EECS 1021 Lab F: Arduino & Java

Richard Robinson and James Andrew Smith

February 2021

Contents

Listings

1 Marking Guide

2 Part 1: Creating a countdown clock with Arduino

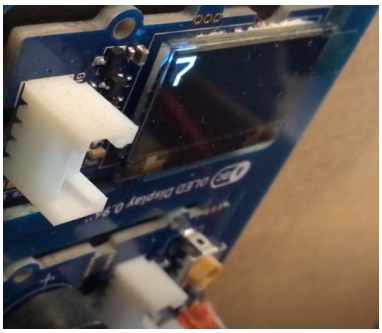
2.1 Objective

The objective of this part of the lab is to create a pure Arduino program using C++ which displays a countdown clock using the OLED and LED on the device. This will familiarize you with several Arduino

Part	Weight	details
Complete Part 1	0.1	All or nothing.
Complete Part 2	0.5	All or nothing.
Complete Part 3	0.4	All or nothing.
Total	1.00	

Table 1: Grading Scheme

functions and expand your knowledge of the OLED.



2.2 Specifications

Listing 1 provides you with an Arduino IDE source file that is similar to what you need to complete this part of the lab. You are to modify this file to meet the following specifications:

- The countdown timer should run for 10 seconds
- At the end of the countdown (when there is 0 seconds remaining), the OLED should stay at displaying the value '0'. As well, the LED should light up.

2.3 Procedure

Listing 1 provides you with a sample program that mostly implements what you need. **Modify** and **complete** the implementation to achieve the desired effect. This will require, in part, to become familiar with several new Arduino functions such as `digitalWrite` and `pinMode`:

- digitalWrite : `https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/`
- pinMode : `https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/`

In planning out your work, contrast the output of the source code that has been supplied to you

- `https://youtu.be/pye5dCUcbk4` (the example provided)

with the desired version:

- `https://youtu.be/6-0xkbXySv0` (the desired output for Part 1)

3 Part 2: Sending data from Java to Arduino

3.1 Objective

The goal of this part of the lab is to use a Java program to send data to an Arduino program. Specifically, we want to create a countdown clock on the Arduino OLED display and LED, via Java. It will require you to create a Java class from scratch. Use the tools you have (namely, your IDE) to your advantage!

The YouTube video (`https://youtu.be/-_zYHVVfbig`) shows how the OLED should behave in response to signals from the Java program to your Arduino.

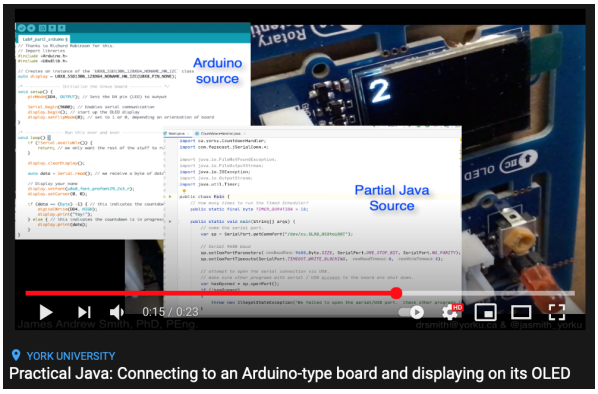


Figure 1: In Part 2, your OLED should display on the Arduino/Grove board should display values sent by the Java program, as shown in https://youtu.be/-_zYHVVfbig

3.2 Specifications

You are already given the following Arduino C++ program in **Listing 2**, which should remain unchanged. You are to create a Java program which should function as follows:

- The Java program should open up a connection with the Arduino
  - jSerialComm Maven page: `https://search.maven.org/artifact/com.fazecast/jSerialComm/2.6.2/jar`.
- Every one second, the program should send a byte of data representing the number of seconds until the countdown completes.
- The countdown should run for 10 seconds.
- At the end of the countdown (when there is 0 seconds remaining), the program should send a sentinel value to the Arduino indicating as such. If this is done correctly, the OLED will display "Yay" and the LED will light up.
  - The sentinel value can be a single byte with a value of -1.

3.3 Procedure

Create a Java program which satisfies the above specifications. For your convenience, starter code is provided in **Listing 3**. Everything needed to open and close the port is given. This should not be changed (except for the port name, which varies by device). The comments in the code provide some hints on what is expected of you.

You should also review the following video material

- Planning the Java program: `https://youtu.be/0d5n-eGwkCo`

- Implementing the Java program: `https://youtu.be/0hWVB_dOD_s`

These two videos will guide you in modifying the listings provided in this document so that you can have a successful interaction between your Java program and the Arduino.

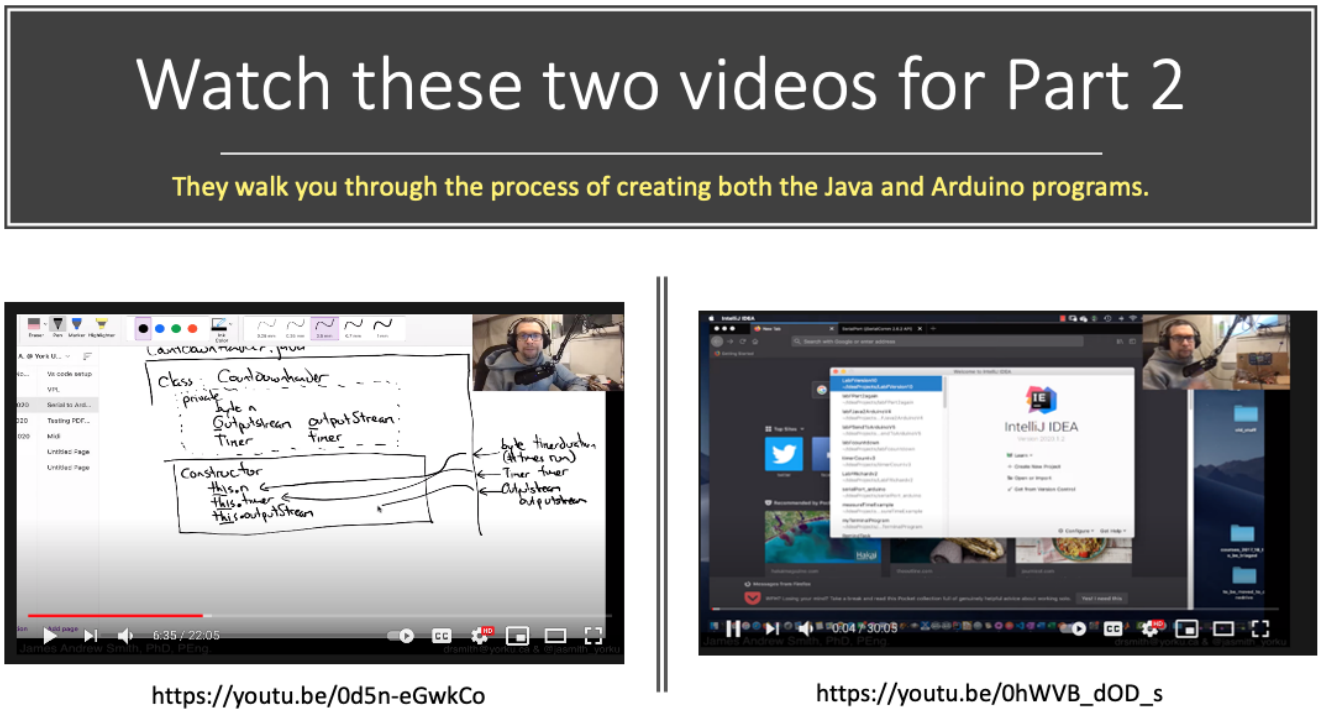


Figure 2: Watch these two videos, https://youtu.be/0d5n-eGwkCo and https://youtu.be/0hWVB_dOD_s , to set up your programs (Java and Arduino) for Part 2.

Want to understand a bit more about the Timer and TimerTask classes? Have a look at this video:

- Timer & TimerTask overview: `https://youtu.be/FDrw-sfTj4o`

4 Part 3: Receiving data from Arduino

4.1 Objective

This section builds on the previous one. By now, you should have a class handling your countdown functionality. In this section, you will be receiving input in Java from your Arduino. Specifically, when the user presses the button on the Arduino, your Java code should reset the countdown back to 10 seconds. To reiterate:

- Send the values 9 through 0 as "bytes" over USB to the Arduino from your Java program. Update the count once per second.
- When you get to 0, send -1 from Java to the Arduino.
- When the Arduino receives 0 to 9, have it display those digits on the OLED display.
- When the Arduino receives -1, have it display Yay!.
- If the user presses the button on the Arduino board, have the Arduino send a single byte (any value) to the Java program.
- When the Java program receives that data, have it update the countdown count within itself.
- WIth the updated countdown value, send the updated countdown value to the Arduino. For instance, if the Arduino display 5 and the button is pressed, then the count should be updated within one second so that the next time it updates its display it will display 9.

4.2 Specifications

The code you developed in Part 2 will remain largely unchanged, with several additions. The only additional specification in this section is that pressing the button resets the countdown clock and turns off the LED.

A more simple set of specs was used to get a demo running. You can see that here: `https://youtu.be/9pbsasv2izk`.

4.3 Procedure

Start by using the code you have made from the previous section. On the Java side, you should have a class which handles the countdown functionality:

1. Augment this class by implementing the `SerialPortDataListener` interface. When new data is received, the class should reset the countdown back to its initial state.
2. Register the class as a listener to the `SerialPort` by using the `addDataListener` methods.

On the Arduino C++ side, you should make the following changes:

1. In the loop function, add a check to see if the button is pressed. (**Hint:** make sure to set the `pinMode` for the button!)
2. In the case of the button being pressed, send a byte of data to `Serial`. As well, turn the LED off.

The video, `https://youtu.be/9pbsasv2izk`, was created to show you a complete Java program that can receive data from an Arduino. There are three listings in the back of this lab document that you can use to reproduce what you watched in the video. Combine the new elements in this video with the program from Part 2 to create the countdown program for Part 3.

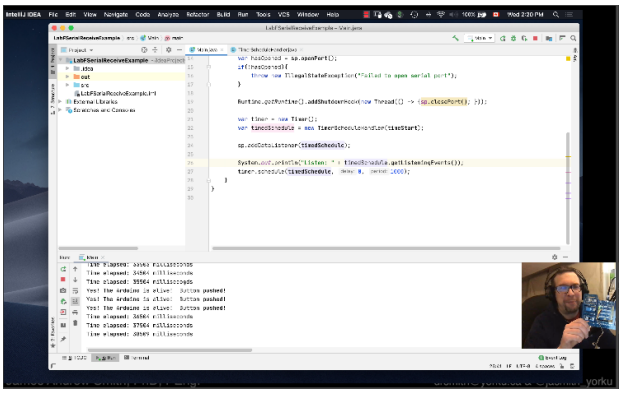


Figure 3: To get you going in Part 3, we've created a video that illustrates the new elements that allow your Java program to receive serial data. Have a look at the video here: https://youtu.be/9pbsasv2izk

A Listings for Students

A.1 Part 1: Arduino Countdown

```
// Import libraries
#include <Arduino.h>
#include <U8x8lib.h>

// Creates an instance of the 'U8X8_SSD1306_128X64_NONAME_HW_I2C' class
auto display = U8X8_SSD1306_128X64_NONAME_HW_I2C(U8X8_PIN_NONE);

int n = 10;
char myCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};

/* --------------- Initialize the Grove board ------------- */
void setup() {
    pinMode(DD4, OUTPUT); // Sets the D4 pin (LED) to output

    Serial.begin(9600); // Enables serial communication
    display.begin(); // start up the OLED display
    display.setFlipMode(0); // set to 1 or 0, depending on orientation of board
    display.clearDisplay();

}

/* --------------- Run this over and over ------------------- */
void loop() {
    // Set up a countdown on the OLED


    display.setFont(u8x8_font_profont29_2x3_r);
    display.setCursor(0,0);


    if(n>=0)
    {
      display.print(myCharArray[n]);
      n=n-1;
    }
    else    // start again
    {
      n=10;
    }

    // pause between updates on the screen...
```

```
42      delay(700);
43
44
45
46  }
```

Listing 1: The example (incomplete) Arduino C++ program for Part 1. Modify this so that the OLED counts down from 9 to 0 in one second intervals. Make sure it stays at 0 at the end and that an LED lights up to signal that the countdown is done.

A.2 Part 2: Java to Arduino

```
// Import libraries
#include <Arduino.h>
#include <U8x8lib.h>

// Creates an instance of the ‘U8X8_SSD1306_128X64_NONAME_HW_I2C‘ class
auto display = U8X8_SSD1306_128X64_NONAME_HW_I2C(U8X8_PIN_NONE);

/* -------------- Initialize the Grove board ------------ */
void setup() {
    pinMode(DD4, OUTPUT); // Sets the D4 pin (LED) to output

    Serial.begin(9600); // Enables serial communication
    display.begin(); // start up the OLED display
    display.setFlipMode(0); // set to 1 or 0, depending on orientation of board
}

/* --------------- Run this over and over ------------------- */
void loop() {
    if (!Serial.available()) {
        return; // we only want the rest of the stuff to run if ‘Serial‘ is available
    }

    display.clearDisplay();

    auto data = Serial.read(); // we receive a byte of data via Java

    // Display your name
    display.setFont(u8x8_font_profont29_2x3_r);
    display.setCursor(0, 0);

    if (data == (byte) -1) { // this indicates the countdown is over
        digitalWrite(DD4, HIGH);
        display.print("Yay");
    } else { // this indicates the countdown is in progress
        display.print(data);
    }
}
```

Listing 2: The Arduino C++ program for Part 2.

```java
package com.richardrobinson;

import com.fazecast.jSerialComm.SerialPort;

import java.io.IOException;
import java.util.Timer;

public class Main {
    public static final byte TIMER_DURATION = 10;

    public static void main(String[] args) {
        var sp = SerialPort.getCommPort("/dev/cu.usbserial-1410");

        sp.setComPortParameters(9600, Byte.SIZE, SerialPort.ONE_STOP_BIT, SerialPort.NO_PARITY);
        sp.setComPortTimeouts(SerialPort.TIMEOUT_WRITE_BLOCKING, 0, 0);

        /*
        We open the serial port. If it fails, our program is in an illegal state, and we throw
        an exception.
         */
        var hasOpened = sp.openPort();
        if (!hasOpened) {
            throw new IllegalStateException("Failed to open port.");
        }

        var outputStream = sp.getOutputStream();

        /*
        When a program ends, it is important to `close` all resources. Because this program
        only terminates manually, we add this code which executes when you terminate the program,
        which closes the output stream and the serial port.
         */
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            try {
                outputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            sp.closePort();
        }));

        /*
        This Timer repeats every 1000 ms, indefinitely. Here lies your challenge:
        Replace `null` with a reasonable parameter to get your code to function as expected.

        Hint: create a class which extends `TimerTask`.
         */
        new Timer().schedule(null, 0, 1000);
    }
}
```

Listing 3: Main.java (for Part 2).

A.3 Part 3: Arduino to Java

```
#include <Arduino.h>

void setup() {
  // put your setup code here, to run once:
   pinMode(DD6, INPUT);  // button.
   Serial.begin(9600);  // enable serial
}

void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(DD6) == HIGH) {
    Serial.write("Button!\n");   // button was pressed
    delay(400); // debouncing, long presses.
  }
  else{
    // nothing
  }

  if (!Serial.available()){
    return;
  }
}
```

Listing 4: The example Arduino C++ program for Part 3.

```java
import ca.yorku.eecs1021.TimerScheduleHandler;
import com.fazecast.jSerialComm.SerialPort;
import java.util.Timer;

public class MainPart3StudentExample {

    public static void main(String[] args) {
        long timeStart = System.currentTimeMillis();

        var sp = SerialPort.getCommPort("/dev/cu.SLAB_USBtoUART");
        sp.setComPortParameters(9600, Byte.SIZE, SerialPort.ONE_STOP_BIT, SerialPort.
    NO_PARITY);
        sp.setComPortTimeouts(SerialPort.TIMEOUT_WRITE_BLOCKING,0,0);

        var hasOpened = sp.openPort();
        if(!hasOpened){
            throw new IllegalStateException("Failed to open serial port");
        }

        Runtime.getRuntime().addShutdownHook(new Thread(() -> {sp.closePort(); }));

        var timer = new Timer();
        var timedSchedule = new TimerScheduleHandler(timeStart);

        sp.addDataListener(timedSchedule);

        System.out.println("Listen: " + timedSchedule.getListeningEvents());
        timer.schedule(timedSchedule, 0, 1000);
    }
}
```

Listing 5: The Example "Main" class in Java for Part 3, MainPart3StudentExample.java.

```java
package ca.yorku.eecs1021;

import java.util.TimerTask;
import com.fazecast.jSerialComm.SerialPort;
import com.fazecast.jSerialComm.SerialPortDataListener;
import com.fazecast.jSerialComm.SerialPortEvent;

public class TimerScheduleHandler extends TimerTask implements SerialPortDataListener {

    private final long timeStart;
    // constructor
    public TimerScheduleHandler(long timeStart) {
        this.timeStart = timeStart;
    }

    // Override run method in TimerTask
    @Override
    public void run() {
        System.out.println("Time elapsed: " +(System.currentTimeMillis() - this.timeStart) +
     " milliseconds");
    }

    @Override
    public int getListeningEvents(){
        return SerialPort.LISTENING_EVENT_DATA_RECEIVED;
    }

    @Override
    public void serialEvent(SerialPortEvent serialPortEvent) {
        if (serialPortEvent.getEventType() == SerialPort.LISTENING_EVENT_DATA_RECEIVED){
            System.out.println("Yes! The Arduino is alive!  Button pushed!");
        }
    }

}
```

Listing 6: The "CountdownHandler" class in Java for Part 3, TimerScheduleHandler.java.