```
package cs3524.solutions.mud;

import java.util.List;
import java.util.ArrayList;

public class ClientImpl implements ClientInterface {
        private String clientUserName;

        public ClientImpl( String userName )
        {
                clientUserName = userName;
        }

        public String getUserName()
        {
                String name = clientUserName;
                return name;
        }

        public void sendMessage(String message)
        {
                System.out.println(message);

        }
}
```

```
package cs3524.solutions.mud;

import java.util.List;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ClientInterface extends Remote
{
        public String getUserName() throws RemoteException;
        public void sendMessage( String message ) throws RemoteException;

}
```

```
package cs3524.solutions.mud;

import java.rmi.Naming;
import java.lang.SecurityManager;
import java.rmi.server.UnicastRemoteObject;
import java.util.List;
import java.util.Scanner;
/**
 * Author: Marcel Zak
 * version 0.0
 */

public class ClientMainline
{
                public static void main(String args[])
                {
                if (args.length < 3) {
                        System.err.println( "Usage:\njava ClientMainline <registryhost> <registr
yport> <callbackport>" ) ;
                        return;
                }

                try {
                        String hostname = args[0];
                        int registryport = Integer.parseInt( args[1] ) ;
                        int callbackport = Integer.parseInt( args[2] ) ;

                        System.setProperty( "java.security.policy", "mud.policy" ) ;
                        System.setSecurityManager( new SecurityManager() ) ;

                        String clientUserName = System.console().readLine("Your n
ame: ").trim();


                        ClientImpl newClient = new ClientImpl( clientUserName );
                        ClientInterface clientStub = (ClientInterface)UnicastRem
oteObject.exportObject( newClient, callbackport );

                        String regURL = "rmi://" + hostname + ":" + registryport +
 "/MUDServer";
                        MUDServerInterface serverStub = (MUDServerInterface)Nami
ng.lookup( regURL );

                        List<String> servers = serverStub.listServers();
                        Integer i = 1;
                        for( String srv : servers )
                        {
                                System.out.println("("+i+") "+srv);
                                if (servers.size() == i){
                                        ++i;
                                        System.out.println("("+i+") Create own server"
);
                                }
                                ++i;
                        }

                        //choose a server or create your own
                        String chosenServerString = null;
                        boolean response = false;
                        while(chosenServerString == null)
                        {
                                chosenServerString = System.console().readLine("
```



```
Connect to server number: ").trim();
                                if (Integer.parseInt(chosenServerString) <= serv
ers.size())
                                {       // you have chosen one of the existing s
ervers
                                        Integer chosenServerInt = Integer.parseI
nt(chosenServerString);
                                        --chosenServerInt;      //decrement the
value to match index
                                        response = serverStub.joinServer(servers
.get(chosenServerInt), clientStub);
                                        if ( response == false ){System.exit(0);
}
                                }
                                else if (Integer.parseInt(chosenServerString) ==
 servers.size()+1)
                                {       // you have chosen to create your own se
rver. It is created on runtime
                                        response = serverStub.joinServer("new",
clientStub);
                                        if ( response == false ){System.exit(0);
}
                                }
                                else {  // invalid input
                                        chosenServerString = null;
                                        System.out.println("Invalid choice! Try again.")
;
                                }
                        }

                        //control commands
                        System.out.println( "For help just type 'help'" );
                        String userInput;
                        while(true)
                        {
                                userInput = System.console().readLine("What do you
 want to do?\n").trim();

                                if ( userInput.equals( "help" ) )
                                {       // help prints all the options
                                        System.out.println( "\nview \nmove \ntake \nsho
w inventory \nonline users \nmessage \nexit\n" );
                                }

                                if ( userInput.equals( "view" ) )
                                {       //view waht you have around you
                                        serverStub.view( clientUserName, "paths"
);
                                        serverStub.view( clientUserName, "things"
 );
                                }

                                if ( userInput.equals( "move" ) )
                                {       // move somewhere
                                        System.out.println( "You can move:\n" );
                                        serverStub.view( clientUserName, "paths"
);
                                        userInput = System.console().readLine( "
Where do you want to move?\n" ).trim();
                                        if ( userInput.equals("north") || userInp
ut.equals("east") || userInput.equals("south") || userInput.equals("west") )
                                        {
```

```
                                        serverStub.moveUser( clientUserN
ame, userInput );
                                    }
                                }

                                if ( userInput.equals( "take" ) )
                                {       // take item around you
                                        serverStub.view( clientUserName, "things"
 );
                                        userInput = System.console().readLine( "
What would you like to take?\n" ).trim();
                                        serverStub.getThing( clientUserName, use
rInput );
                                }

                                if ( userInput.equals( "show inventory" ) )
                                {       //show your items in inventory
                                        serverStub.showInventory( clientUserName
 );
                                }

                                if ( userInput.equals( "online users" ) )
                                {       // show all online users on the server
                                        serverStub.listUsers( clientUserName );
                                }

                                if ( userInput.equals( "message" ) )
                                {       // send a message to online user
                                        System.out.println( "You can message to:\n" )
;
                                        serverStub.listUsers( clientUserName );
                                        String to = System.console().readLine( "
Write the name:\n" ).trim();
                                        String message = System.console().readLi
ne( "Write the message:\n" ).trim();
                                        serverStub.message(clientUserName, to, m
essage );
                                }

                                if ( userInput.equals( "exit" ) )
                                {       // stop the client
                                        serverStub.leaveServer( clientUserName )
;
                                        System.exit(0);
                                }

                        }

                }
                catch(java.rmi.NotBoundException e) {
                        System.err.println( "Can't find the auctioneer in the registry." );
                }
                catch (java.io.IOException e) {
                        System.out.println( "Failed to register." );
                }
        }
}
```

```
/**************************************************************************
 * cs3524.solutions.mud.Edge
 *************************************************************************/

package cs3524.solutions.mud;

// Represents an path in the MUD (an edge in a graph).
class Edge
{
        public Vertex _dest;   // Your destination if you walk down this path
        public String _view;   // What you see if you look down this path
        public Edge( Vertex d, String v )
                {
                        _dest = d;
                        _view = v;
                }
}
```

```
/*************************************************************************
 * cs3524.solutions.mud.MUD
 *************************************************************************/

package cs3524.solutions.mud;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.StringTokenizer;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Vector;
import java.util.HashMap;

/**
 * A class that can be used to represent a MUD; essenially, this is a
 * graph.
 */

public class MUD
{
        /**
         * Private stuff
         */

        // A record of all the vertices in the MUD graph. HashMaps are not
        // synchronized, but we don't really need this to be synchronised.
        private Map<String,Vertex> vertexMap = new HashMap<String,Vertex>();

        private String _startLocation = "";

        /**
         * Add a new edge to the graph.
         */
        private void addEdge( String sourceName,
                              String destName,
                              String direction,
                              String view )
        {
                Vertex v = getOrCreateVertex( sourceName );
                Vertex w = getOrCreateVertex( destName );
                v._routes.put( direction, new Edge( w, view ) );
        }

        /**
         * Create a new thing at a location.
         */
        private void createThing( String loc, String thing )
        {
                Vertex v = getOrCreateVertex( loc );
                v._things.add( thing );
        }

        /**
         * Change the message associated with a location.
         */
        private void changeMessage( String loc, String msg )
        {
                Vertex v = getOrCreateVertex( loc );
```



```
                v._msg = msg;
        }

        /**
         * If vertexName is not present, add it to vertexMap.  In either
         * case, return the Vertex. Used only for creating the MUD.
         */
        private Vertex getOrCreateVertex( String vertexName )
        {
                Vertex v = vertexMap.get( vertexName );
                if (v == null) {
                        v = new Vertex( vertexName );
                        vertexMap.put( vertexName, v );
                }
                return v;
        }

        /**
         *
         */
        private Vertex getVertex( String vertexName )
        {
                return vertexMap.get( vertexName );
        }

        /**
         * Creates the edges of the graph on the basis of a file with the
         * following fromat:
         * source direction destination message
         */
        private void createEdges( String edgesfile )
        {
        try {
                FileReader fin = new FileReader( edgesfile );
                        BufferedReader edges = new BufferedReader( fin );
                        String line;
                        while((line = edges.readLine()) != null) {
                                StringTokenizer st = new StringTokenizer( line )
;
                if( st.countTokens( ) < 3 ) {
                        System.err.println( "Skipping ill-formatted line " + line );
                        continue;
                }
                String source = st.nextToken();
                String dir        = st.nextToken();
                String dest   = st.nextToken();
                String msg = "";
                while (st.hasMoreTokens()) {
                        msg = msg + st.nextToken() + " ";
                                }
                addEdge( source, dest, dir, msg );
                }
        }
        catch( IOException e ) {
                System.err.println( "Graph.createEdges( String " +
                                edgesfile + ")\n" + e.getMessage() );
        }
        }

        /**
         * Records the messages assocated with vertices in the graph on
         * the basis of a file with the following format:
```

```
         * location message
         * The first location is assumed to be the starting point for
         * users joining the MUD.
         */
        private void recordMessages( String messagesfile )
        {
        try {
                FileReader fin = new FileReader( messagesfile );
                        BufferedReader messages = new BufferedReader( fin );
                        String line;
                boolean first = true; // For recording the start location.
                        while((line = messages.readLine()) != null) {
                                StringTokenizer st = new StringTokenizer( line )
;
                if( st.countTokens( ) < 2 ) {
                        System.err.println( "Skipping ill-formatted line " + line );
                        continue;
                }
                String loc = st.nextToken();
                String msg = "";
                while (st.hasMoreTokens()) {
                        msg = msg + st.nextToken() + " ";
                                }
                changeMessage( loc, msg );
                if (first) {      // Record the start location.
                        _startLocation = loc;
                        first = false;
                }
                }
        }
        catch( IOException e ) {
                System.err.println( "Graph.recordMessages( String " +
                                messagesfile + ")\n" + e.getMessage() );
        }
        }

        /**
         * Records the things assocated with vertices in the graph on
         * the basis of a file with the following format:
         * location thing1 thing2 ...
         */
        private void recordThings( String thingsfile )
        {
        try {
                FileReader fin = new FileReader( thingsfile );
                        BufferedReader things = new BufferedReader( fin );
                        String line;
                        while((line = things.readLine()) != null) {
                                StringTokenizer st = new StringTokenizer( line )
;
                if( st.countTokens( ) < 2 ) {
                        System.err.println( "Skipping ill-formatted line " + line );
                        continue;
                }
                String loc = st.nextToken();
                while (st.hasMoreTokens()) {
                        addThing( loc, st.nextToken());
                                }
                }
        }
        catch( IOException e ) {
                System.err.println( "Graph.recordThings( String " +
```



```
                                thingsfile + ")\n" + e.getMessage() );
        }
        }

        /**
         * All the public stuff. These methods are designed to hide the
         * internal structure of the MUD. Could declare these on an
         * interface and have external objects interact with the MUD via
         * the interface.
         */

        /**
         * A constructor that creates the MUD.
         */
        public MUD( String edgesfile, String messagesfile, String thingsfile )
        {
        createEdges( edgesfile );
        recordMessages( messagesfile );
        recordThings( thingsfile );

        System.out.println( "Files read..." );
        System.out.println( vertexMap.size( ) + " vertices\n" );
        }

        // This method enables us to display the entire MUD (mostly used
        // for testing purposes so that we can check that the structure
        // defined has been successfully parsed.
        public String toString()
        {
        String summary = "";
        Iterator iter = vertexMap.keySet().iterator();
        String loc;
        while (iter.hasNext()) {
                loc = (String)iter.next();
                summary = summary + "Node: " + loc;
                summary += ((Vertex)vertexMap.get( loc )).toString();
        }
        summary += "Start location = " + _startLocation;
        return summary;
        }

        /**
         * A method to provide a string describing a particular location.
         */
        public String locationInfo( String loc )
        {
        return getVertex( loc ).toString();
        }

        //method that provides infor where a player can move
        public String locationPaths( String loc )
        {
                String message = getVertex( loc )._msg + "\n";
                for (Map.Entry<String, Edge> vertex : getVertex(loc)._routes.ent
rySet())
                {
                                message += "You can move to the " + vertex.getKey() + " there is "
 + vertex.getValue()._view + "\n";
                }
                return message;
        }
```

```
        //method that provides info about things on location
        public List locationThings( String loc )
        {
                List<String> things = getVertex(loc)._things;
                return things;
        }

        /**
         * Get the start location for new MUD users.
         */
        public String startLocation()
        {
        return _startLocation;
        }

        /**
         * Add a thing to a location; used to enable us to add new users.
         */
        public void addThing( String loc,
                              String thing )
        {
        Vertex v = getVertex( loc );
        v._things.add( thing );
        }

        /**
         * Remove a thing from a location.
         */
        public void delThing( String loc,
                              String thing )
        {
        Vertex v = getVertex( loc );
        v._things.remove( thing );
        }

        /**
         * A method to enable a player to move through the MUD (a player
         * is a thing). Checks that there is a route to travel on. Returns
         * the location moved to.
         */
        public String moveThing( String loc, String dir, String thing )
        {
        Vertex v = getVertex( loc );
        Edge e = v._routes.get( dir );
        if (e == null)   // if there is no route in that direction
                return loc;  // no move is made; return current location.
        v._things.remove( thing );
        e._dest._things.add( thing );
        return e._dest._name;
        }

        /**
         * A main method that can be used to testing purposes to ensure
         * that the MUD is specified correctly.
         */
        public static void main(String[] args)
        {
        if (args.length != 3) {
                System.err.println("Usage: java Graph <edgesfile> <messagesfile> <thingsfile>");
                return;
        }
        MUD m = new MUD( args[0], args[1], args[2] );
```



```
                System.out.println( m.toString() );
                }
}
```

```
/*      Author: Marcel Zak
*       version: 0.0
*/
package cs3524.solutions.mud;

import java.rmi.*;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.ArrayList;
import java.util.Set;
import java.util.HashMap;


public class MUDServerImpl implements MUDServerInterface {

        private Integer maxPlayers = 2; // restrict number of players per server
        private Integer maxServers = 3; // restrict number of MUDs which can be
created
        // serverMap holds all the MUDs. Key is a name and value is MUD object
        private Map<String, MUD> serversMap = new HashMap<String, MUD>();
        // clientToServerMap holds records about where the client is
        private Map<String, String> clientToServerMap = new HashMap<String, Stri
ng>();
        // serverToClientMap is a nested HashMap which holds records about
        // server name -> HashMap of all clients names logged in -> ClientInterf
ace
        private Map<String, HashMap<String, ClientInterface>> serverToClientsMap
 = new HashMap<String, HashMap<String, ClientInterface>>();
        // clientInventoryMap holds records of all clients inventories Name -> I
nventory
        private Map<String, ArrayList<String>> clientInventoryMap = new HashMap<
String, ArrayList<String>>();
        // clientPositionMap holds records of all clients positions
        private Map<String, String> clientPositionMap = new HashMap<String, Stri
ng>();

        public MUDServerImpl() throws RemoteException
        {       //create two servers at start
                serversMap.put("Hades", new MUD("servers/hades/hades.edg","servers/hades/hade
s.msg","servers/hades/hades.thg"));
                HashMap<String, ClientInterface> clientsMap = new HashMap<String
, ClientInterface>();
                serverToClientsMap.put( "Hades", clientsMap);
                serversMap.put("Pathos", new MUD("servers/pathos/pathos.edg","servers/pathos/p
athos.msg","servers/pathos/pathos.thg"));
                clientsMap = new HashMap<String, ClientInterface>();
                serverToClientsMap.put( "Pathos", clientsMap);
        }

        public List<String> listServers() throws RemoteException
        {       // list all the online MUDs
                Set<String> serversSet = serversMap.keySet();
                return new ArrayList<String>(serversSet);
        }

        public boolean joinServer(String serverName, ClientInterface client) thr
ows RemoteException
        {       // join or create own server
                MUD server = serversMap.get(serverName);
                String clientUserName = client.getUserName();
                HashMap<String, ClientInterface> clientsMap;
```



```
                if (server == null)
                {       // server does not exist that means you want to create o
wn MUD
                        if ( serversMap.size() >= maxServers )
                        {       // check if maxServers is reached
                                client.sendMessage( "Maximum number of servers reached. T
ry later." );
                                return false;
                        }
                        serversMap.put(clientUserName+"'s server", new MUD("servers/vo
id/void.edg","servers/void/void.msg","servers/void/void.thg"));
                        clientToServerMap.put( clientUserName, clientUserName+"'
s server" );
                        server = serversMap.get(clientUserName+"'s server");

                        clientsMap = new HashMap<String, ClientInterface>();
                        clientsMap.put( clientUserName, client );
                        serverToClientsMap.put( clientUserName+"'s server", clients
Map );

                        clientPositionMap.put( clientUserName, server.startLocat
ion() );

                        clientInventoryMap.put( clientUserName, new ArrayList<St
ring>() );

                        server.addThing( server.startLocation(), "User: "+clientU
serName );

                        String message = "\n~~~Welcome to "+clientUserName+"'s Server~
~~\n";
                        message += "You are currently at "+clientPositionMap.get( clien
tUserName )+" location\n";
                        client.sendMessage( message );  // inform the client
                        return true;
                }
                if(serverToClientsMap.get(serverName).size() >= maxPlayers)
                {       // check if maxPlayers is reached
                        client.sendMessage( "Sorry maximum players reached. Try later or join a
nother server" );
                        return false;
                }
                if ( clientToServerMap.get( clientUserName ) != null )
                {       // check if clientUserName already exist. It must be uni
que
                        client.sendMessage( "Change name please! User already exist!" );
                        return false;
                }
                // create user
                clientToServerMap.put( clientUserName, serverName );
                clientsMap = serverToClientsMap.get( serverName );
                clientsMap.put( clientUserName, client );
                serverToClientsMap.put( serverName, clientsMap );
                clientPositionMap.put( clientUserName, server.startLocation() );
                clientInventoryMap.put( clientUserName, new ArrayList<String>()
);
                server.addThing( server.startLocation(), "User: "+clientUserName
);
                // prepare the message
                String message = ( "\n~~~Welcome to "+serverName+" Server~~~\n" );
                message += "Current number of player on this server is "+serverToClientsMap.ge
t( serverName ).size()+"\n";
```

```
                message += "You are currently at "+clientPositionMap.get( clientUserNam
e )+" location\n";
                // send the message to the client
                client.sendMessage( message );
                return true;
        }

        public boolean leaveServer( String clientUserName ) throws RemoteExcepti
on
        {       // leave server
                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface client = clientsMap.get( clientUserName );
                String position = clientPositionMap.get( clientUserName );

                if ( serverName  == null )
                {       // user does not exist
                        client.sendMessage( "Error the user does not exist at the server\n" );
                        return false;
                }

                if ( serverName.equals( clientUserName+"'s server" ) )
                {       // used had own server
                        serversMap.remove( clientUserName+"'s server" );
                        serverToClientsMap.remove( clientUserName+"'s server" );
                }
                else
                {       // delete users records
                        server.delThing( position  ,"User: "+clientUserName );
                        serversMap.put( serverName, server);
                        clientsMap.remove( clientUserName );
                        serverToClientsMap.put( serverName, clientsMap );
                }
                // send bye message
                client.sendMessage( "BB see you soon!" );
                clientToServerMap.remove( clientUserName );
                clientInventoryMap.remove( clientUserName );
                clientPositionMap.remove( clientUserName );
                return true;

        }

        public boolean view( String clientUserName, String what ) throws RemoteE
xception
        {       // view what is at particular location
                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface client = clientsMap.get( clientUserName );
                String position = clientPositionMap.get( clientUserName );
                String message = null;

                if ( what.equals("paths") )
                {       // send info about possible paths
                        message = server.locationPaths( position );
                        client.sendMessage( message );
                        return true;
                }
```



```
                if ( what.equals("things") )
                {       // send info about things
                        message = "There is:\n";
                        List<String> things = server.locationThings( position );
                        for ( String t : things )
                        {       // construct the message and send
                                message += t + "\n";
                        }
                        client.sendMessage( message );
                        return true;
                }
                return false;
        }

        public boolean moveUser(String clientUserName, String position) throws R
emoteException
        {       // move the user
                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface client = clientsMap.get( clientUserName );
                String origin = clientPositionMap.get( clientUserName );
                String message = "";
                // try to move the user and check response
                message = server.moveThing( origin, position, "User: "+clientUser
Name );
                clientPositionMap.put( clientUserName, message );
                if ( message.equals( origin ) )
                {       // user is at the same place because there is no path
                        client.sendMessage( "You cannot move there.\n" );
                        return false;
                }
                client.sendMessage( "You moved to " + message + "\n");
                return true;
        }



        public boolean getThing( String clientUserName, String thing) throws Rem
oteException
        {       // client can take a thing but not a user

                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface client = clientsMap.get( clientUserName );
                ArrayList<String> inventory = clientInventoryMap.get( clientUser
Name );
                List<String> things = server.locationThings( clientPositionMap.g
et( clientUserName ) );

                for ( String t : things )
                {       // iterate through things
                        if ( thing.equals( t ) && !thing.contains("User:") )
                        {       // check if there is the thing client wants to t
ake
                                // && check of the thing is not user
                                server.delThing( clientPositionMap.get( clientUs
erName ), t );
                                inventory.add( t );
```

```
                                clientInventoryMap.put( clientUserName, inventor
y );
                                client.sendMessage( "You have: "+inventory.toStrin
g() );
                                return true;
                        }
                }
                // the thing was not there or it was a user
                client.sendMessage( "No!\nYou have: "+inventory.toString() );
                return false;

        }

        public boolean showInventory( String clientUserName ) throws RemoteExcep
tion
        {       // list all the collected items
                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface client = clientsMap.get( clientUserName );
                List<String> inventory =  clientInventoryMap.get(clientUserName)
;
                String message = "In your inventory is:\n";
                client.sendMessage( message+inventory.toString() );
                return true;
        }

        public boolean listUsers( String clientUserName ) throws RemoteException
        {       // list all the online users at the server where client is.
                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface client = clientsMap.get( clientUserName );
                String message = "\nThese users are online:\n";
                Set<String> clientsSet = clientsMap.keySet();
                for (String c : clientsSet )
                {
                        message += c+"\n";
                }
                client.sendMessage( message );
                return true;
        }

        public boolean message( String clientUserName, String to, String message
 ) throws RemoteException
        {       // send a message to a user
                String serverName = clientToServerMap.get( clientUserName );
                MUD server = serversMap.get( serverName );
                HashMap<String, ClientInterface> clientsMap = serverToClientsMap
.get( serverName );
                ClientInterface fromClient = clientsMap.get( clientUserName );
                ClientInterface toClient = clientsMap.get( to );
                String formatedMessage = "Message from " + clientUserName + ":\n" +
message;
                toClient.sendMessage( formatedMessage );
                fromClient.sendMessage( "\nMessage sent\n" );
                return true;
        }
}
```

```
/*      Author: Marcel Zak
*       version: 0.0
*/

package cs3524.solutions.mud;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface MUDServerInterface extends Remote
{

        public List<String> listServers() throws RemoteException;
        public boolean joinServer( String serverName, ClientInterface client ) t
hrows RemoteException;
        public boolean leaveServer( String clientUserName ) throws RemoteExcepti
on;
        public boolean view( String clientUserName, String way ) throws RemoteEx
ception;
        public boolean moveUser( String clientUserName, String position ) throws
 RemoteException;
        public boolean getThing( String clientUserName, String thing ) throws Re
moteException;
        public boolean showInventory( String clientUserName ) throws RemoteExcep
tion;
        public boolean listUsers( String clientUserName ) throws RemoteException
;
        public boolean message( String clientUserName, String to, String message
 ) throws RemoteException;
}
```

```
/*      Author: Marcel Zak
*       version: 0.0
*/

package cs3524.solutions.mud;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class MUDServerMainline {

        static BufferedReader in = new BufferedReader( new InputStreamReader( Sy
stem.in ));


        public static void main(String args[]){

        if(args.length < 2){
                System.err.println("Usage:\n java MUDServerMainline <registryport> <serverport>"
);
                return;
        }

        try {
                String hostname = (InetAddress.getLocalHost()).getCanonicalHostN
ame();
                int registryPort = Integer.parseInt(args[0]);
                int serverPort = Integer.parseInt(args[1]);

                // Setup Security
                System.setProperty("java.security.policy", "mud.policy");
                System.setSecurityManager( new RMISecurityManager() );

                // Generate the remote objects
                MUDServerImpl mudserver = new MUDServerImpl();
                MUDServerInterface mudstub = (MUDServerInterface)UnicastRemoteOb
ject.exportObject(mudserver, serverPort);

                String regURL = "rmi://" + hostname + ":" + registryPort + "/MUDSer
ver";
                System.out.println("Registering " + regURL);
                Naming.rebind(regURL, mudstub);

        }
        catch (java.net.UnknownHostException e) {
                System.err.println("Java can't determine the local host!");
        }
        catch (java.io.IOException e){
                System.err.println("Failed to regitser.");
        }

        }

}
```

```
/**********************************************************************
 * cs3524.solutions.mud.Vertex
 **********************************************************************/

package cs3524.solutions.mud;

import java.util.Map;
import java.util.HashMap;
import java.util.List;
import java.util.Vector;
import java.util.Iterator;

// Represents a location in the MUD (a vertex in the graph).
class Vertex
{
        public String _name;              // Vertex name
        public String _msg = "";                  // Message about this location
        public Map<String,Edge> _routes; // Association between direction
                                         // (e.g. "north") and a path
                                         // (Edge)
        public List<String> _things;     // The things (e.g. players) at
                                         // this location

        public Vertex( String nm )
        {
                _name = nm;
                _routes = new HashMap<String,Edge>(); // Not synchronised
                _things = new Vector<String>();           // Synchronised
        }

}
```