CS 580 Client-Server Programming
Spring Semester, 2006
Comments on Assignment 2 part 1
Feb 14, 2006

Names

```
class myClient {
        public String[] Count(String url)
blab

class myServer {
        void processRequest(InputStream in, OutputStream out) {
                int num = 0;                    // num occurrences of space ' '
                while (true) {
                        int c = in.read();
                        if (c == ' ' ) {
                                num++
                        }
```

```
class Server {
        void processRequest(InputStream in, OutputStream out) {
                int spaceCount = 0;
                while (true) {
                        int c = in.read();
                        if (c == ' ' ) {
                                spaceCount++
                        }
```

Inheritance Abused

public class Server extends Client { blah }

A is a B => A is a subclass of B

A has a B => A has a field of type B

Use Returns to Simplify Code

```
private ResultObj processCommand(String cmd) {
        if (null== cmd)
                return null;
        ResultObj ro = null;
        if (cmd.startsWith("count") && cmd.endsWith(" ;")) {
                ro = count(cmd.substring(7);
        } else if (cmd.startsWith("reset") && cmd.endsWith(" ;")) {
                ro = reset(cmd.substring(7);
        } else {
                ro = null;
        }
        return ro;
}
```

```
private ResultObj processCommand(String cmd) {
        if (null== cmd)
                return null;
        if (isCountCommand(cmd)) {
                return count(cmd.substring(7);
        }
        if (isResetCommand(cmd)) {
                return reset(cmd.substring(7);
        }
        return null;
}
```

Avoid nested If statements

```
if (st.hasMoreTokens() ) {
        tempString = st.nextToken();
        if (tempString).compareTo("count") == 0) {
                if (st.hasMoreTokens() ) {
                        tempHostName = st.nextToken();
                        if (st.hasMoreTokens() ) {
                                tempString = st.nextToken();
                                if (tempString).compareTo(";") == 0) {
                                        UrlCount urlDB = (UrlCount) hosts.get(tempHostName);
                                        if (urlDB != null {
                                                blah
                                                blah
                                        } else {
                                                hosts.put(tempHostName, new UrlCount(tempHostName));
                                                tempString = "1 " + tempHostName + " ;";
                                        }
                                        out = new PrintWriter(connection.getOutputStream());
                                        out.println(tempString);
                                        out.flush();
                        }
} else {
        out = new PrintWriter(connection.getOutputStream());
        out.println("Invalid command");
        out.flush();
```

Error prone
Hard to read
Hard to maintain

Some Refactoring

```
if (!st.hasMoreTokens() ) {
        return "Invalid command";
}
command = st.nextToken();
if (!st.hasMoreTokens() ) {
        return "Invalid command";
}
url = st.nextToken();
if (!st.hasMoreTokens() ) {
        return "Invalid command";
}
if (st.nextToken().compareTo(";") != 0) {
        return "Invalid command";
}
UrlCount urlDB = (UrlCount) hosts.get(tempHostName);
if (urlDB != null {
        blah
        blah
        } else {
                hosts.put(tempHostName, new UrlCount(tempHostName));
                return  "1 " + tempHostName + " ;";
        }
```

More Refactoring

```
if (requestParts.countTokens() != 3 ) {
        return "Invalid command";
}
command = requestParts.nextToken();
url = requestParts.nextToken();
terminator = requestParts.nextToken();
if (terminator.compareTo(";") != 0) {
        return "Invalid command";
}
UrlCount urlDB = (UrlCount) hosts.get(tempHostName);
if (urlDB != null {
        blah
        blah
        } else {
                hosts.put(tempHostName, new UrlCount(tempHostName));
                return  "1 " + tempHostName + " ;";
        }
```

Duh

```
StringTokenizer st = new StringTokenizer(blah);  //create a tokenizer


int port;            //port number

// Default constructor
public Client() { }

//Declare variables
StringBuffer request = new StringBuffer();
Socket connection = new Socket(server.port);
```

Keep Comments brief

```
/**
 * Constructor takes IP address and port number of the server
 * @param addr the IP address of the server
 * @param port the port number of the server
 */
public Client(String addr, int port) { blah}
```

```
/**
 * @param address the IP address of the server
 * @param port the port number of the server
 */
public Client(String address, int port) { blah}
```

```
public Client(String serverIPAddres, int serverPort) { blah}
```

Comment result not how the method works

```
// Count method sends a string command to the server to increment the count
// of the url by 1
public String count(String url) {
        return send ("count " + url + " ;");
}
```

```
// Returns number of times url has been accessed on server
public String count(String url) {
        return send ("count " + url + " ;");
}
```

Do it once

```
public class Client {
        public Client() {
                host = "bismarck.sdsu.edu";
        }

        public Client(String hostname) {
                host = hostname;
        }
        blah
```

```
public class Client {
        public Client() {
                this("bismarck.sdsu.edu");
        }

        public Client(String hostname) {
                host = hostname;
        }
        blah
```

Constants

```
while ( (c = in.read()) != 59)
        instr.append( (char) c);
```

```
while ( (c = in.read()) != SEMICOLON)
        buffer.append( (char) c);
```

```
while ( (c = in.read()) != COMMAND_TERMINATOR)
        buffer.append( (char) c);
```

```
which = parse_request(request)
case which
        when 1
                blah
        when 2
                blah
        else
                blah
end
```

```
which = parse_request(request)
case which
        when COUNT
                blah
        when RESET
                blah
        else
                blah
end
```

Issues?

```
public class Client {
        int port;
        String serverAddress;
        Socket socket;

        public void connect() {
                socket = new Socket(serverAddress, port);
        }

        public String count(String url ) {
                return send("count " + url + " ;");
        }

        private String send(String text) {
                OutputStream rawOut = socket.getOutputStream();
                blah
        }
        public void close() {
                socket.close();
        }
```

```
Client urlCounter = new Client(blah, 4444);
urlCounter.connect();
urlCounter.count("/foo/bar");
urlCounter.close();

urlCounter.connect();
urlCounter.count("/foo/bar");
urlCounter.close();
```

Do it Once, Don't force users to know internal workings

```
public class Client {
        int port;
        String serverAddress;
        Socket socket;

        private void connect() {
                socket = new Socket(serverAddress, port);
        }

        public String count(String url ) {
                connect();
                String result =  send("count " + url + " ;");
                close();
                return result;
        }

        private String send(String text) {
                OutputStream rawOut = socket.getOutputStream();
                blah
        }
        private void close() {
                socket.close();
        }
```

```
Client urlCounter = new Client(blah, 4444);
urlCounter.count("/foo/bar");

urlCounter.count("/foo/bar");
```

State verses method arguments & local variables

```
public class Client {
        int port;
        String serverAddress;
        Socket socket;
        OutputStream rawOut;

        private void connect() {
                socket = new Socket(serverAddress, port);
        }

        public String count(String url ) {
                connect();
                String result =  send("count " + url + " ;");
                close();
                return result;
        }

        private String send(String text) {
                rawOut = socket.getOutputStream();
                PrintStream out = new PrintStream(rawOut);
        }
        private void close() {
                socket.close();
        }
```

Removing improper state

```
public class Client {
        int port;
        String serverAddress;

        private void connect() {
                return new Socket(serverAddress, port);
        }

        public String count(String url ) {
        Socket serverConnection = connect();
        String result =  send("count " + url + " ;", serverConnection);
        serverConnection.close();
        return result;
}

private String send(String text, Socket socket) {
        OutputStream rawOut = socket.getOutputStream();
        PrintStream out = new PrintStream(rawOut);
}
```

Fields are global variables!

Do it Once

```
public class Client {
        int port;
        String serverAddress;

        public String count(String url ) {
                Socket serverConnection = new Socket(serverAddress, port);
                String result =  send("count " + url + " ;", serverConnection);
                serverConnection.close();
                return result;
        }

        public String reset(String url ) {
                Socket serverConnection = new Socket(serverAddress, port);
                String result =  send("reset " + url + " ;", serverConnection);
                serverConnection.close();
                return result;
        }

        private String send(String text, Socket socket) {
                OutputStream rawOut = socket.getOutputStream();
                PrintStream out = new PrintStream(rawOut);
        }
```

Just do it once

```
public class Client {
        int port;
        String serverAddress;

        public String count(String url ) {
                return send("count " + url + " ;", serverConnection);
        }

public String reset(String url ) {
        return send("count " + url + " ;", serverConnection);
}

private String send(String text) {
        Socket serverConnection = new Socket(serverAddress, port);
        OutputStream rawOut = serverConnection.getOutputStream();
        PrintStream out = new PrintStream(rawOut);
        blah
}
```

What is System.out good for?

```
public class Server {

        public void run() {
                blah
                try {
                        while (true) {
                                Socket clientConnection = socket.accept();
                                blah
                        }
                } catch (IOException e) {
                        System.out.println(e);
                } catch (Exception e) {
                        System.out.println(e);
                }
                blah
```

Not the Protocol - will not work with other clients/servers

```
public class Client {
        private String send(String text) {
                blah

                out.print(text);
                out.flush();
                String answer = in.readLine();
                blah;
}
```

```
public class Server
        void processRequest(InputStream in,OutputStream out)  {
                blah

                String inputLine = parsedInput.readLine();
                blah
                parsedOutput.println(result.toString());
                }
        }
}
```

Why Does this work?

```
public class Client {
        private String send(String text) {
                blah

                out.print(text);
                out.flush();
                int c;
                while (( c = in.read() ) != -1 ) {
                        buffer.append( (char) c);
                }
                blah;
}
```

When will this Fail?

```
public class Server
        void processRequest(InputStream in,OutputStream out)  {
                blah

                char[ ] buffer = new char[4096];
                try {
                        in.read(buffer, 0, 4096);
                } catch { IOException e) {
                        blah
                }
                blah
        }
}
```

Do it once and only once

```
public class Client {
        private String send(String text) {
                code to send request
                // now read response
                int c;
                while (( c = in. read()) != -1 ) {
                        buffer.append((char) c);
                        if (ch == ';' )
                                break;
                }
                blah
```

```
public class Server {
        private void processRequest(blah) {
                blah
                int c;
                while (( c = in. read()) != -1 ) {
                        buffer.append((char) c);
                        if (ch == ';' )
                                break;
                }
                blah
```

Put Functionality in Stream

```
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();
        }

        public byte[] upTo(String end) throws IOException {
                //A bit more complex
        }
```

Only Once?

```
public class Client {
        private String send(String text) {
                code to send request
                // now read response
                UpToInputStream in = new UpToInputStream(blah);
                request = in.upTo(" ;");
                }
                blah
```

```
public class Server {
        private void processRequest(blah) {
                blah
                UpToInputStream in = new UpToInputStream(blah);
                request = in.upTo(" ;");
                }
                blah
```

Testing the Server Idea 1

Keep network layer thin
Test below network layer

```
public class TestJavaServer {
    public void testOneCount() {
        InputStream in = new ByteArrayInputStream("count /foo ;".getBytes()));
        ByteArrayOutputStream fakeOut = new ByteArrayOutputStream();
        Server counter = new Server(4444);
        counter.processRequestOn(in, fakeOut);
        assertTrue(fakeOut.toString() == "1 2006 02 14")
    }
}
```

Testing the Server Idea 1 - Ruby

```
require 'test/unit'
require 'DateServer'

class MockSocket
  def gets(aString)
    "date\n"
  end

  def send(aString, anInteger)
    @sent = aString
  end

  def sent()
    @sent
  end
end

class TestExample < Test::Unit::TestCase
  def testDateServer
    server = DateServer.new(1111)
    fake_data = MockSocket.new
    server.process_request_on(fake_data)
    assert fake_data.sent==(Time.now.strftime("%x") + "\n")
  end
end
```

Mock Object

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

Mock Object Home
http://www.mockobjects.com/FrontPage.html

```
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
```

Flex Mock & Server

```
require 'flexmock'
require 'test/unit'

class TestExample < Test::Unit::TestCase
  def testShowMockObject()
    socket = FlexMock.new
    socket.should_receive(:gets).with(" ;").returns{"count /foo ;"}.ordered
    socket.should_receive(:send).with("1 2006 02 14 ;").ordered
    server = Server.new(4444)
    server.process_request_on(socket)
  end
end
```

Test fails if send does not have correct argument

Should regular expression or compute date rather hardcode value

Test Network Layer with Mock Object

```
class Server
  def initialize(port)
    @port = port
  end

  def run()
    server = serverSocket()
    puts("start " + @port.to_s)
    while (continueServer?)
      session = server.accept
      Thread.new(session) do |connection|
        process_request_on(connection)
        connection.close
      end
    end
  end
end
```

```
private
  def continueServer?()
    true
  end

  def serverSocket()
    TCPServer.new( @port)
  end

  def process_request_on(socket)
    blah
  end
end
```

Subclass with Mock Sockets

```
class TestServer < Server
  def intialize(port)
    super(port)
    @connectionCount = 0
  end

  def continueServer?()
    @connectionCount += 1
    @connectionCount < 2
  end

  def serverSocket()
    clientSocket = FlexMock.new
    clientSocket.should_receive(:gets).with(" ;").returns{"count /foo ;"}.ordered
    clientSocket.should_receive(:send).with("1 2006 02 14 ;").ordered
    serverSocket = FlexMock.new
    serverSocket.should_receive(:accept).returns{clientSocket}
    serverSocket
  end
end
```

Test Network Layer with Client and Server - Use threads

```
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
```

Put non-network code in separate class

```
class UrlCounter
  def initialize
    @urls = { }
  end

  def count(url)
    code to add url to hash table
    return correct count
  end

  def reset(url)
    urls.delete(url)
  end
end
```

Easy to test this code