

```

/*****
 * mud.MUD
 *****/

```

```
package 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; essentially, this is a
 * graph.
 */
```

```
public class MUD
{
```

```
    /**
     * Private stuff
     */
```

```
    // record of all players currently logged in to this MUD
    // stores their name and location
    public Map<String, String> players = 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,
                          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 format:
 * 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 associated 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 associated with vertices in the graph on
 * the basis of a file with the following format:
 * location thing1 thing2 ...
 */
private void recordThings( String thingsfile )
{
    try {
        FileReader fin = new FileReader( thingsfile );
        BufferedReader things = new BufferedReader( fin );
        String line;
        while((line = things.readLine()) != null) {
            StringTokenizer st = new StringTokenizer( line );
            if( st.countTokens( ) < 2 ) {
                System.err.println( "Skipping ill-formatted line " + line );
                continue;
            }
            String loc = st.nextToken();
            while (st.hasMoreTokens()) {
                addThing( loc, st.nextToken());
            }
        }
    }
}

```

```

        catch( IOException e ) {
            System.err.println( "Graph.recordThings( String " +
                               thingsfile + ")\n" + e.getMessage() );
        }
    }

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

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

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

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

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

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

/**
 * Add a thing to a location; used to enable us to add new users.
 */

```

```

public void addThing( String loc,
                     String thing )
{
    Vertex v = getVertex( loc );
    v._things.add( thing );
}

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

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

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

```