**Arduino Intermediate and Advanced Programming**

Oct 7, 2015

At last weeks meeting, Frank brought up the fact that his BittyBot didn’t seem to go forward, or backward at the same rate of speed or time, and that the timing seemed off.

part of the reason for this is in the sketch he was talking about, DELAY was used for timings.

And I always knew DELAY is not what I wanted to use, but it’s quick, and good for testing.

what DELAY does is stop the microcontroller, it will not advance to the next step in the program until the DELAY is meet.

Frank’s questioning the timing last week was a good reason to go back and rewrite the DEMO code using a different method for timing. Once before we did a demo for Frank that had different timings and a button push for a LED. In that we used MILLIS() and watched the clock our.

The idea is to watch the clock ourself this lets the sketch advance and do more in the same amount of time.

So let’s look at something simple (BLINK):

int LED = 13;

int State = 0;

void setup() {

pinMode(LED, OUTPUT);

digitalWrite(LED, State);

}

void loop() {

State = !State;

digitalWrite(LED, State);

delay(1000);

}

Ok, pretty simple, but each time a DELAY is reached the sketch stops and waits, nothing else happens. - Let’s look at the same sketch but without a DELAY, using MILLIS()

int LED = 13;

int State = 0;

long previousMillis = 0; // this will store the last time the LED was updated

long interval = 1000; //this is the DELAY in milliseconds (just like using DELAY)

void setup() {

pinMode(LED, OUTPUT);

digitalWrite(LED, State);

}

void loop() {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis > interval) {

previousMillis = currentMillis;

State = !State;

digitalWrite(LED, State);

}

}

What is going on here for each loop of the sketch it checks how long the microcontroller has been on, minus the last time the LED was changed, it says that it needs to be greater than the DELAY time (1000) - changes the State of the LED (toggle), and updates the previousMillis.

It looks like a harder way to just blink a LED, but what it’s really shows is that if it hasn’t been at least the INTERVAL time, I can move on and do more code.

So if you wanted to also check a ultrasonic, or move a servo, or check the battery level, you could do all this while still blinking a LED.

Why do we need to know this well because it leads to a more advanced concept called Object Oriented programming. In the above example, everything is still done inside the loop function. With OOP you set things called CLASSes, DATA, METHODs, and OBJECTs.

A Class is a group of Methods, a Method is a group of code that relates to DATA in the Class. A Object is the “friendly” name for a class (at least that is how I understand it)

We honestly uses these all the time, but most of us don’t know it. Libraries use Classes, this is something we don’t normally see, but we use.

The example I will use is SoftwareSerial (<https://www.arduino.cc/en/Tutorial/SoftwareSerialExample>)

when you invoke a instance of SoftwareSerial you are making a object that object talks to the SoftwareSerial CLASS.

IE: SoftwareSerial mySerial(RX,TX);

mySerial is the “friendly” name we are using, it can be anything we want - serial2, camera, or whatever.

SoftwareSerial camera(RX,TX); - in some cases just a general object like mySerial is good, and in others it’s nice to know which/or what you are talking to camera.

Generally you’ll see a comment telling what the object is talking to, so someone can follow what you are doing in your sketch. Sometimes it’s clear and others not so much.

We can create more than one SoftwareSerial object - IE:

SoftwareSerial Serial1(RX1, TX1);

SoftwareSerial Serial2(RX2, TX2);

In this case each object has it’s own set of DATA it can use, IE: it could be 2 different baud rates, or whatever. And the two instances shouldn’t affect the other.

With the SoftwareSerial library, we have a few different methods.

Serial1.read();

Serial1.begin(4800);

Serial1.print();

Serial1.println();

etc….. Each method in the library is a function that interacts with the SoftwareSerial Class.

What is inside the () for each method call could contain DATA or not.

The library also uses what is called PRIVATE DATA, this is really more advanced than we need to know right now, but for the most part this is DATA that only the class needs to know, Settings, timing checks, etc. As a rule we don’t have access to this DATA and only the CLASS can change it.

A library is more advanced that most of us program, someone else has already figured out what needs to be done to make sensors work, or just what timings need changed to add additional PWM or change the PWM, or whatever. They are nice, because it makes life easy, we want to use a I2C device, we include the Wire library, we want to send information over a 433Mhz link, we use the VirtualWire Library. We don’t need to know anything else.

So that is the very basic overview of a library. Why is it important, well, as I said, we use Object oriented programming now, we just don’t know it.

We can make our own Libraries -

<https://www.arduino.cc/en/Hacking/LibraryTutorial>

<http://playground.arduino.cc/Code/Library>

And what isn’t clear is we can make our own CLASSes, They work very much like how we use libraries now, the biggest different is they are contained in the .INO file.

At one time I read they needed to be at the top of the file, before you declare your variables and include other libraries, I can’t find that now, but I think it’s true.

So, I’m not going to go into too much about some of the Intermediate and Advanced Programming features for the Arduino. There are literally hundreds of good tutorials on line that can explain some of this much better than I can. But here are some basics Arduino is programmed in C++, C++ is a object orientated programming language.

So what does that mean really?

From Wikipedia:

**Object-oriented programming** (**OOP**) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_%28computer_science%29)", which are [data structures](https://en.wikipedia.org/wiki/Data_structure) that contain [data](https://en.wikipedia.org/wiki/Data), in the form of [fields](https://en.wikipedia.org/wiki/Field_%28computer_science%29), often known as *attributes;* and code, in the form of procedures, often known as [*methods*](https://en.wikipedia.org/wiki/Method_%28computer_science%29)*.* A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "[this](https://en.wikipedia.org/wiki/This_%28computer_programming%29)" or "self"). In OO programming, computer programs are designed by making them out of objects that interact with one another.[[1]](https://en.wikipedia.org/wiki/Object-oriented_programming#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Object-oriented_programming#cite_note-2) There is significant diversity in object-oriented programming, but most popular languages are [class-based](https://en.wikipedia.org/wiki/Class-based_programming), meaning that objects are [instances](https://en.wikipedia.org/wiki/Instance_%28computer_science%29) of [classes](https://en.wikipedia.org/wiki/Class_%28computer_science%29), which typically also determines their [type](https://en.wikipedia.org/wiki/Data_type).

Many of the most widely used programming languages are [multi-paradigm programming languages](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language) that support object-oriented programming to a greater or lesser degree, typically in combination with [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming). Significant object-oriented languages include [Python](https://en.wikipedia.org/wiki/Python_%28programming_language%29), [C++](https://en.wikipedia.org/wiki/C%2B%2B), [Objective-C](https://en.wikipedia.org/wiki/Objective-C), [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk), [Delphi](https://en.wikipedia.org/wiki/Delphi_%28programming_language%29), [Java](https://en.wikipedia.org/wiki/Java_%28programming_language%29), [Swift](https://en.wikipedia.org/wiki/Swift_%28programming_language%29), [C#](https://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29), [Perl](https://en.wikipedia.org/wiki/Perl), [Ruby](https://en.wikipedia.org/wiki/Ruby_%28programming_language%29) and [PHP](https://en.wikipedia.org/wiki/PHP).

<https://en.wikipedia.org/wiki/Object-oriented_programming>

Well that clears it up doesn’t it -

What it really means is you have a group of things that are all related to each other. (DATA)

It also has something called “Members” this is what to do with the DATA.

When both of these things are together they go into a CLASS.

So if we look something like the BLINK without DELAY sketch again:

int LED = 13;

int State = 0;

long previousMillis = 0; // this will store the last time the LED was updated

long interval = 1000; //this is the DELAY in milliseconds (just like using DELAY)

void setup() {

pinMode(LED, OUTPUT);

digitalWrite(LED, State);

}

void loop() {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis > interval) {

previousMillis = currentMillis;

State = !State;

digitalWrite(LED, State);

}

}

**We can make this a CLASS:**

Class Blink {

int LED;

long interval;

int State;

unsigned long previousMillis;

//This is the constructor - and it initializes the members and variables and states

public:

Blink(int pin, long on) {

LED = pin;

pinMode(LED, OUTPUT);

digitalWrite(LED, LOW);

interval = on;

State = 0;

previousMillis = 0;

}

void Update() {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis > onTime) {

previousMillis = currentMillis;

State = !State;

digitalWrite(LED, State);

}

};

Blink led1(12, 1000); //Create a object called led1 attached to Pin 12, with a delay of 1000

Blink led2(13, 1200);//Create a object called led2 attached to Pin 13 with a delay of 1200

void setup() {

}

void loop() {

led1.Update();

led2.Update();

}

So as you can see, the class looks very much like the Blink with out Delay sketch, with a couple of notable exceptions.

There is now something called PUBLIC in this: This is were the functions we have access to will go. In this case, we have the constructor (Blink, and the Update function)

In the case of SoftwareSerial some of the PUBLIC functions were already listed.

There is also a PRIVATE constructor, this is a place that only the CLASS has access to. We don’t need any PRIVATE things for this CLASS.

In the above example, just like with SoftwareSerial, we call the Blink class and create a object called led1 and led2, these objects both have 2 variables to them.

and like SoftwareSerial, we use the “friendly” object names to interact with Blink class.

led1.Update();

The loop in this example can run very quickly and the class keeps track of both states of LEDs, and the timings used.

What will this do for the BittyBot - well, I re-wrote the drive sketch, (which showed how to make the Bitty move) into a CLASS, This allows additional code to be ran, while still watching how long the motors are on. I demoed this with a re-write of the Ultrasonic code - If you remember the 1st ultrasonic code would sometimes run into things, the code worked for the most part, but still would at times hit things. Using the CLASS (which uses MILLIS) and better timings, the ultrasonic rarely hits anything now, it still not perfect, but is much better.

Also using the class I wrote a sketch that can use a Joystick to remotely control the BittyBot.

(actually, there are a few intermediate/advanced programming things beyond the CLASS in those sketches as well)

I plan on doing a re-write of the ESP8266 sketches using the CLASS, while the DELAY for the one sketch didn’t seem too bad, I think it will improve with using CLASS.

HERE are a few (like I said there are hundreds of tutorials) tutorials I found for CLASS, and CLASS with Arduino:

<http://www.tutorialspoint.com/cplusplus/cpp_classes_objects.htm>

<http://www.cplusplus.com/doc/tutorial/classes/>

<https://learn.adafruit.com/multi-tasking-the-arduino-part-1/overview> (This is the one I used to base my code on, and my example above, it’s not the same as what I did above, but it is what I based it on)

TIME BASED INTERRUPTS and PIN CHANGE INTERRUPTS:

SO CLASS is a fairly advanced programming concept, but there are even more ADVANCED concepts that the Arduino can do, and you can improve your code greatly if you understand how you can use the timers built into the Arduino for yourself.

It should be noted that making changes to some of these timers can really mess things up too.

Here is a sites I found that explains some of what can be done with timers:

<http://letsmakerobots.com/content/arduino-101-timers-and-interrupts>

<http://gammon.com.au/interrupts> (I got more from this site than the one above)

There are other sites that explain it too, most of this is over my head. But it’s a worthy exercise to learn.

Also in the MORE Advanced category:

Using PIN Change Interrupts. So we know the Arduino UNO has 2 interrupt pins, that are documented, and easy to use. However, All of the PINs can be used as CHANGE interrupts.

They are 3 BANKS of pins that can be used, and the trick is be able to figure out which pin in the BANK triggered the interrupt.

<http://playground.arduino.cc/Main/PinChangeInterrupt>

A lot of the advance concepts fall not only in the hardware realm, but also in just how powerful C++ is.