# Computer Communications

CSE 1302

## Today’s Outline

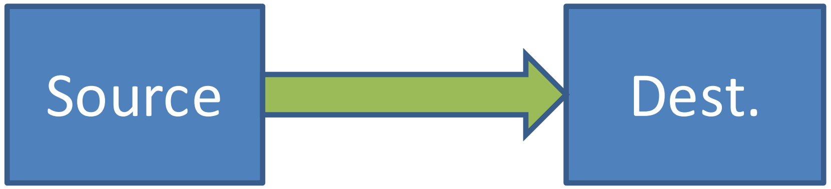
- Communicating between PC and Arduino
  - Java on PC (either Windows or Mac)
- Streams in Java
- Observability
  - SerialComm overview
- Protocol Design
- Images

# 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 the keyboard

## 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 in = new InputStream(…);
BufferedStream dataIn = new BufferedStream(in);
```

- Watch video for examples of user input streams

# Wrapping Streams

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

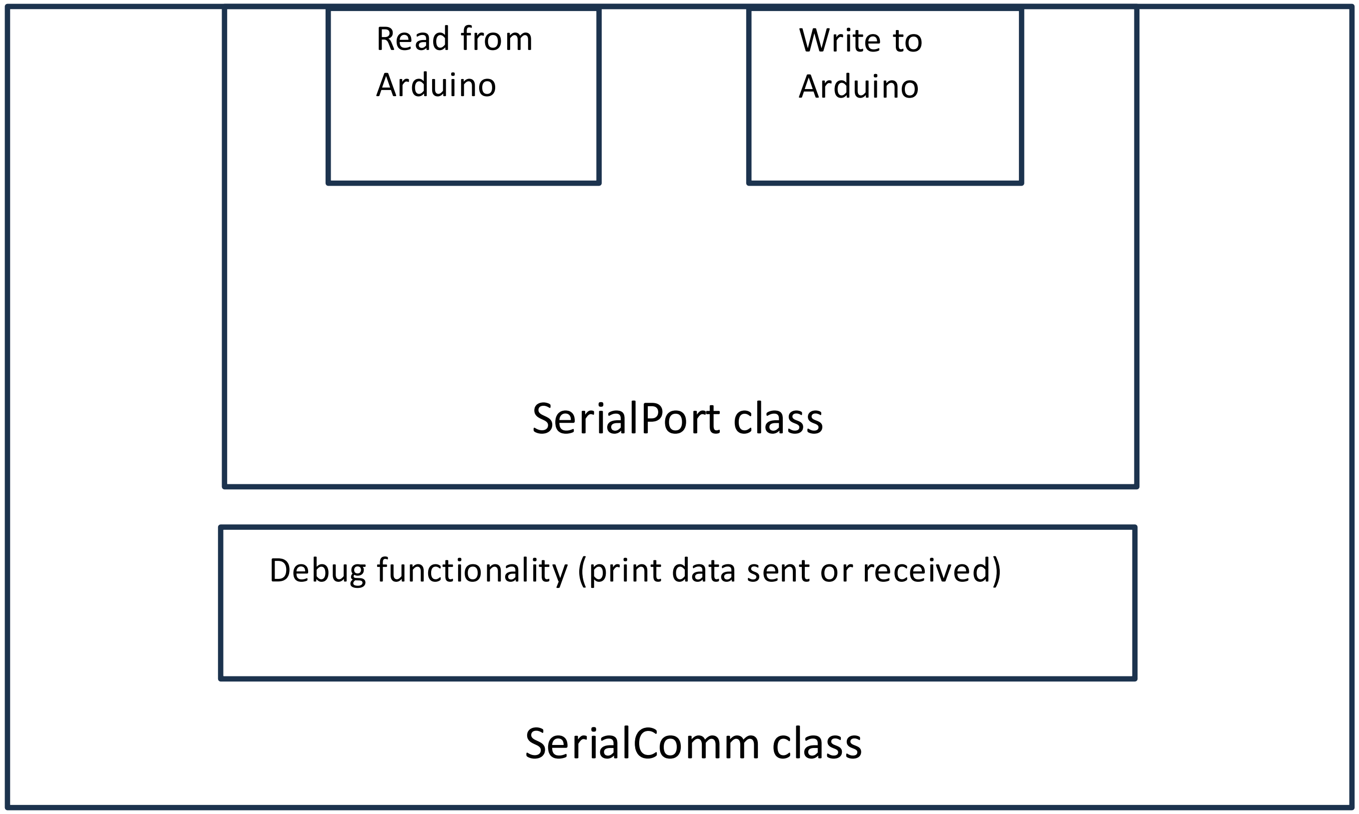
```
SerialComm sc = new SerialComm(new SerialPort(…));
```

- This is called “decorator” pattern
- We will use SerialComm and SerialPort in upcoming studios and assignments – starting with 4

## Communications in Java

- Open COM port with SerialPort object
  - Use SerialPort class, which we provide
  - Works in Windows and Mac
- Wrap SerialPort with a SerialComm object
  - You will write SerialComm class (some of it, anyway)
  - What “properties” does SerialComm provide?
    - Fixes up a data type inconvenience on input bytes
    - Most importantly, provides a debugging capability

# SerialComm class Functionality



## Back to Communications

- Streams are sequences of bytes
- We need data at a higher level of abstraction
  - Integers
  - Floats, Doubles
  - Characters
  - Strings
  - More
- Protocols must be designed to enable this
  - Build bigger things out of streams of bytes

## Individual Data Elements

- Byte – basic network element
  - writeByte(), readByte() in SerialComm class
  - Serial.read(), Serial.write() in Arduino C
- Character
  - Two bytes in Java
  - One byte in C
- Integer
  - Four bytes in Java
  - Two bytes in C

# 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 in Communications

- Need to know what is really going across the communication link
- On sender and receiver:
  - Display what is going out the output stream
  - Display what is coming in the input stream
  - Show the raw data (sequence of bytes)
- You will build these tools
  - This is the primary purpose of SerialComm class

## Quiz Time

- Go to Canvas and answer the single question on Quiz 4A

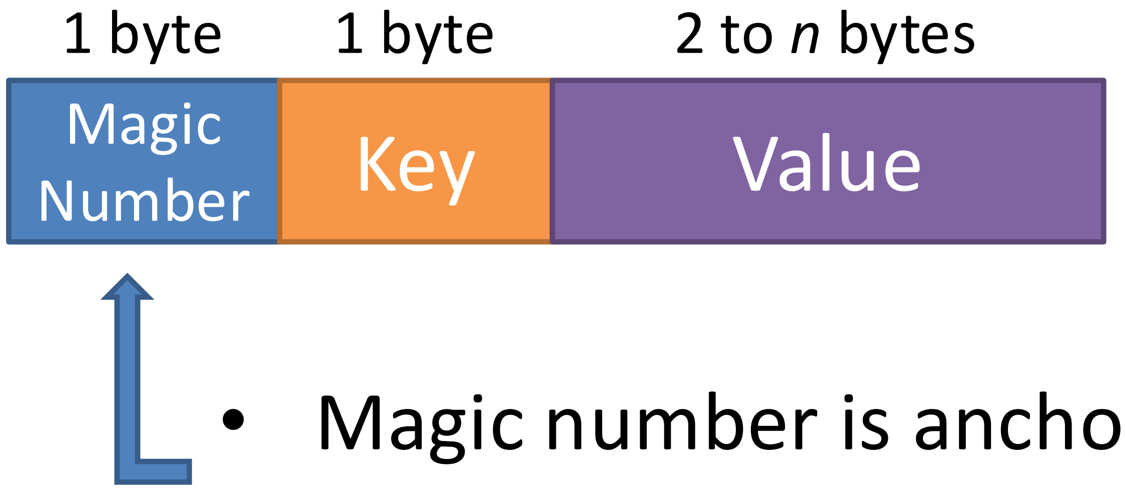
- A stream delivers bytes in a "first-in, first-out" order:
  - True
  - False

# Protocol Design

- What do we want to communicate?
- How do we want to say it?

# A Protocol for Us

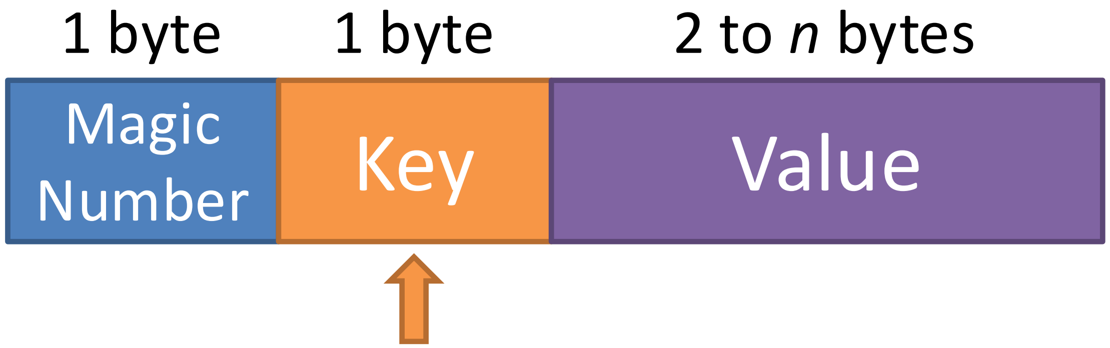
Message format:



- Magic number is anchor of message
- Always first byte
- Unlikely in rest of message
- Reader can ignore bytes until it sees magic number and then receive

# A Protocol for Us

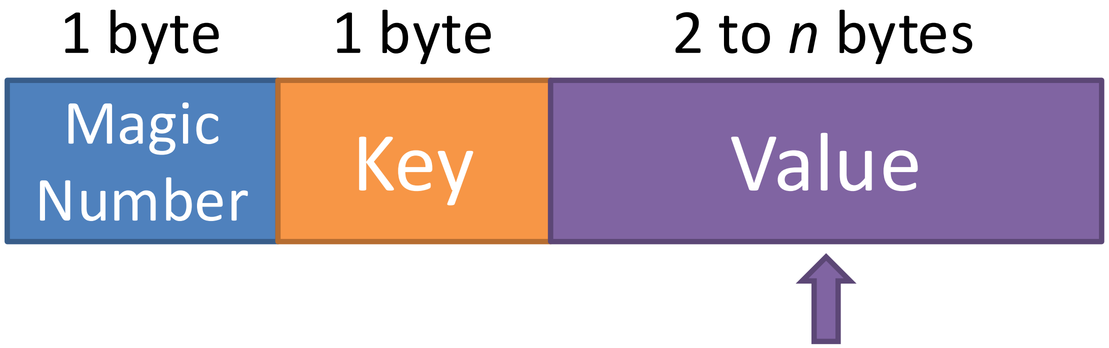
Message format:



- Key tells what type of message
- Indicates both size and interpretation
- E.g., 2-byte temperature value
- E.g., 4-byte timestamp
- E.g., UTF-8 encoded error string
- Table of legal keys must be maintained

# A Protocol for Us

Message format:



- Actual content of message
- Key tells how to interpret

## Images

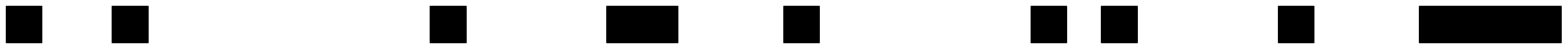
- Consider the following bits:

0x002400081881423c

0000 0000 0010 0100 0000 0000 0000 1000

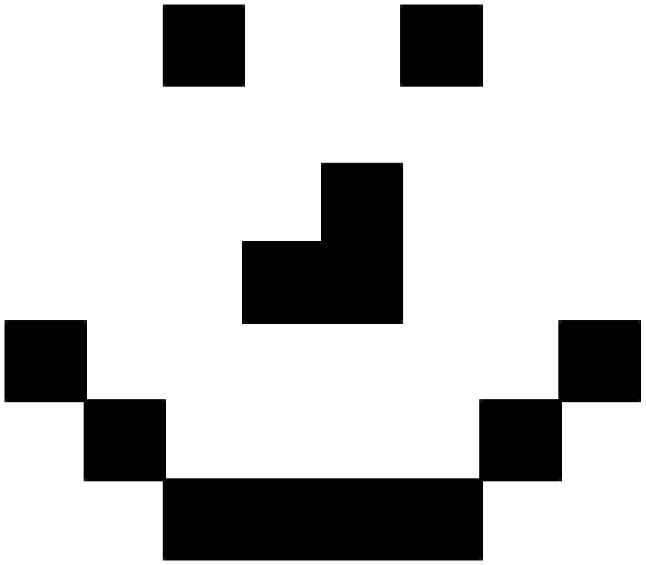
0001 1000 1000 0001 0100 0010 0011 1100

- Make 1 dark and 0 light:



# Images

- Arrange in rows, one byte per row:

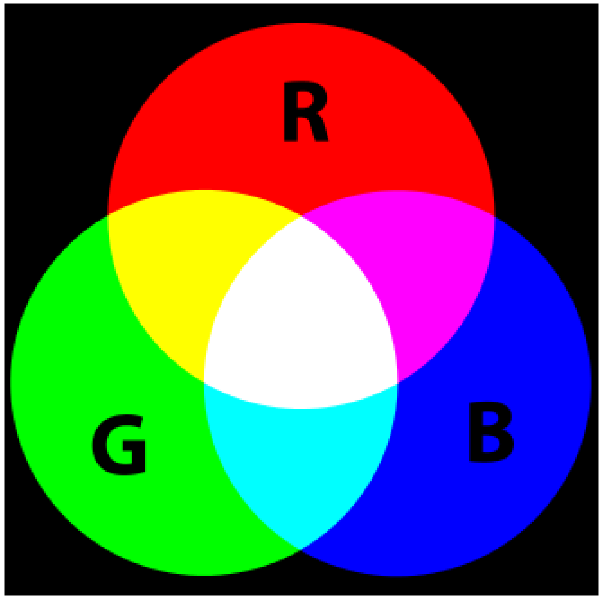


- Each bit is a "pixel" in the image

## Add color and more pixels

# Color

- Additive color – primaries Red, Green, Blue



- Position close together and put diffuser above
  - This builds one pixel