Chapter 2

ACTIVITY DESCRIPTION

2.1 UDP Socket program for multi-user chatting

Online chat may refer to any kind of communication over the Internet that offers a real-time transmission of text messages from sender to receiver. Chat messages are generally short in order to enable other participants to respond quickly. Thereby, a feeling similar to a spoken conversation is created, which distinguishes chatting from other text-based online communication forms such as Internet forums and email. Online chat may address point-to-point communications as well as multicast communications from one sender to many receivers and voice and video chat, or may be a feature of a web conferencing service.

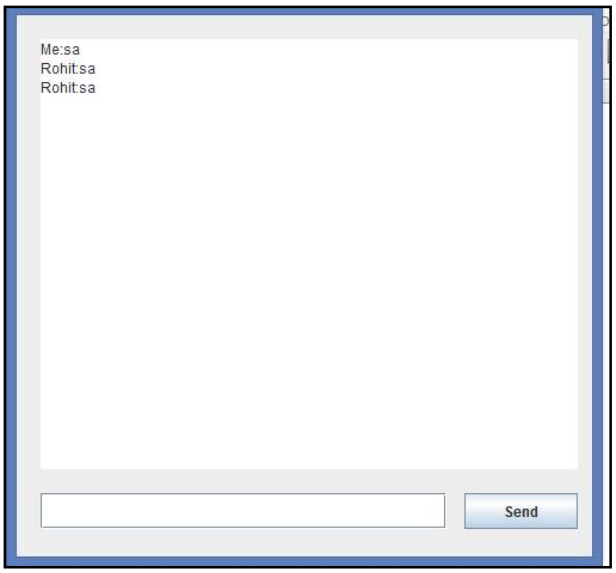
Online chat in a less stringent definition may be primarily any direct text-based or video-based (webcams), one-on-one chat or one-to-many group chat(formally also known as synchronous conferencing), using tools such as instant messengers, Internet Relay Chat (IRC), talkers and possibly MUDs. The expression online chat comes from the word chat which means "informal conversation". Online chat includes web-based applications that allow communication – often directly addressed, but anonymous between users in a multi-user environment. Web conferencing is a more specific online service, that is often sold as a service, hosted on a web server controlled by the vendor.

```
package javaapplication7;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.*;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

public class JavaApplication7 extends JFrame implements ActionListener {
    static DatagramSocket ds;
```

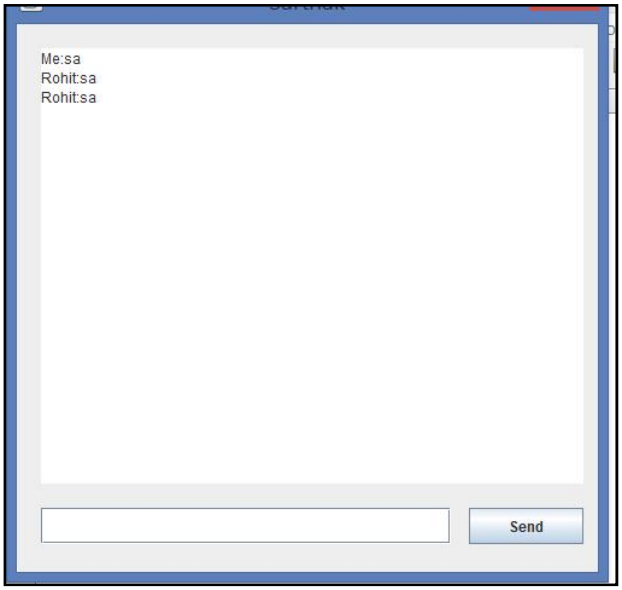
```
        JPanel panel;
        JTextField NewMsg;
        JTextArea ChatHistory;
        JButton Send;
        DataInputStream dis;
        DataOutputStream dos;

        public JavaApplication7() throws UnknownHostException, IOException {
    ds=new DatagramSocket(8000);
                panel = new JPanel();
                NewMsg = new JTextField();
                ChatHistory = new JTextArea();
                Send = new JButton("Send");
                this.setSize(500, 500);
                this.setVisible(true);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                panel.setLayout(null);
                this.add(panel);
                ChatHistory.setBounds(20, 20, 450, 360);
                panel.add(ChatHistory);
                NewMsg.setBounds(20, 400, 340, 30);
                panel.add(NewMsg);
                Send.setBounds(375, 400, 95, 30);
                panel.add(Send);
                this.setTitle("Sarthak");
                Send.addActionListener(this);
                while (true) {
                        try {
    byte b[]=new byte[1024];
    DatagramPacket r=new DatagramPacket(b,b.length);
    ds.receive(r);
    int port=ds.getPort();
    String string = new String(r.getData());
    if(port==7000){
    ChatHistory.setText(ChatHistory.getText()  + "Ranga:"+ string+"\n");
    }
    else{
    ChatHistory.setText(ChatHistory.getText()  + "Rohit:"+ string+"\n");
    }

} catch (Exception e1) {

ChatHistory.setText(ChatHistory.getText() + "Message sending fail:Network Error\n");

try {

        Thread.sleep(3000);
        System.exit(0);
```

```
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

  }

 }

}

        @Override

        public void actionPerformed(ActionEvent e) {
                        if ((e.getSource() == Send) && (NewMsg.getText() != "")) {
                         ChatHistory.setText(ChatHistory.getText()  + "Me:"
                                        + NewMsg.getText()+"\n");
                         try {
                           String msg=new String(NewMsg.getText());
    InetAddress ip=InetAddress.getByName("192.168.3.78");
    int port=ds.getPort();
    DatagramPacket s=new DatagramPacket(msg.getBytes(),msg.length(),ip,9000);
    ds.send(s);
    s.setPort(7000);
    ds.send(s);
    } catch (Exception e1) {
    try {
            Thread.sleep(3000);
            System.exit(0);
        } catch (InterruptedException e2) {
            e2.printStackTrace();
        }
    }
    NewMsg.setText("");

   }

}

public static void main(String[] args) throws UnknownHostException,

IOException {

        new JavaApplication7();
    }

}
```

Output :-



First user is connected to a different user in the LAN by typing the IP address of the user and then both the user can chat to each other via LAN.



Second user is also connected to a the user in the LAN by typing the IP address of the user and then both the user chat to each other via LAN.

2.2 TCP Socket program for Tic-Tac-Toe

Tic-tac-toe (also known as noughts and crosses or Xs and Os) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

Because of the simplicity of tic-tac-toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship and the branch of artificial intelligence that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly, to enumerate the 765 essentially different positions (the state space complexity), or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space.

The game can be generalized to an m,n,k-game in which two players alternate placing stones of their own color on an m×n board, with the goal of getting k of their own color in a row. Tic-tac-toe is the (3,3,3)-game. Harary's generalized tic-tac-toe is an even broader generalization of tic tac toe. It can also be generalized as a n^d game. Tic-tac-toe is the game where n equals 3 and d equals 2. If played properly, the game will end in a draw making tic-tac-toe a futile game.

Tic-Tac-Toe Client Code :-

```
package tictactoeclient;

import java.awt.*;

import java.awt.event.*;

import java.net.*;

import java.io.*;

import javax.swing.*;

// Client class to let a user play Tic-Tac-Toe with

// another user across a network.

public class TicTacToeClient extends JApplet
```

```
                    implements Runnable {
private JTextField id;
private JTextArea display;
private JPanel boardPanel, panel2;
private Square board[][], currentSquare;
private Socket connection;
private DataInputStream input;
private DataOutputStream output;
private Thread outputThread;
private char myMark;
private boolean myTurn;
// Set up user-interface and board
public void init()
{
  display = new JTextArea( 4, 30 );
  display.setEditable( false );
  getContentPane().add( new JScrollPane( display ),
                    BorderLayout.SOUTH );

  boardPanel = new JPanel();
  GridLayout layout = new GridLayout( 3, 3, 0, 0 );
  boardPanel.setLayout( layout );
```

```
board = new Square[ 3 ][ 3 ];
// When creating a Square, the location argument to the
// constructor is a value from 0 to 8 indicating the
// position of the Square on the board. Values 0, 1,
// and 2 are the first row, values 3, 4, and 5 are the
// second row. Values 6, 7, and 8 are the third row.
for ( int row = 0; row < board.length; row++ )
{
  for ( int col = 0;
          col < board[ row ].length; col++ ) {
    board[ row ][ col ] =
      new Square( ' ', row * 3 + col );
    board[ row ][ col ].addMouseListener(
      new SquareListener(
        this, board[ row ][ col ] ) );

    boardPanel.add( board[ row ][ col ] );
  }
}
id = new JTextField();
id.setEditable( false );
getContentPane().add( id, BorderLayout.NORTH );
panel2 = new JPanel();
```

```
    panel2.add( boardPanel, BorderLayout.CENTER );
    getContentPane().add( panel2, BorderLayout.CENTER );
  }
  // Make connection to server and get associated streams.
  // Start separate thread to allow this applet to
  // continually update its output in text area display.
  public void start()
  {
    try {
      connection = new Socket(
        InetAddress.getByName( "127.0.0.1" ), 5000 );
      input = new DataInputStream(
                connection.getInputStream() );
      output = new DataOutputStream(
                connection.getOutputStream() );
    }
    catch ( IOException e ) {
      e.printStackTrace();
    }
    outputThread = new Thread( this );
    outputThread.start();
  }
  // Control thread that allows continuous update of the
```

```
// text area display.
public void run()
{
  // First get player's mark (X or O)
  try {
    myMark = input.readChar();
    id.setText( "You are player \"" + myMark + "\"" );
    myTurn = ( myMark == 'X' ? true  : false );
  }
  catch ( IOException e ) {
    e.printStackTrace();
  }
  // Receive messages sent to client
  while ( true ) {
    try {
      String s = input.readUTF();
      processMessage( s );
    }
    catch ( IOException e ) {
      e.printStackTrace();
    }
  }
}
```

```
// Process messages sent to client
public void processMessage( String s )
{
  if ( s.equals( "Valid move." ) ) {
    display.append( "Valid move, please wait.\n" );
    currentSquare.setMark( myMark );
    currentSquare.repaint();
  }
  else if ( s.equals( "Invalid move, try again" ) ) {
    display.append( s + "\n" );
    myTurn = true;
  }
  else if ( s.equals( "Opponent moved" ) ) {
    try {
      int loc = input.readInt();
      board[ loc / 3 ][ loc % 3 ].setMark(
          ( myMark == 'X' ? 'O' : 'X' ) );
      board[ loc / 3 ][ loc % 3 ].repaint();
      display.append(
        "Opponent moved. Your turn.\n" );
      myTurn = true;
    }
    catch ( IOException e ) {
```

```
            e.printStackTrace();
          }
        }
        else
          display.append( s + "\n" );

        display.setCaretPosition(
          display.getText().length() );
      }
      public void sendClickedSquare( int loc )
      {
        if ( myTurn )
          try {
            output.writeInt( loc );
            myTurn = false;
          }
          catch ( IOException ie ) {
            ie.printStackTrace();
          }
      }
      public void setCurrentSquare( Square s )
      {
        currentSquare = s;
```

```
    }
  }
  // Maintains one square on the board
  class Square extends JPanel {
    private char mark;
    private int location;
    public Square( char m, int loc)
    {
      mark = m;
      location = loc;
      setSize ( 30, 30 );
      setVisible(true);
    }
    public Dimension getPreferredSize() {
      return ( new Dimension( 30, 30 ) );
    }
    public Dimension getMinimumSize() {
      return ( getPreferredSize() );
    }
    public void setMark( char c ) { mark = c; }
    public int getSquareLocation() { return location; }
    public void paintComponent( Graphics g )
    {
```

```
            super.paintComponent( g );
            g.drawRect( 0, 0, 29, 29 );
            g.drawString( String.valueOf( mark ), 11, 20 );
        }
    }
    class SquareListener extends MouseAdapter {
        private TicTacToeClient applet;
        private Square square;
        public SquareListener( TicTacToeClient t, Square s )
        {
            applet = t;
            square = s;
        }
        public void mouseReleased( MouseEvent e )
        {
            applet.setCurrentSquare( square );
            applet.sendClickedSquare( square.getSquareLocation() );
        }
    }
```

Tic-Tac-Toe Server Code :-

```
package tictactoeserver;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
```

```
import java.io.*;
import javax.swing.*;
public class TicTacToeServer extends JFrame {
   private byte board[];
   private boolean xMove;
   private JTextArea output;
   private Player players[];
   private ServerSocket server;
   private int currentPlayer;
   private int flag=0;
   public TicTacToeServer()
   {
      super( "Tic-Tac-Toe Server" );
      board = new byte[ 9 ];
      xMove = true;
      players = new Player[ 2 ];
      currentPlayer = 0;
      // set up ServerSocket
      try {
         server = new ServerSocket( 5000, 2 );
      }
      catch( IOException e ) {
         e.printStackTrace();
         System.exit( 1 );
      }
      output = new JTextArea();
      getContentPane().add( output, BorderLayout.CENTER );
```

```
    output.setText( "Server awaiting connections\n" );
    setSize( 300, 300 );
    show();
  }
  // wait for two connections so game can be played
  public void execute()
  {
    for ( int i = 0; i < players.length; i++ ) {
      try {
        players[ i ] =
          new Player( server.accept(), this, i );
        players[ i ].start();
      }
      catch( IOException e ) {
        e.printStackTrace();
        System.exit( 1 );
      }
    }
    // Player X is suspended until Player O connects.
    // Resume player X now.
    synchronized ( players[ 0 ] ) {
      players[ 0 ].threadSuspended = false;
      players[ 0 ].notify();
    }
  }
  public void display( String s )
  {
```

```
    output.append( s + "\n" );
  }
  // Determine if a move is valid.
  // This method is synchronized because only one move can be
  // made at a time.
  public synchronized boolean validMove( int loc, int player )
  {
    boolean moveDone = false;
    while ( player != currentPlayer ) {
      try {
        wait();
      }
      catch( InterruptedException e ) {
        e.printStackTrace();
      }
    }
    if ( !isOccupied( loc ) ) {
      board[ loc ] =
        (byte) ( currentPlayer == 0 ? 'X' : 'O' );
      currentPlayer = ( currentPlayer + 1 ) % 2;
      players[ currentPlayer ].otherPlayerMoved( loc );
      notify();    // tell waiting player to continue
      return true;
    }
    else
      return false;
  }
```

```
public boolean isOccupied( int loc )
{
  if ( board[ loc ] == 'X' || board [ loc ] == 'O' )
     return true;
  else
     return false;
}
public boolean gameOver()
{
   String win;
   if(currentPlayer==0)
      win="O";
   else
      win="X";
 if((board[0]=='X')&&(board[0]==board[1])&&(board[1]==board[2])&&flag==0)
  {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
 else if((board[3]=='X')&&(board[3]==board[4])&&(board[4]==board[5])&&flag==0)
  {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
 else if((board[6]=='X')&&(board[6]==board[7])&&(board[7]==board[8])&&flag==0)
  {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
 else if((board[0]=='X')&&(board[0]==board[3])&&(board[3]==board[6])&&flag==0)
  {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
 else if((board[1]=='X')&&(board[1]==board[4])&&(board[4]==board[7])&&flag==0)
  {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
 else if((board[2]=='X')&&(board[2]==board[5])&&(board[5]==board[8])&&flag==0)
  {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
 else if((board[0]=='X')&&(board[0]==board[4])&&(board[4]==board[8])&&flag==0)
```

```
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[2]=='X')&&(board[2]==board[4])&&(board[4]==board[6])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   if((board[0]=='O')&&(board[0]==board[1])&&(board[1]==board[2])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[3]=='O')&&(board[3]==board[4])&&(board[4]==board[5])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[6]=='O')&&(board[6]==board[7])&&(board[7]==board[8])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[0]=='O')&&(board[0]==board[3])&&(board[3]==board[6])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[1]=='O')&&(board[1]==board[4])&&(board[4]==board[7])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[2]=='O')&&(board[2]==board[5])&&(board[5]==board[8])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[0]=='O')&&(board[0]==board[4])&&(board[4]==board[8])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
   else if((board[2]=='O')&&(board[2]==board[4])&&(board[4]==board[6])&&flag==0)
    {JOptionPane.showMessageDialog(null,"Player "+win+" won");flag=1;return true;}
// Place code here to test for a winner of the game
    return false;
  }
  public static void main(String args[])
  {
    TicTacToeServer game = new TicTacToeServer();
    game.addWindowListener( new WindowAdapter() {
      public void windowClosing( WindowEvent e )
```

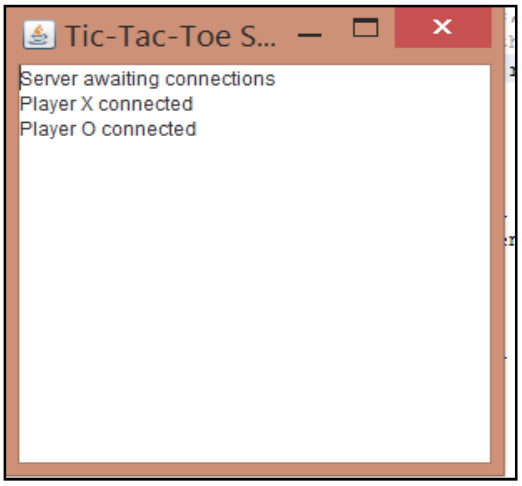
```
            {
              System.exit( 0 );
            }
          }
        );
        game.execute();
    }
}
// Player class to manage each Player as a thread
class Player extends Thread {
    private Socket connection;
    private DataInputStream input;
    private DataOutputStream output;
    private TicTacToeServer control;
    private int number;
    private char mark;
    protected boolean threadSuspended = true;
    public Player( Socket s, TicTacToeServer t, int num )
    {
        mark = ( num == 0 ? 'X' : 'O' );
        connection = s;
        try {
            input = new DataInputStream(
                        connection.getInputStream() );
            output = new DataOutputStream(
                        connection.getOutputStream() );
        }
```

```
    catch( IOException e ) {
      e.printStackTrace();
      System.exit( 1 );
    }
    control = t;
    number = num;
  }
  public void otherPlayerMoved( int loc )
  {
    try {
      output.writeUTF( "Opponent moved" );
      output.writeInt( loc );
    }
    catch ( IOException e ) { e.printStackTrace(); }
  }
  public void run()
  {
    boolean done = false;
    try {
      control.display( "Player " +
        ( number == 0 ? 'X' : 'O' ) + " connected" );
      output.writeChar( mark );
      output.writeUTF( "Player " +
        ( number == 0 ? "X connected\n" :
                  "O connected, please wait\n" ) );
      // wait for another player to arrive
      if ( mark == 'X' ) {
```

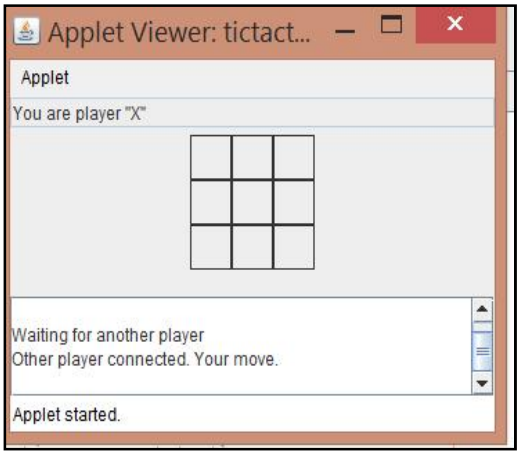
```
    output.writeUTF( "Waiting for another player" );
    try {
      synchronized( this ) {
        while ( threadSuspended )
          wait();
      }
    }
    catch ( InterruptedException e ) {
      e.printStackTrace();
    }
    output.writeUTF(
      "Other player connected. Your move." );
  }
  // Play game
  while ( !done ) {
    int location = input.readInt();

    if ( control.validMove( location, number ) ) {
      control.display( "loc: " + location );
      output.writeUTF( "Valid move." );
    }
    else
      output.writeUTF( "Invalid move, try again" );
    if ( control.gameOver() )
    {done = true;
    }
  }
```

```
        connection.close();
      }
      catch( IOException e ) {
        e.printStackTrace();
        System.exit( 1 );
      }
    }
}
```

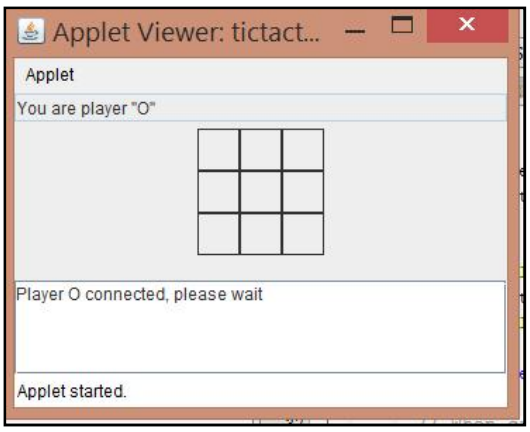
Output :-



Tic-Tac-Toe Server program is run first so that both the user can connect to it and play against each other.



Tic-Tac-Toe Client program is run in one computer and is connected to the server as player 'X'.



Tic-Tac-Toe Client program is run in another computer and is connected to the server as player 'O'.

Now, both the user are connected and can play the game against each other online.