bad example of code in any game, but it is good at demonstrating what we are talking about. Nobody would take the time to code this like this, but you as the beginner can see what we are doing here.

That **->** arrow symbol in between goblin and **isAsleep()** is called the member access variable. It does what it sounds like, it allows you to access members of an object.

So **isAsleep()** is returning a true or false, it looks like this.

**bool goblin::isAsleep(){ return sleeping; }**

The data member sleeping is where we store our true and false, **isAsleep()** is just returning a copy of that state.

This of course is not real code, all of this is just an example of how we use objects in c++, how the idea of Object Orientation becomes code in C++;

Remember always that c++ uses c code, when we do math like abs() and pow() we are including the cmath header, this is literally the math.h header from c.

What really makes c++ is the Class and the use of templates. Templates are another day but class is the keyword we use to make objects. The reason why is on page 11 and 12 of the Introduction to Mathematical Philosophy by bertrand russel:

*“Many philosophers, when attempting to define number, are really setting to work to define plurality, which is quite a different thing. Number is what is characteristic of numbers, as man is what is characteristic of men. A plurality is not an instance of number, but of some particular number. A trio of men, for example, is an instance of the number 3, and the number 3 is an instance of number; but the trio is not an instance of number. This point may seem elementary and scarcely worth mentioning; yet it has proved too subtle for the philosophers, with few exceptions. A particular number is not identical with any collection of terms having that number: the number 3 is not identical with the trio consisting of Brown, Jones, and Robinson. The number 3 is something which all trios have in common, and which distinguishes them from other collections. A number is something that characterizes certain collections, namely, those that have that number. Instead of speaking of a “collection,” we shall as a rule speak of a “class,” or sometimes a “set.” Other words used in mathematics for the same thing are “aggregate” and “manifold.” We shall have much to say later on about classes.”*

(leave this on-screen for a minute or two)

So you can pause and read that if you want, but the important part is that last, how he is associating the word class with number. Remember two things, one that there were no digital machines in Bertrand’s day, and two, there is no way to store anything in the digital computer that is not numeric. We must save our data in some way to store and work with it, the only way we have that works with the hardware is in binary, the binary representation can always be understood as a number. So in math depending on how far you got, at some point you start to deal with compound numbers. These are numeric values, but they are complex and they have more than one part, so in the computer you can have modular binary numbers that store multiple parts in the same way. However the computer is not like you and I, we can infer things, the computer cannot. The computer cannot understand that you just gave it a number that has multiple parts that mean different things, instead of a regular old integer.

Thus when you create a new type, a new class of number, you have to tell the computer how to work with that number, that object. Thus Object Oriented Programming. And you can see from that philosophy blurb, why we use the word class to create objects, because you are creating a new class of number.

At first this seems hard and scary, and it is. But once you get your feet underneath you in this aspect the whole world of programming will open up ahead of you. You can think of the last few lessons as learning to walk and run, today we are learning to fly… So sorry about the long lecture, flying is harder than running. But now to the code let's work with the ideas we were talking about.

Please pay attention to this setup process if you are a beginner. To begin, open up your VS and create a new C++ program with nothing in it, and call it shape project. I like to leave this box here (project and solution in the same folder) this checked for simple projects. Now we are in our project, open up the solution explorer and click on the name of the project if you need to, notice all these folders and what they are named, now we won't use all of these but we will use some and so you might want to get used to how they look. Now source code is where you do our executable code, this is the real source of our operations, this is where main is, and this is where we actually tell the computer what we want it to do. So first I am going to right click the project, not the folders here but the project itself, and then say add, then new item, now make a .cpp file and down at the bottom of this popup screen name it main. Notice that it is in the source folder.

As we remember main is where things are done, and there are some built in objects we can work with here but that won’t do much for you so we are going to go over here and create an object. So right click our project again and add again, but this time choose this other option class. In the pop-up box we will call this object ‘Shape with an uppercase s. DO NOT check this box here that says define inline, if it is checked uncheck it. And click ok. Notice now that VS made us two files, one in source and another in the header folder. This is because headers are for declaration and source is for definition. So open up shape.h and shape.cpp. These files together make a class and the first one shape.h is where we tell the computer what this class is, so shape won't really do much for us, but we do want it to exist.

So we will say:

**//this will tell the computer that there is a class called shape with a constructor**

**class Shape**

**{  
 Shape();**

**}**

Now the compiler knows what we are about to do, but it lacks the exact commands, it has an idea that we want this object shape and we want to construct it, but it has no idea what exactly to capital D doo. We give it that in our .cpp file,

So in shape.cpp we say:

**//this is the definition of the constructor**

**Shape::Shape(){}**

Notice it does nothing, but now in main, we can say

**Shape shape;**

That may not seem like much but we just created an object of class shape, called shape, we can also say:

**Shape shape1;**

**Shape shape3;**

**Shape oblong;**

And so on and so forth. And we are making imaginary shapes in the computer memory.

What about a more real one?

Let's make a square.

Now right click your project, add a new class, and name it capital S Square.

Now in square.h type this.

**class Square**

**{**

**public:**

**//default constructor**

**Square();**

**//constructor and functional members**

**Square(double);**

**double area();**

**double perimeter();**

**//data members**

**double b;**

**};**

Double is of course a fancy way of saying decimal number, so we can have a square that takes a decimal. Take note also of the public: flag, that will become important in a moment and I will explain it then. And now we will define these in our Square.cpp file.

**#include "square.h"**

**//default constructor**

**Square::Square(){};**

**//parameter constructor**

**Square::Square(double x){ b = x; };**

**//return area of square**

**double Square::area(){ return b \* b ; }**

**//return Perimeter of square**

**double Square::perimeter(){ return b \* 4 ; }**

Now in main we can delete all these shapes and we can call:

**Square square(4);**

Now we have a shape we can use for math, a shape we can show on the console.

**std::cout square->area() <<std::endl;**

And I want to just be clear here I could have a Square object named blahblahblah, like it could literally be anything that is allowed as a name in C++:

Square blahBlahBlah444(4);

std::cout blahBlahBlah444->area() << std::endl;

We get the same result because blahBlahBlah444 is an object in the same class and is doing all the same operations as the little s square we made right here.

So now we have made an object and done operations on it, but what about if we had an Object that was a little like square but it was longer, like a rectangle. Let's make a new class, again, add, new class, and name this one Rectangle.

So in Rectangle.h code this:

**#include "Square.h"**

**Class Rectangle: public Square**

**{**

**public:**

**Rectangle(){};**

**Rectangle( double x, double y);**

**double h;**

**double area();**

**double perimeter();**

**};**

Notice first that the **#include “square.h”** and the **public** tag, as well that we only have one data member. The **Public** tag is going to allow us to access all the members declared under it in **Square.h**. That is why it is important, and what it is doing, I will do like a 20 minute mini lesson on **Public Private** and **Protected** sometime but for now just understand that by default all members in a **class** are **private**, nobody but that **class** can access it, and there are reasons for that, and you can change the state of the member by putting it under one of the tags, **private, public** or **protected**. Under a **public** tag anybody can access the member, we want to be able to call these functions in **main** so we want them to be **public**. Now return your attention to the **h** variable here, remember in Square.h we

So now that we understand all that, in our rectangle.cpp file we will code this.

**#include "Rectangle.h"**

**//default constructor**

**Rectangle::Rectangle(){};**

**//para construct**

**Rectangle::Rectangle(double x, double y){**

**b=x;**

**h=y**

**}**

**//area**

**double Rectangle::area(){**

**return (b \* h);**

**}**

**//perimeter**

**double Rectangle::perimeter(){**

**return ((2 \* b)+(2 \* h));**

**}**

Now in main you can create rectangles right.

**Rectangle rect(4, 5);**

**std::cout << rect.area() << std::endl;**

So that could be the end of our code, but there is something else we could do, for education's sake.

So when you look at Rectangle.cpp and Square.cpp you begin to notice that the functions really are very similar. If you did some thinking you might even come to the idea that we dont really need Square as an object if we just had a rectangle that only took one parameter and we declared the base. So lets do that.

(in rectangle.h)

**Rectangle(double);**

**double b;**

(delete)

**#include “Square”**

**:public Square**

(in rectangle.cpp)

**Rectangle::Rectangle(double x){ b=x; h=x;}**

Now in main we replace every spot where we have a Square with Rectangle, compile and run, and we have our output.

Now we could literally if we want go over here and delete these files, I am not going to but that is just a little example of how we can use reason to use less code, and that is the real high level in Programming of any kind, is to write small amounts of code that do lots of stuff. Right because if you notice before I said, what if we had a square that was longer, well really you can think of a square as just a rectangle where the base and the height are the same. By changing your understanding of the objects you can write better code. Programming is a mix of Math and Philosophy, and I love it personally, but my point is that by having an understanding of both basic operations in math and how to work with ideas in philosophy you can go a long way in programming.

Anyways Homework.

For your homework I want you to create a class called Circle, that is a circle, and make sure you can return the area and parameter.

That is all have a good day or night.