```
    1: /*
    2: * @author Haniel Rameshbabu
    3: */
    4:
    5: package cs3524.solutions.mud;
    6:
    7: import java.rmi.Naming;
    8: import java.lang.SecurityManager;
    9: import java.rmi.server.UnicastRemoteObject;
   10: import java.util.List;
   11: import java.util.Scanner;
   12:
   13: public class Client {
   14:
   15:         public static void main(String args[]) {
   16:                 if (args.length < 3) {
   17:                         System.err.println( "Usage:\njava Client <registryhostname
> <registryport> <callbackport>" ) ;
   18:                         return;
   19:                 }
   20:
   21:                 try {
   22:
   23:                         String hostname = args[0];
   24:                         int registryport = Integer.parseInt( args[1] );
   25:                         int callbackport = Integer.parseInt( args[2] );
   26:
   27:                         System.setProperty( "java.security.policy", "mud.policy" )
 ;
   28:                         System.setSecurityManager( new SecurityManager() ) ;
   29:
   30:                         // User Instance
   31:                         String userName = System.console().readLine("Username: ").
trim();
   32:                         UserImpl user = new UserImpl( userName );
   33:                         UserInterface userstub = (UserInterface)UnicastRemoteObjec
t.exportObject( user, callbackport );
   34:
   35:
   36:                         // Stub
   37:                         String regURL = "rmi://" + hostname + ":" + registryport +
 "/MUDServer";
   38:                         System.out.println("Looking up " + regURL );
   39:                         MUDServerInterface serverstub = (MUDServerInterface)Naming
.lookup( regURL );
   40:
   41:
   42:                         List<String> servers = serverstub.listServers();
   43:                         Integer i = 1;
   44:                         for( String srv : servers )
   45:                         {
   46:                                 System.out.println("("+i+") "+srv);
   47:                                 ++i;
   48:                         }
   49:
   50:                         //choose a server or create your own
   51:                         String chosenServerString = null;
   52:                         boolean response = false;
   53:                         while(chosenServerString == null)
   54:                         {
   55:                                 chosenServerString = System.console().readLine("Co
nnect to server number: ").trim();
   56:                                 if (Integer.parseInt(chosenServerString) <= server
s.size())
   57:                                 {       // you have chosen one of the existing ser
vers
   58:                                         Integer chosenServerInt = Integer.parseInt
(chosenServerString);
   59:                                         --chosenServerInt;      //decrement the va
lue to match index
   60:                                         response = serverstub.joinServer(servers.g
et(chosenServerInt), userstub);
   61:                                         if ( response == false ){System.exit(0);}
   62:                                 }
   63:                                 else {  // invalid input
   64:                                         chosenServerString = null;
   65:                                         System.out.println("Invalid choice! Try ag
ain.");
   66:                                 }
   67:                         }
   68:
   69:
   70:                         // Main interface
   71:                         System.out.println( "Type 'help' for controls/commands" );
   72:                         String input;
   73:                         while (true) {
   74:                                 input = System.console().readLine("\nEnter command
:\n\n").trim();
   75:
   76:                                 // Basic commands sys out
   77:                                 if ( input.equals("help")  ) {
   78:                                         System.out.println( "\nview \nmove \ntake
\nshow inventory \nonline users \nmessage \nexit\n" );
   79:                                 }
   80:
   81:                                 else if ( input.equals( "view" ) )
   82:                                 {       //view waht you have around you
   83:                                         serverstub.view( userName, "paths" );
   84:                                         serverstub.view( userName, "things" );
   85:                                 }
   86:
   87:                                 else if ( input.equals( "move" ) )
   88:                                 {       // move somewhere
   89:                                         System.out.println( "You can move:\n" );
   90:                                         serverstub.view( userName, "paths" );
   91:                                         input = System.console().readLine( "Where
do you want to move?\n" ).trim();
   92:                                         if ( input.equals("north") || input.equals
("east") || input.equals("south") || input.equals("west") )
   93:                                         {
   94:                                                 serverstub.moveUser( userName, inp
ut );
   95:                                         }
   96:                                 }
   97:
   98:                                 else if ( input.equals( "take" ) )
   99:                                 {       // take item around you
  100:                                         serverstub.view( userName, "things" );
  101:                                         input = System.console().readLine( "What w
ould you like to take?\n" ).trim();
  102:                                         serverstub.getThing( userName, input );
  103:                                 }
  104:
  105:                                 else if ( input.equals( "show inventory" ) )
  106:                                 {       //show your items in inventory
  107:                                         serverstub.showInventory( userName );
  108:                                 }
  109:
  110:                                 else if ( input.equals( "online users" ) )
  111:                                 {       // show all online users on the server
  112:                                         serverstub.listUsers( userName );
  113:                                 }
  114:
  115:                                 else if ( input.equals( "message" ) )
```

```
  116:                                    {       // send a message to online user
  117:                                            System.out.println( "You can message to:\n
" );
  118:                                            serverstub.listUsers( userName );
  119:                                            String to = System.console().readLine( "Wr
ite the name:\n" ).trim();
  120:                                            String message = System.console().readLine
( "Write the message:\n" ).trim();
  121:                                            serverstub.message(userName, to, message )
;
  122:                                    }
  123:
  124:
  125:
  126:                                    // Exit
  127:                                    else if ( input.equals( "exit" ) ) {
  128:                                            //serverStub.leaveServer( userName );
  129:                                            System.exit(0);
  130:                                    }
  131:
  132:                                    // Invalid command
  133:                                    else {
  134:                                            System.out.println("Please enter valid com
mand.\n");
  135:                                    }
  136:                            }
  137:
  138:
  139:                    }
  140:                    catch(java.rmi.NotBoundException e) {
  141:                            //System.err.println( "Can't find the auctioneer in the re
gistry." );
  142:                            System.err.println( "Error: server NotBoundException" );
  143:                    }
  144:                    catch (java.io.IOException e) {
  145:                            System.out.println( "Failed to register." );
  146:                    }
  147:            }
  148:
  149: }
```

```
 1: /**********************************************************************
 2:  * cs3524.solutions.mud.Edge
 3:  **********************************************************************/
 4:
 5: package cs3524.solutions.mud;
 6:
 7: // Represents an path in the MUD (an edge in a graph).
 8: class Edge
 9: {
10:     public Vertex _dest;   // Your destination if you walk down this path
11:     public String _view;   // What you see if you look down this path
12:
13:     public Edge( Vertex d, String v )
14:     {
15:         _dest = d;
16:                 _view = v;
17:     }
18: }
19:
```

```
/**************************************************************************
 * 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;
```

```
135:             boolean first = true; // For recording the start location.
136:                 while((line = messages.readLine()) != null) {
137:                     StringTokenizer st = new StringTokenizer( line );
138:                 if( st.countTokens( ) < 2 ) {
139:                     System.err.println( "Skipping ill-formatted line " + line );
140:                     continue;
141:                 }
142:                 String loc = st.nextToken();
143:                 String msg = "";
144:                 while (st.hasMoreTokens()) {
145:                     msg = msg + st.nextToken() + " ";
146:                     }
147:                 changeMessage( loc, msg );
148:                 if (first) {      // Record the start location.
149:                     _startLocation = loc;
150:                     first = false;
151:                 }
152:             }
153:         }
154:         catch( IOException e ) {
155:             System.err.println( "Graph.recordMessages( String " +
156:                                 messagesfile + ")\n" + e.getMessage() );
157:         }
158:     }
159:
160:     /**
161:      * Records the things assocated with vertices in the graph on
162:      * the basis of a file with the following format:
163:      * location thing1 thing2 ...
164:      */
165:     private void recordThings( String thingsfile )
166:     {
167:         try {
168:             FileReader fin = new FileReader( thingsfile );
169:                 BufferedReader things = new BufferedReader( fin );
170:                 String line;
171:                 while((line = things.readLine()) != null) {
172:                     StringTokenizer st = new StringTokenizer( line );
173:                 if( st.countTokens( ) < 2 ) {
174:                     System.err.println( "Skipping ill-formatted line " + line );
175:                     continue;
176:                 }
177:                 String loc = st.nextToken();
178:                 while (st.hasMoreTokens()) {
179:                     addThing( loc, st.nextToken());
180:                     }
181:             }
182:         }
183:         catch( IOException e ) {
184:             System.err.println( "Graph.recordThings( String " +
185:                                 thingsfile + ")\n" + e.getMessage() );
186:         }
187:     }
188:
189:     /**
190:      * All the public stuff. These methods are designed to hide the
191:      * internal structure of the MUD. Could declare these on an
192:      * interface and have external objects interact with the MUD via
193:      * the interface.
194:      */
195:
196:     /**
197:      * A constructor that creates the MUD.
198:      */
199:     public MUD( String edgesfile, String messagesfile, String thingsfile )
200:     {
201:         createEdges( edgesfile );
202:         recordMessages( messagesfile );
203:         recordThings( thingsfile );
204:
205:         System.out.println( "Files read..." );
206:         System.out.println( vertexMap.size( ) + " vertices\n" );
207:         }
208:
209:         // This method enables us to display the entire MUD (mostly used
210:         // for testing purposes so that we can check that the structure
211:         // defined has been successfully parsed.
212:         public String toString()
213:         {
214:         String summary = "";
215:         Iterator iter = vertexMap.keySet().iterator();
216:         String loc;
217:         while (iter.hasNext()) {
218:             loc = (String)iter.next();
219:             summary = summary + "Node: " + loc;
220:             summary += ((Vertex)vertexMap.get( loc )).toString();
221:         }
222:         summary += "Start location = " + _startLocation;
223:         return summary;
224:     }
225:
226:     /**
227:      * A method to provide a string describing a particular location.
228:      */
229:     public String locationInfo( String loc )
230:     {
231:             return getVertex( loc ).toString();
232:     }
233:
234:     /**
235:      * Get the start location for new MUD users.
236:      */
237:     public String startLocation()
238:     {
239:             return _startLocation;
240:     }
241:
242:
243:     /**
244:      * method that provides infor where a player can move
245:      */
246:     public String locationPaths( String loc )
247:     {
248:         String message = getVertex( loc )._msg + "\n";
249:         for (Map.Entry<String, Edge> vertex : getVertex(loc)._routes.entrySet())
250:         {
251:             message += "You can move to the " + vertex.getKey() + " there is " + v
ertex.getValue()._view + "\n";
252:         }
253:         return message;
254:     }
255:
256:     //method that provides info about things on location
257:     public List locationThings( String loc )
258:     {
259:         List<String> things = getVertex(loc)._things;
260:         return things;
261:     }
262:
263:     /**
264:      * Add a thing to a location; used to enable us to add new users.
265:      */
266:     public void addThing( String loc,
267:                           String thing )
```

```
   268:     {
   269:         Vertex v = getVertex( loc );
   270:         v._things.add( thing );
   271:     }
   272:
   273:     /**
   274:      * Remove a thing from a location.
   275:      */
   276:     public void delThing( String loc,
   277:                           String thing )
   278:     {
   279:         Vertex v = getVertex( loc );
   280:         v._things.remove( thing );
   281:     }
   282:
   283:     /**
   284:      * A method to enable a player to move through the MUD (a player
   285:      * is a thing). Checks that there is a route to travel on. Returns
   286:      * the location moved to.
   287:      */
   288:     public String moveThing( String loc, String dir, String thing )
   289:     {
   290:         Vertex v = getVertex( loc );
   291:         Edge e = v._routes.get( dir );
   292:         if (e == null)   // if there is no route in that direction
   293:             return loc;  // no move is made; return current location.
   294:         v._things.remove( thing );
   295:         e._dest._things.add( thing );
   296:         return e._dest._name;
   297:     }
   298:
   299:     /**
   300:      * A main method that can be used to testing purposes to ensure
   301:      * that the MUD is specified correctly.
   302:      */
   303:     public static void main(String[] args)
   304:     {
   305:         if (args.length != 3) {
   306:             System.err.println("Usage: java Graph <edgesfile> <messagesfile> <thin
gsfile>");
   307:             return;
   308:         }
   309:         MUD m = new MUD( args[0], args[1], args[2] );
   310:         System.out.println( m.toString() );
   311:     }
   312: }
```

```
    1: /*
    2: * @author Haniel Rameshbabu
    3: * Implementation of MUDServerInterface
    4: */
    5:
    6: package cs3524.solutions.mud;
    7:
    8: import java.rmi.Remote;
    9: import java.rmi.RemoteException;
   10: import java.util.Iterator;
   11: import java.util.List;
   12: import java.util.Map;
   13: import java.util.ArrayList;
   14: import java.util.Set;
   15: import java.util.HashMap;
   16:
   17:
   18: public class MUDServerImpl implements MUDServerInterface {
   19:
   20:   // Key => Value; name => MUD object
   21:   private Map<String, MUD> servers = new HashMap<String, MUD>();
   22:
   23:   // whereabouts of the User
   24:         private Map<String, String> userMap = new HashMap<String, String>();
   25:   // records of all users positions
   26:         private Map<String, String> userPosMap = new HashMap<String, String>();
   27:
   28:         // server => Hash of all usernames logged
   29:         private Map<String, HashMap<String, UserInterface>> loggedUsers = new Hash
Map<String, HashMap<String, UserInterface>>();
   30:
   31:         // holds records of all user inventories
   32:   // @format InventpryName => InventoryItem
   33:         private Map<String, ArrayList<String>> inventoryMap = new HashMap<String,
ArrayList<String>>();
   34:
   35:   // Create all servers at start; i.e 3;
   36:   public MUDServerImpl() throws RemoteException {
   37:     // Server 1 (Rosa)
   38:     servers.put("Rosa", new MUD("MUDs/rosa/rosa.edg","MUDs/rosa/rosa.msg","MUDs/ro
sa/rosa.thg"));
   39:     // Username => Object
   40:     HashMap<String, UserInterface> RosaObj = new HashMap<String, UserInterface>();
   41:                 loggedUsers.put( "Rosa", RosaObj);
   42:
   43:     // Server 2
   44:     servers.put("Tros", new MUD("MUDs/tros/tros.edg","MUDs/tros/tros.msg","MUDs/tr
os/tros.thg"));
   45:     // Username => Object
   46:     HashMap<String, UserInterface> TrosObj = new HashMap<String, UserInterface>();
   47:     loggedUsers.put( "Tros", TrosObj);
   48:
   49:     // Server 3
   50:     servers.put("Gina", new MUD("MUDs/gina/gina.edg","MUDs/gina/gina.msg","MUDs/gi
na/gina.thg"));
   51:     // Username => Object
   52:     HashMap<String, UserInterface> GinaObj = new HashMap<String, UserInterface>();
   53:     loggedUsers.put( "Gina", GinaObj);
   54:
   55:   }
   56:
   57:   public List<String> listServers() throws RemoteException {
   58:     Set<String> set = servers.keySet();
   59:                 return new ArrayList<String>(set);
   60:   }
   61:
   62:   public boolean joinServer(String servername, UserInterface user) throws RemoteEx
ception {
   63:     MUD server = servers.get(servername);
   64:     String userName = user.getName();
   65:     HashMap<String, UserInterface> userObj;
   66:
   67:
   68:     /*if(loggedUsers.get(name).size() >= maxPlayers) {  // check if maxPlayers is
reached
   69:       client.sendMessage( "Sorry maximum players reached. Try later or join anothe
r server" );
   70:       return false;
   71:     }
   72:
   73:     if ( userMap.get( userName ) != null ) {    // check if userName already exist
. It must be unique
   74:       client.sendMessage( "Change name please! User already exist!" );
   75:       return false;
   76:     }*/
   77:
   78:     // User
   79:     userMap.put( userName, servername );
   80:     userObj = loggedUsers.get( servername );
   81:
   82:     userObj.put( userName, user);
   83:
   84:     loggedUsers.put( servername, userObj );
   85:
   86:     userPosMap.put( userName, server.startLocation() );
   87:
   88:     inventoryMap.put( userName, new ArrayList<String>() );
   89:
   90:     server.addThing( server.startLocation(), "User: "+userName );
   91:
   92:
   93:     // prepare the message
   94:     String message = ( "\n***** Welcome to " + servername + " Server *****\n" );
   95:     message += "Current number of player on this server is "+loggedUsers.get( serv
ername ).size()+"\n";
   96:     message += "You are currently at "+userPosMap.get( userName )+" location\n";
   97:
   98:     // send the message to the client
   99:     user.sendMessage( message );
  100:     return true;
  101:   }
  102:
  103:   // view what is at particular location
  104:   public boolean view( String userName, String what ) throws RemoteException{
  105:     String serverName = userMap.get( userName );
  106:     MUD server = servers.get( serverName );
  107:     HashMap<String, UserInterface> clientsMap = loggedUsers.get( serverName );
  108:     UserInterface client = clientsMap.get( userName );
  109:     String position = userPosMap.get( userName );
  110:     String message = null;
  111:
  112:     if ( what.equals("paths") )
  113:     { // send info about possible paths
  114:       message = server.locationPaths( position );
  115:       client.sendMessage( message );
  116:       return true;
  117:     }
  118:
  119:     if ( what.equals("things") )
  120:     { // send info about things
  121:       message = "There is:\n";
  122:       List<String> things = server.locationThings( position );
  123:       for ( String t : things )
  124:       { // construct the message and send
```

```
125:         message += t + "\n";
126:       }
127:       client.sendMessage( message );
128:       return true;
129:     }
130:     return false;
131:   }
132:
133:   public boolean moveUser(String userName, String position) throws RemoteException
134:   { // move the user
135:     String serverName = userMap.get( userName );
136:     MUD server = servers.get( serverName );
137:     HashMap<String, UserInterface> clientsMap = loggedUsers.get( serverName );
138:     UserInterface client = clientsMap.get( userName );
139:     String origin = userPosMap.get( userName );
140:     String message = "";
141:     // try to move the user and check response
142:     message = server.moveThing( origin, position, "User: "+userName );
143:     userPosMap.put( userName, message );
144:     if ( message.equals( origin ) )
145:     { // user is at the same place because there is no path
146:       client.sendMessage( "You cannot move there.\n" );
147:       return false;
148:     }
149:     client.sendMessage( "You moved to " + message + "\n");
150:     return true;
151:   }
152:
153:
154:
155:   public boolean getThing( String userName, String thing) throws RemoteException
156:   { // client can take a thing but not a user
157:
158:     String serverName = userMap.get( userName );
159:     MUD server = servers.get( serverName );
160:     HashMap<String, UserInterface> clientsMap = loggedUsers.get( serverName );
161:     UserInterface client = clientsMap.get( userName );
162:     ArrayList<String> inventory = inventoryMap.get( userName );
163:     List<String> things = server.locationThings( userPosMap.get( userName ) );
164:
165:     for ( String t : things )
166:     { // iterate through things
167:       if ( thing.equals( t ) && !thing.contains("User:") )
168:       { // check if there is the thing client wants to take
169:         // && check of the thing is not user
170:         server.delThing( userPosMap.get( userName ), t );
171:         inventory.add( t );
172:         inventoryMap.put( userName, inventory );
173:         client.sendMessage( "You have: "+inventory.toString() );
174:         return true;
175:       }
176:     }
177:     // the thing was not there or it was a user
178:     client.sendMessage( "No!\nYou have: "+inventory.toString() );
179:     return false;
180:
181:   }
182:
183:   public boolean showInventory( String userName ) throws RemoteException
184:   { // list all the collected items
185:     String serverName = userMap.get( userName );
186:     MUD server = servers.get( serverName );
187:     HashMap<String, UserInterface> clientsMap = loggedUsers.get( serverName );
188:     UserInterface client = clientsMap.get( userName );
189:     List<String> inventory =  inventoryMap.get(userName);
190:     String message = "In your inventory is:\n";
191:     client.sendMessage( message+inventory.toString() );
192:     return true;
193:   }
194:
195:   public boolean listUsers( String userName ) throws RemoteException
196:   { // list all the online users at the server where client is.
197:     String serverName = userMap.get( userName );
198:     MUD server = servers.get( serverName );
199:     HashMap<String, UserInterface> clientsMap = loggedUsers.get( serverName );
200:     UserInterface client = clientsMap.get( userName );
201:     String message = "\nThese users are online:\n";
202:     Set<String> clientsSet = clientsMap.keySet();
203:     for (String c : clientsSet )
204:     {
205:       message += c+"\n";
206:     }
207:     client.sendMessage( message );
208:     return true;
209:   }
210:
211:   public boolean message( String userName, String to, String message ) throws Remo
teException
212:   { // send a message to a user
213:     String serverName = userMap.get( userName );
214:     MUD server = servers.get( serverName );
215:     HashMap<String, UserInterface> clientsMap = loggedUsers.get( serverName );
216:     UserInterface fromClient = clientsMap.get( userName );
217:     UserInterface toClient = clientsMap.get( to );
218:     String formatedMessage = "Message from " + userName + ":\n" + message;
219:     toClient.sendMessage( formatedMessage );
220:     fromClient.sendMessage( "\nMessage sent\n" );
221:     return true;
222:   }
223:
224:
225: }
```

```
    1: /*
    2: * @author Haniel Rameshbabu
    3: * Remote Interface for MUD game server
    4: */
    5:
    6: package cs3524.solutions.mud;
    7:
    8: import java.rmi.Remote;
    9: import java.rmi.RemoteException;
   10: import java.util.List;
   11:
   12:
   13: public interface MUDServerInterface extends Remote
   14: {
   15:
   16:         public List<String> listServers() throws RemoteException;
   17:         public boolean joinServer( String servername, UserInterface user ) throws
RemoteException;
   18:         //public boolean leaveServer( String userName ) throws RemoteException;
   19:         public boolean view( String userName, String way ) throws RemoteException;
   20:         public boolean moveUser( String userName, String position ) throws RemoteE
xception;
   21:         public boolean getThing( String userName, String thing ) throws RemoteExce
ption;
   22:         public boolean showInventory( String userName ) throws RemoteException;
   23:         public boolean listUsers( String userName ) throws RemoteException;
   24:         public boolean message( String userName, String to, String message ) throw
s RemoteException;
   25: }
```

```
    1: /*
    2: * @author Haniel Rameshbabu
    3: * @title Server Mainline for MUD game server
    4: * @source Adapted Practicals 1 code
    5: */
    6:
    7: package cs3524.solutions.mud;
    8:
    9: import java.io.InputStreamReader;
   10: import java.net.InetAddress;
   11: import java.rmi.Naming;
   12: import java.rmi.RMISecurityManager;
   13: import java.rmi.server.UnicastRemoteObject;
   14:
   15: public class MUDServerMainline {
   16:
   17:
   18:
   19:         public static void main(String args[]){
   20:
   21:                 if (args.length < 2) {
   22:                         System.err.println("Usage: \njava MUDServerMainline <regis
tryport> <serverport>");
   23:                         return;
   24:                 }
   25:
   26:                 try {
   27:                         String hostname = (InetAddress.getLocalHost()).getCanonica
lHostName();
   28:                         int registryport = Integer.parseInt(args[0]);
   29:                         int serverport = Integer.parseInt(args[1]);
   30:
   31:                         // Security Policy
   32:                         System.setProperty("java.security.policy", "mud.policy");
   33:                         System.setSecurityManager( new SecurityManager());
   34:
   35:                         // Generate remote Objects
   36:                         MUDServerImpl server = new MUDServerImpl();
   37:                         MUDServerInterface stub = (MUDServerInterface)UnicastRemot
eObject.exportObject(server, serverport);
   38:
   39:                         String regURL = "rmi://" + hostname + ":" + registryport +
 "/MUDServer";
   40:                         System.out.println("Registering " + regURL);
   41:                         Naming.rebind(regURL, stub);
   42:
   43:                 }
   44:
   45:                 // Error Catching
   46:                 catch(java.net.UnknownHostException e) {
   47:                     System.err.println( "Cannot determine localhost name." );
   48:                     System.err.println( e.getMessage() );
   49:                 }
   50:                 catch (java.io.IOException e) {
   51:             System.err.println( "Failed to register!" );
   52:                     System.err.println( e.getMessage() );
   53:             }
   54:
   55:
   56:         }
   57:
   58: }
```

```
 1: /*
 2: * @author Haniel Rameshbabu
 3: */
 4:
 5: package cs3524.solutions.mud;
 6:
 7: //import java.util.List;
 8: //import java.util.ArrayList;
 9:
10: public class UserImpl implements UserInterface {
11:         private String userName;
12:
13:         public UserImpl( String name ) {
14:                 userName = name;
15:         }
16:
17:         public String getName() {
18:                     return userName;
19:         }
20:
21:         public void sendMessage(String message) {
22:                 System.out.println(message);
23:         }
24: }
```

```
 1: /*
 2: * @author Haniel Rameshbabu
 3: */
 4:
 5: package cs3524.solutions.mud;
 6:
 7: //import java.util.List;
 8: import java.rmi.Remote;
 9: import java.rmi.RemoteException;
10:
11: public interface UserInterface extends Remote
12: {
13:         public String getName() throws RemoteException;
14:         public void sendMessage( String message ) throws RemoteException;
15:
16: }
```

```
    1: /**********************************************************************
    2:  * cs3524.solutions.mud.Vertex
    3:  **********************************************************************/
    4:
    5: package cs3524.solutions.mud;
    6:
    7: import java.util.Map;
    8: import java.util.HashMap;
    9: import java.util.List;
   10: import java.util.Vector;
   11: import java.util.Iterator;
   12:
   13: // Represents a location in the MUD (a vertex in the graph).
   14: class Vertex
   15: {
   16:     public String _name;             // Vertex name
   17:     public String _msg = "";         // Message about this location
   18:     public Map<String,Edge> _routes; // Association between direction
   19:                                      // (e.g. "north") and a path
   20:                                      // (Edge)
   21:     public List<String> _things;     // The things (e.g. players) at
   22:                                      // this location
   23:
   24:     public Vertex( String nm )
   25:     {
   26:         _name = nm;
   27:         _routes = new HashMap<String,Edge>(); // Not synchronised
   28:         _things = new Vector<String>();       // Synchronised
   29:     }
   30:
   31:     public String toString()
   32:     {
   33:         String summary = "\n";
   34:         summary += _msg + "\n";
   35:         Iterator iter = _routes.keySet().iterator();
   36:         String direction;
   37:         while (iter.hasNext()) {
   38:             direction = (String)iter.next();
   39:             summary += "To the " + direction + " there is " + ((Edge)_routes.get(
direction ))._view + "\n";
   40:         }
   41:         iter = _things.iterator();
   42:         if (iter.hasNext()) {
   43:             summary += "You can see: ";
   44:             do {
   45:                 summary += iter.next() + " ";
   46:             } while (iter.hasNext());
   47:         }
   48:         summary += "\n\n";
   49:         return summary;
   50:     }
   51: }
   52:
```