Chat

Client 1

Server

Client 2

text
text
SDChat Commands

"available"
"login"
"register"
"nickname"
"startconversation"
"quit"
"waitinglist"
"acceptconversation"
"message"
"rejectconnection"
"endconversation"
Command & Response Structure

identifier;key:value;key2:value2;; login;nickname:foo;password:foopass;;

identifier;; quit;;

identifier:value;; ok:quit;;

identifier:value;key:value;; ok:1;nickname:bar;
Metacharacters

character with special meaning to a computer program

SDChat metacharacters
   \ : ;
Metacharacters & values

What happens when a nickname contains metacharacter?

nickname = foo;password:cat;

How to parse:

login;nickname:foo;password:cat;;password:foopass;;
Metacharacters & values

Metacharacters in values must be escaped with "\"

login;nickname:foo\;password\:cat\;;password:foopass;;

Before sending command/response clients & server have to escape values
When reading from network clients need to unescape values
Metacharacters, Identifiers & keys

Identifiers & keys do not contain metacharacters
Connection (server) States

1. **Start**
   - nickname
   - register

2. **authenticated**
   - login
   - available
   - waitinglist
   - startconversation
   - rejectconversation
   - endconversation
   - acceptconversation

3. **Conversation**
   - message

4. **Conversation handshake**
   - quit
Sample Timeline

Client bar
- login
- ok:success;;
- available
- ok:success;;
- requestconversation:user:foo;;
- acceptconversation;;

Server

Client foo
- login
- ok:success;;
- waitinglist
- ok:1;nickname:bar;;
- startconversation:bar;;
- acceptconversation;;
- acceptconversation;;
Sample Timeline Continued

Client bar

- requestconversation:user:foo;;
- acceptconversation;;
- message; text: a; sender: foo; time: xx;;
- message; text: b;;
- endconversation;;

Server

- startconversation: bar;;
- acceptconversation;;
- message; text: a;;
- message; text: b; sender: bar; time: xx;;
- endconversation;;

Client foo
No newline in protocol

message end in ":;:" 

So readline will not work
Entire protocol is in text

The "1" is the string representation of the number one
Timestamp format

02/08/2010 20:13:37
Java Streams
InputStream & Bytes

Read just bytes

int available()
void close()
abstract int read()
int read(byte[] b)
int read(byte[] b, int off, int len)
long skip(long n)

void mark(int readlimit)
boolean markSupported()
void reset()

byte[] input = new byte[10];
for (int k = 0; k < input.length; k++) {
    int b = in.read();
    if (b == -1) break;
    input[k] = (byte) b;
}
read returns an int

casts to signed byte
  -128 to 127

Works fine if value is between 0 and 127

int shifted = b >= 0 ? b : 256 + b;

byte[] input = new byte[10];
for (int k = 0; k < input.length; k++) {
  int b = in.read();
  if (b == -1) break;
  input[k] = (byte) b;
}
Issue - Performance

Reading one byte at a time is slow

```java
int bytesRead = 0;
int bytesToRead=1024;
byte[ ] input = new byte[bytesToRead];
while (bytesRead < bytesToRead) {
    int readSize = in.read(input, bytesRead, bytesToRead - bytesRead);
    if (readSize = -1 ) break;
    bytesRead += readSize;
}
```
Issue - How far to read?

Normally don't know the size of a message

Some protocols allow multiple requests to be sent as same time
void mark(int readlimit)  
boolean markSupported()  
void reset()  

Most streams don't support mark  
Be careful
Peek (look ahead) is Useful

login;nickname:foo;password:foopass;;  When we read a ";:" are we
don done with the message
Just done with one segment

ok:1;nickname:bar;;  Don't know until we read next
c character
Would Be Nice

But you need "peek"

```java
class MyClass {
    public void method() {
        while (!atEndOfMessage(in)) {
            messageText += readUpto(";", in);
        }
    }
}
```

`atEndOfMessage(stream)`
- returns true if next character in stream is ";
- Does not remove characters from the stream

`readUpto(char, stream)`
- reads up through the next occurrence of character
How do we get peek, readUpto?

PushbackInputStream - helps for peek

Subclass FilterReader, FilterInputStream
Some Smalltalk ReadStream Methods

peek
upTo: aCharacter
upToAll: aCollection
through: aCharacter
throughAll: aCollection
next
next: anInteger
PrintStream

"PrintStream is evil and network programmers should avoid it like the plague!"

Elliotte Harold
Readers & Writers

Java's streams do not handle unicode.

If protocol uses unicode use readers and writers.
Java's Data Streams

Read/Write binary

Do not use if protocol is text based

If protocol is binary DataStreams format may not be correct
Parsing
Some low level Java Parsing

"cat;man;ran".split(";");

Returns an array of String [ “cat”, “man”, “ran”];
parts = new java.util.StringTokenizer("cat,man;ran;,fan", ",;"); 
while (parts.hasMoreElements())
{
    System.out.println(parts.nextToken());
}

Output

cat
man
ran
fan
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\s*fish\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();

Output
1
2
red
blue
Socket connection = new Socket(server, port);
InputStream rawIn = connection.getInputStream();
UpToReader in = new UpToReader(
    new InputStreamReader(rawIn));
String answer = in.upTo(';');
sdsu.io.ChunkReader

read = new sdsu.io.ChunkReader("catEOMmatEOM", "EOM")
while (read.hasMoreElements())
{
    System.out.println(read.readChunk());
}

Output

cat
mat
Subclass FilterInputStream

public class UpToInputStream extends FilterInputStream {
    public UpToInputStream(InputStream stream) {
        super(stream);
    }

    public byte[] upto(char end) throws IOException {
        int EOF = -1;
        ByteBuffer buffer = new ByteBuffer();
        int c;
        while ((c = super.read()) != EOF) {
            buffer.append((byte)c);
            if (c == end)
                break;
        }
        if (c == EOF & (buffer.isEmpty()))
            return new byte[0];
        return buffer.getBytes();
    }
}

Issue - What if User's text contains ";";

password = trou;;ble

login;nickname:whitney;password:trou\\;\\ble;;

text = duh;now what

message:duh\\;now what;;

You need to escape/unescape the ";;"

UpTo has to know about escaped characters
Relax

Clear your mind

Get ready for big idea
Why limit reading to characters?
Why not read Message Objects?

InputStream rawIn = connection.getInputStream();
SDChatReader in = new SDChatReader(rawIn);
Message answer = in.next();
SDChatMessage

Login

Nickname

Message
Message Responsibilities

Hide all message syntax

Read message and convert to object

    Message message =
    Message.from("message:duh\;now what;;");

Create message from values

    Message message = new Message("duh;now what");

Convert object to required protocol string

    message.toString()  // returns "message:duh\;now what;;"

Access information about message

    message.isLogin();
    message.name();
Client Side

Socket connection = new Socket(server, port);
OutputStream rawOut = connection.getOutputStream();
PrintWriter out = new PrintWriter(new BufferedOutputStream(rawOut));
InputStream rawIn = connection.getInputStream();

SDChatReader in = new SDChatReader(rawIn);
SDChatMessage login = new LoginMessage("whitney", "foo");
out.print(login.toString());
out.flush();

SDChatMessage result = in.next();
if (result.isError() ) then
    deal with error
else
    blah
Server Side

SDChatMessage request = in.next();
if (request.isLogin() ) {
    etc
}
else if (request.isTransmit() ) {
    etc
}
blah
Consequences

Main code operates at higher level

Isolates protocol syntax

Testing becomes easier

More Classes

Logic is spread across multiple classes
public void testAdd() {
    Message add = new Message("cat");
    assertTrue( add.toString() == "message:cat;";
}
public class DateServer {

    public void run(int port) throws IOException {
        ServerSocket input = new ServerSocket(port);

        while (true) {
            Socket client = input.accept();
            BufferedReader parsedInput =
                new BufferedReader(new InputStreamReader(client.getInputStream()));

            boolean autoflushOn = true;
            PrintWriter parsedOutput = new PrintWriter(client.getOutputStream());

            String inputLine = parsedInput.readLine();

            if (inputLine.startsWith("date")) {
                Date now = new Date();
                parsedOutput.println(now.toString());
                client.close();
            }
        }
    }
}
Testing DateServer

Must use network to test server

OK for date server, but not for more complex servers
Idea 1 - Keep Network Layer Thin

```java
public class DateServer {
    private static Logger log = Logger.getLogger("dateLogger");

    public void run(int port) throws IOException {
        ServerSocket input = new ServerSocket(port);

        while (true) {
            Socket client = input.accept();
            log.info("Request from " + client.getInetAddress());
            processRequest(client.getInputStream(), client.getOutputStream());
            client.close();
        }
    }

    void processRequest(InputStream in, OutputStream out) throws IOException {
        BufferedReader parsedInput = new BufferedReader(new InputStreamReader(in));
        boolean autoflushOn = true;
        PrintWriter parsedOutput = new PrintWriter(out, autoflushOn);
        etc.
    }
}
```
Idea 1 - Keep Network Layer Thin

public class TestDateServer {
    public void testDate() {
        InputStream in = new ByteArrayInputStream("date;".getBytes()));
        ByteArrayOutputStream fakeOut = new ByteArrayOutputStream();
        DateServer counter = new DateServer();
        counter.processRequestOn(in, fakeOut);
        assertTrue(fakeOut.toString() == "2006 02 14;")
    }
}

Idea 2 - Separate IO from Action

```java
class SDChatServer {
    boolean login(String name, String password) {
        code here
    }

    boolean transmit(String message) {
        code here
    }

    etc.
}

Now can test action without going through protocol strings
```
Scale Changes Everything

As a Server grows in complexity testing through socketsstreams is too hard
Idea 3 Fake it

Create a fake Socket class that
returns fixed output
records input

Build class from scratch or use Mock Objects

Ruby FlexMock
http://onestepback.org/software/flexmock/

Mock Object Home
http://www.mockobjects.com/
require 'flexmock'
require 'test/unit'

class TestExample < Test::Unit::TestCase
  def testShowMockObject()
    a = FlexMock.new
    a.should_receive(:foo).with(4).returns{|x| x + 1}
    a.should_receive(:foo).with(10).returns{'cat'}
    a.should_receive(:bar).returns{'dog'}
    assert( a.bar == 'dog')
    assert( a.foo(4) == 5)
    assert( a.foo(10) == 'cat')
    assert( a.foo(4) == 5)
    assert( a.bar == 'dog')
  end
end
Idea 4 - Run Client & Server in test case

require 'flexmock'
require 'test/unit'
require 'server'
require 'client'

class TestExample < Test::Unit::TestCase
  def setup()
    @server = Server.new(4444)
    @serverThread = Thread.new { @server.run }
  end

  def teardown()
    @serverThread.terminate
  end

  def testServer()
    client = Client.new("localhost", 4444)
    result = client.count("/foo")
    blah
  end
end

Look out for deadlock
Worry about scaling