```
package rmiServer;
import java.rmi.*;
import java.rmi.server.*;
import rmiClient.ChatObjectInterface;
import javax.swing.JOptionPane;
/**
 * This class will publish its service on the rmiregistery.
 * the service methods available are listed in ServerInterface.java
 * an instance of the server will contain an array of 50 remote
 * stubs of clients represented as ChatObjectInterface.
 *
 * @author Patrick and Tenzin
 * @date 25 nov 2008
 * @version 1.0
 */
public class Server extends UnicastRemoteObject implements ServerInterface{
    private static final long serialVersionUID = 1L;
    private Object clients[] = new Object[50];
    private int clientCount=0;

    /**
     * constructor. creats a server for chat application
     */
    public Server() throws RemoteException{
    }
    /**
     * Implements the remote server method connect() that clients can use to connect
     * to the server.It passes to the server the clients own remoter interface so that
     * server can send messeges to this client.on successful registrations sends a welcome message to the user.
     *
     * @param  name String could be used to enforce unique nickname for each user
     * @param  c    ChatObjectInterface
     */
    public void connect(String name,ChatObjectInterface c)throws RemoteException {
              try{
                    System.out.println(name+" entered (id " + clientCount + ")");
                   for( int i=0;i<50; i++){
                           if (clients[i]==null) {
                                   clients[i]=c;
                                   clientCount++;

                           sendInternal(c,"\nWelcome to the server "+ name + "!");
                           return;
                           }
                       }
                  }catch(Exception e){
                    System.out.println("Error While connecting: "+e);
                  }

    }
    /**
     * A private send method that sends the messege to one specified client only.
     * Is used by the remote method send and connect. on error, will drop the client
     *  who is the cause of error.
     */
    private void sendInternal(ChatObjectInterface c, String messege) {
        try{c.handle(messege);}
        catch(RemoteException e){
            System.out.println("Error While sending to client i: "+e+"\n. will terminate this client now");
            disconnectInternal(c);
            }
    }
    /**
     *This method is called by a remote client to send a string messege
     *to all the other chat users.It specifically crawls through the servers client list and send the messege
```

```
     *to all valid clients through the client remote interface
     *@param message String
     */
    public void send (String message)throws RemoteException {
        int i=0;
        ChatObjectInterface thisclient;
        //in the array the clients could be at index {1,10,28} while those spaces in between are null.
       while(i<50) {
            if(clients[i]!=null) {
                thisclient=(ChatObjectInterface) clients[i];
                sendInternal(thisclient,message);
            }
            i++;
        }
    }
    /*a private method that removes a client proxy from the server.
     * Is used by the remote method send and connect. on error, will drop the client
     *  who is the cause of error.*/

    private void disconnectInternal(ChatObjectInterface c) {
          try{
                int i=0;
                if(clientCount==0) return ;
                 while(i<50) {
                     if(clients[i]!=null ){
                         if (c==null)
                              {  clients[i]=null;
                                 clientCount--;
                                 System.out.println("user disconnected (id "+ i +")");}
                         else if(clients[i].equals(c)) {
                             //also if possible confirm to the client that he has been disconnected.
                             clients[i]=null;

                             clientCount--;
                             System.out.println("user disconnected (id "+ i +")");
                             return;
                            }
                        }i++;
                    }
              }catch(Exception e){
                System.out.println("Error While disconnecting: "+e);
     }
}
/**
 * a remote method that a client can use to tell the server that it is quiting the chat service.
 * It just removes th clients remote interface from the servers list of client.
 *
 * @param c ChatObjectInterface of the client who wants to exit ther service
 */
    public void disconnect(ChatObjectInterface c) throws RemoteException{
        if (c==null) {
            System.out.println("received a null c");
        }else disconnectInternal(c);
    }
/**
 * Called by the server to tell all clients that it is shutting down the service.
 * It is not strictly necessary but makes the application stop more gracefully.
 */
public void stopServer() {
    try {
        disconnectInternal(null);
    }catch (Exception e) {
        System.out.println("exceptions while stoping server" + e.getMessage());
    } System.exit(0);
}
private void initserver() {
```

```
    for(int i=0;i<50;i++ ) {
        clients[i]=null;
        }
}
/**
 *The application launch point. It creats an instance of this class, registers itself as RMICHAT and opens a simple control panel containins a switch toshut down this server.
 *It takes no arguments.
 */
    public static void main (String[] args) {
        try {
            Server server = new Server();
            server.initserver();
            Naming.rebind("RMICHAT", server);
            int i;
         do{
            i=JOptionPane.showConfirmDialog(null,"Server running, press yes to stop", "information",JOptionPane.YES_OPTION, JOptionPane.INFORMATION_MESSAGE);
            }while(i!=JOptionPane.YES_OPTION);{
                    server.stopServer();
                };
        } catch (Exception e) {
            System.out.println("Error while naming: "+e);

        }
    }


}
```