```java
/*******************************************************************************
 * Assessment.mud.cs3524.solutions.mud
 *******************************************************************************/

package mud;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;

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

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

    // User: <UserName, Location>
    public Map<String, String> users = new HashMap<String, String>();

    // 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,
```

```
38                              String destName,
39                              String direction,
40                              String view) {
41          Vertex v = getOrCreateVertex(sourceName);
42          Vertex w = getOrCreateVertex(destName);
43          v._routes.put(direction, new Edge(w, view));
44      }
45
46      /**
47       * Create a new thing at a location.
48       */
49      private void createThing(String loc,
50                               String thing) {
51          Vertex v = getOrCreateVertex(loc);
52          v._things.add(thing);
53      }
54
55      /**
56       * Change the message associated with a location.
57       */
58      private void changeMessage(String loc, String msg) {
59          Vertex v = getOrCreateVertex(loc);
60          v._msg = msg;
61      }
62
63      /**
64       * If vertexName is not present, add it to vertexMap.  In either
65       * case, return the Vertex. Used only for creating the MUD.
66       */
67      private Vertex getOrCreateVertex(String vertexName) {
68          Vertex v = vertexMap.get(vertexName);
69          if (v == null) {
70              v = new Vertex(vertexName);
71              vertexMap.put(vertexName, v);
72          }
73          return v;
74      }
```

```
 75
 76      /**
 77       *
 78       */
 79      protected Vertex getVertex(String vertexName) {
 80          return vertexMap.get(vertexName);
 81      }
 82
 83      /**
 84       * Creates the edges of the graph on the basis of a file with the
 85       * following format:
 86       * source direction destination message
 87       */
 88      private void createEdges(String edgesFile) {
 89          try {
 90              FileReader fin = new FileReader(edgesFile);
 91              BufferedReader edges = new BufferedReader(fin);
 92              String line;
 93              while ((line = edges.readLine()) != null) {
 94                  StringTokenizer st = new StringTokenizer(line);
 95                  if (st.countTokens() < 3) {
 96                      System.err.println("Skipping ill-formatted line " + line);
 97                      continue;
 98                  }
 99                  String source = st.nextToken();
100                  String dir = st.nextToken();
101                  String dest = st.nextToken();
102                  String msg = "";
103                  while (st.hasMoreTokens()) {
104                      msg = msg + st.nextToken() + " ";
105                  }
106                  addEdge(source, dest, dir, msg);
107              }
108          } catch (IOException e) {
109              System.err.println("Graph.createEdges( String " +
110                      edgesFile + ")\n" + e.getMessage());
111          }
```

```
112     }
113
114     /**
115      * Records the messages associated with vertices in the graph on
116      * the basis of a file with the following format:
117      * location message
118      * The first location is assumed to be the starting point for
119      * users joining the MUD.
120      */
121     private void recordMessages(String messagesFile) {
122         try {
123             FileReader fin = new FileReader(messagesFile);
124             BufferedReader messages = new BufferedReader(fin);
125             String line;
126             boolean first = true; // For recording the start location.
127             while ((line = messages.readLine()) != null) {
128                 StringTokenizer st = new StringTokenizer(line);
129                 if (st.countTokens() < 2) {
130                     System.err.println("Skipping ill-formatted line " + line);
131                     continue;
132                 }
133                 String loc = st.nextToken();
134                 String msg = "";
135                 while (st.hasMoreTokens()) {
136                     msg = msg + st.nextToken() + " ";
137                 }
138                 changeMessage(loc, msg);
139                 if (first) {      // Record the start location.
140                     _startLocation = loc;
141                     first = false;
142                 }
143             }
144         } catch (IOException e) {
145             System.err.println("Graph.recordMessages( String " +
146                     messagesFile + ")\n" + e.getMessage());
147         }
148     }
```

```
149
150     /**
151      * Records the things associated with vertices in the graph on
152      * the basis of a file with the following format:
153      * location thing1 thing2 ...
154      */
155     private void recordThings(String thingsFile) {
156         try {
157             FileReader fin = new FileReader(thingsFile);
158             BufferedReader things = new BufferedReader(fin);
159             String line;
160             while ((line = things.readLine()) != null) {
161                 StringTokenizer st = new StringTokenizer(line);
162                 if (st.countTokens() < 2) {
163                     System.err.println("Skipping ill-formatted line " + line);
164                     continue;
165                 }
166                 String loc = st.nextToken();
167                 while (st.hasMoreTokens()) {
168                     addThing(loc, st.nextToken());
169                 }
170             }
171         } catch (IOException e) {
172             System.err.println("Graph.recordThings( String " +
173                     thingsFile + ")\n" + e.getMessage());
174         }
175     }
176
177     /**
178      * All the public stuff. These methods are designed to hide the
179      * internal structure of the MUD. Could declare these on an
180      * interface and have external objects interact with the MUD via
181      * the interface.
182      */
183
184     /**
185      * A constructor that creates the MUD.
```

```
186       */
187      public MUD(String edgesFile, String messagesFile, String thingsFile) {
188          createEdges(edgesFile);
189          recordMessages(messagesFile);
190          recordThings(thingsFile);
191
192          System.out.println("Files read! :)...");
193          System.out.println(vertexMap.size() + " vertices\n");
194      }
195
196      // This method enables us to display the entire MUD (mostly used
197      // for testing purposes so that we can check that the structure
198      // defined has been successfully parsed.
199      public String toString() {
200          String summary = "";
201          Iterator iter = vertexMap.keySet().iterator();
202          String loc;
203          while (iter.hasNext()) {
204              loc = (String) iter.next();
205              summary = summary + "Node: " + loc;
206              summary += ((Vertex) vertexMap.get(loc)).toString();
207          }
208          summary += "Start location = " + _startLocation;
209          return summary;
210      }
211
212      /**
213       * A method to provide a string describing a particular location.
214       */
215      public String locationInfo(String loc) {
216          return getVertex(loc).toString();
217      }
218
219      /**
220       * Get the start location for new MUD users.
221       */
222      public String startLocation() {
```

```
223         return _startLocation;
224     }
225 
226     /**
227      * Add a thing to a location; used to enable us to add new users.
228      */
229     public void addThing(String loc,
230                          String thing) {
231         Vertex v = getVertex(loc);
232         v._things.add(thing);
233     }
234 
235     /**
236      * Remove a thing from a location.
237      */
238     public void deleteThing(String loc,
239                             String thing) {
240         Vertex v = getVertex(loc);
241         v._things.remove(thing);
242     }
243 
244     /**
245      * A method to enable a player to move through the MUD (a player
246      * is a thing). Checks that there is a route to travel on. Returns
247      * the location moved to.
248      */
249     public String moveThing(String loc, String direction, String thing) {
250         Vertex v = getVertex(loc);
251         Edge e = v._routes.get(direction);
252 
253         // if there is no route in that direction
254         if (e == null) {
255             System.out.println("You cannot move that way!");
256             return loc;  // no move is made; return current location.
257         }
258         v._things.remove(thing);
259         e._dest._things.add(thing);
```

```
260             return e._dest._name;
261         }
262 
263         /**
264          * A main method that can be used to testing purposes to ensure
265          * that the MUD is specified correctly.
266          */
267         public static void main(String[] args) {
268             if (args.length != 3) {
269                 System.err.println("Usage: java Graph <edgesfile> <messagesfile> <thingsfile>");
270                 return;
271             }
272             MUD m = new MUD(args[0], args[1], args[2]);
273             System.out.println(m.toString());
274         }
275 }
276 
```