c h a p t e r 3

Sending and Receiving Data

# 3.1 Encoding Information

The only data types to which these operations can be applied are bytes and arrays of bytes.

* Need method to converted to byte arrays

## 3.1.1 Primitive Integers

The sender and receiver have to agree on several things:

* Size
* Little-endian
* The numbers transmitted will be signed or unsigned

Finally, essentially everything in this subsection applies also to the BigInteger class, which supports arbitrarily large integers. As with the primitive integer types, sender and receiver have to agree on a specific size (number of bytes) to represent the value.

## 3.1.2 Strings and Text

we can send almost any other kind of data: first represent it as text, then encode the text.

every String instance corresponds to a sequenceof char. A char value in Java is represented internally as an integer (by using ASCII, Unicode)

* So sender and receiver have to agree on encoding scheme

## 3.1.3 Bit-Diddling: Encoding Booleans

Bitmaps are a very compact way to encode boolean information, which is often used in protocols. Let’s number the bits of a value of type int from 0 to 31, where bit 0 is the least significant bit. In general, the int value that has a 1 in bit position i, and a zero in all other bit positions, is just 2i.

# 3.2 Composing I/O Streams

# 3.3 Framing and Parsing

- Framing refers to the problem of enabling the receiver to locate the beginning and end of a message.

Two general techniques enable a receiver to unambiguously find the end of the

message:

**Delimiter-based:** The end of the message is indicated by a unique marker, an explicit byte

sequence that the sender transmits immediately following the data. The marker must be

known not to occur in the data.

**Explicit length**: The variable-length field or message is preceded by a (fixed-size) length field that tells how many bytes it contains.

- With this framing method, the sender does not have to inspect the content of the message being framed; it needs only to check that the message does not exceed the length limit.

# 3.4 Java-Specific Encodings

When you know that:

(i) both ends of the communication will be implemented in Java

(ii) you have complete control over the protocol

* Can make use of Java’s built-in facilities like the Serializable interface or the Remote Method Invocation (RMI) facility

Sum-up:

- Custom-designed classes have to provide their own implementations of the serialization interfaces

- There are certainly situations where these built-in facilities are useful, but sometimes it is simpler, easier, or more efficient to “roll your own”.

# 3.5 Constructing and Parsing Protocol Messages

## 3.5.1 Text-Based Representation

The protocol specifies that the text be encoded using the US-ASCII charset.

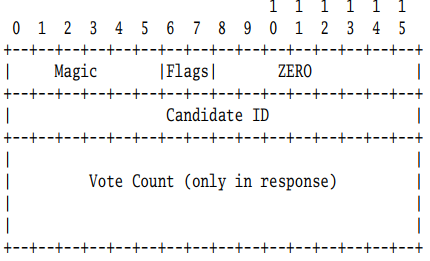
The message begins with a sequence of characters that allows a recipient to quickly recognize the message as a Voting protocol message.

The Vote/Inquiry boolean is encoded with the character ‘v’ for a vote or ‘i’ for an inquiry.

## 3.5.2 Binary Representation

The binary format uses fixed-size messages.

Each message begins with a one-byte field that contains the “magic” value 010101 in its high-order six bits.

The two low-order bits of the first byte encode the two booleans. The second byte of the message always contains zeros, and the third and fourth bytes contain the candidateID. The final eight bytes of a response message (only) contain the vote count.

## 3.5.3 Sending and Receiving

- Sending a message over a stream is adding appropriate framing information, and writing it.

- Receiving does things in the opposite order.

=> This approach applies to TCP; in UDP explicit framing is not necessary.