← Week 5 Overview

Assignment 5: Dot Dot Dot, Dash Dash Dash, Dot Dot Dot

Logistics

Assigned: Wed, Oct 5

Due: Wed, Oct 12

1 The idea

In this assignment you'll use the combined power of Java and Arduino's built-in streams. Java's streams will send data to the Arduino, and the Arduino's streams will process those bytes and display their contents in **Morse code**. Morse Code is a lot like a substitution ciper because individual letters are replaced by a specific, fixed "word".

By the end of this assignment you should have a better understanding of how streams work and how they can be used. In addition, you will: (again) practice delta timing, practice iterating through arrays, practice working with ASCII characters, and have to carefully consider the logic required for timing.

2 The background

2.1 The Java side of the stream

We will be using Java's `OutputStream` class, which is very similar to, but in the opposite direction of, the `InputStream` class from last week. The `SerialComm` class from last week includes a `getOutputStream()` method that returns a reference to an `OutputStream` object. This `OutputStream` will take a sequence of bytes and deliver them to a Serial port, and your Arduino will be connected to the receiving end of the serial port. The `write(int)` method of

`OutputStream` will put a single byte (the `int` argument) in the stream, which will be transmitted to the Arduino.

As with `InputStream`, an `OutputStream` can be wrapped (using the decorator pattern—we "decorate" the original class with more functionality) to do more things. We will "decorate" the `OutputStream` using a pair of classes `ViewOutputStream`, which you will author. `ViewOutputStream` is like last week's `ViewInputStream`. It will allow you to view the bytes entering the stream, but this time you will watch the bytes leaving the PC. You will also decorate the `OutputStream` using `DataOutputStream`, which is a built-in class and provides features to send multi-byte data elements out the stream (e.g., integers, floats, etc.).

2.2 The Arduino

2.2.1 Serial stream

The primary functions for interacting with Arduino's incoming `Serial` data are `Serial.available()` and `Serial.read()`. The `Serial` object operations are also built on top of `Stream` objects.

`Serial.available()` returns the number of bytes currently available to the `Serial.read()` function. `Serial.read()` reads the next byte from the stream. They behave very similarly to Java's `IntputStream`'s `available()` and `read()` methods.

2.2.2 Additional Functions

You will also use `morseEncode(char symbol)`, which has already been written for you. `morseEncode()` will return a `String` that contains data corresponding to the Morse code for a given symbol. Unfortunately Morse Code doesn't distinguish between capital letters and lower case letters. The `morseEncode()` function will call a function named `toUpper()` to will convert lower case to upper case, but *you* have to write the `toUpper()` function. `toUpper()` should have the following prototype:

```
// Argument: Any character
// Return Value: Either:
//        1) If the character is a letter, the upper case equivalent.
//        2) If the character is not a letter, the original value.
char toUpper(char c) {
    ...
}
```

The easiest way to complete `toUpper()` is to use the properties of ASCII values. If you look closely at the ASCII Table you'll notice that a lower case letter can be converted to an upper case letter by changing just one bit, so you may want to review bitwise operations. Since the letters are numeric and in lexicographic order they can also be compared via the normal relational operators, like:

```
char c;
...
```

```
// Does this character come "at or after"
// the character for 0? (The character 1 would come after 0, etc.)
if(c>='0') {
```

Using these properties of ASCII (being able to do comparisons, lexicographic ordering, and being able to manipulate bits) you can complete `toUpper()` with just a few lines of code.

2.2.3 Including files

There are three files in the `assignment5/MorseCoder/` directory:

- `MorseCoder.ino`: This contains a nearly empty sketch. You will complete all your Arduino work here.
- `MorseCodes.h`: This is called a header file (hence the ".h" extension) and contains things that need to read at the top (or head) of a source file to define features used later in the file. The `MorseCoder.ino` Arduino file "includes" it to have access to the function defined in `MorseCodes.cpp`. A header file and corresponding `#include` are used somewhat like the `import` statement in Java. **Read through the comments which describe the function(s) you may need to use for this assignment.**
- `MorseCodes.cpp`: This contains a Morse code table and code that uses it. You do not need to modify it. **Although you don't need to understand the exact details of how `morseEncode()` works to be able to use it, take a few minutes and read through it. You should notice that it uses the properties of ASCII in a few ways.**

Header files are an important part of code organization in C. They're usually paired with a **source file** (Often with a ".c", ".cc", or ".cpp" extension). This separation helps for several reasons: it speeds up compile time, keeps code organized, and it can prevent errors due to order of declarations. Most importantly, the header is usually used to specify a public **interface** to functions whereas the corresponding source file provides private **declarations**. Often when you just need to use a function and don't care about how it works you can simply refer to the header file.

Often a library of related functionality is contained in one source file (.cpp file) that may be used in many different projects. The MorseCodes files are a library that can be used to encode and decode Morse code.

3 The Arduino Assignment

3.1 Getting the Arduino to recognize input from the Serial Monitor

1. Start by making the sketch in `MorseCoder.ino` read in characters from the Serial Monitor and echo them back, as in Studio 4.
2. Complete the `toUpper()` function and test it. Use it to ensure all the echoed characters are in upper case. For example, if the following is entered:

```
Don't shout at me!
```

It would be echoed back as:

```
DON'T SHOUT AT ME!
```

3. Use the `morseEncode()` function and echo back the Morse code that corresponds to the characters instead of the characters. `morseEncode()` will use your `toUpper()` function, so there's no need for you to call `toUpper()` yourself. Read the comments in `MorseCodes.h` for details of how you need to call `morseEncode()`.

4. Update your code to blink an LED for the given Morse code. You will have to process each character of the Morse code separately. For example, the `morseEncode()` function will return the S[illegible] You will have to loop through the returned string, retrieve each character, and use the character to turn on an LED for an appropriate amount of time. You can access an individual letter by using array-like notation. For example, this would print each letter in a string one-at-a-time:

```
String words = "Hello World!";
for(int i=0;i<words.length();i++) {
    Serial.print(words[i]);
}
```

Some notes about timing of Morse code:

- the length of a dot is 1 unit (you choose the unit. 500ms is a good choice)
- a dash is 3 units
- the space between parts of the same symbol's code is 1 unit (so at least 1 unit between any dot and dash)
- the space between different symbol's codes is 3 units (3 units between the codes for two letters)
- the space between words is 7 units (or 4 units more than the end of a symbol)

When not displaying a dot or a dash the LED should be turned off.

Although you may initially use `delay()` timing, when you're done you should only be using delta timing. (I.e., **NO `delay()` allowed**)

4 The PC Assignment

Your final task is to author a Java program that performs very close to the same function as the Serial Monitor.

1. Create a new Package called `assignment5.morse/`.
2. Create a new Java class called `SerialOutput` in `java/assignment5`. It will be somewhat like your `JavaHexTX` from Studio 5 with a few minor changes:
   - It will read an entire line at a time.
   - It will send send each character to serial port.
   - The port's output stream (use it's `getOutputStream()` method) will be wrapped in a `ViewOutputStream` object that will allow you to see the bytes as they enter the stream.
3. Create a new Java class called `ViewOutputStream` in `java/assignment5`. It should `extends FilterOutputStream`. `OutputStream`s have `write()` methods that are used to write data into the stream. We want to replace (Override) `write()` but also still defer to the `write()` defined in `FilterOutputStream`. That is, when `write()` is called your version of `write()` will run. It will show the data in a `PrintStreamPanel` as well as passing the data on to the original version of `write()`. Since the original version of `write()` is in the parent (or

"super") class, you will have to call it using the keyword `super`: `super.write( ... )`.

The constructor should be nearly identical to the one in your `ViewInputStream` (you may want to rename the window's title to be "Output Stream" rather than "Input Stream");

You will need to *Override* the parent class' `write(int)` method:

```
@Override
public void write(int i) throws IOException {
    // Send the Hex values corresponding to i (the parameter)
    // to be displayed in the panel and also send it to the parent
    // class so it can do its "write"
}
```

5 The check-in

1. Commit all your code. **Do not ask to be checked out by a TA until after you have made certain that your work is committed**. Failing to do this may result in you losing points on your assignment.

2. Follow the checklist below to see if you have everything done before demo your assignment to a TA.
   - Your sketch is able to read input from Java or the Serial Monitor
   - `Serial.read()` should only be invoked when data is `available()`
   - LED flashes properly (dot and dash are distinguishable)
   - Timing between letters and between words is correct
   - No `delay()` is used in your work
   - Your Java program reads lines of characters and sends them through the serial port.
   - You Java program opens `PrintStreamPanel` window and displays the outgoing data
   - All of your files are committed

3. Check out with a TA.

New files:

```
└── assign5/
    ├── MorseCoder/
    │   ├── MorseCoder/MorseCoder.ino
    │   ├── MorseCoder/MorseCodes.cpp
    │   └── MorseCoder/MorseCodes.h
    ├── SerialOutput.java
    └── ViewOutputStream.java
```

5.1 The rubric

- 100pts: Did the demoed assignment work?
  - `toUpper()` meets requirements (10pts)
  - Proper Morse code. Dashes differ from dots, blinking the correct patterns. (15 pts)
  - Proper timing between dots/dashes, between symbols, and between words (15 pts)
  - Arduino code uses delta timing (No `delay()`s) (15 pts)
  - Arduino code is well-organized, variables are appropriately named, and comments are used to aid understanding (10 pts)
  - Java code reads from the console and sends data to the Arduino (25 pts)
  - `ViewOutputStream` class extends the `FilterOutputStream` and is displaying the bytes that enter the stream. (`ViewOutputStream` has its `write(int)` overridden) (10 pts)

Generated at 2016-10-10 02:56:38 -0500.

Page written by Julia Vogl & Bill Siever. Site design by Ben Stolovitz.