```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/


package mud.cs3524.solutions.mud;

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

public interface ClientInterface extends java.rmi.Remote {
        public String getWorld() throws java.rmi.RemoteException;
        public String setWorld(String newWorld) throws java.rmi.RemoteException;

        public List<String> getThings() throws java.rmi.RemoteException;
        public void carry(String thing) throws java.rmi.RemoteException;
        public Boolean drop(String thing) throws java.rmi.RemoteException;

        public void printmess(String mess) throws java.rmi.RemoteException;

        public String getName() throws java.rmi.RemoteException;

        public String getLocation() throws java.rmi.RemoteException;

        public String setLocation(String newlocation) throws java.rmi.RemoteExce
ption;
        public void print(String output) throws java.rmi.RemoteException;
}
/*declaring all the functions on the client side, extending java rmi remote*/
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/
package mud.cs3524.solutions.mud;

import java.util.List;
import java.util.ArrayList;
/* implementation of all the functions defined in the interface, all the functio
n that
I need for client to work, as well as print function, when i want something to p
rint
on the clients side.
*/
public class Client implements ClientInterface {
        private String name;
  private String location;
  private String world;
  private List<String> things = new ArrayList<String>();

  public Client(String pname, String startingLocation) {
        name = pname;
        location = startingLocation;
  }// defines player by name and assigned starting location of the world


  public String getName() {
        return name;
  }// getname function, used in messaging

  public void printmess(String mess){
    System.out.println(mess);

  }//print message function, used for both global and private messaging

        public String getLocation() {
        return location;
        }// to get location of a player in the world


  public String setLocation(String newLocation) {
        return location = newLocation;
  }// to move player in the world

  public void print(String output){

    System.out.print(output);

  } // to print players status after each move, the same as printmess, but its m
ore readible this way

  public String getWorld() {
    return world;
  }//to find out in which world a player is

  public String setWorld(String newWorld) {
    return world = newWorld;
  }//to spawn player in world

  public List<String> getThings() {
    return things;
```



```
  }//to get things at given location

  public void carry(String thing) {
    things.add(thing);
  }// to pick up things, and add them to a bag

  public Boolean drop(String thing) {
    if (things.contains(thing)) {
      things.remove(thing);
      return true;
    } else { return false; }
  }// to drop things
}
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/


package mud.cs3524.solutions.mud;

import java.rmi.Naming;
import java.lang.SecurityManager;
import java.rmi.server.UnicastRemoteObject;
import java.io.*;
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;
import java.util.concurrent.TimeUnit;

/* the usual way to connect to server, as we learned on the practicals, user wil
l be asked
to enter name, and pick a game world, there are two predefined, Hell and Heaven,
 or the user
can create his own on runtime, then there is a 'infinite' while loop, where the
code waits
for input from user, that can be either pick a thing, drop a  thing, go 'somewhe
re', messageall, or message 'someuser'
or user can exit by typing quit or exit.
*/


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

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

            System.setProperty( "java.security.policy", "rmishout.policy" ) ;
            System.setSecurityManager( new SecurityManager() ) ;
            //using rmishout security policy from first practical, grants all pr
iviledges
            System.out.println("What is your name?");
                Scanner pn= new Scanner(System.in);
                String pname= pn.nextLine();
                //enter name
        Client client = new Client(pname,null);
        //creates client
            ClientInterface clientstub = (ClientInterface)UnicastRemoteObject.ex
portObject( client, callbackport );
            //creates stub
            String regURL = "rmi://" + hostname + ":" + registryport + "/GameServerM
ainline";
            GameServerInterface gamestub = (GameServerInterface)Naming.lookup( r
egURL );
```



```
            //local host
            List<String> worlds = gamestub.getWorlds();
            for(String world : worlds) {
                    System.out.println(world);
            }//list of worlds on the server

            Boolean customWorldsAllowed = worlds.size() < 3;
            if (customWorldsAllowed == true) {
                    System.out.println("Custom");
            }       else {
                    System.out.println("The world server is currently at maximum capacity"
);
            }//3 custom worlds are max allowed


            String world = null;
            while(world == null) {
                    world = System.console().readLine("Choose World: ").trim();
                    world = world.substring(0, 1).toUpperCase() + world.subs
tring(1);
                    if (world.equals("Custom") && customWorldsAllowed == tru
e) {
                            world = gamestub.createWorld(buildWorld());
            }       else if (!worlds.contains(world)) {
                    world = null;
                    System.out.println("That is not a valid world. Try again");
                    }
            }//promt to choose a world
            System.out.println("Joining: " + world);

            if (gamestub.getPlayers() > 2){
                    System.out.println("Maximum number of players reached");
                    TimeUnit.SECONDS.sleep(1);
                    System.exit(0);
            }//terminates if more than 3 players joined
            //System.out.println(gamestub.getPlayers());

            gamestub.spawn(clientstub, world); //connects client to a world
of his choice

            String input = "";
            Boolean update = true;
            while(true){
                    input = System.console().readLine();
                    if ((input.contains("exit")) || (input.contains("quit"))){
                            update = gamestub.removePlayer(clientstub);
                            break;

                    } //input from keyboard
                    if (input.contains("pick")) {
                            input = input.replace("pick ", "").trim(); //repl
aces pick with '' and trims the spaces => gets just the input ei. 'pen' insted o
f drop pen.
                            update = gamestub.pick(clientstub, input);
                    } else if (input.contains("drop")) {
                            input = input.replace("drop ", "").trim();
                            update = gamestub.drop(clientstub, input);
                    } else if (input.contains("messageall")) {
                            input = input.replace("messageall ", "").trim();
                            update = gamestub.messaging(clientstub, input);
                    } else if (input.contains("message")) {
                            input = input.replace("message ", "").trim();
```

```
                                String who = input.split(" ")[0].trim();
                                String text = input.replace(who, "").trim();
                                update = gamestub.messagingsomeone(clientstub, w
ho, text);
                        }else if (input.contains("go")) {
                                input = input.replace("go ", "").trim();
                                update = gamestub.move(clientstub, input);
                        }else if (input.contains("?")) {
                                System.out.println(" go <somewhere>\n pick <thing>\n drop
<thing>\n messageall <textOFmessage>\n message <player> <text> ");
                        }else {

                                System.out.println(" wrong command\n for help try '?' ");
                        }
                }
                System.out.println("exiting...");
                System.exit(0);
                }
                catch (Exception e) { System.err.println(e.getMessage()); }

        }


    //world creation functions
    private static List<String> buildWorld() {
    System.out.println("Welcome to World Creator!");
    System.out.println("Give your world a name like this:");
    System.out.println("name: <examplename>");
    System.out.println("Add links between places like this:");
    System.out.println("edge: A <something> B action description");
    System.out.println("Give places names like this:");
    System.out.println("message: A <name>");
    System.out.println("Place starting objects like this:");
    System.out.println("thing: <nameofthething>");
    System.out.println("Set a starting location like this:");
    System.out.println("start: <A,B,C,D>");
    System.out.println("When you're done type exit");
    System.out.println("Remember:You must complete all the attributes");

    String input = "";
    List<String> markup = new ArrayList<String>();
    while((!input.equals("exit"))
        || (markupContains(markup, "name: ").equals(false))
        || (markupContains(markup, "start: ").equals(false))
        || (markupContains(markup, "edge: ").equals(false))
        || (markupContains(markup, "message: ").equals(false))) {
      input = System.console().readLine("âM–^^¤ ").trim();
      markup.add(input);
    }
    System.out.println("done building");
    return markup;
  }

  public static Boolean markupContains(List<String> markup, String subString) {
    Boolean result = false;
    for(String statement : markup) {
      if (statement.contains(subString)) { result = true; }
    }
    return result;
  }
}
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/
package mud.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;
    }
}
/* file that was provided*/
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/
package mud.cs3524.solutions.mud;

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


/*implementation of all the functions needed on the serverside such as status, w
hich gives back
information about user location, and what is around him, the move function, the
drop function, the spawn,
and both the messaging functions
*/

public class GameServerImplementation implements GameServerInterface {
  public Map<String, World> worlds;
  List<ClientInterface> clients;

  public GameServerImplementation() {
    clients = new ArrayList<ClientInterface>();
    worlds = new HashMap<String, World>();

    worlds.put("Hell", new World("hell.world"));
    worlds.put("Heaven", new World("heaven.world"));
  }

  public List<String> getWorlds() {
    return new ArrayList<String>(worlds.keySet());
  }//returns the worlds on the server

  public int getPlayers(){
    int number = 0;
    for (ClientInterface dude : clients){
      number++;

    }
    return number;
  }//returns number of players on server

  public String createWorld(List<String> markup) {
    String name = "";
    if (worlds.size() < 3) {
      World world = new World(markup);
      worlds.put(world._name, world);
      name = world._name;
    }
    return name;
  }

  public void exportWorlds() {
    System.out.println("Exporting worlds...");
    for(World world : worlds.values()) {
      world.export();
    }
  }
```



```
      public void status(ClientInterface client, String message) {
    try {

                  String status = "######\nWorld:" + client.getWorld() + "\n";
                  status += worlds.get(client.getWorld()).locationStatus(client.
getLocation());
                  List<String> things = new ArrayList<String>(worlds.get(client.
getWorld()).locationThings(client.getLocation()));
                  things.remove(client.getName());

                  String strThings = "";
                  String strPlayers = "";
                  String strBag = "";

                  for(String thing : things) {
                    if (isClient(thing).equals(true)) {
                      strPlayers += thing + " ";
                    } else {
                      strThings += thing + " ";
                    }
                  }

                  for(String thing : client.getThings()) {
                    strBag += thing + " ";
                  }

                  if (!strThings.equals("")) {
                    status += "Objects:" + " " + strThings + "\n";
                  }

                  if (!strPlayers.equals("")) {
                    status += "Players:" + " " + strPlayers + "\n";
                  }

                  if (!strBag.equals("")) {
                    status += "Bag:" + " " + strBag + "\n";
                  }

                  if (!message.equals("")) {
                    status += "Message:" + " " + message + "\n";
                  }

      client.print(status);
    } catch (Exception e) { System.out.println("status: " + e); }
        }//the function for the information about player, gathers everything fro
m the players, if there is some
      //one else at this location it lets you know, send the status string for p
rint on the client side

      public Boolean spawn(ClientInterface client, String world) {
    try {
      clients.add(client);
      client.setWorld(world);
      client.setLocation(worlds.get(client.getWorld()).startLocation());
      worlds.get(client.getWorld()).addThing(worlds.get(client.getWorld()).start
Location(), client.getName());
      updatePlayers(worlds.get(client.getWorld()).startLocation(), client.getWor
ld(), "");
      System.out.println(client.getName() + " is in " + client.getWorld() + " world...
");
    } catch (Exception e) { System.out.println("spawn: " + e); return false; }
```

```
    return true;
        }//function to create a player in the world and updates the number of pl
ayers there

  public Boolean pick(ClientInterface client, String thing) {
    Boolean response  = true;
    try {
      if (isClient(thing).equals(false) && worlds.get(client.getWorld()).locatio
nThings(client.getLocation()).contains(thing)) {
        worlds.get(client.getWorld()).delThing(client.getLocation(), thing);
        client.carry(thing);
        updatePlayers(client.getLocation(), client.getWorld(), "");
      } else {
        status(client, "Invalid Item Selection");
        response = false;
      }
    } catch (Exception e) { System.out.println("pick: " + e); return false; }
    return response;
  }//to pick up a thing at the location

  public Boolean drop(ClientInterface client, String thing) {
    Boolean response  = true;
    try {
      Boolean has = client.drop(thing);
      if (has.equals(true)) {
        worlds.get(client.getWorld()).addThing(client.getLocation(), thing);
        updatePlayers(client.getLocation(), client.getWorld(), "");
      } else {
        status(client, ("You do not have a " + thing + " to drop."));
        response = false;
      }
    } catch (Exception e) { System.out.println("drop: " + e); return false; }
    return response;
  }//to drop a thing
  public Boolean messaging (ClientInterface client, String text){
    try{
      for (ClientInterface dude : clients){
      dude.printmess("Global messaging:" + " " + client.getName() + " said: " + text);
      //System.out.println(text);
    }
    }//sends message to global chat

    catch (Exception e) { System.err.println(e.getMessage()); return false; }
    return true;
  }

  public Boolean messagingsomeone (ClientInterface player, String who, String te
xt){
    try{
      for (ClientInterface client : clients)  {
          if (client.getName().equals(who)) {
            client.printmess(player.getName() + " said: " + text);
            return true;
          }
      }
    }
    catch (Exception e) { System.err.println(e.getMessage()); return false; }
    return true;
  }//sends message to a single player

        public Boolean move(ClientInterface client, String dir) {
    try{
```



```
      String endpoint = worlds.get(client.getWorld()).moveThing(client.getLocati
on(), dir, client.getName());
      //updatePlayers(client.getLocation(), client.getWorld(), "");
      client.setLocation(endpoint);
      updatePlayers(endpoint, client.getWorld(), "");
    } catch (Exception e) { System.out.println("pick: " + e); return false; }
    return true;
        }//moves player from position to position

  private void updatePlayers(String location, String world, String message) {
    try {
      for(ClientInterface client : clients)  {
        if (client.getLocation().equals(location) && client.getWorld().equals(wo
rld)) {
          status(client, message);

        }
      }
    } catch (Exception e) { System.out.println("updatePlayers: " + e); }
        }

  private Boolean isClient(String thing) {
    try {
      for(ClientInterface client : clients)  {
        if (client.getName().equals(thing)) {
          return true;
        }
      }
    } catch (Exception e) { System.out.println("isPlayer: " + e); }
    return false;
        }//checks if an object is a player

  public Boolean removePlayer(ClientInterface player) {
    try {
      String pname = player.getName();
      String pWorld = player.getWorld();
      clients.remove(player);
      System.out.println(pname + " left " + pWorld);
      for (ClientInterface dude : clients){
      dude.printmess(pname + " has left the game");
      //System.out.println(text);
      }

    }
    catch (Exception e) { System.out.println("removePlayer: " + e); return false; }

    return false;
  }



}
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/
package mud.cs3524.solutions.mud;

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

public interface GameServerInterface extends Remote
{
        public Boolean messaging(ClientInterface client, String text) throws jav
a.rmi.RemoteException;
        public List<String> getWorlds() throws java.rmi.RemoteException;
        public int getPlayers() throws java.rmi.RemoteException;
        public Boolean pick(ClientInterface client, String thing) throws java.rm
i.RemoteException;
        public String createWorld(List<String> markup) throws java.rmi.RemoteExc
eption;
        public void exportWorlds() throws java.rmi.RemoteException;
        public Boolean drop(ClientInterface client, String thing) throws java.rm
i.RemoteException;
        public void status(ClientInterface client, String message) throws java.r
mi.RemoteException;
        public Boolean spawn(ClientInterface client, String world) throws java.r
mi.RemoteException;
        public Boolean move(ClientInterface client, String dir) throws java.rmi.
RemoteException;
        public Boolean messagingsomeone(ClientInterface client, String who, Stri
ng text) throws java.rmi.RemoteException;
        public Boolean removePlayer(ClientInterface player) throws java.rmi.Remo
teException;
}
/* declaring all the functions needed on the Server side, the functions messagin
g and messaging someone are the ones that are additional for the A4-A1*/
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/


package mud.cs3524.solutions.mud;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.net.InetAddress;

/*the server mainline, binds as a stub to the rmi adress so it can be called by
client, hostaname is set to
localhost, security is again rmishout that grants all priviledges
*/
public class GameServerMainline
{
    public static void main(String args[])
    {
        if (args.length < 2) {
            System.err.println( "Usage:\njava GameServerMainline <registryport> <serverport>" )
;
            return;
        }

        try {
            String hostname = (InetAddress.getLocalHost()).getCanonicalHostName(
) ;
            int registryport = Integer.parseInt( args[0] ) ;
            int serverport = Integer.parseInt( args[1] ) ;

            System.setProperty( "java.security.policy", "rmishout.policy" ) ;
            System.setSecurityManager( new RMISecurityManager() ) ;

        GameServerImplementation serv = new GameServerImplementation();

        GameServerInterface stub = (GameServerInterface)java.rmi.server.UnicastR
emoteObject.exportObject( serv, serverport );
        Naming.rebind( "rmi://" + hostname + ":" + registryport + "/GameServerMainline"
, stub );

        System.out.println("Server is running");

        }
        catch(java.net.UnknownHostException e) {
            System.err.println( "It seems that Java can't determine the local host!" );
        }
        catch (java.io.IOException e) {
            System.out.println( "Failed to register." );
        }
    }
}
//java mud.cs3524.solutions.mud.GameServerMainline 50011 50012
//java mud.cs3524.solutions.mud.ClientMainline localhost 50011 50013
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/

package mud.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
    }

}

/*another file that was provided*/
```

```
/*
 @author: Martin, 51444972
 @version: 1.0.1

*/



package mud.cs3524.solutions.mud;

import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.Scanner;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Vector;
import java.util.HashMap;
import java.io.File;
import java.io.PrintWriter;
import java.util.Arrays;

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

public class World
{
    /**
     * Private stuff
     */
    public String _name = "Generic World";
    // 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;
    }

    public String startLocation(){
        return _startLocation;
    }
    /**
     *
     */
    private Vertex getVertex( String vertexName )
    {
        return vertexMap.get( vertexName );
    }
    private void saveEdge(String[] attributes) {
        String message = "";
        for(String n: Arrays.copyOfRange(attributes, 4, attributes.length)) mess
age += n + " ";
        addEdge(attributes[1], attributes[3], attributes[2], message.trim());
    }

    private void saveMessage(String[] attributes) {
        String message = "";
        for(String n: Arrays.copyOfRange(attributes, 2, attributes.length)) mess
age += n + " ";
        changeMessage(attributes[1], message.trim());
    }

    private void saveThing(String[] attributes) {
        String message = "";
        for(String n: Arrays.copyOfRange(attributes, 2, attributes.length)) mess
age += n + " ";
        addThing(attributes[1], message.trim());
    }
    private void loadRecord(String record) {
        String[] attributes = record.split(" ");
        if (attributes[0].equals("name:"))
            _name = attributes[1];
        if (attributes[0].equals("edge:"))
            saveEdge(attributes);
        if (attributes[0].equals("message:"))
```

```
            saveMessage(attributes);
        if (attributes[0].equals("thing:"))
            saveThing(attributes);
        if (attributes[0].equals("start:"))
            _startLocation = attributes[1];
    }
    /**
     * 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 World( String worldfile )
    {
        try {
        Scanner sc = new Scanner(new File(worldfile));
        while (sc.hasNextLine()) {
            loadRecord(sc.nextLine());
        }
      }
```

```
        catch(java.io.FileNotFoundException e) { System.out.println(e.getMessage()
); }

        System.out.println("Imported: " + _name + " (" + vertexMap.size() + " locations)")
;
    }

    public World(List<String> markup) {
        for(String statement : markup) {
        loadRecord(statement);
        }
    }

    // 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();
    }
    public String locationStatus(String loc) {
        String summary = getVertex(loc)._msg + "\n";
        for(Map.Entry<String, Edge> vertex : getVertex(loc)._routes.entrySet())
{
            summary += "To the " + vertex.getKey() + " there is " + vertex.getValue().
_view + "\n";
        }
        return summary;
    }
    /**
     * Get the start location for new MUD users.
     */

    public List<String> locationThings(String loc) {
        return getVertex(loc)._things;
    }
    /**
     * 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;
    }

    public void export() {
    System.out.println("Saving: " + _name);
    try {
      PrintWriter writer = new PrintWriter(_name.toLowerCase() + ".world", "UTF-8
");
      // name
      writer.println("name: " + _name + "\n");
      // edges
      for(Map.Entry<String, Vertex> vertex : vertexMap.entrySet()) {
        for(Map.Entry<String, Edge> route : getVertex(vertex.getKey())._routes.e
ntrySet()) {
          writer.println("edge: " + vertex.getKey() + " " + route.getKey() + " " +
route.getValue()._dest._name + " " + route.getValue()._view);
        }
      }
      writer.println("");
      // messages
      for(Map.Entry<String, Vertex> vertex : vertexMap.entrySet()) {
        writer.println("message: " + vertex.getKey() + " " + vertex.getValue()._msg
);
      }
      writer.println("");
      // things
      for(Map.Entry<String, Vertex> vertex : vertexMap.entrySet()) {
        for(String thing : vertex.getValue()._things) {
          writer.println("thing: " + vertex.getKey() + " " + thing);
        }
      }
      // start
      writer.println("\nstart: " + _startLocation);
      writer.close();
    }
    catch(java.io.FileNotFoundException e) { System.out.println(e.getMessage());
 }
    catch(java.io.UnsupportedEncodingException e) { System.out.println(e.getMess
```

```
age()); }
  }
    /**
     * A main method that can be used to testing purposes to ensure
     * that the MUD is specified correctly.
     */
}
```