Faculty of

Computing, Engineering

and the Built Environment

BIRMINGHAM CITY
University

Coursework Assessment Brief

Academic Year 2016-17

Module:	**CMP5308 - Advanced Programming UG2**	
Assessment Title:	**Developing a** **Text Messaging Server in Java** A Client/Server Internet Utility Tool	
Assessment Identifier:	**CWRK001**	Weighting: 100%
School:	Computing and Digital Technology	
Module Co-ordinator:	Abdel-Rahman Tawil	
Hand in deadline date:	12pm Midday on date of submission. Please see the Assessment Block on your module Moodle page for details: http://moodle.bcu.ac.uk	
Hand back date:	Marks and feedback on your work will normally be provided within 20 working days of your submission deadline. See iCity/Moodle on the intranet for details. http://moodle.bcu.ac.uk	
Assessment hand in deadline date:	12pm Midday on **24th April 2017**	
Support available for students required to submit a re-assessment:	Timetabled revisions sessions will be arranged for the period immediately preceding the hand in date	

NOTE:	At the first assessment attempt, the full range of marks is available. At the re-assessment attempt the mark is capped and the maximum mark that can be achieved is 40%.
Assessment Summary	This is a group project. The maximum number of students in each group should not exceed four members; groups can be formed of less than four but with a minimum of two members in a group. Use the *CMP5308 - Advanced Programming Coursework (Groups)* Group choice system to enrol yourself and join a specific group. For this assignment *you will design, implement and demonstrate a (Text massaging client/ server) chat Application in Java. The assignment will be assessed in 2 parts:* *1. **Demonstration and Viva** [100% of module mark] – 5% of the assessment mark will be based on the completion of a selected lab session(s).*

IMPORTANT STATEMENTS

Standard Undergraduate Assessment Regulations

Your studies will be governed by version 5 of the Standard Undergraduate Assessment Regulations (SUAR 5).

Under these regulations you are permitted two attempts at assessment for each module: a first sit and re-assessment attempt.

This means that you will be required to withdraw from the course if, following the reassessment attempt, you have not passed.

Cheating and Plagiarism

Both cheating and plagiarism are totally unacceptable and the University maintains a strict policy against them. It is YOUR responsibility to be aware of this policy and to act accordingly. Please refer to the Academic Registry Guidance at https://icity.bcu.ac.uk/Academic-Registry/Information-for-Students/Assessment/Avoiding-Allegations-of-Cheating

The basic principles are:

- Don’t pass off anyone else’s work as your own, including work from “essay banks”. This is plagiarism and is viewed extremely seriously by the University.
- Don’t submit a piece of work in whole or in part that has already been submitted for assessment elsewhere. This is called duplication and, like plagiarism, is viewed extremely seriously by the University.
- Always acknowledge all of the sources that you have used in your coursework assignment or project.
- If you are using the exact words of another person, always put them in quotation marks.
- Check that you know whether the coursework is to be produced individually or whether you can work with others.
- If you are doing group work, be sure about what you are supposed to do on your own.
- Never make up or falsify data to prove your point.
- Never allow others to copy your work.
- Never lend disks, memory sticks or copies of your coursework to any other student in the University; this may lead you being accused of collusion.

By submitting coursework, either physically or electronically, you are confirming that it is your own work (or, in the case of a group submission, that it is the result of joint work undertaken by members of the group that you represent) and that you have read and understand the University's guidance on plagiarism and cheating.

Students should be aware that, at the discretion of the module co-ordinator, coursework may be submitted to an electronic detection system in order to help ascertain if any plagiarised material is present.

Electronic Submission of Work

Students should also be aware that it is their responsibility to ensure that work submitted in electronic format can be opened on a faculty computer and to check that any electronic submissions have been successfully uploaded. If it cannot be opened it will not be marked. Any required file formats will be specified in the assignment brief and failure to comply with these submission requirements will result in work not being marked.

Students must retain a copy of all electronic work they have submitted and resubmit if requested.

Learning Outcomes to be Assessed:

1. Demonstrate an understanding of basic principles for software development and its applications.
2. Use of appropriate programming / development environments (such as Eclipse, Java) to design and develop simple programming solutions.
3. Understand and use advanced programming concepts to create a programming solution.
4. Understand and use core java.net libraries and concepts to design and implement a networked solution.
5. Understand the need for basic analysis and design for advanced network program development.
6. Write simple software documentation to test an implemented programming solution

This assessment will assess all of the module's learning outcomes.

1. Problem Description

Your group task is to implement a text-messaging server that can be used for exchanging text messaging. A text message contains a single line of ASCII text, sent to a single user. The two components of this system are; a message server that manages the messages and a telnet client that can be used to connect to the server and to send and receive messages.

A partial implementation of the server will be made available to the students and in it its, in its current state, it allows users to *Log In*, *Log Out*, *send a message*, *check for waiting messages* and *get the next message* from the server. The server can be compiled and run, you can connect to it using telnet commands 101, 102, 103, 104 and 105 (see below) which are fully implemented. The code is available and can be downloaded from Moodle.

In its implemented state, the server requires a password file. The current location of the password file must be configured in the file *MsgProtocol.java*. The default location is "h:\\pwd.txt" – this needs to be changed with the current location on your local machine.

2.1. The Current Text Messaging Protocol

Here is the definition for the commands that a client may send to the message server:

```
{QUERY} ::= {LOGIN} | {SEND} | {OTHER}
{LOGIN} ::= "101" {CR} {U} {CR} {PWD} {CR}
{SEND} ::= "103" {CR} {U} {CR} {U} {CR} {M} {CR}
{OTHER} ::= {ID} {CR} {U} {CR}
{U} ::= username
{PWD} ::= password
{ID} ::= "102" | "104" | "105" | "106"
{M} ::= [<CHAR> | <CHAR> {M}]
{CR} := <CRLF>
```

Note that <CHAR> is any ASCII character excluding control characters and <CRLF> is carriage return followed by line feed which is "\r\n" in java. Here is a key to the client command IDs:

ID	Translation	Example
101	Login Command	"101 \r\n tony \r\n mypasswd \r\n"
102	Logout Command	"102 \r\n tony \r\n"
103	Send Command	"103 \r\n tony \r\n fred \r\n A message to fred \r\n"
104	Do I have waiting messages?	"104 \r\n tony \r\n"
105	Get next message	"105 \r\n tony \r\n"
106	Get all messages	"106 \r\n tony \r\n"

2.2. Current Available Server responses

```
{RESPONSE} ::= {OK} | {MSG} | {ERROR}

{OK} ::= "200" {CR}
{MSG} ::= "201" {CR} <n> {CR} {MSGDATA}
{ERROR} ::= "500" {CR} {M} {CR}

{D} ::= Date

{MSGDATA} ::= {U} {CR} {D} {CR} {M} {CR} [{MSGDATA}]
```

Note <n> is an integer with a value greater than zero

Here is a key to the server response IDs:

ID	Translation
200	Request successfully executed
201	n messages are being sent, each message consists of the senders username, the date the message was sent and the message content.
500	An error response and the accompanying message should explain the error

Examples of Java strings for valid responses are:

"200 \r\n"

"201 \r\n 1 \r\n fred \r\n Thu Feb 27 14:04:32 GMT 2003 \r\n A Message \r\n" or,

if there is more than one message:
"201 \r\n 2 \r\n fred \r\n Thu Feb 27 14:06:32 GMT 2003 \r\n Hello from Fred \r\n"
"fred \r\n Thu Feb 27 14:07:47 GMT 2003 \r\n Another Message \r\n"

"500 \r\n Incorrect Password \r\n"

Commands 101, 102, 103, 104 will all be followed by either an {OK} response or an {ERROR} response.

Note that for request 104, an {OK} response is a confirmation that there are messages waiting and an {ERROR} response means
No messages are waiting.

Commands 105 and 106 will be followed by eithera {MSG} response or an {ERROR} response.

2.3. The server

Main server classes are:

MsgProtocol
The MsgProtocol class defines constants for all of the client and server command identifiers. Use these constants in your code.

LoginCommand and LogoutCommand
The LoginCommand and LogoutCommand classes process the login and logout commands. You should look at these classes to see how to implement a command.

CommandFactory
This class exists to read the command identifier sent by the client and return a command class that can process the rest of the command. For example, if the command identifier is 101, a LoginCommand class will be returned. Currently command classes are only implemented for logging in and logging out.

Handling messages – MessageCollection
The classes you need for storing messages on the server has been fully implemented. These are called Message and MessageCollection. The Message class models the individual messages, has sender, date and content. The date is added to a message automatically.

The send command construct a new Message object whenever a new message is sent and that message is then added to the MessageCollection. The MessageCollection class provides a way of holding all the messages that have been sent but not yet read.

Those messages can be accessed using the recipient name as a key. The class has all the methods you will need for adding a new message to the collection, retrieving messages for a particular user and finding out how many messages a particular user currently has waiting in the collection. Note that getting a message from the collection also removes it from the collection.

Server connection - classes
The MsgSvrConnection class handles an individual connection between the server and a client. It has methods to set the current user and get the current user. Another method returns the MessageServer object because that object provides access to the message collection. The MessageServer class is the main server class. It knows about the MessageCollection and each MsgSvrConnection.

3. Command classes to be Completed

Your group task is to implement the networking code for the remaining classes below. This should include defining the protocols that handles these commands that your server must process. These commands are:

1. Get all messages (106) – Partially implemented

   (10 Marks)

2. Devise and implement a protocol and command(s) that allows the user to register for the text-messaging server. Basic registration would include username and password. Full registration can include Date-of-birth, telephone information, address details – Not implemented

   (20 Marks)

3. Devise and implement a protocol and command(s) that allows the user to update registration details – Not implemented

   (10 Marks)

4. Devise and implement a protocol and command(s) that allows the user to set reminders for particular event. Users can be notified of reminders by (text message, sound alert or a popup window) – Not implemented

   (20 Marks)

5. Devise and implement a protocol that allows the user to access and update reminders. Users can be notified of reminders by (text message, sound alert or a popup window) – Not implemented

   (10 Marks)

6. Extend the implementation of the text-messaging server to enable JDBC connection for at least two command classes.

   (30 Marks)

When you design the protocol you need to take into account error messages and server response status codes. The functionalities for each step should be thoroughly tested based on a studies testing strategy for software implementation.

Assessment Criteria:

Table of Assessment Criteria and Associated Grading Criteria

FOR GUIDANCE ONLY: Grading Criteria (100%)

FOR GUIDANCE ONLY: Grading Criteria: DEMONSTRATION AND VIVA

Assessment Criteria →	**1. Get All Messages**	**2. User Registration**	**3. Registration Update**	**4. User Reminders**
Weighting:	**10%**	**20%**	**10%**	**20%**
Grading Criteria **0 – 29%** **30 – 39%**	A basic (possibly unsuccessful) but *relevant* attempt at implementing the get all messages method – *evident in code*.	A basic protocol specification and implementation of user registration (possibly unsuccessful) but *relevant* attempt – *evident in code*.	A basic protocol specification and implementation of registration update (possibly unsuccessful) but *relevant* attempt – *evident in code*.	A basic protocol specification and implementation of user reminders (possibly unsuccessful) but *relevant* attempt – *evident in code*.
40 – 49% **50 – 59%**	A good (mostly successful) attempt at implementing the get all messages method – *able to successfully display some but not all messages – evident in code*.	A basic and functional implementation incorporating evidence *at basic user registration* username and password – *evident in code*.	A basic and functional implementation incorporating evidence *at basic user details update* username and password – *evident in code*.	A basic and functional implementation incorporating evidence *at basic reminder setting. Reminders may not trigger – evident in code*.

60 – 69%	A very good (fully successful) attempt at implementing a usable and robust get all messages method (without data validation) – *evident in code.*	A very good and functional implementation incorporating evidence *at basic user registration* username and password and an attempt to implement complete user details registration– *evident in code.*	A very good and functional implementation incorporating evidence *at basic update user registration* username and password and an attempt to implement complete user details update– *evident in code.*	A very good and functional implementation incorporating evidence *at basic reminder protocol* and an implementation of *reminder setting. Reminders should trigger – evident in code.*
70 – 79%	An excellent, inspired, usable and robust get all messages method with data validation – *evident in code.*	An excellent and fully successful attempt at incorporating evidence *user registration* username and password implement complete user details for registration, documented – *evident in code.*	An excellent and fully successful attempt at incorporating evidence *user registration* update username and password also implement complete user details registration update, documented – *evident in code...*	An excellent and fully successful attempt at *advanced reminder protocol* and an implementation of *reminder setting, documented. Reminders should trigger – evident in code.*
80 – 100%	ALL features and validations are implemented code robust & documented – *evident in code.*			
Checklist	*Application Demonstration and Code Inspection*	*Application Demonstration and Code Inspection*	*Application Demonstration and Code Inspection*	*Application Demonstration and Code Inspection*

Assessment Criteria →	**5. Reminders Update**	**6. JDBC Connectivity**
Weighting:	**10%**	**30%**
Grading Criteria **0 – 29%**	A basic protocol specification and implementation of reminders update (possibly unsuccessful) but *relevant* attempt – *evident in code*.	A basic design that described JDBC extension (possibly unsuccessful implementation) but *relevant* attempt – *evident in code*.
30 – 39%		
40 – 49%	A basic and functional implementation incorporating evidence *at basic reminder update. Reminders may not update or trigger – evident in code*.	A basic and functional implementation incorporating evidence *at basic use of JDBC for data access* (at least on task fully implemented) – *evident in code*.
50 – 59%		
60 – 69%	A very good and functional implementation incorporating evidence *at basic reminder update protocol* and an implementation of *reminder update. Reminders should update & trigger – evident in code*.	A very good and functional implementation incorporating evidence *at basic use of JDBC for data access* and a good attempt to implement complete JDBC connectivity and access for two tasks – *evident in code*.
70 – 79%	An excellent and fully successful attempt at *advanced reminder update protocol* and a full implementation of *reminder update, documented. Reminders should trigger – evident in code*.	An excellent and fully successful attempt at incorporating JDBC implemented and fully functional for two tasks, documented – *evident in code*.
80 – 100%		
Checklist	*Application Demonstration and Code Inspection*	*Application Demonstration and Code Inspection*

Submission Details and Instructions:

- **<u>Demonstration / Viva of Completed Application</u>** - the completed Application project (with source code) should be compressed (zipped) and **Uploaded to Moodle (link to be supplied)** by the specified date. The individual demonstration / viva will be held in the following week(s), you will be notified of the schedule for the group Viva nearer the date – **only work that has been uploaded to Moodle on time will be marked, any work not submitted in this way may be capped as late when demonstrated.**

Workload:

The assignment will require at least 30 hours of work effort.

Feedback:

You will be provided brief informal feedback at the time of presentation / demonstration. Formal feedback and provisional marks will be provided via Moodle.

Marks and Feedback on your work will normally be provided within 20 working days of its submission deadline.

(Coursework Total Mark 100%)