Overall reflection for CSE2920 - Phidgets

Stefan Such

As you can probably see: I've done individual reflection logs for each assignment already. This will be more-or-less a compilation of what I've learnt throughout this credit.

Having to manually write polling loops in the beginning of the credit was annoying, but not too difficult. The hardest thing to do was to detect when the button state changed, which involved comparing the previous state of the button to it's current state every refresh.

```
boolean redButtonPreviousState = false;
boolean greenButtonPreviousState = false;

if (redButton.getState() != redButtonPreviousState) {
    System.out.println("Red Button State: " + redButton.getState());
    redButtonPreviousState = redButton.getState(); // refresh this ins
}
```

Another thing that was difficult for me to wrap my head around, was the fact that the pre-written code was written using spaces instead of tabs for indentation. It doesn't work well with eclipse (given that Eclipse has no issues with automatically indenting with tabs), which resulted in me either deleting too many or too few characters at a time.

One thing that I started doing in the first few lessons (and continued throughout the entire credit) was copy/pasting code from previous assignments to save on time. The process for creating, addressing, and opening phidgets never changes, unless you're using a new phidget or have your phidgets plugged into a different port.

The manual polling method became a roadblock when I started work on the Build A Thermostat challenge, because it wanted me to poll the thermometer at a different speed that I poll the buttons. If I could have used button events for this, it would have simplified most of the work I had to do. I managed to simplify my work to this:

```
int printTemperatureEvery_Seconds = 1;
int pollingSpeedInHertz = 100;
int printTemperatureEvery_Refresh = printTemperatureEvery_Seconds * pollingSpeedInHertz;
int refreshNumber = 0;
```

The program will refresh the buttons at the specified polling rate; it will then take the number of seconds needed in between printing the temperature, convert that to the number of refreshes in between printing the temperature, and then count the refreshes so that when the refresh number is divisible by `printTemperatureEvery_Refresh` it'll output the temperature.

The methods unit was a one-and-done for this credit. It teaches you how to use PWM dimming to control the LED brightness, but then it never requires you to do anything with that information. The same goes for teaching you how to close phidgets (though, the lesson itself admits that closing phidgets is mainly useful for power constrained use cases and low-memory cases)

The knowledge gained about *Duty Cycles* in relation to PWM will probably help me in the future, outside of school. For context, one of my hobbies is building high power Nerf blasters, and a project I'm currently planning out involves brushless motors and control boards for said motors, so I'll probably end up using a pot hooked up to an Arduino to adjust the RPM of the motors (IIRC, applying more voltage but using PWM produces more torque than adjusting the voltage to change the RPM)

The events unit was the final unit I needed to do, and it wasn't particularly hard. It was mostly just repeating the same block of code over and over and doing marginally different things with it.

StateChangeListener, TemperatureChangeListener, AttachListener, they all do ~the same thing. Naturally some of these methods have data intervals, and some don't, but they all fundamentally do the same thing.

```
redButton.addStateChangeListener(new DigitalInputStateChangeListener() {
    public void onStateChange(DigitalInputStateChangeEvent e) {
        if (e.getState() == true) {System.out.println("Red button pressed");}
        else {System.out.println("Red button released");}
    }
});
```