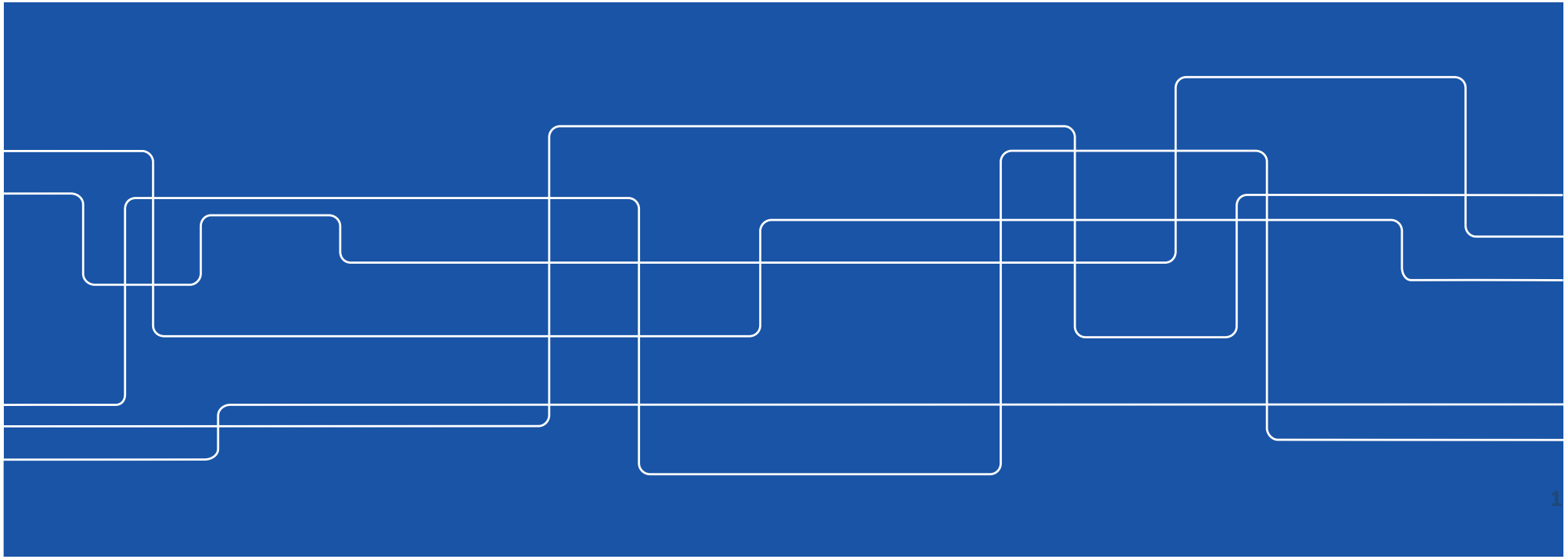
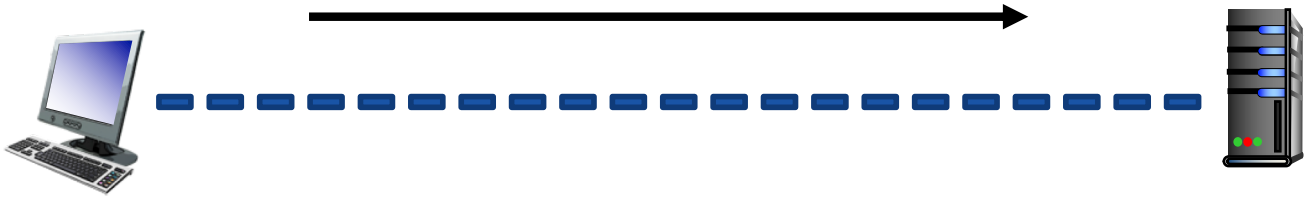
I/O, Encoding and Decoding

IK1203
Peter Sjödin



Byte Communication



- Networks transfers bytes between processes
  - For example, TCP is stream-based
    - Transfers a sequence of bytes (“stream”)
  - UDP is datagram-based
    - Transfers a block of bytes (“datagram”)
- It is up to the application to interpret the bytes as data

Java Streams

- In Java, streams (InputStream and OutputStream) are the basic classes for byte I/O

```
OutputStream()

void write(byte [] buffer)
void write(byte [] buffer, int offset, int length)
void write(int)
```

Write/send all bytes in buffer.

Write/send length bytes from buffer, starting at offset.

Write/send a single byte (0 – 255). Larger integers will be truncated.

Java Streams

- Reads data into an existing (pre-allocated) array
- Returns the number of bytes read
- Returns -1 at end of data (connection closed)

Read/receive bytes into buffer

```
InputStream()

int read(byte [] buffer)
int read(byte [] buffer, int offset, int length)
```

Read/receive at most length bytes into buffer, starting at offset.

Kurose & Ross' Client With Byte I/O

```
class TCPClient {
    private static int BUFFERSIZE=1024;

    public static void main(String argv[]) throws Exception
    {
        // Pre-allocate buffers for reading/receiving
        byte[] fromUserBuffer = new byte[BUFFERSIZE];
        byte[] fromServerBuffer = new byte[BUFFERSIZE];

        Socket clientSocket = new Socket("hostname", 6789);

        int fromUserLength = System.in.read(fromUserBuffer); // User input
        clientSocket.getOutputStream().write(fromUserBuffer, 0, fromUserLength);
        clientSocket.getOutputStream().write('\n');

        int fromServerLength = clientSocket.getInputStream().read(fromServerBuffer);
        System.out.print("FROM SERVER: "); // Use print method since it is a string
        System.out.write(fromServerBuffer, 0, fromServerLength);
        System.out.write('\n');
        clientSocket.close();
    }
}
```

Encoding/Decoding

- Programming languages have data types
  - Integers, floating numbers, strings, booleans, etc.
- How convert between bytes and data types?
- That is what encoding/decoding is about
- Example:
  - Transmit a string over the network
  - Save a floating number to file

Text Encoding/Decoding

- Translate between string symbols (such as ”A”, ”?”, ”d”, ”ö”, ”戏”) and bytes
- Define how each symbols is represented as one or more bytes

ASCII

- American Standard Code for Information Interchange
- 7 bits per symbol
  - In practice, one byte per symbol
  - 8th bit always zero
- ”A” is 65 (41 hex)
- ”z” is 122 (7A hex)
- ASCII 0 to 31 represent control characters
  - 7 (07 hex) is bell (”beep”)
  - 10 (0A hex) is line feed
  - 13 (0D hex) is carriage return

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

UTF-8

- Unicode Transformation Format – 8-bit
- Variable length encoding
- Up to four bytes per symbol
- The first 128 are the same as for ASCII
- Dominating format on the Web

There are Many Other Formats

- Families or series of encodings
- EBCDIC (IBM)
- ISO 8859
- MS-Windows
- Mac OS Roman
- Unicode
  - UTF-8, UTF-16
- …

Controlling Encoding/Decoding

- Many methods that convert between strings and bytes take encoding scheme (”Charset”) as a parameter
  - If unspecified, there is a default encoding
    - Typically UTF-8

```
String string = ”Fruit flies like a banana”;
// encode a string into a byte array
byte [] encodedBytes =
    string.getBytes(StandardCharsets.UTF_8);
// decode a byte array into a string
String decodedString =
    new String(encodedBytes, StandardCharsets.UTF_8);
```

Kurose & Ross' Server With Byte I/O

```
import java.nio.charset.StandardCharsets;
class TCPServer {
    static int BUFFERSIZE=1024;

    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;
        byte[] fromClientBuffer = new byte[BUFFERSIZE]; //pre-allocate buffer

        ServerSocket welcomeSocket = new ServerSocket(6789);

        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            int fromClientLength =
                connectionSocket.getInputStream().read(fromClientBuffer);

            clientSentence = new String(fromClientBuffer, 0,
                                        fromClientLength, StandardCharsets.UTF_8);
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            connectionSocket.getOutputStream().
                write(capitalizedSentence.getBytes(StandardCharsets.UTF_8));
        }
    }
```

Other Data Types

- Any object needs a encoding/decoding scheme in order to, for example, export/import it in a program
- Floating numbers
  - IEEE 754-2008
  - Single-precision (approx 7 decimal digits): 32 bits
  - Double-precision (approx 16 decimal digits): 64 bits
- Integer
  - 16, 32, 64 bits
  - Byte order – big-endian or little-endian?
- …

Remark About Code Examples

- Do not copy-paste this code!
  - Same goes for Kurose-Ross code examples
- Fundamental and important parts missing
- In particular, no error handling
  - Never make assumptions about what the other party does in communication!!!

```
int fromClientLength =
    connectionSocket.getInputStream().read(fromClientBuffer);
if (fromClientLength > 0) {
    capitalizedSentence = clientSentence.toUpperCase() + '\n';
    connectionSocket.getOutputStream().
        write(capitalizedSentence.getBytes(StandardCharsets.UTF_8));
}
else {
     // Handle closed connection
}
```