CS 580 Client-Server Programming
Spring Semester, 2010
Doc 10 Comments on Assignment 3 part 1
2 March, 2010

Testing Exceptions

```
@Test
  public void testChatLogin4() {
      System.out.println("chatLogin");
      String nicknameIN = "foobar";
            String passwordIN = "sdchat";
      boolean expResult = true;
      boolean exceptionOccurred = false;
      SDChatCommon instance = new SDChatCommon();
      try {
          instance.chatLogin(nicknameIN, passwordIN);
      }
      catch (Exception e) {
          exceptionOccurred = true;
      }
      assertEquals(expResult, exceptionOccurred);
  }
```

Shorter Version

```
@Test(expected = Exception.class)
public void testChatLogin4() throws Exception {
String nicknameIN = "foobar";
        String passwordIN = "sdchat";
        SDChatCommon instance = new SDChatCommon();
        instance.chatLogin(nicknameIN, passwordIN);
}
```

Name

```
@Test(expected = Exception.class)
public void testChatLoginNonValidUser() throws Exception {
String nicknameIN = "foobar";
        String passwordIN = "sdchat";
        SDChatCommon instance = new SDChatCommon();
        instance.chatLogin(nicknameIN, passwordIN);
}
```

Client or server Test

```
@Test(expected = Exception.class)
public void testChatLoginEmptyPassword() throws Exception {
String nicknameIN = "foobar";
        String passwordIN = "";
        SDChatCommon instance = new SDChatCommon();
        instance.chatLogin(nicknameIN, passwordIN);
}
```

GUI & Domain Logic

```
public void chatLogin(String nicknameIN, String passwordIN) throws Exception {
        if ( currentState == 0 ) {
            StringBuffer loginLine = new StringBuffer("login");
            if ( nicknameIN == null || nicknameIN.trim().length() == 0 )
                throw new Exception("Error: Nickname cannot be blank!");
            if ( passwordIN == null || passwordIN.trim().length() == 0 )
                throw new Exception("Error: Password cannot be blank!");
            loginLine.append(";nickname:"+escapeString(nicknameIN));
            loginLine.append(";password:"+escapeString(passwordIN));
            loginLine.append(";;");
            String response = sendAndReceive(loginLine.toString());
            if ( response.startsWith("error:") )
                throw new Exception(parseError(response));
            else
                currentState++;
        }
}
```

Modified

```
public boolean chatLogin(String nicknameIN, String passwordIN)  {
        if ( currentState == START ) {
            StringBuffer loginLine = new StringBuffer("login");
            loginLine.append(";nickname:"+escapeString(nicknameIN));
            loginLine.append(";password:"+escapeString(passwordIN));
            loginLine.append(";;");
            String response = sendAndReceive(loginLine.toString());
            if ( response.startsWith("error:") )
                return false;
            currentState = AUTHENTICATED;
            return true
        }
    }
```

Menu Systems

```
        SDChatCommon testChat = new SDChatCommon();
        String textEntered = ""; // Line read from standard in

        System.out.println("You are in the "+testChat.getState()+" state.  Available commands are
"+testChat.getAvailableCommands()+".");
        System.out.print("SDChat client (type 'quit' to exit): ");
        InputStreamReader converter = new InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(converter);

        while ( !textEntered.equals("quit") ) {
            textEntered = in.readLine();
```

```
if ( !textEntered.equals("quit") ) {
    textEntered = textEntered.trim();

    if ( textEntered.length() > 0 ) {
        if ( testChat.isValidCommand(textEntered) ) {
            // Test for available
            if ( textEntered.compareTo("available") == 0 ) {
                if ( !testChat.chatAvailable() )
                    System.err.println("Error setting available status");
            }

            // Test for login
```

Are the comments Right

```
while (true) {
        char current = (char) input.read();
        if (current == '\\') {
            // its escaped so skip the next char whatever is next
            responseBuffer.append(current);
            responseBuffer.append((char) input.read());
        }
        if (current == ';') {
            responseBuffer.append(current);
            current = (char) input.read();
            if (current == ';') {
                // we've hit two semi colons in a row so stop after this
                responseBuffer.append(current);
                break;
            }
    }
```

In SDChatMessage Class

```
public void displayResponse() {
        System.out.println(getResponse());
    }
```

Using NicknameMessage Class

```
public boolean doNickname(String nickname) {
        if (state != State.Start)
            throw new IllegalStateException("Command not allowed in state: "
                    + state);

        if (connection == null) {
            throw new RuntimeException("There is no server connection");
        }

        String escapedNickname = ClientUtil.escape(nickname);
        String outMessage = "nickname;nickname:" + escapedNickname + ";;";
        String response = connection.sendCommand(outMessage);
        NicknameMessage nicknameParser = new NicknameMessage();
        nicknameParser.parseServerMessage(response);
        return nicknameParser.isNicknameIsAvailable();
}
```

Using NicknameMessage Class

```
public boolean doNickname(String nickname) {
        if (state != State.Start)
            throw new IllegalStateException("Command not allowed in state: "
                    + state);

        if (connection == null) {
            throw new RuntimeException("There is no server connection");
        }

        NicknameMessage command = new NicknameMessage(nickname);
        String response = connection.sendCommand(command.toString());
        command.serverResponse(response);
        return command.isNicknameIsAvailable();
}
```

Client Workflow

Why let the user ask for nickname in the wrong state

```
public boolean doNickname(String nickname) {
        NicknameMessage command = new NicknameMessage(nickname);
        String response = connection.sendCommand(command.toString());
        command.serverResponse(response);
        return command.isNicknameIsAvailable();
    }
```

How is this a test?

```
	public void testParsing() {
		String message =
"login;nickname:foo;nickname:foo1;nickname:foo2;password:foo;;";
		Scanner scanner = new Scanner(message).useDelimiter("nickname:");
		scanner.next();// skip the first one
		while (scanner.hasNext())
			System.out.println("user: " + scanner.next());
		scanner = new Scanner(message).useDelimiter("password:");
		scanner.next(); // skip the first one
		while (scanner.hasNext())
			System.out.println("pw: " + scanner.next());
	}
```

Now it is a test

```
    @Test
    public void testParsing() {
        String message =
"login;nickname:foo;nickname:foo1;nickname:foo2;password:foo;;";
        Scanner scanner = new Scanner(message).useDelimiter("nickname:");
        assertEquals("login;", scanner.next());
        assertTrue(scanner.hasNext());
        assertEquals("foo;", scanner.next());
        assertTrue(scanner.hasNext());
        assertEquals("foo1;", scanner.next());
        assertTrue(scanner.hasNext());
        assertEquals("foo2;password:foo;;", scanner.next());
        assertFalse(scanner.hasNext());

        scanner = new Scanner(message).useDelimiter("password:");
        scanner.next(); // skip the first one
        assertTrue(scanner.hasNext());
        assertEquals("foo;;", scanner.next());
        assertFalse(scanner.hasNext());
    }
```

Issues

```
private boolean sendBasicCommand(SDChatCommand command) {
        boolean success = false;
        System.out.println("Command: " + command.getCommand());
        out.print(command.getCommand());
        out.flush();

        List<SDChatMessage> message = in.next();

        for(int i = 0; i < message.size(); i++){
                System.out.println("Message: " + message.get(i).getName() + ":" +
message.get(i).getValue());
        }

        if (message.get(0).getValue().equalsIgnoreCase("success")) {
                success = true;
        } else if (message.get(0).getName().equalsIgnoreCase("ok")) {
                success = true;
        } else {
                System.err.println("Error: " + message.get(0).getValue());
        }

        return success;
}
```

Issues

```
private boolean sendBasicCommand(SDChatCommand command) {
        out.print(command.getCommand());
        out.flush();

        List<SDChatMessage> message = in.next();

        for(int i = 0; i < message.size(); i++){
                System.out.println("Message: " + message.get(i).getName() + ":" +
message.get(i).getValue());
        }

        if (message.get(0).getValue().equalsIgnoreCase("success")) {
                return true;
        } else if (message.get(0).getName().equalsIgnoreCase("ok")) {
                return true;
        } else {
                System.err.println("Error: " + message.get(0).getValue());
        }

        return false;
}
```

command verses basic command?

```
private List<SDChatMessage> sendCommand(SDChatCommand command) {
        out.print(command.getCommand());
        out.flush();

        List<SDChatMessage> message = in.next();

        return message;
}
```

Reader, Write and Argument

```
public SDChatReader(OutputStream outputStream, InputStream inputStream,
        SDChatMessage message) {
    this.outputStream = outputStream;
    this.inputStream = inputStream;
    this.message = message;
}

public void write() {
    try {
        Writer out = new BufferedWriter(new OutputStreamWriter(
                outputStream, CHAR_ENCODING));
        out.write(message.toString());
        out.flush();
    } catch (UnsupportedEncodingException e) {
        throw new SdChatClientException(e);
    } catch (IOException e) {
        throw new SdChatClientException(e);
    }
}
```

Changed Argument

```
public SDChatReader(OutputStream outputStream, InputStream inputStream ) {
        this.outputStream = outputStream;
        this.inputStream = inputStream;
}

public void write(SDChatMessage message) {
        try {
                Writer out = new BufferedWriter(new OutputStreamWriter(
                                outputStream, CHAR_ENCODING));
                out.write(message.toString());
                out.flush();
        } catch (UnsupportedEncodingException e) {
                throw new SdChatClientException(e);
        } catch (IOException e) {
                throw new SdChatClientException(e);
        }
}
```

Issues

```
public ArrayList<String> getWaitingList() {
      ArrayList<String> waitingList = new ArrayList<String>();
      try {
            messageOutputStream.write( new WaitingListMessage() );
            String response = messageInputStream.read();

            if( response.startsWith( "ok:N;" ) ) {
                  String[] temp = response.split(";");

                  for( int i = 1; i < temp.length; i++ ) {
                        if( temp[i].contains( ":" ) ) {
                              String[] temp2 = temp[i].split(":");
                              waitingList.add(temp2[1]);
                        }
                  }
      }
} catch( IOException e ) { }
```

In SDChatMenu Class

```
private void available() {
    HelperClass helper = new HelperClass();
   String tempString = null;
   tempString=helper.formRequest("available");
         if (protoLayer.sendRequest(tempString)) {
                  if ((tempString=protoLayer.getResponse()).equals("ok")){
                  menuChoice='w';
                  menuFlag=true;
                  return;
                  } else {
                       helper.printString("Connection " + tempString + ":");
                  tempString= protoLayer.getResponseData()[0];
                  helper.printlnString(tempString);
                  menuChoice='m';
                  menuFlag=true;
                     return;
                  }
            }
}
```

Formating

```
                    }
                }
}
private void nicknameCheck()
{
    HelperClass helper = new HelperClass();
    String tempString = null;
```

Issues

```
public SDChatStreamLayer() {
        // TODO Auto-generated constructor stub
        try {
                helper      = new HelperClass();
                readBuffer= new byte[BUFFERSIZE];
                lastResponse = new StringBuffer();
                lastRequest  = new String();
                chatServerPort = PresentationLayer.SERVERPORT;
                chatServerIp= PresentationLayer.SERVERIP;
                if (Init())
                {
                streamReader= cSocket.getInputStream();
            streamWriter= cSocket.getOutputStream();
                }
                else
                {

                }
                }
        catch(IOException e) {
                helper.printException(e);
        }
}
```

Duh Comments

```
//Default constructor
public SDChatProtocolLayer() {
        // Server IP & Port taken from Presentation Layer.
        chatServerIp= PresentationLayer.SERVERIP;
        chatServerPort=PresentationLayer.SERVERPORT;
        chatClientState = ProtocolLayerState.start;
        streamLayer= new SDChatStreamLayer(chatServerIp,chatServerPort);
        }

//State of the ProtocolLayer
public String getState() {

        return chatClientState.toString();
        }
```

Comments & Names

```
public String toString() {
        String out = "";
        for (MessageNode thisNode : messages)
                // Put each node together
                out = out.concat(thisNode.toString());
        // Add last semicolon (denote end of command)
        out = out.concat(";");
        return out;
}
```

```
public String toString() {
        String out = "";
        for (MessageNode thisNode : messages)
                // Put each node together
                out = out.concat(thisNode.toString());
        out = out.concat(COMMAND_END);
        return out;
}
```

Comments & Names

```
public class Message {
	List<MessageNode> messages;

	public Message() throws IOException {
		this.messages = new ArrayList<MessageNode>();
	}
```

```
public class Message {
	List<MessageKeyValue> messageParts;

	public Message()  {
		this.messages = new ArrayList<MessageKeyValue>();
	}
```

Make it Complete

```
/* You need to include nonstandard libraries with your code
 import org.jmock.Mockery;
 import org.jmock.Expectations;
 */
```

Names

```
class fakeSocket extends Socket {
    String message;

    //NOTE: Fake output stream to throw away writes.
    class fakeOutputStream extends OutputStream {
```