CS 580 Client-Server Programming
Spring Semester, 2009
Doc 8 Client Review
19 Feb, 2009

Time-Date Client

```
import java.io.*;
import java.net.Socket;

class DateClient {
        String server;
        int port;

        public DateClient(String serverAddress, int port) {
                server = serverAddress;
                this.port = port;
        }

public String date() {
        return send("date\n");
}

public String time() {
        return send("time\n");
}
```

Client Continued

```
        private String send(String text) {
                try {
                        Socket connection = new Socket(server, port);
                        OutputStream rawOut = connection.getOutputStream();
                        PrintStream out = new PrintStream(new BufferedOutputStream(rawOut));
                        InputStream rawIn = connection.getInputStream();
                        BufferedReader in = new BufferedReader(new
InputStreamReader(rawIn));

                        out.print(text);
                        out.flush();
                        String answer = in.readLine();
                        out.close();
                        in.close();
                        return answer;
                }
                catch (IOException e) {
                        return "Error in connecting to server";
                }
        }
}
```

Issue - Client will not work on all platforms

String answer = in.readLine();

What ends a line?

Don't Do this

```
String answer = in.readLine();
```


I did it to keep the example small. One can not get much code on a slide using 24 point font. Plus the Ruby example is shorter than this.

Netcat

Swiss Army knife Network Utilty

Sends/receives TCP & UDP packets

Blog post

http://www.catonmat.net/blog/unix-utilities-netcat/

Man Page

http://linux.die.net/man/1/nc

Netcat home

http://netcat.sourceforge.net/

Netcat as Telnet

Al pro 11->nc bismarck.sdsu.edu 8010

I typed → **messages;block:1;;**

Server Response

ok:20;text:Unit test message:sender:james:time:02/19/2009 8\:41\:14;text:Unit testing my code \:):sender:bblaine:time:02/19/2009 0\:40\:31;text:Unit testing my code \:):sender:bblaine:time:02/19/2009 0\:40\:12;text:Unit testing my code \:):sender:bblaine:time:02/19/2009 0\:39\:02;text:Unit testing my code \:):sender:bblaine:time:02/19/2009 0\:38\:37;text:error\:ErrorString\;\;text\:Hello\:sender \:foo\:time\:02/03/2009 13\:29\:45\;I wonder how many\; people\;\; will\:have\:problems \;with\:thisok\:success\;\;:sender:whitney:time:02/18/2009 23\:10\:10;text:error \:ErrorString:sender:whitney:time:02/18/2009 23\:07\:38;text:did this fix a bug?:sender:james:time:02/18/2009 21\:01\:34;text:top chef is making me hungry:sender:james:time:02/18/2009 20\:49\:16;text:blah:sender:james:time:02/18/2009 20\:41\:39;text:woot:sender:james:time:02/18/2009 20\:35\:56;text:Hello \::sender:test:time:02/18/2009 17\:16\:29;text:Hello World:sender:test:time:02/18/2009 17\:16\:29;text:Hello \::sender:test:time:02/18/2009 17\:15\:34;text:Hello World:sender:test:time:02/18/2009 17\:15\:34;text:Hello \::sender:test:time:02/18/2009 17\:14\:17;text:Hello World:sender:test:time:02/18/2009 17\:14\:16;text:Hello World:sender:test:time:02/18/2009 17\:13\:57;text:Hello World:sender:test:time:02/18/2009 17\:12\:38;text:Hello 2:sender:test:time:02/18/2009 17\:09\:32;;

Netcat as Server

Al pro 12->netcat -l -p 12345

```
client := TwitterClient server: '127.0.0.1' port: 12345.
client newUser: 'foo'  password: 'b:ar' .
```

```
newUser;screenName:foo;password:b\:ar;;
```


Yes you can then type the server response

Three SDwitter Clients

Telnet/nc client

Date Server type client

A lot of work client

Metrics

Testability

Hiding protocol from user

Separation of domain logic from GUI layer

Reusability (in server)

Amount of work

Telnet/nc Client

```
public class TelnetLikeClient {
    private Inputstream fromServer;
    private OutputStream toServer;

    public TelnetLikeClient(String server, int port) {
        code to initialize things
    }

    public String send(String command) {
        toServer.write(command);
        toServer.flush();

        a loop to read the server response until ';;'
        return response;
    }
```

Telnet/nc Client

```
public static main(String[] args) {
    TelnetLikeClient client = new TelnetLikeClient("bismarck.sdsu.edu", 8010);
    while (true) {
        System.out.println("Enter a command");
        String command = System.in.readLine();
        String serverResponse = client.send(command);
        System.out.println( serverResponse);
    }
}
```

Metrics

Testability

Hiding protocol from user

Separation of domain logic from GUI layer

Reusability (in server)

Amount of work

DateTimeLikeClient

```
public class DateTimeLikeClient {
    private Inputstream fromServer;
    private OutputStream toServer;

    public DateTimeLikeClient(String server, int port) {
        code to initialize things
    }

    private String send(String command) {
        toServer.write(command);
        toServer.flush();

        a loop to read the server response until ';;'
        return response;
    }
```

DateTimeLikeClient

```
public string login(String name, String password) {
    String escapedName = escape(name);
    String escapedPassword = escape(password);
    String command = "login;screenName:" + escapedName +
                              ";password:" + escapedPassword + ";;";
    return send(command);
}

public string newUser(String name, String password) { etc.}
public string transmitMessage(String message) { etc. }

etc.
```

Metrics

Testability

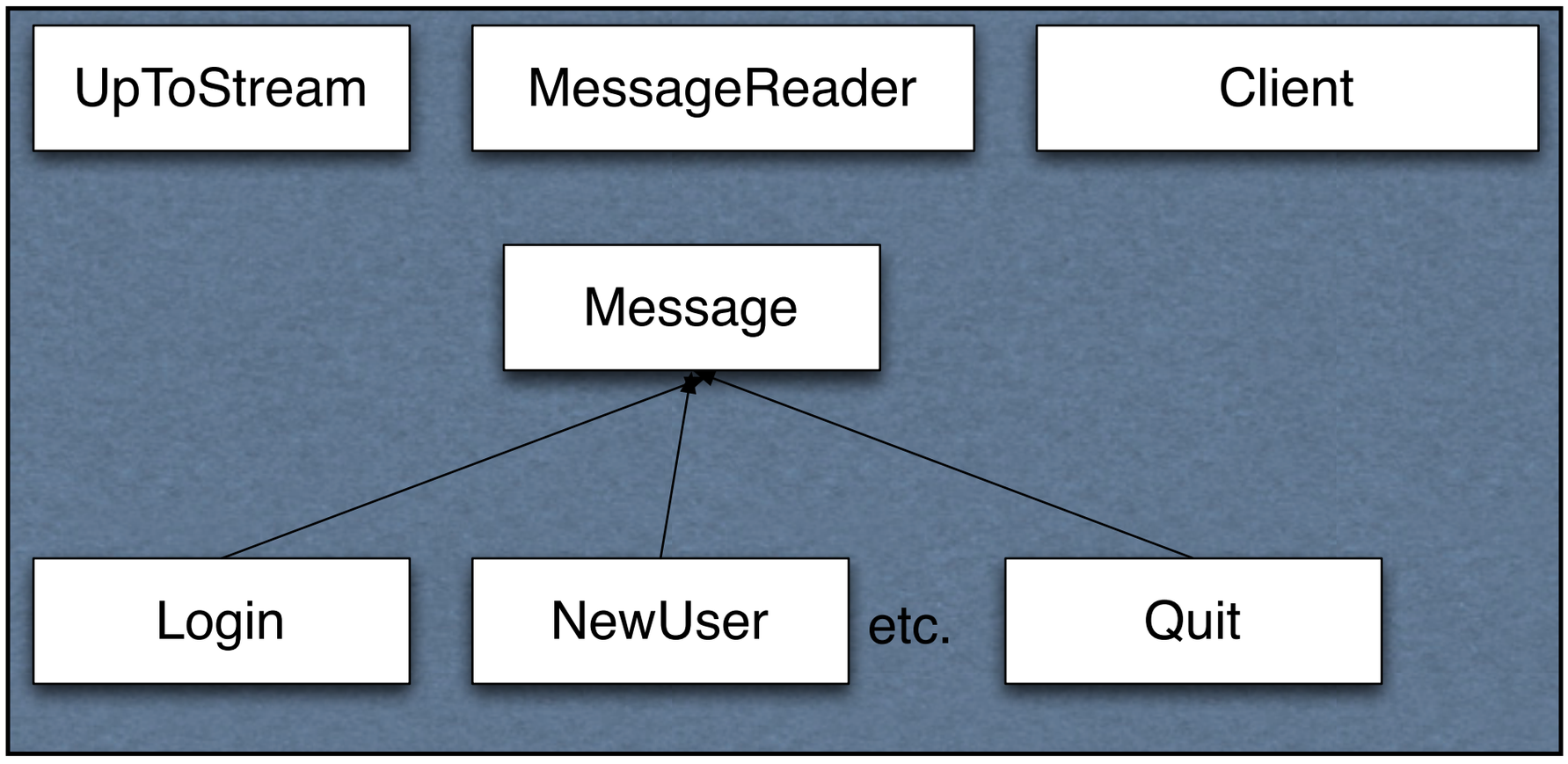
Hiding protocol from user

Separation of domain logic from GUI layer

Reusability (in server)

Amount of work

A lot of work client



A lot of work client

```
public class UpToStream extends InputStream  {
    public UpToStream(InputStream in)  { super(in); }

    public String upto(String end) throws IOException {
        loop to read chars until one gets to "end"
        return result;
    }

    etc.
}
```

A lot of work client

```
public class MessageReader
{
    UpToStream in;

        public MessageReader(InputStream input) {
            in = new UpToStream(input);
    }

    public Message next() {
        String messageString = in.upto(";;");
        return Message.from(messageString);
    }

etc.
```

A lot of work client

```
public class Login extends Message  {
    private String screenName;
    private String password;

    public Login(String screenName, String password) {
        this.screenName = screenName;
        this.password = password;
    }

    public String toString()  {
        return "login;screenName:" + escape(screenName) + ";password:" +
            escape(password) + ";;";
    }

    etc.
}
```

A lot of work client

```
public class LotOfWorkClient {

    public void login(String screenName, String password) throws IOException {
        send(new Login(screenName, password));
    }

    private void send(Message messageToSend) throws IOException {
        out.write(messageToSend.toString());
        out.flush();
    }

    etc.
```

Metrics

Testability

Hiding protocol from user

Separation of domain logic from GUI layer

Reusability (in server)

Amount of work

Which takes longer to implement

Will LotOfWorkClient implementer recoup the time?

How to Test a Client?

Use a lot of small pieces - can test pieces

How to test the pieces working together

Using automated tests of course

Some background

ByteArrayOutputStream

output stream in which the data is written into a byte array
data can be retrieved using toByteArray() and toString()

```
test = new ByteArrayOutputStream();
test.write("cat".getBytes(), 0 , 3);
test.toString();
```

Some background

ByteArrayInputStream

contains an internal buffer that contains bytes that may be read from the stream

```
ByteArrayInputStream test = new ByteArrayInputStream("dog".getBytes());
int c = test.read();
System.out.println((char) c);
```

Base Client

```
public class DateTimeLikeClient {
    private Inputstream fromServer;
    private OutputStream toServer;

    public DateTimeLikeClient(String server, int port) {
        code to initialize things
    }

    private String send(String command) {
        toServer.write(command);
        toServer.flush();
        a loop to read the server response until ';;'
        return response;
    }

    public string login(String name, String password) {etc}
    public string newUser(String name, String password) { etc.}
    public string transmitMessage(String message) { etc. }
```

TestClient

```
public class TestClient extends DateTimeLikeClient {

    public TestClient(String text ) {
        super();
        fromServer = new ByteArrayInputStream(text.getBytes()));
        toServer = new ByteArrayOutputStream();
    }

    public String clientRequest() {
        return toServer.toString();
    }
}
```

Sample Test

```
public SampleTest extends TestCase {

    public testClientLoginString() {
        TestClient client = new TestClient("ok:success;;");
        client.login("foo", "b:ar");
        assertTrue( client. clientRequest() == "login;screenName:foo;password:b\:ar;;");
    }
```