```java
package cs3524.solutions.mud;

import java.io.*;
import java.net.InetAddress;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class MUDServerMainline
{
        static BufferedReader in = new BufferedReader( new InputStreamReader (System.in));

        public static void main (String args[])
        {
                //Prevents user from starting up the server without having entered a host and port number
                if (args.length < 2)
                {
                        System.err.println("Missing arguments. PLease specify both <host> <port>");
                        return;
                }

                //Variable that contains the port number
                int registryPort = Integer.parseInt(args[0]);

                //Variable that contains the hpst number
                int serverPort = Integer.parseInt(args[1]);

                //Give the user some feedback so that they know the system is still operational
                System.out.println("Starting server on port " + Integer.toString(registryPort) + "...");

                try
                {
                        //Make hostName equall to localhost
                        String hostName = (InetAddress.getLocalHost()).getCanonicalHostName();

                        //Specify security policy
                System.setProperty( "java.security.policy", "muddy.policy" ) ;
                System.setSecurityManager( new RMISecurityManager() ) ;

                        //Create an instance of MUDServiceImpl()
                        MUDServiceImpl mudService = new MUDServiceImpl();

                        //Create a stub for mudService
                        MUDService mudstub = (MUDService)UnicastRemoteObject.exportObject(mudService, serverPort);

                        //Buile the url
                        String regUrl = "rmi://" + hostName + ":" + registryPort + "/MudService";

                        try
                        {
                                //Bind the mudstub to that url
                                Naming.rebind(regUrl, mudstub);
                        }
                        catch (Exception e)
                        {
                                //If the bind failed print the exception message
                                System.out.println(e.getMessage());
                        }

                        //Notify the user that the server is now running & where it is running
                        System.out.println("Server running at: " + regUrl);

                        //More text for the user to indicate system progress
                        System.out.println("Creating default MUD...");

                        //Call createMud() which creates an instance of MUD
                        mudService.createMUD("default");
                }
                catch(Exception b)
                {
                        //If there was a problem with instance creation then alert the user
                        System.err.println(b.getMessage());
                }
        }
}

//TO RUN: RUN FROM mud(1)                java cs3524.solutions.mud.MUDServerMainlin50010 50011
```

```java
/***********************************************************************
 * 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> users = 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.
     */
    public 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 );
            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 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> <thin
gsfile>");
            return;
        }
        MUD m = new MUD( args[0], args[1], args[2] );
        System.out.println( m.toString() );
    }
}
```

```java
package cs3524.solutions.mud;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface MUDService extends Remote
{
    public void createMUD(String mudName) throws RemoteException;

    public String getStartLocation() throws RemoteException;

    public String location(String location) throws RemoteException;

    public String locationInfo( String loc ) throws RemoteException;

    public String move(String current, String direction) throws RemoteException;

    public String pickMUD(String inputMud) throws RemoteException;

    public boolean pickUp(String item, String location) throws RemoteException;

    public void refreshLocation(String username, String location) throws RemoteExc
eption ;

    public String welcome() throws RemoteException;

    public String whosThere(String location) throws RemoteException;
}
```

```java
package cs3524.solutions.mud;

import java.rmi.*;
import java.util.*;

public class MUDServiceImpl implements MUDService
{
        private MUD mudInstance;
    public Map<String, MUD> MUDs = new HashMap<String, MUD>();
    public Integer mudLimiter = 4;
    public Integer mudCounter = 0;

        public MUDServiceImpl()    throws RemoteException
    {
        //Boilerplate as there is nothing to construct
        }

        public void createMUD(String mudName) throws RemoteException
    {
                try
        {
                        //Try and make a instance of MUD In MUD.java
            if(mudCounter == mudLimiter)
            {
                System.out.println("Too many muds created, you can’t create anymore.");

            }

            else
            {
                MUDs.put(mudName, new MUD("mymud.edg","mymud.msg","mymud.thg"));
                System.out.println("MUD " + mudName + " created");
                mudCounter = mudCounter + 1;
            }
                }

        catch (Exception ex)
        {
                        System.err.println("Error creating MUD. Error details: " + ex.getMessage());
                }

        }

    public String welcome() throws RemoteException
    {
        String output = "";
        output = ("----- MUDs -----\n");

        for(Map.Entry<String, MUD> entry : MUDs.entrySet())
        {
            String key = entry.getKey();
            output += (key + "\n");
        }

        output += ("---------------\n");
        output += ("Select a MUD to connect to: ");

        return output;
    }

    public String pickMUD(String inputMud) throws RemoteException
    {

        String output = "";

        if(inputMud != "")
        {
            mudInstance = MUDs.get(inputMud);
            output = ( "Welcome to " + inputMud + " MUD Server!\n" );
            output += ( "Please enter a username: " );
        }

        return output;
    }


        public String getStartLocation() throws RemoteException
    {
        //Used to get the position of an instantiated user
        return mudInstance.startLocation();
    }



    public String location(String location) throws RemoteException
    {
        //Asks for the location of a user in a MUD world
        return mudInstance.getVertex(location).toString();
    }


    public String move(String current, String direction) throws RemoteException
    {
        //Allows users to traverse the MUD enviroment given the start location and the desired location
        //User will move if the direction is valid given the location
        Vertex currentVertex = mudInstance.getVertex(current);

        if(currentVertex._routes.containsKey(direction))
        {
            Edge newLocation = currentVertex._routes.get(direction);
            Vertex newVert = (newLocation._dest);
            return newVert._name;
        }

        else
        {
            return current;
        }
        }


    public void refreshLocation(String username, String location) throws RemoteException
    {
        //Remove the user from a location
        mudInstance.users.remove(username);
         //Add them to the new location
        mudInstance.users.put(username, location);

        //System.out.println(mudInstance.users);
        }


    public String whosThere(String location) throws RemoteException
    {

        //Creare an array of all players playing
        ArrayList<String> Players = new ArrayList<String>();
        //Instantiate a variable for username
        String username;

        StringBuilder usernameList = new StringBuilder();

        Iterator i = mudInstance.users.keySet().iterator();
        //Keep adding users who are at a given location, until there is no more users at location
        while (i.hasNext())
        {
                username = i.next().toString();

            if(mudInstance.users.get(username).equalsIgnoreCase(location))
            {
                Players.add(username);
                usernameList.append(username);
                usernameList.append(", ");
            }

            }

        usernameList.setLength(usernameList.length() - 2);

        //return the string of all players at that location
        return "You can see: " + usernameList.toString();

    }


    public String locationInfo( String location )
    {
    //Return the correct message, given your location, from Mud.thg
        return mudInstance.getVertex( location ).toString();
    }


    public boolean pickUp(String item, String location) throws RemoteException
    {
        Vertex currentVertex = mudInstance.getVertex(location);
        //Get all items at current location
        List<String> things = currentVertex._things;
        //If there is something at that location
        if(things.contains(item))
        {
            //Remove it
            mudInstance.delThing(location, item);

            if(location.equals("D"))
            {
                //Change the message at location D to indicate that the tresure is no longer here
                mudInstance.changeMessage(location, "Looks like there used to be treasure here");
            }

            else if(location.equals("A"))
            {
                //Change the message at location A to indicate that the ring is no longer here
                mudInstance.changeMessage(location, "Looks like there used to be a ring here");
            }

            return true;
         }

        return false;
    }
}
```

```java
package 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
    }

    public String toString()
    {
        String summary = "\n";
        summary += _msg + "\n";
        Iterator iter = _routes.keySet().iterator();
        String direction;
        while (iter.hasNext()) {
            direction = (String)iter.next();
            summary += "To the " + direction + " there is " + ((Edge)_routes.get(
direction ))._view + "\n";
        }
        iter = _things.iterator();
        if (iter.hasNext()) {
            summary += "You can see: ";
            do {
                summary += iter.next() + " ";
            } while (iter.hasNext());
        }
        summary += "\n\n";
        return summary;
    }
}

```

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

package 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;
    }
}

```

```java
package cs3524.solutions.mud;

import java.rmi.*;
import java.util.*;
import java.io.*;;
import java.net.InetAddress;

public class MUDClient
{

        //Creates an instance of MUDService called service
        static MUDService service;

        //Variable declarations
        static BufferedReader in = new BufferedReader( new InputStreamReader( System.in ));
    private static String username;
    private static String location;
    private static String nameOfMUD;

        public static void main(String args[]) throws Exception
        {
                // An if statement that check if the user has provided a valid host and port
                if(args.length < 2)
                {
                        System.err.println("Missing arguments. PLease specify both <host> <port>");
                        return;
                }

                //The first argument will be made equall to hostname
                String hostName = args[0];

                //The seccond argument will be made equall to port
                int port = Integer.parseInt(args[1]);

                //Specify the security policy and set security manager
                System.setProperty( "java.security.policy", "muddy.policy" ) ;
        System.setSecurityManager( new RMISecurityManager() ) ;


                try
                {
                        //Create registration URL from hostname, port
                        String regUrl = "rmi://" + hostName + ":" + port + "/MudService";
                        service = (MUDService)Naming.lookup(regUrl);

                        //Once connected call function startUp()
                        startUp();
                }
                catch (java.io.IOException e)
                {
                        System.err.println("Input error");
                        System.err.println(e.getMessage());
                }
        }

        static void startUp() throws Exception
        {

                //Print out the String returned by intoroduction function in MUDServiceImpl.java
                System.out.println(service.welcome());
                nameOfMUD = in.readLine();
                try
                {
                        System.out.println(service.pickMUD(nameOfMUD));

                        //Ask user to set username
                        username = in.readLine();

                        //Use the String returned by getStartLocation from MUDServiceImpl to set the value for location
                        location = service.getStartLocation();
                        gamePlay();
                }
                catch(Exception e)
                {
                        System.out.println("Server is down, try again soon");
                }

        }

        static void gamePlay() throws Exception
        {

                //A variable that represents whether or not the user is still playing, used to maintain a loop checking for input
                boolean stillPlaying = true;

                //Get user start location and them assign it to location
                String location = service.getStartLocation();

                //Register User location with
                service.refreshLocation(username, location);
                System.out.println(service.locationInfo(location));

                while(stillPlaying)
                {
                        System.out.println("\nWhat would you like to do?");

                        //Chcks user imput and sets equal to command
                        String command = in.readLine();

                        if (command.equals("exit"))
                        {
                                //If command equals exit print message then leave game
                                System.out.println("You'll be back.....");
                                System.exit(0);
                        }

                        else if (command.equalsIgnoreCase("move"))
                        {
                                //If command equals move then ask the user where they want to move
                                //Accepts 'north' 'south' 'east' 'west'
                                System.out
                                .println("Which way would you like to move.....");
                                String direction = in.readLine();

                                //Use service method move direction and pass it the current location of the player and the direction they wans to go
                                //Both are Strings
                                String newLocation = service.move(location, direction.toLowerCase());
                                location = newLocation;

                                //Print out to the user what the surrounding location is like
                                System.out.println(service.locationInfo(location));

                                service.refreshLocation(username, location);
                        }

                        else if (command.equalsIgnoreCase("Yell"))
                        {
                                //If command is yell ask user what they would like to yell
                                System.out.println("What would you like to yell?");

                                //Read in user input
                                String yell = in.readLine();
                                if (yell.equalsIgnoreCase("Who’s there?"))
                                {
                                        //If user asks who’s there then print who is at the location, including their own username
                                        System.out.println(service.whosThere(location));
                                }

                                else
                                {
                                        //If they don’t type a valid input then alert the user
                                        System.out.println("You can only yell ’Who’s there?’");
                                }
                        }

                        else if (command.equalsIgnoreCase("take"))
                        {
                                //If command is take then ask the user what they would like to take
                                System.out.println("What would you like to take?");

                                //Read in user input
                                String item = in.readLine();
                                if(service.pickUp(item,location))
                                {
                                        //If pickUp returns true then tell the user they own the item they tried to take
                                        System.out.println("You now own the " + item+"\n");
                                }

                                else
                                {
                                        //If the item is not at location then tell the user they can’t take the item
                                        System.out.println("You could not take " + item+"\n");
                                }
                        }

                        else if (command.equalsIgnoreCase("create mud"))
                        {
                                System.out.println("What wouould you like to call it?");
                                String mudName = in.readLine();
                                service.createMUD(mudName);
                        }


                        //CHANGES SUCCESSFULLY BUT THIS IS EXPERIMENTAL AT BEST AND DOES NOT SOLVE MANY PROBLEMS FACED WITH MOVEING MUD
                        else if (command.equalsIgnoreCase("Change MUD"))
                        {
                                startUp();
                        }
                }
        }
}

//TO RUN: RUN FROM mud(1)    java cs3524.solutions.mud.MUDClient jack-U05FA 50010

```