```
    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:
```

```
 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:
```

```
    1: package cs3524.solutions.mud;
    2:
    3: import java.rmi.RMISecurityManager;
    4: import java.rmi.Naming;
    5:
    6: public class MudClient{
    7:   static MudServerInterface server;
    8:   private static String player, currentLocation, mudChoice;
    9:
   10:   public static void main(String args[]) throws Exception {
   11:     if (args.length < 2) {
   12:       System.err.println("Usage: \njava MudClient <host> <port>");
   13:       return;
   14:     }
   15:
   16:     String hostname = args[0];
   17:     int port = Integer.parseInt(args[1]);
   18:
   19:     System.setProperty("java.security.policy", "mud.policy");
   20:     System.setSecurityManager(new RMISecurityManager());
   21:
   22:     try {
   23:       String registryURL = "rmi://" + hostname + ":" + port + "/MudService";
   24:       server = (MudServerInterface) Naming.lookup(registryURL);
   25:       setup();
   26:     } catch (Exception e) {
   27:       System.err.println(e.getMessage());
   28:     }
   29:   }
   30:
   31:   //Sets up the players game
   32:   static void setup() throws Exception{
   33:     //Print list of servers and request a choice from the user
   34:     System.out.println(server.getServers());
   35:
   36:     mudChoice = System.console().readLine("Please select a server: ").toLowerCase(
);
   37:     server.changeMUD(mudChoice);
   38:
   39:     //Request a username from the player and set their starting location
   40:     player = System.console().readLine("Please enter your username: ").toLowerCase
();
   41:     currentLocation = server.getLocation();
   42:
   43:     //If addPlayer() returns true call StartGame(), otherwise print an error messa
ge to the user
   44:     if (server.addPlayer(player)) {
   45:       startGame();
   46:     }else {
   47:       System.out.println("Error connecting " + player + " to " + mudChoice);
   48:     }
   49:   }
   50:
   51:   static void startGame() throws Exception{
   52:     try{
   53:       System.out.println("\nWelcome to " + mudChoice);
   54:       //Print the users location and the command needed to get instructions
   55:       System.out.println(server.status(currentLocation));
   56:       System.out.println("\nTo get instructions, type 'help'.\n");
   57:       String input = "";
   58:
   59:         //Main game loop, ends when user inputs "exit"
   60:         while (!input.equalsIgnoreCase("exit")) {
   61:           input = System.console().readLine("\nWhat would you like to do? ");
   62:           server.changeMUD(mudChoice);
   63:
   64:           //if user inputs help, game instructions are printed
   65:           if (input.equalsIgnoreCase("help")) {
   66:             System.out.println("\nTo exit the game, type 'exit'.");
   67:             System.out.println("To move, type the direction you wish to move in. '
north', 'south', 'east', 'west'.");
   68:             System.out.println("To pickup an item, type 'pickup' and the item you
wish to pickup.");
   69:             System.out.println("To see players at your location type 'players'");
   70:             System.out.println("To see your current location, type 'where'");
   71:           }
   72:
   73:           //if user inputs one of 4 directions, the value of currentLocation is up
dated.
   74:           else if (input.equalsIgnoreCase("north") || input.equalsIgnoreCase("east
") || input.equalsIgnoreCase("south") || input.equalsIgnoreCase("west")) {
   75:             String move = server.move(currentLocation, input.toLowerCase());
   76:
   77:             //if the attempted move doesnt change the users location, inform the u
ser. otherwise update the users location
   78:             if (move.equals(currentLocation)) {
   79:               System.out.println("Cannot move there");
   80:             } else {
   81:               currentLocation = server.move(currentLocation, input.toLowerCase());
   82:               server.updatePlayerLocation(player, currentLocation);
   83:               System.out.println(server.status(currentLocation));
   84:             }
   85:           }
   86:
   87:           //if the user inputs pickup and an item, that item is removed from the c
urrentLocation.
   88:           else if (input.toLowerCase().contains("pickup")) {
   89:             input = input.toLowerCase().replace("pickup ", "");
   90:             System.out.println(server.pickup(currentLocation, input.toLowerCase(),
 player));
   91:           }
   92:
   93:           //Display the all the players at the users location if requested
   94:           else if (input.equalsIgnoreCase("players")) {
   95:             System.out.println("Players at this Location:");
   96:             System.out.println(server.getPlayers(currentLocation));
   97:           } else if (input.equalsIgnoreCase("where")) {
   98:             System.out.println(server.status(currentLocation));
   99:           }
  100:
  101:           //if any other input is received, message is printed informing the user.
  102:           else if (!input.equalsIgnoreCase("exit")) {
  103:             System.out.println("\nInvalid Action");
  104:           }
  105:         }
  106:
  107:       //Removes the player from the mud when they exit
  108:       System.out.println("Goodbye " + player);
  109:       server.delPlayer(player);
  110:     }
  111:     catch(Exception e) {
  112:         System.err.println(e.getMessage());
  113:     }
  114:   }
  115: }
```

```
    1: package cs3524.solutions.mud;
    2:
    3: import java.rmi.RMISecurityManager;
    4:
    5: public class MudServerMainline{
    6:   public static void main(String args[]){
    7:     System.out.println("mainline");
    8:     if (args.length <2){
    9:       System.err.println("Usage: \njava MudServerMainline <registryport> <serverpo
rt>");
   10:       return;
   11:     }
   12:     int registryPort = Integer.parseInt(args[0]);
   13:     int serverPort = Integer.parseInt(args[1]);
   14:
   15:     try{
   16:       String hostname = (java.net.InetAddress.getLocalHost()).getCanonicalHostName
();
   17:
   18:       System.setProperty("java.security.policy", "mud.policy");
   19:       System.setSecurityManager(new RMISecurityManager());
   20:
   21:       MudServerImpl mudServer = new MudServerImpl();
   22:       MudServerInterface mudServerStub = (MudServerInterface)java.rmi.server.Unica
stRemoteObject.exportObject(mudServer, serverPort);
   23:       String regURL = "rmi://" + hostname + ":" + registryPort + "/MudService";
   24:       System.out.println("Registering " + regURL);
   25:       java.rmi.Naming.rebind(regURL, mudServerStub);
   26:
   27:       System.out.println("\nTo create a new MUD, type 'create <name> <edgesfile> <
messagesfile> <thingsfile>' with 1 space between each variable");
   28:       //loop to allow multiple MUDs to be created
   29:       while (true){
   30:         String input = "";
   31:         input = System.console().readLine("\n");
   32:
   33:         //if the user requests to create a mud, split the input into an array so e
ach component of the mud can be added
   34:         if (input.toLowerCase().contains("create")){
   35:           String[] components = input.split(" ");
   36:
   37:           if (mudServer.servers.containsKey(components[1].toLowerCase())){
   38:             System.out.println("Mud with that name aleady exists");
   39:           }
   40:           else if (mudServer.servers.size()<5 || components.length != 5){
   41:             MUD m = new MUD(components[2], components[3], components[4]);
   42:             mudServer.servers.put(components[1].toLowerCase(), m);
   43:             System.out.println("Mud created with name " + components[1].toLowerCas
e());
   44:           }
   45:           else{
   46:             System.out.println("Mud cannot be created");
   47:           }
   48:         }
   49:         else{
   50:           System.out.println("Not a valid Command");
   51:         }
   52:       }
   53:     }catch(Exception e){
   54:       System.err.println(e.getMessage());
   55:     }
   56:   }
   57: }
```

```
/************************************************************************
 * 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 = "";

    public Map<String, String> players = new HashMap<String, String>();

    /**
     * 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;
    }

    /**
     *
     */
    public 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 );
```

```
135:             BufferedReader messages = new BufferedReader( fin );
136:             String line;
137:             boolean first = true; // For recording the start location.
138:             while((line = messages.readLine()) != null) {
139:                 StringTokenizer st = new StringTokenizer( line );
140:                 if( st.countTokens( ) < 2 ) {
141:                     System.err.println( "Skipping ill-formatted line " + line );
142:                     continue;
143:                 }
144:                 String loc = st.nextToken();
145:                 String msg = "";
146:                 while (st.hasMoreTokens()) {
147:                     msg = msg + st.nextToken() + " ";
148:                 }
149:                 changeMessage( loc, msg );
150:                 if (first) {      // Record the start location.
151:                     _startLocation = loc;
152:                     first = false;
153:                 }
154:             }
155:         }
156:         catch( IOException e ) {
157:             System.err.println( "Graph.recordMessages( String " +
158:                                 messagesfile + ")\n" + e.getMessage() );
159:         }
160:     }
161:
162:     /**
163:      * Records the things assocated with vertices in the graph on
164:      * the basis of a file with the following format:
165:      * location thing1 thing2 ...
166:      */
167:     private void recordThings( String thingsfile )
168:     {
169:         try {
170:             FileReader fin = new FileReader( thingsfile );
171:             BufferedReader things = new BufferedReader( fin );
172:             String line;
173:             while((line = things.readLine()) != null) {
174:                 StringTokenizer st = new StringTokenizer( line );
175:                 if( st.countTokens( ) < 2 ) {
176:                     System.err.println( "Skipping ill-formatted line " + line );
177:                     continue;
178:                 }
179:                 String loc = st.nextToken();
180:                 while (st.hasMoreTokens()) {
181:                     addThing( loc, st.nextToken());
182:                 }
183:             }
184:         }
185:         catch( IOException e ) {
186:             System.err.println( "Graph.recordThings( String " +
187:                                 thingsfile + ")\n" + e.getMessage() );
188:         }
189:     }
190:
191:     /**
192:      * All the public stuff. These methods are designed to hide the
193:      * internal structure of the MUD. Could declare these on an
194:      * interface and have external objects interact with the MUD via
195:      * the interface.
196:      */
197:
198:     /**
199:      * A constructor that creates the MUD.
200:      */
201:     public MUD( String edgesfile, String messagesfile, String thingsfile )
202:     {
203:         createEdges( edgesfile );
204:         recordMessages( messagesfile );
205:         recordThings( thingsfile );
206:
207:         System.out.println( "Files read..." );
208:         System.out.println( vertexMap.size( ) + " vertices\n" );
209:     }
210:
211:     // This method enables us to display the entire MUD (mostly used
212:     // for testing purposes so that we can check that the structure
213:     // defined has been successfully parsed.
214:     public String toString()
215:     {
216:         String summary = "";
217:         Iterator iter = vertexMap.keySet().iterator();
218:         String loc;
219:         while (iter.hasNext()) {
220:             loc = (String)iter.next();
221:             summary = summary + "Node: " + loc;
222:             summary += ((Vertex)vertexMap.get( loc )).toString();
223:         }
224:         summary += "Start location = " + _startLocation;
225:         return summary;
226:     }
227:
228:     /**
229:      * A method to provide a string describing a particular location.
230:      */
231:     public String locationInfo( String loc )
232:     {
233:         return getVertex( loc ).toString();
234:     }
235:
236:     /**
237:      * Get the start location for new MUD users.
238:      */
239:     public String startLocation()
240:     {
241:         return _startLocation;
242:     }
243:
244:     /**
245:      * Add a thing to a location; used to enable us to add new users.
246:      */
247:     public void addThing( String loc,
248:                           String thing )
249:     {
250:         Vertex v = getVertex( loc );
251:         v._things.add( thing );
252:     }
253:
254:     /**
255:      * Remove a thing from a location.
256:      */
257:     public void delThing( String loc,
258:                           String thing )
259:     {
260:         Vertex v = getVertex( loc );
261:         v._things.remove( thing );
262:     }
263:
264:     /**
265:      * A method to enable a player to move through the MUD (a player
266:      * is a thing). Checks that there is a route to travel on. Returns
267:      * the location moved to.
268:      */
```

```
  269:     public String moveThing( String loc, String dir)
  270:     {
  271:         Vertex v = getVertex( loc );
  272:         Edge e = v._routes.get( dir );
  273:         if (e == null) {// if there is no route in that direction
  274:                 return loc;  // no move is made; return current location.
  275:         }
  276:         return e._dest._name;
  277:     }
  278:
  279:     /**
  280:      * A main method that can be used to testing purposes to ensure
  281:      * that the MUD is specified correctly.
  282:      */
  283:     public static void main(String[] args)
  284:     {
  285:         if (args.length != 3) {
  286:             System.err.println("Usage: java Graph <edgesfile> <messagesfile> <thin
gsfile>");
  287:             return;
  288:         }
  289:         MUD m = new MUD( args[0], args[1], args[2] );
  290:         System.out.println( m.toString() );
  291:     }
  292: }
```

```
    1: package cs3524.solutions.mud;
    2:
    3: import java.rmi.Remote;
    4: import java.rmi.RemoteException;
    5:
    6: public interface MudServerInterface extends Remote{
    7:    String status(String location) throws RemoteException;
    8:    String move(String location, String direction) throws RemoteException;
    9:    String pickup(String location, String thing, String player) throws RemoteExcept
ion;
   10:    boolean addPlayer(String player) throws RemoteException;
   11:    String getLocation() throws RemoteException;
   12:    String getPlayers(String location) throws RemoteException;
   13:    void updatePlayerLocation(String player, String Location) throws RemoteExceptio
n;
   14:    void delPlayer(String player) throws RemoteException;
   15:    String getServers() throws RemoteException;
   16:    void changeMUD(String mudChoice) throws RemoteException;
   17: }
```

```
 1: package cs3524.solutions.mud;
 2:
 3: import java.util.*;
 4:
 5: public class MudServerImpl implements MudServerInterface {
 6:   private MUD m;
 7:   public Map<String, MUD> servers = new HashMap<String, MUD>();
 8:
 9:   //Add two muds to the game
10:   public MudServerImpl(){
11:     servers.put("wood", new MUD("wood.edg","wood.msg","wood.thg"));
12:     servers.put("beach", new MUD("beach.edg","beach.msg","beach.thg"));
13:   }
14:
15:   //get mud starting location
16:   public String getLocation() {
17:     return m.startLocation();
18:   }
19:
20:   //Adds player to a mud
21:   public boolean addPlayer(String player) {
22:     //If the username already exists return false
23:     if (m.players.containsKey(player))
24:       return false;
25:     //Cap number of players per MUD, if a mud is full return false
26:     if (m.players.size() >= 5){
27:       return false;
28:     }
29:     //add player with a starting location in the mud
30:     else{
31:       m.players.put(player, m.startLocation());
32:       return true;
33:     }
34:   }
35:
36:   //Removes player from their mud
37:   public void delPlayer(String player){
38:     m.players.remove(player);
39:   }
40:
41:   //Retrieve a list of players to display to the user
42:   public String getPlayers(String location) {
43:     ArrayList<String> Players = new ArrayList<String>();
44:     String player;
45:
46:     StringBuilder sb = new StringBuilder();
47:
48:     Iterator itter = m.players.keySet().iterator();
49:
50:     while (itter.hasNext()) {
51:       player = itter.next().toString();
52:       if(m.players.get(player).equalsIgnoreCase(location)){
53:         Players.add(player);
54:         sb.append(player);
55:         sb.append(", ");
56:       }
57:     }
58:     sb.setLength(sb.length() - 2);
59:     return sb.toString();
60:
61:   }
62:
63:   //Update player location when they move
64:   public void updatePlayerLocation(String player, String location){
65:     m.players.remove(player);
66:     m.players.put(player, location);
67:   }
68:
69:   //Returns information about the requested location.
70:   public String status(String location){
71:     return m.getVertex(location).toString();
72:   }
73:
74:   //Moves the player given a location, direction and thing.
75:   public String move(String location, String direction){
76:     return m.moveThing(location, direction);
77:   }
78:
79:   //Removes the item being picked up from it's current location.
80:   public String pickup(String location, String thing, String player){
81:     //if the item does not exist, inform the user
82:     if (!m.getVertex(location)._things.contains(thing)){
83:       return thing + " does not exist or cannot be picked up";
84:     }
85:     //if it does exist, inform the user they have picked up the item
86:     else {
87:       m.delThing(location, thing);
88:       return player + " picked up " + thing;
89:     }
90:   }
91:
92:   //Retrieve list of mud servers available
93:   public String getServers() {
94:     StringBuilder sb = new StringBuilder();
95:
96:     Iterator itter = servers.keySet().iterator();
97:
98:     while (itter.hasNext()) {
99:       sb.append(itter.next().toString());
100:      sb.append(", ");
101:      }
102:    sb.setLength(sb.length() - 2);
103:    return "Servers: " + sb.toString();
104:  }
105:
106:  //change m to the mud the user selected
107:  public void changeMUD(String mudChoice){
108:    m = servers.get(mudChoice);
109:  }
110: }
```