CS 580 Client-Server Programming
Spring Semester, 2007
Doc 7 Some Parsing & Testing
Feb 15, 2007

```
Socket connection = new Socket(server, port);
InputStream rawIn =
connection.getInputStream();
BufferedReader in = new BufferedReader(
        new InputStreamReader(rawIn));
String answer = in.readLine();
```

Now parse answer

Some low level Java Parsing

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

Returns an array of String [“cat”, “man”, “ran”];

StringTokenizer

```
parts = new java.util.StringTokenizer("cat,man;ran;,fan", ",;");
while (parts.hasMoreElements())
	{
	System.out.println( parts.nextToken());
	}
```

Output

```
cat
man
ran
fan
```

java.util.Scanner

```
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
```

Ruby Streams

```
def send(text)
    connection = TCPSocket.new(@server, @port)
    connection.print(text)
    connection.flush
    answer = connection.gets("\n")
    connection.close
    answer
  end
```

Java UpToReader?

```
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

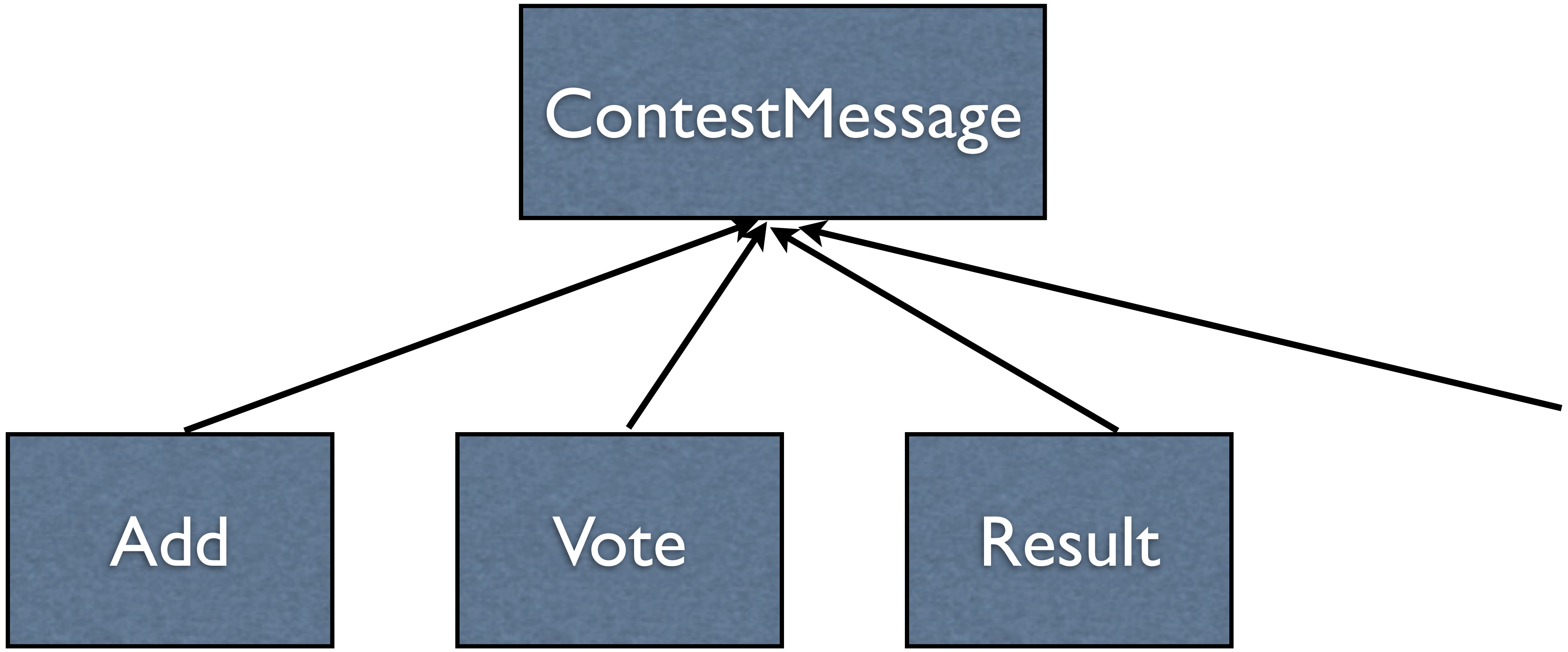
```
public class UpToInputStream extends FilterInputStream {
    public UpToInputStream(InputStream stream)
        { super(stream); }

    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();
        }
    }
```

Why not read Message Objects?

```
InputStream rawIn = connection.getInputStream();
ContestMessageReader in = new ContestMessageReader(rawIn);
Message answer = in.next();
```


ContestMessage
Add
Vote
Result

Client Side

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

ContestMessageReader in = new ContestMessageReader(rawIn);
AddMessage add = new AddMessage(“whitney”);
out.print(add.toString());
out.flush();

ContestMessage result = in.next();
if (result.isError() ) then
    deal with error
else
    blah
```

Server Side

```
ContestMessage request = in.next();
if (request.isAdd() ) {
      AddMessage add = (AddMessage) request;
      String userName = add.name();
      etc
}
else if (request.isvote() ) {
      etc
}
 blah
```

Message Responsibilities

Hide all message syntax

Read message and convert to object

```
VoteMessage message =  VoteMessage.from(“vote cat;”);
```

Create message from values

```
VoteMessage vote = new VoteMessage();
vote.for("dog");
```

Access information about message

```
message.isLogin();
message.name();
```

Consequences

Main code operates at higher level

Isolates protocol syntax

Testing becomes easier

More Classes

Logic is spread across multiple classes

Testing

Can test more parts without using network

```
public void testAdd() {
	AddMessage add = new AddMessage("cat");
	assertTrue( add.toString() == "add cat;";
	AddMessage fromString = new AddMessage.from(add.toString());
	assertTrue( fromString.name() == "cat");
}
```

Testing Servers

```
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

```
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

```
class VoteServer {
    boolean add(String name) {
        code to add the name
        return true if added successfully
    }

    boolean voteFor(String name) {
        code to vote for name
        return true if successfull
    }

    etc.
```

Now can test action without

going through protocol strings

Scale Changes Everything

As a Server grows in complexity testing through sockets/streams 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/

Example of Mock Object

```
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

Some OOP & Server Architecture

Those Pesky If Statements

```
class VoteServer {
    boolean add(String name) { blah }

    boolean voteFor(String name) { blah }

    void proccessRequest( blah ) {
        blah
        ContestMessage request = in.next();
        if (request.isAdd() ) then
            foo = add( blah);
            blah
        else if (request.isList() )
            bar = list():
            blah
        else if etc.
```

Use Double Dispatch

```
void proccessRequest( blah ) {
        blah
        ContestMessage request = in.next();
        ContestMessage result = request.executeUsing(self);
        now send back result
}


class AddMessage {
    public ContestMessage executeUsing(Server x) {
        boolean added = x.add(name);
        if (added)
            foo
        else
            bar
```