CS3524 Assessment 2 Multiplayer Game Server

In this assessment, you will implement a simple game server that allows multiple players (clients) access to a MUD (Multi-User Dungeon or Dimension or Domain). This assessment will examine your ability to design and implement distributed systems using Java RMI.

You are expected to use and adapt code provided in practical exercises to construct your solution to the assessment. When you do, however, you must acknowledge the classes that your solution is based on using a comment in your source code.

How to start your assessment

In developing your solution, start from the java code provided in mud.tgz. You can download this file from MyAberdeen. The mud package contains three classes:

- MUD, which provides a graph-based representation of a MUD world
- In order to represent this game graph in memory, the MUD class uses the two classes Vertex and Edge:
    - The Vertex class represents a vertex in the graph
        - These are the “locations” in the MUD game
    - The Edge, class represents an edge in the graph
        - These are the paths between locations in the MUD game.

The archive also contains three files that capture information about an example MUD world:

- mymud.edg contains the edges in the graph representing the MUD game (locations)
- mymud.msg contains the messages or information associated with each location
- mymud.thg contains the things that pre-allocated to location

You can define your own MUD using these file formats and check that it parses the MUD specification by running the class on its own (see the main() method in MUD.java, which is provided just for testing purposes).

Class MUD contains the following public methods: addThing(), delThing,() moveThing(), loactionInfo(). A “thing” in terms of this MUD implementation is everything that is situated at a location – this includes the players that move through the game, or items that can be picked up at a location (see mymud.thg).

Please note that your solution has to be executed when marked. Therefore, provide information how to install / setup and run / play your solution.

Assessment Requirements

CGS D3-D1

For achieving a CGS D3-D1, create a simple client – server application, where the server implements a remote method that is called by the client.

A MUD game server and a client has to be implemented, based on Java RMI. This game server will allow a client to connect to the server and call remote methods.

a) Implement a MUD game server: in your implementation, create a remote interface that extends java.rmi.Remote and declares a set of remote methods, create a class that is the implementation of this interface, and the server mainline to register remote objects with the rmiregistry.
In developing your solution to this, start from the java code provided in mud.tgz. It is available for download from MyAberdeen.

b) Implement a client that utilises the remote methods provided by the remote object. The client should be implemented to allow a player via console input to make at least one move in at least one direction within the MUD game. At start, the client should print out information from the start location, and print out the information associated with the new location after the move (see mymud.msg).

In order to achieve a CGS D3-D1, implement the server and client to allow a client to perform a single move in the MUD game, and display the location information (mymud.msg) of the new location. Your solution has to contain the correct setup of a security manager, including a security policy file and the correct setup and use of RMI.

CGS C3-C1

For CGS C3-C1, create a client – server application that is multi-user: multiple clients may access remote methods on a server and see each others' actions.

Extend the solution for CGS D, so that
- Users can move around in the MUD world in any direction
- Users can see other users in the MUD world
- Users can pick up things in the MUD

In your client implementation, provide a set of commands that allow a user to make moves, list the users currently at a location, list the things that are stored at a location, pick up things, show a player's inventory (what they picked up) [1]. When the client starts the game, allow a user to specify a user name, and print the location info of the start location (see mymud.msg). Just pressing RETURN (no command) should again print the location info for the current location (acting as a refresh of the console output).

[1] The "things" at a location are game items (such as "treasure"), but – to simplify the implementation – also players that arrived at this location. Players are supposed to collect only things at a location that are game items.

If an item is collected and transferred into a player's inventory, it should be removed properly from the location. Provide a help command that prints out information what commands are available and how to use them.

CGS B3-B1

For CGS B3-B1, create a client – server application that is multi-user and multi-game: multiple clients may access remote methods on a server and operate with one of multiple MUD games the server instantiates (all MUD games can use the same mymud.* files).

Extend the solution for CGS C so that

- The game server instantiates more than one MUD game
- The game server provides the user with the ability to
    - find out what MUDs are running at the Server, and
    - select a MUD to join for game playing
    - functionality to leave a MUD and end playing the MUD game

In order to achieve a B1, a user should also be able, while playing in one MUD, to switch to / join another MUD, being able to switch the user's "game focus" between multiple games.

CGS A5

In order to achieve a CGS A5, create a client – server application that is multi-user, multi-game, and where new MUD games can be (a) instantiated and (b) joined / played.

Extend your implementation so that

- a user can issue commands to the server in the client application to create new MUD games (all MUD games can use the same mymud.* files)

Your solution should provide:

- Client can create new games
- Client can join, exit game and join other / new game
- Client can have multiple games open

A user can create / switch to / join another MUD and continue playing in the new MUD, effectively playing multiple MUDs, and being able to switch the users "game focus" between multiple games.

The server should also restrict the number of MUDs that can run at any time and the total number of users logged on to MUDs.

Organise your console interface so that a user can easily create and/or join / leave MUD games.

CGS A4-A1

In order to achieve a CGS A4 – A1, create a responsive and robust distributed system where clients can be aborted, restarted and reconnected to the server.

Responsiveness - Automatic refresh of player consoles:

- introduce automatic update of all the consoles of all players playing a MUD game if changes occur, such as a player arriving at a location, or a player collecting items.
- Use a callback implementation to allow a server to push changes in a MUD game to all the clients that are currently playing the game

Robustness: make your implementation robust against clients leaving a game or client applications being aborted

- Handle users leaving MUDs correctly, or closing or aborting client applications
    - servers should recognise such a situation and clean up client logins, unregister any remote objects in the rmiregistry if necessary
- Clients handle exit and abort correctly

Communication:

- Introduce commands so that players can send messages to each other.

Submission Procedure

You are required to submit in electronic as form:

- one zip file called **`cs3524_assessment2_<your_username>.zip`**

Submit your work, using the following method:

- In MyAberdeen, go to “Assessment”, and do the following:
    - Go to the area after the text explaining the submission procedure
    - click on “CS3524 Assessment 2 (submit here)”
    - click on “Browse My Computer” (this is underneath “2. Assignment Submission”)
    - find your zip file and complete this upload

Please use your University email address to submit your assessment.

All submissions should be documented, and this documentation must include your name and userid at the top of each file. Please submit (a) the complete source code (plus any necessary make files, text files, configuration files etc to compile and run your submission) and (b) a report describing your submission and how to operate your application. Make a zip file containing all this information and send it to the email address above.