Computer Communications

CSE 132

Today's Outline

- Communicating between PC and Arduino
  - Java on PC (either Windows or Mac)
- Streams in Java
  - An aside on class hierarchies
- Protocol Design
- Observability

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 that 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 adds functionality to the wrapped stream
- E.g.,

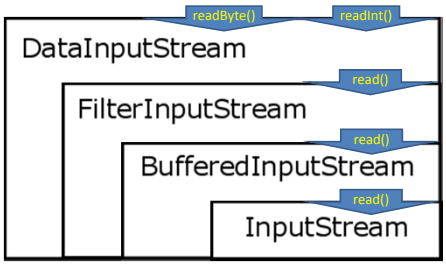
```
DataOutputStream out =
        new DataOutputStream(
                new BufferedOutputStream(
                        new FileOutputStream(…)
                )
        );
```

- This is called "decorator" pattern

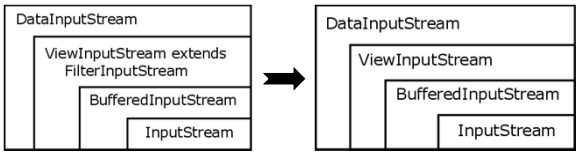
Communications in Java

- Open COM port with both InputStream and OutputStream objects
  - Use SerialComm class, which we provide
  - Works in Windows and Mac
- Wrap InputStream with BufferedInputStream
- Wrap BufferedInputStream with ViewInputStream
- Wrap ViewInputStream with DataInputStream
  - You will write ViewInputStream, extending FilterInputStream

Decorating InputStream

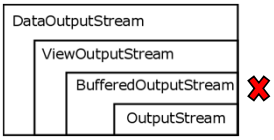


Authoring ViewInputStream



Decorating OutputStream

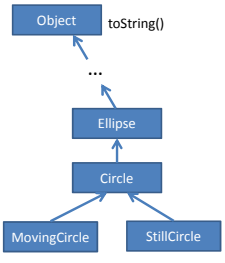
- Wrap OutputStream with ViewOutputStream
- Don't use BufferedOutputStream
- Wrap ViewOutputStream with DataOutputStream
  - You will write ViewOutputStream, extending FilterOutputStream



Aside on Java Class Hierarchies

- Scaling up programs
- Lots of objects?
  - Use data structures such as:
  - Lists
  - Queues
- Lots of classes?
  - Group related types – Java packages
  - Design hierarchically and exploit structure

Example Hierarchy



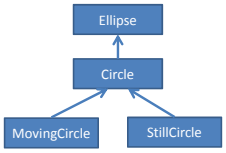
- Circle "extends" Ellipse
- All Circles are Ellipses
- Circle inherits from Ellipse
  - Instance variables
  - Methods
- Circle can override methods in Ellipse
- Circle can add new things

Benefits of Class Hierarchy

- Capture relationships to simplify reasoning
- Save implementation effort (less code) by inheriting functionality
- Polymorphism

  Circle c = any Circle, including subtypes of Circle

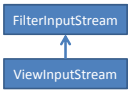
  c.anyMethodDefinedOnCircle()



How it relates to today's studio

- You will author ViewInputStream class
- It extends FilterInputStream class (which already exists)
- Child can use parent methods

  super.read() in child invokes read() method in parent



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 (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

Communicating Individual Data Elements 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 = (first << 8) + second
- Long Integer – four bytes – value -2^{31} to $2^{31}-1$
  - Read bytes
  - value = (first<<24) + (sec<<16) + (third<<8) + fourth

Strings

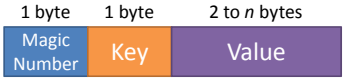
- Not just a sequence of two-byte characters!
- Network communication is language agnostic, so must acknowledge that others do things in different ways
- UTF-8 is common character encoding
- String is
  - 2-byte length (of bytes in string), followed by
  - Characters in UTF-8 encoding
  - Supported by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

Protocol Design

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

A Protocol for Us

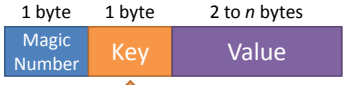
Message format:



- 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

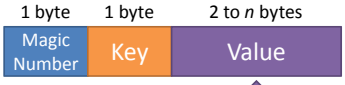
Message format:



- 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

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, receiver, or maybe both:
  - Display what is going out the output stream
  - Display what is coming in the input stream
  - Show the raw data (sequence of bytes)
- 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 byte(s) as a hex values (0x00 to 0xff)
  - This is the studio exercise this week
  - Required for assignment
- ViewOutputStream will be next week's task

This Week

- Studio
  - Use SerialComm to receive bytes in Java from Arduino
  - Author ViewInputStream
- Assignment
  - Use protocol to send temperature, potentiometer value, and high-alarm string to Java
  - Note: will need to unify analog input reference for the two readings