

```

1: package cs3524.solutions.mud;
2:
3: import java.io.*;
4: import java.net.InetAddress;
5: import java.rmi.*;
6: import java.rmi.server.UnicastRemoteObject;
7:
8: public class MUDServerMainline
9: {
10:     static BufferedReader in = new BufferedReader( new InputStreamReader (Syst
em.in));
11:
12:     public static void main (String args[])
13:     {
14:         //Prevents user from starting up the server without having entered
a host and port number
15:         if (args.length < 2)
16:         {
17:             System.err.println("Missing arguments. Please specify both
<host> <port>");
18:             return;
19:         }
20:
21:         //Variable that contains the port number
22:         int registryPort = Integer.parseInt(args[0]);
23:
24:         //Variable that contains the hpst number
25:         int serverPort = Integer.parseInt(args[1]);
26:
27:         //Give the user some feedback so that they know the system is stil
l operational
28:         System.out.println("Starting server on port " + Integer.toString(r
egistryPort) + "...");
29:
30:         try
31:         {
32:             //Make hostName equall to localhost
33:             String hostName = (InetAddress.getLocalHost()).getCanonica
lHostName();
34:
35:             //Specify security policy
36:             System.setProperty( "java.security.policy", "muddy.policy" ) ;
37:             System.setSecurityManager( new RMISecurityManager() ) ;
38:
39:             //Create an instance of MUDServiceImpl()
40:             MUDServiceImpl mudService = new MUDServiceImpl();
41:
42:             //Create a stub for mudService
43:             MUDService mudstub = (MUDService)UnicastRemoteObject.expor
tObject(mudService, serverPort);
44:
45:             //Buile the url
46:             String regUrl = "rmi://" + hostName + ":" + registryPort +
"/MUDService";
47:
48:             try
49:             {
50:                 //Bind the mudstub to that url
51:                 Naming.rebind(regUrl, mudstub);
52:             }
53:             catch (Exception e)
54:             {
55:                 //If the bind failed print the exception message
56:                 System.out.println(e.getMessage());
57:             }
58:
59:             //Notify the user that the server is now running & where i
t is running
60:
61:             System.out.println("Server running at: " + regUrl);
62:
63:             //More text for the user to indicate system progress
64:             System.out.println("Creating default MUD...");
65:
66:             //Call createMud() which creates an instance of MUD
67:             mudService.createMUD("default");
68:         }
69:         catch(Exception b)
70:         {
71:             //If there was a problem with instance creation then alert
the user
72:             System.err.println(b.getMessage());
73:         }
74:     }
75:
76: //TO RUN: RUN FROM mud(1)
50010 50011
77:
78: java cs3524.solutions.mud.MUDServerMainlin

```



```

1: /*****
2:  * cs3524.solutions.mud.MUD
3:  *****/
4:
5: package cs3524.solutions.mud;
6:
7: import java.io.FileReader;
8: import java.io.BufferedReader;
9: import java.io.IOException;
10: import java.util.StringTokenizer;
11:
12: import java.util.Iterator;
13: import java.util.List;
14: import java.util.Map;
15: import java.util.Vector;
16: import java.util.HashMap;
17:
18: /**
19:  * A class that can be used to represent a MUD; essentially, this is a
20:  * graph.
21:  */
22:
23: public class MUD
24: {
25:     /**
26:      * Private stuff
27:      */
28:
29:     // A record of all the vertices in the MUD graph. HashMaps are not
30:     // synchronized, but we don't really need this to be synchronised.
31:     private Map<String,Vertex> vertexMap = new HashMap<String,Vertex>();
32:
33:     private String _startLocation = "";
34:
35:     public Map<String,String> users = new HashMap<String,String>();
36:
37:     /**
38:      * Add a new edge to the graph.
39:      */
40:     private void addEdge( String sourceName,
41:                          String destName,
42:                          String direction,
43:                          String view )
44:     {
45:         Vertex v = getOrCreateVertex( sourceName );
46:         Vertex w = getOrCreateVertex( destName );
47:         v._routes.put( direction, new Edge( w, view ) );
48:     }
49:
50:     /**
51:      * Create a new thing at a location.
52:      */
53:     private void createThing( String loc,
54:                              String thing )
55:     {
56:         Vertex v = getOrCreateVertex( loc );
57:         v._things.add( thing );
58:     }
59:
60:     /**
61:      * Change the message associated with a location.
62:      */
63:     public void changeMessage( String loc, String msg )
64:     {
65:         Vertex v = getOrCreateVertex( loc );
66:         v._msg = msg;
67:     }

```

```

68:
69:
70:     /**
71:      * If vertexName is not present, add it to vertexMap. In either
72:      * case, return the Vertex. Used only for creating the MUD.
73:      */
74:     private Vertex getOrCreateVertex( String vertexName )
75:     {
76:         Vertex v = vertexMap.get( vertexName );
77:         if ( v == null ) {
78:             v = new Vertex( vertexName );
79:             vertexMap.put( vertexName, v );
80:         }
81:         return v;
82:     }
83:
84:     /**
85:      */
86:     public Vertex getVertex( String vertexName )
87:     {
88:         return vertexMap.get( vertexName );
89:     }
90:
91:     /**
92:      * Creates the edges of the graph on the basis of a file with the
93:      * following format:
94:      * source direction destination message
95:      */
96:     private void createEdges( String edgesfile )
97:     {
98:         try {
99:             FileReader fin = new FileReader( edgesfile );
100:             BufferedReader edges = new BufferedReader( fin );
101:             String line;
102:             while( (line = edges.readLine()) != null ) {
103:                 StringTokenizer st = new StringTokenizer( line );
104:                 if( st.countTokens() < 3 ) {
105:                     System.err.println( "Skipping ill-formatted line " + line );
106:                     continue;
107:                 }
108:                 String source = st.nextToken();
109:                 String dir = st.nextToken();
110:                 String dest = st.nextToken();
111:                 String msg = "";
112:                 while (st.hasMoreTokens()) {
113:                     msg = msg + st.nextToken() + " ";
114:                 }
115:                 addEdge( source, dest, dir, msg );
116:             }
117:         }
118:         catch( IOException e ) {
119:             System.err.println( "Graph.createEdges( String " +
120:                                edgesfile + ")\n" + e.getMessage() );
121:         }
122:     }
123:
124:     /**
125:      * Records the messages associated with vertices in the graph on
126:      * the basis of a file with the following format:
127:      * location message
128:      * The first location is assumed to be the starting point for
129:      * users joining the MUD.
130:      */
131:     private void recordMessages( String messagesfile )
132:     {
133:         try {
134:             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 associated 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, String thing )
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:         v._things.remove( thing );
276:         e._dest._things.add( thing );
277:         return e._dest._name;
278:     }
279:
280:     /**
281:      * A main method that can be used to testing purposes to ensure
282:      * that the MUD is specified correctly.
283:      */
284:     public static void main(String[] args)
285:     {
286:         if (args.length != 3) {
287:             System.err.println("Usage: java Graph <edgesfile> <messagesfile> <thin
gsfile>");
288:             return;
289:         }
290:         MUD m = new MUD( args[0], args[1], args[2] );
291:         System.out.println( m.toString() );
292:     }
293: }
```



```
1: package cs3524.solutions.mud;
2:
3: import java.rmi.Remote;
4: import java.rmi.RemoteException;
5:
6: public interface MUDService extends Remote
7: {
8:     public void createMUD(String mudName) throws RemoteException;
9:
10:    public String getStartLocation() throws RemoteException;
11:
12:    public String location(String location) throws RemoteException;
13:
14:    public String locationInfo( String loc ) throws RemoteException;
15:
16:    public String move(String current, String direction) throws RemoteException;
17:
18:    public String pickMUD(String inputMud) throws RemoteException;
19:
20:    public boolean pickUp(String item, String location) throws RemoteException;
21:
22:    public void refreshLocation(String username, String location) throws RemoteExc
ption ;
23:
24:    public String welcome() throws RemoteException;
25:
26:    public String whosThere(String location) throws RemoteException;
27: }
```



```

1: package cs3524.solutions.mud;
2:
3: import java.rmi.*;
4: import java.util.*;
5:
6: public class MUDServiceImpl implements MUDService
7: {
8:     private MUD mudInstance;
9:     public Map<String, MUD> MUDs = new HashMap<String, MUD>();
10:    public Integer mudLimiter = 4;
11:    public Integer mudCounter = 0;
12:
13:    public MUDServiceImpl() throws RemoteException
14:    {
15:        //Boilerplate as there is nothing to construct
16:    }
17:
18:    public void createMUD(String mudName) throws RemoteException
19:    {
20:        try
21:        {
22:            //Try and make a instance of MUD In MUD.java
23:            if(mudCounter == mudLimiter)
24:            {
25:                System.out.println("Too many muds created, you can't create anymore
e.");
26:            }
27:
28:            else
29:            {
30:                MUDs.put(mudName, new MUD("mymud.edg", "mymud.msg", "mymud.thg"));
31:                System.out.println("MUD " + mudName + " created");
32:                mudCounter = mudCounter + 1;
33:            }
34:        }
35:
36:        catch (Exception ex)
37:        {
38:            System.err.println("Error creating MUD. Error details: " +
ex.getMessage());
39:        }
40:
41:    }
42:
43:    public String welcome() throws RemoteException
44:    {
45:        String output = "";
46:        output = ("----- MUDs -----\\n");
47:
48:        for(Map.Entry<String, MUD> entry : MUDs.entrySet())
49:        {
50:            String key = entry.getKey();
51:            output += (key + "\\n");
52:        }
53:
54:        output += ("-----\\n");
55:        output += ("Select a MUD to connect to: ");
56:
57:        return output;
58:    }
59:
60:    public String pickMUD(String inputMud) throws RemoteException
61:    {
62:
63:        String output = "";
64:
65:        if(inputMud != "")

```

```

66:    {
67:        mudInstance = MUDs.get(inputMud);
68:        output = ( "Welcome to " + inputMud + " MUD Server!\\n" );
69:        output += ( "Please enter a username: " );
70:    }
71:
72:    return output;
73: }
74:
75:
76:    public String getStartLocation() throws RemoteException
77:    {
78:        //Used to get the position of an instantiated user
79:        return mudInstance.startLocation();
80:    }
81:
82:
83:
84:    public String location(String location) throws RemoteException
85:    {
86:        //Asks for the location of a user in a MUD world
87:        return mudInstance.getVertex(location).toString();
88:    }
89:
90:
91:    public String move(String current, String direction) throws RemoteException
92:    {
93:        //Allows users to traverse the MUD enviroment given the start location and
the desired location
94:        //User will move if the direction is valid given the location
95:        Vertex currentVertex = mudInstance.getVertex(current);
96:
97:        if(currentVertex._routes.containsKey(direction))
98:        {
99:            Edge newLocation = currentVertex._routes.get(direction);
100:            Vertex newVert = (newLocation._dest);
101:            return newVert._name;
102:        }
103:
104:        else
105:        {
106:            return current;
107:        }
108:    }
109:
110:
111:    public void refreshLocation(String username, String location) throws RemoteExc
eption
112:    {
113:        //Remove the user from a location
114:        mudInstance.users.remove(username);
115:        //Add them to the new location
116:        mudInstance.users.put(username, location);
117:
118:        //System.out.println(mudInstance.users);
119:    }
120:
121:
122:    public String whosThere(String location) throws RemoteException
123:    {
124:
125:        //Create an array of all players playing
126:        ArrayList<String> Players = new ArrayList<String>();
127:        //Instantiate a variable for username
128:        String username;
129:
130:        StringBuilder usernameList = new StringBuilder();

```

```
131:
132:     Iterator i = mudInstance.users.keySet().iterator();
133:     //Keep adding users who are at a given location, until there is no more us
ers at location
134:     while (i.hasNext())
135:     {
136:         username = i.next().toString();
137:
138:         if(mudInstance.users.get(username).equalsIgnoreCase(location))
139:         {
140:             Players.add(username);
141:             usernameList.append(username);
142:             usernameList.append(", ");
143:         }
144:
145:     }
146:
147:     usernameList.setLength(usernameList.length() - 2);
148:
149:     //return the string of all players at that location
150:     return "You can see: " + usernameList.toString();
151:
152: }
153:
154:
155: public String locationInfo( String location )
156: {
157:     //Return the correct message, given your location, from Mud.thg
158:     return mudInstance.getVertex( location ).toString();
159: }
160:
161:
162: public boolean pickUp(String item, String location) throws RemoteException
163: {
164:     Vertex currentVertex = mudInstance.getVertex(location);
165:     //Get all items at current location
166:     List<String> things = currentVertex._things;
167:     //If there is something at that location
168:     if(things.contains(item))
169:     {
170:         //Remove it
171:         mudInstance.delThing(location, item);
172:
173:         if(location.equals("D"))
174:         {
175:             //Change the message at location D to indicate that the treasure is
no longer here
176:             mudInstance.changeMessage(location, "Looks like there used to be a
treasure here");
177:         }
178:
179:         else if(location.equals("A"))
180:         {
181:             //Change the message at location A to indicate that the ring is no
longer here
182:             mudInstance.changeMessage(location, "Looks like there used to be a
ring here");
183:         }
184:
185:         return true;
186:     }
187:
188:     return false;
189: }
190: }
```

```
1: package cs3524.solutions.mud;
2:
3: import java.util.Map;
4: import java.util.HashMap;
5: import java.util.List;
6: import java.util.Vector;
7: import java.util.Iterator;
8:
9: // Represents a location in the MUD (a vertex in the graph).
10: class Vertex
11: {
12:     public String _name;           // Vertex name
13:     public String _msg = "";       // Message about this location
14:     public Map<String,Edge> _routes; // Association between direction
15:                                     // (e.g. "north") and a path
16:                                     // (Edge)
17:     public List<String> _things;    // The things (e.g. players) at
18:                                     // this location
19:
20:     public Vertex( String nm )
21:     {
22:         _name = nm;
23:         _routes = new HashMap<String,Edge>(); // Not synchronised
24:         _things = new Vector<String>();       // Synchronised
25:     }
26:
27:     public String toString()
28:     {
29:         String summary = "\n";
30:         summary += _msg + "\n";
31:         Iterator iter = _routes.keySet().iterator();
32:         String direction;
33:         while (iter.hasNext()) {
34:             direction = (String)iter.next();
35:             summary += "To the " + direction + " there is " + ((Edge)_routes.get(
direction))._view + "\n";
36:         }
37:         iter = _things.iterator();
38:         if (iter.hasNext()) {
39:             summary += "You can see: ";
40:             do {
41:                 summary += iter.next() + " ";
42:             } while (iter.hasNext());
43:         }
44:         summary += "\n\n";
45:         return summary;
46:     }
47: }
48:
```



```
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.*;
4: import java.util.*;
5: import java.io.*;
6: import java.net.InetAddress;
7:
8: public class MUDClient
9: {
10:
11:     //Creates an instance of MUDService called service
12:     static MUDService service;
13:
14:     //Variable declarations
15:     static BufferedReader in = new BufferedReader( new InputStreamReader( Syst
em.in ));
16:     private static String username;
17:     private static String location;
18:     private static String nameOfMUD;
19:
20:     public static void main(String args[] throws Exception
21:     {
22:         // An if statement that check if the user has provided a valid hos
t and port
23:         if(args.length < 2)
24:         {
25:             System.err.println("Missing arguments. PLease specify both
<host> <port>");
26:             return;
27:         }
28:
29:         //The first argument will be made equall to hostname
30:         String hostName = args[0];
31:
32:         //The seccond argument will be made equall to port
33:         int port = Integer.parseInt(args[1]);
34:
35:         //Specify the security policy and set security manager
36:         System.setProperty( "java.security.policy", "muddy.policy" );
37:         System.setSecurityManager( new RMISecurityManager() );
38:
39:         try
40:         {
41:             //Create registration URL from hostname, port
42:             String regUrl = "rmi://" + hostName + ":" + port + "/MudSe
rvice";
43:
44:             service = (MUDService)Naming.lookup(regUrl);
45:
46:             //Once connected call function startUp()
47:             startUp();
48:         }
49:         catch (java.io.IOException e)
50:         {
51:             System.err.println("Input error");
52:             System.err.println(e.getMessage());
53:         }
54:     }
55:
56:     static void startUp() throws Exception
57:     {
58:
59:         //Print out the String returned by intoroduction function in MUDSe
rviceImpl.java
60:         System.out.println(service.welcome());
61:         nameOfMUD = in.readLine();
62:         try
63:         {
64:             System.out.println(service.pickMUD(nameOfMUD));
65:
66:             //Ask user to set username
67:             username = in.readLine();
68:
69:             //Use the String returned by getStartLocation from MUDServ
iceImpl to set the value for location
70:             location = service.getStartLocation();
71:             gamePlay();
72:         }
73:         catch(Exception e)
74:         {
75:             System.out.println("Server is down, try again soon");
76:         }
77:     }
78:
79:
80:     static void gamePlay() throws Exception
81:     {
82:
83:         //A variable that represents whether or not the user is still play
ing, used to maintain a loop checking for input
84:         boolean stillPlaying = true;
85:
86:         //Get user start location and them assign it to location
87:         String location = service.getStartLocation();
88:
89:         //Register User location with
90:         service.refreshLocation(username, location);
91:         System.out.println(service.locationInfo(location));
92:
93:         while(stillPlaying)
94:         {
95:             System.out.println("\nWhat would you like to do?");
96:
97:             //Chcks user imput and sets equal to command
98:             String command = in.readLine();
99:
100:             if (command.equals("exit"))
101:             {
102:                 //If command equals exit print message then leave
103:                 System.out.println("You'll be back.....");
104:                 System.exit(0);
105:             }
106:
107:             else if (command.equalsIgnoreCase("move"))
108:             {
109:                 //If command equals move then ask the user where t
hey want to move
110:                 //Accepts 'north' 'south' 'east' 'west'
111:                 System.out
112:                 .println("Which way would you like to move.....");
113:                 String direction = in.readLine();
114:
115:                 //Use service method move direction and pass it th
e current location of the player and the direction they wans to go
116:                 //Both are Strings
117:                 String newLocation = service.move(location, direct
ion.toLowerCase());
118:                 location = newLocation;
119:
120:                 //Print out to the user what the surrounding locat
ion is like
121:                 System.out.println(service.locationInfo(location))
;

```

```

122:
123:         service.refreshLocation(username, location);
124:     }
125:
126:     else if (command.equalsIgnoreCase("Yell"))
127:     {
128:         //If command is yell ask user what they would like
129:         System.out.println("What would you like to yell?")
130:
131:         //Read in user input
132:         String yell = in.readLine();
133:         if (yell.equalsIgnoreCase("Who's there?"))
134:         {
135:             //If user asks who's there then print who
136:             System.out.println(service.whosThere(location));
137:         }
138:
139:         else
140:         {
141:             //If they don't type a valid input then alert the user
142:             System.out.println("You can only yell 'Who's there?'");
143:         }
144:     }
145:
146:     else if (command.equalsIgnoreCase("take"))
147:     {
148:         //If command is take then ask the user what they would like to take
149:         System.out.println("What would you like to take?")
150:
151:         //Read in user input
152:         String item = in.readLine();
153:         if(service.pickUp(item,location))
154:         {
155:             //If pickUp returns true then tell the user they own the item they tried to take
156:             System.out.println("You now own the " + item+"\n");
157:         }
158:
159:         else
160:         {
161:             //If the item is not at location then tell the user they can't take the item
162:             System.out.println("You could not take " + item+"\n");
163:         }
164:     }
165:
166:     else if (command.equalsIgnoreCase("create mud"))
167:     {
168:         System.out.println("What would you like to call it?");
169:         String mudName = in.readLine();
170:         service.createMUD(mudName);
171:     }
172:
173:
174:     //CHANGES SUCCESSFULLY BUT THIS IS EXPERIMENTAL AT BEST AND DOES NOT SOLVE MANY PROBLEMS FACED WITH MOVING MUD

```

```

175:         else if (command.equalsIgnoreCase("Change MUD"))
176:         {
177:             startUp();
178:         }
179:     }
180: }
181: }
182:
183: //TO RUN: RUN FROM mud(1) java cs3524.solutions.mud.MUDClient jack-U05FA 50010
184:

```