```
1  import ChatApp.*;          // The package containing our stubs.
2  import org.omg.CosNaming.*;// HelloServer will use the naming service.
3  import org.omg.CosNaming.NamingContextPackage.*; // ..for exceptions.
4  import org.omg.CORBA.*;    // All CORBA applications need these classes.
5  import org.omg.PortableServer.*;
6  import org.omg.PortableServer.POA;
7  import java.util.*;
8
9  class User {
10   private String name;
11
12   public ChatCallback clientobj;
13
14   public User(String name, ChatCallback clientobj)
15   {
16     this.name = name;
17     this.clientobj = clientobj;
18   }
19
20   public String getName()
21   {
22     return name;
23   }
24 }
25
26 class ChatImpl extends ChatPOA {
27     private ORB orb;
28
29   private List<User> users = new ArrayList<User>();
30   //Init a new gomoku game
31   private Gomoku gomoku = new Gomoku();
32
33   public void setORB(ORB orb_val) {
34     orb = orb_val;
35   }
36
37   public void gomoku(ChatCallback callobj, String color)
38   {
39     //Client needs to be registered to join a Gomoku game
40     if (!isRegistered(callobj))
41     {
42       callobj.callback("Error: You must join to play. Use command: 'join
       <username>' to join");
43       return;
44     }
45     //Client must have a valid color to join a Gomoku game
46     else if (!gomoku.validColor(color))
47     {
48       callobj.callback("Error: You can't join with color '" + color + "', please
       use 'x' or 'o'");
49       return;
50     }
51     //Client can't join if client is already playing
52     else if(gomoku.isActivePlayer(getUserByCallObject(callobj).getName()))
53     {
54       callobj.callback("Error: You are already in a Gomoku game session");
55       return;
```

```
56        }
57        //Add a player to the game with specified username and team color
58        gomoku.addPlayer(getUserByCallObject(callobj).getName(), color.charAt(0));
59
60        callobj.callback("You entered a Gomoku game session with color '" + color + "'"
          );
61        //Update gameboard only for joined player
62        updateGameBoard(callobj);
63      }
64
65      public void add(ChatCallback callobj, int x, int y) {
66        //Client can't add a tile if not in a game session
67        if (!gomoku.isActivePlayer(getUserByCallObject(callobj).getName()))
68        {
69          callobj.callback("Error: You need to join to play the game. Use command:
            'join <username>' to join");
70          return;
71        }
72        //If position of tile is invalid
73        else if(!gomoku.canAddTile(x, y))
74        {
75          callobj.callback("Error: You can't add tile on tile " + "[" + (x+1) + "," + (
            y+1) + "]");
76          return;
77        }
78        //Add tile by specified username and cordinates
79        gomoku.addTile(getUserByCallObject(callobj).getName(), x, y);
80
81        //Print gameboard to all users since a new tile has been updated
82        for (User user : users)
83        {
84          updateGameBoard(user.clientobj);
85        }
86
87        //Print gameboard to server by specifying callobject to null.
88        //(I know it's ugly fix but whatever, it's at least better then
89        //overriding method with another parameter,
90        //unfortunately Java doesn't support default paremters')
91        updateGameBoard(null);
92
93        //A team has won
94        if (gomoku.decideWinner())
95        {
96          //Broadcast winning message to all players
97          broadcastMessage(gomoku.getWinningMessage(), null);
98          //Reset the Gomoku game
99          gomoku.reset();
100       }
101     }
102
103     private void updateGameBoard(ChatCallback callobj) {
104       char[][] gameBoard = gomoku.getGameBoard();
105       String strGameBoard = "";
106       //Go through whole gameboard and get row by row
107       for (int x = 0; x < gomoku.NUM_OF_X_TILES; x++)
108       {
109         for(int y = 0; y < gomoku.NUM_OF_Y_TILES; y++)
```

```
110         {
111           strGameBoard+=Character.toString(gameBoard[x][y]) + " ";
112         }
113         strGameBoard+="\n";
114       }
115       //Print gameboard for either client or server
116       if (callobj == null) { System.out.println(strGameBoard); }
117       else { callobj.callback(strGameBoard); }
118     }
119
120     public void join(ChatCallback callobj, String userName) {
121       //If user with specified username already is registered
122       if (nameTaken(userName))
123       {
124         callobj.callback("Error: user " + userName + " is already an active chatter");
125         return;
126       }
127       //Client has already joined
128       else if(isRegistered(callobj))
129       {
130         callobj.callback("Error: you are already joined as user '" +
          getUserByCallObject(callobj).getName() + "'");
131         return;
132       }
133       callobj.callback("Welcome " + userName + "!");
134       //Add new client with specified name and specified call object
135       users.add(new User(userName, callobj));
136
137       //Broadcast message joining message to all clients except joined client (callobj)
138       broadcastMessage(getUserByCallObject(callobj).getName() + " joined", callobj);
139
140       System.out.println(userName + " joined the server");
141     }
142
143     public void leave(ChatCallback callobj) {
144       //Client can't leave if client hasn't joined yet
145       if (!isRegistered(callobj))
146       {
147         callobj.callback("Error: You need to join to leave. Use command: 'join
          <username>' to join");
148         return;
149       }
150       callobj.callback("Goodbye " + getUserByCallObject(callobj).getName());
151
152       //Broadcast message to all users except the one leaving (callobj)
153       broadcastMessage(getUserByCallObject(callobj).getName() + " left", callobj);
154
155       System.out.println(getUserByCallObject(callobj).getName() + " left the server");
156       //Remove user from the server
157       users.remove(getUserByCallObject(callobj));
158     }
159
160     public void post(ChatCallback callobj, String message) {
161       //Client can't post if not joined
162       if(!isRegistered(callobj))
163       {
164         callobj.callback("Error: You need to join to post. Use command: 'join
```

```
        <username>' to join");
165       return;
166     }
167     //Broadcast the message to all users (exception is null means there's not exception)
168     broadcastMessage(getUserByCallObject(callobj).getName() + " said: " + message, null);
169   }
170
171   public void list(ChatCallback callobj) {
172     String listMsg="There are " + users.size() + " users currently registered:";
173     //Add all names of registered users
174     for (User user : users)
175     {
176       listMsg+="\n" + user.getName();
177     }
178     //Print list to only client called the method
179     callobj.callback(listMsg);
180   }
181
182   private User getUserByCallObject(ChatCallback callobj) {
183     //Look which user the callobj is belonging to (null if not any user)
184     for (User user : users)
185     {
186       if (user.clientobj.equals(callobj))
187       {
188         return user;
189       }
190     }
191     return null;
192   }
193
194   //Set exception to null if you don't want any exceptions
195   private void broadcastMessage(String message, ChatCallback exception) {
196     for (User user : users)
197     {
198       //To not broadcoast to specified user
199       if (!user.clientobj.equals(exception))
200       {
201         user.clientobj.callback(message);
202       }
203     }
204   }
205
206   private boolean nameTaken(String userName) {
207     //Loop for user with same name as specified name
208     for (User user : users)
209     {
210       if (user.getName().equals(userName))
211       {
212         return true;
213       }
214     }
215     return false;
216   }
217
218   private boolean isRegistered(ChatCallback callobj) {
```

```
219       //Check if callobject belongs to any of the users
220       for (User user : users)
221       {
222         if (user.clientobj.equals(callobj))
223         {
224           return true;
225         }
226       }
227       return false;
228     }
229   }
230
231   public class ChatServer {
232     public static void main(String args[]) {
233       try
234       {
235         // create and initialize the ORB
236         ORB orb = ORB.init(args, null);
237
238         // create servant (impl) and register it with the ORB
239         ChatImpl chatImpl = new ChatImpl();
240         chatImpl.setORB(orb);
241
242         // get reference to rootpoa & activate the POAManager
243         POA rootpoa =
244       POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
245         rootpoa.the_POAManager().activate();
246
247         // get the root naming context
248         org.omg.CORBA.Object objRef =
249                   orb.resolve_initial_references("NameService");
250         NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
251
252         // obtain object reference from the servant (impl)
253         org.omg.CORBA.Object ref =
254       rootpoa.servant_to_reference(chatImpl);
255         Chat cref = ChatHelper.narrow(ref);
256
257         // bind the object reference in naming
258         String name = "Chat";
259         NameComponent path[] = ncRef.to_name(name);
260         ncRef.rebind(path, cref);
261
262         // Application code goes below
263         System.out.println("ChatServer ready and waiting ...");
264
265         // wait for invocations from clients
266         orb.run();
267       }
268
269       catch(Exception e) {
270         System.err.println("ERROR : " + e);
271         e.printStackTrace(System.out);
272       }
273     System.out.println("ChatServer Exiting ...");
274     }
275   }
```