

Observability

CSE 102

Today's Outline

- Observability – smart debugging
 - On Arduino – when we cannot use serial port
 - On PC in Java – when we are talking to Arduino
- Communicating between PC and Arduino
 - Java on PC (either Windows or Mac)

Observability

- What is **really** going on?
- Option 1: stare at the code until inspired
 - When that doesn't work, make random change
- Option 2: don't assume the code you actually wrote does what you think it does!
 - Alter code so that you discover what it really does
 - On PC in Java, use the debugger!
 - Or use `System.out.print()` to display on console
 - On Arduino in C, use `Serial.print()`

Observability on Arduino

- What about when PC isn't available?
- 2x16 character LCD display (class `ST7036`)
 - `print()` method is available
 - Accepts multiple data types: string, int, etc.
- Initialization and use
 - Constructor: `ST7036 lcd = ST7036(2,16,0x7c);`
 - In `setup()`:


```
lcd.init();
lcd.setContrast(0);
```
 - In `loop()`:


```
lcd.setCursor(line, column);
lcd.print("Hi!");
```

Computer Communications

- Link that provides byte-level data delivery
 - Network
 - Serial port
- Ability to send and receive on each endpoint
- Must use a protocol to understand anything other than individual bytes
 - Individual data elements (ints, chars, strings, etc.)
 - Higher-level, application-specific messages
 - The user just pressed button "X"
 - The pressure in vessel X is Y psi at time Z
- Needs to work across platforms
 - E.g., Java on PC and C on Arduino

Java Communications uses Streams

- Upstream writer, downstream reader



- Source writes to stream
- Destination reads from stream
- Either endpoint might be a file or some other input/output device, e.g.,
 - Dest. could be Arduino connected via serial port
 - Source could be a temperature sensor

Stream Conventions

- FIFO ordering (First-In-First-Out)
- Protocol must be same at both ends of stream for effective communication to take place
 - Stream of bytes? chars? integers? what is a char?
- Properties supported by streams that “wrap” other streams, e.g.,


```
InputStream stream = new InputStream(...);
DataInputStream dataIn = new DataInputStream(stream);
```

Wrapping Streams

- A stream can take another stream as a parameter to its constructor
- The outer stream delegates to the wrapped one
- E.g.,


```
DataOutputStream out =
    new DataOutputStream(
        new BufferedOutputStream(
            new FileOutputStream(...))
    );
```
- This is called “decorator” pattern

Communications in Java

- Open COM port with both `InputStream` and `OutputStream` objects
 - Works in Windows, Linux, and Mac
- Wrap `InputStream` with `DataInputStream`
- Wrap `OutputStream` with `DataOutputStream`

Individual Data Elements (in Java Stream)

- Byte – basic network element
 - `writeByte()`, `readByte()` in `Data(Input/Output)Stream`
- Character – two bytes in Java
 - `writeChar()`, `readChar()`, high byte first
- Short Integer – two bytes – bits can be anything from `0x0000` to `0xffff`
 - `writeShort()`, `readShort()`
- Integer – four bytes in Java – value -2^{31} to $2^{31}-1$
 - `writeInt()`, `readInt()`, most significant byte (MSB) first

Communications in Arduino C

- Byte – basic network element
 - `Stream.read()`, `Stream.write()`
- Character – two bytes in Java
 - Only 1 byte in C! Read and toss first byte, save second
- Integer – two bytes – bits can be anything from `0x0000` to `0xffff`
 - Read both bytes – value = $(\text{first} \ll 8) + \text{second}$
- Long Integer – four bytes – value -2^{31} to $2^{31}-1$
 - Read bytes
 - value = $(\text{first} \ll 24) + (\text{sec} \ll 16) + (\text{third} \ll 8) + \text{fourth}$

Strings

- Not just a sequence of two-byte characters!
- Network communication is language agnostic, so must acknowledge that others do things in different ways
- UTF-8 is common character encoding
- String is
 - 2-byte length (of bytes in string), followed by
 - Characters in UTF-8 encoding
 - Supported by `writeUTF()`, `readUTF()`
 - Need to build on Arduino side

Observability in Communications

- Need to know what is **really** going across the communication link
- On server, client, or maybe both:
 - Display what is going out the output stream
 - Display what is coming in the input stream
 - Support multiple interpretations of the raw data
- You can build these tools
 - Do a good job and it will help you the rest of the semester!

Observability Tools in Java

- One for `InputStream` and one for `OutputStream`
- Extend `FilterInputStream` (and its counterpart) as `ViewInputStream`
- `ViewInputStream`'s `read()` method should:
 - `read()` from the provided `InputStream`
 - Display the character(s) (in a specifiable form)
 - As an ASCII character
 - As a decimal value (0 to 255)
 - As a hexadecimal value (0x00 to 0xff)
 - As an integer (collecting 4 bytes before displaying)
 - Control format via constructor (or something else)