# Programação Orientada a Objetos

Programação em Redes

# 6. Networking

**Sumário**

# 6.1 Introduction

- Networking package is `java.net`
  - Socket-based communications
    - Applications view networking as streams of data
    - Connection-based protocol
    - Uses TCP (Transmission Control Protocol)
  - Packet-based communications
    - Individual packets transmitted
    - Connectionless service
    - Uses UDP (User Datagram Protocol)

# 6.2 Establishing a Simple Server Using Stream Sockets

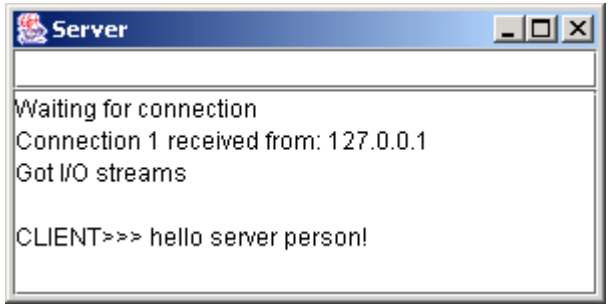
- Five steps to create a simple server in Java
  - `ServerSocket` object
    - Registers an available port and a maximum number of clients
  - Each client connection handled with `Socket` object
    - Server blocks until client connects
  - Sending and receiving data
    - `OutputStream` to send and `InputStream` to receive data
    - Methods `getInputStream` and `getOutputstream`
      - Use on `Socket` object
  - Process phase
    - Server and Client communicate via streams
  - Close streams and connections

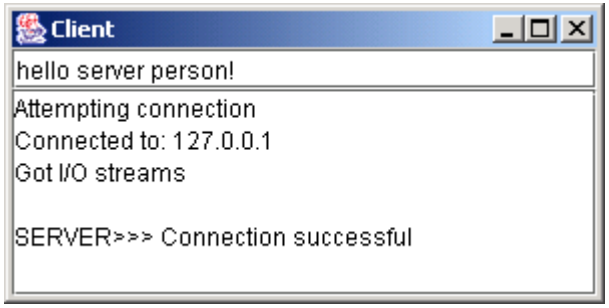
# 6.5 Establishing a Simple Client Using Stream Sockets

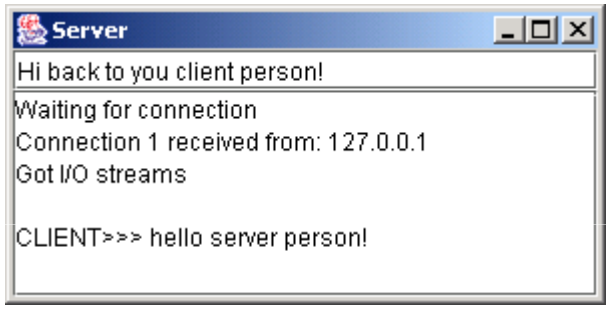
- Four steps to create a simple client in Java
  - Create a `Socket` object for the client
  - Obtain `Socket`'s `InputStream` and `Outputstream`
  - Process information communicated
  - Close streams and `Socket`

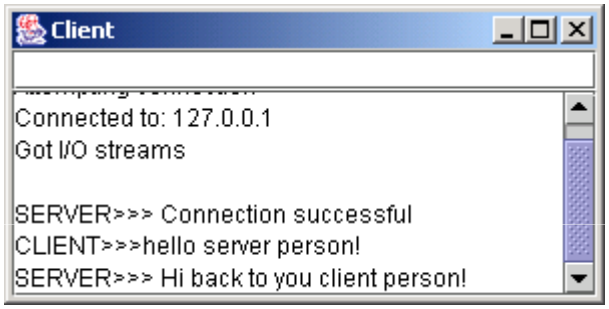
# 6.6 Client/Server Interaction with Stream Socket Connections

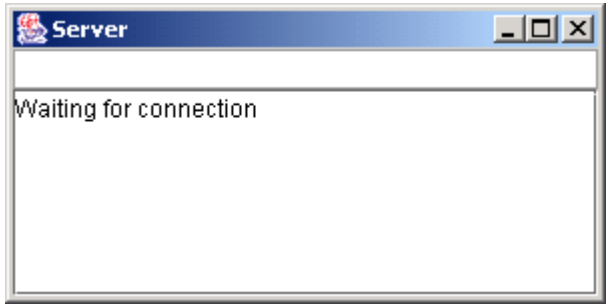
- Client/server chat application
  - Uses stream sockets as described in last two sections

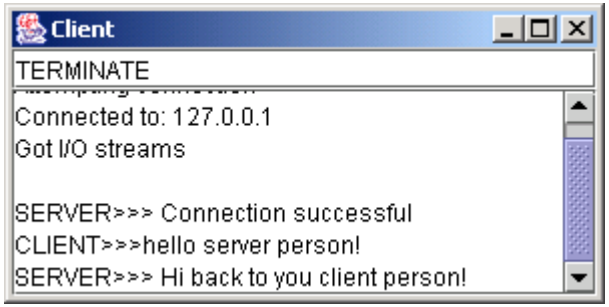

Server
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams
CLIENT>>> hello server person!


Client
hello server person!
Attempting connection
Connected to: 127.0.0.1
Got I/O streams
SERVER>>> Connection successful


Server
Hi back to you client person!
Waiting for connection
Connection 1 received from: 127.0.0.1
Got I/O streams
CLIENT>>> hello server person!


Client
Connected to: 127.0.0.1
Got I/O streams
SERVER>>> Connection successful
CLIENT>>>hello server person!
SERVER>>> Hi back to you client person!


Server
Waiting for connection


Client
TERMINATE
Connected to: 127.0.0.1
Got I/O streams
SERVER>>> Connection successful
CLIENT>>>hello server person!
SERVER>>> Hi back to you client person!

```
1   // Server.java
2   // Set up a Server that will receive a connection from a client, send
3   // a string to the client, and close the connection.
4   import java.io.*;
5   import java.net.*;
6   import java.awt.*;
7   import java.awt.event.*;
8   import javax.swing.*;
9
10  public class Server extends JFrame {
11     private JTextField enterField;
12     private JTextArea displayArea;
13     private ObjectOutputStream output;
14     private ObjectInputStream input;
15     private ServerSocket server;
16     private Socket connection;
17     private int counter = 1;
18
19     // set up GUI
20     public Server()
21     {
22        super( "Server" );
23
24        Container container = getContentPane();
25
```

Listen on a ServerSocket; the connection is a Socket

```
26          // create enterField and register listener
27          enterField = new JTextField();
28          enterField.setEditable( false );
29          enterField.addActionListener(
30             new ActionListener() {
31
32                // send message to client
33                public void actionPerformed( ActionEvent event )
34                {
35                   sendData( event.getActionCommand() );
36                   enterField.setText( "" );
37                }
38             }
39          );
40
41          container.add( enterField, BorderLayout.NORTH );
42
43          // create displayArea
44          displayArea = new JTextArea();
45          container.add( new JScrollPane( displayArea ),
46             BorderLayout.CENTER );
47
48          setSize( 300, 150 );
49          setVisible( true );
50
51       } // end Server constructor
52
```

```
53    // set up and run server
54    public void runServer()
55    {
56       // set up server to receive connections; process connections
57       try {
58
59          // Step 1: Create a ServerSocket.
60          server = new ServerSocket( 12345, 100 );
61
62          while ( true ) {
63
64             try {
65                waitForConnection(); // Step 2: Wait for a connection.
66                getStreams();        // Step 3: Get input & output streams.
67                processConnection(); // Step 4: Process connection.
68             }
69
70             // process EOFException when client closes connection
71             catch ( EOFException eofException ) {
72                System.err.println( "Server terminated connection" );
73             }
74
75             finally {
76                closeConnection();   // Step 5: Close connection.
77                ++counter;
78             }
```

Create ServerSocket at port 12345 with queue of length 100

```
79
80          } // end while
81
82       } // end try
83
84       // process problems with I/O
85       catch ( IOException ioException ) {
86          ioException.printStackTrace();
87       }
88
89    } // end method runServer
90
91    // wait for connection to arrive, then display connection info
92    private void waitForConnection() throws IOException
93    {
94       displayMessage( "Waiting for connection\n" );
95       connection = server.accept(); // allow server to accept connec
96       displayMessage( "Connection " + counter + " received from: " +
97          connection.getInetAddress().getHostName() );
98    }
99
100   // get streams to send and receive data
101   private void getStreams() throws IOException
102   {
```

Method accept waits for connection

Output name of computer that connected

```
103        // set up output stream for objects
104        output = new ObjectOutputStream( connection.getOutputStream() );
105        output.flush(); // flush output buffer to send header information
106
107        // set up input stream for objects
108        input = new ObjectInputStream( connection.getInputStream() );
109
110        displayMessage( "\nGot I/O streams\n" );
111     }
112
113     // process connection with client
114     private void processConnection() throws IOException
115     {
116        // send connection successful message to client
117        String message = "Connection successful";
118        sendData( message );
119
120        // enable enterField so server user can send messages
121        setTextFieldEditable( true );
122
123        do { // process messages sent from client
124
```

Method flush empties output buffer and sends header information

```
125          // read message and display it
126          try {
127             message = ( String ) input.readObject();
128             displayMessage( "\n" + message );
129          }
130
131          // catch problems reading from client
132          catch ( ClassNotFoundException classNotFoundException ) {
133             displayMessage( "\nUnknown object type received" );
134          }
135
136       } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
137
138    } // end method processConnection
139
140    // close streams and socket
141    private void closeConnection()
142    {
143       displayMessage( "\nTerminating connection\n" );
144       setTextFieldEditable( false ); // disable enterField
145
146       try {
147          output.close();
148          input.close();
149          connection.close();
150       }
```

Read String from client and display it

Method closeConnection closes streams and sockets

```
151        catch( IOException ioException ) {
152           ioException.printStackTrace();
153        }
154     }
155
156     // send message to client
157     private void sendData( String message )
158     {
159        // send object to client
160        try {
161           output.writeObject( "SERVER>>> " + message );
162           output.flush();
163           displayMessage( "\nSERVER>>> " + message );
164        }
165
166        // process problems sending object
167        catch ( IOException ioException ) {
168           displayArea.append( "\nError writing object" );
169        }
170     }
171
172     // utility method called from other threads to manipulate
173     // displayArea in the event-dispatch thread
174     private void displayMessage( final String messageToDisplay )
175     {
```

Method flush empties output buffer and sends header information

```
176       // display message from event-dispatch thread of execution
177       SwingUtilities.invokeLater(
178          new Runnable() {  // inner class to ensure GUI updates properly
179
180             public void run() // updates displayArea
181             {
182                displayArea.append( messageToDisplay );
183                displayArea.setCaretPosition(
184                   displayArea.getText().length() );
185             }
186
187          }  // end inner class
188
189       ); // end call to SwingUtilities.invokeLater
190    }
191
192    // utility method called from other threads to manipulate
193    // enterField in the event-dispatch thread
194    private void setTextFieldEditable( final boolean editable )
195    {
196       // display message from event-dispatch  thread of execution
197       SwingUtilities.invokeLater(
198          new Runnable() {  // inner class to ensure GUI updates properly
199
```

```
200             public void run()  // sets enterField's editability
201             {
202                enterField.setEditable( editable );
203             }
204
205          }  // end inner class
206
207       ); // end call to SwingUtilities.invokeLater
208    }
209
210    public static void main( String args[] )
211    {
212       Server application = new Server();
213       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
214       application.runServer();
215    }
216
217 }  // end class Server
```

```
1   // Client.java
2   // Client that reads and displays information sent from a Server.
3   import java.io.*;
4   import java.net.*;
5   import java.awt.*;
6   import java.awt.event.*;
7   import javax.swing.*;
8
9   public class Client extends JFrame {
10     private JTextField enterField;
11     private JTextArea displayArea;
12     private ObjectOutputStream output;
13     private ObjectInputStream input;
14     private String message = "";
15     private String chatServer;
16     private Socket client;
17
18     // initialize chatServer and set up GUI
19     public Client( String host )
20     {
21        super( "Client" );
22
23        chatServer = host; // set server to which this client connects
24
25        Container container = getContentPane();
```

The client is a Socket

```
26
27          // create enterField and register listener
28          enterField = new JTextField();
29          enterField.setEditable( false );
30          enterField.addActionListener(
31             new ActionListener() {
32
33                // send message to server
34                public void actionPerformed( ActionEvent event )
35                {
36                   sendData( event.getActionCommand() );
37                   enterField.setText( "" );
38                }
39             }
40          );
41
42          container.add( enterField, BorderLayout.NORTH );
43
44          // create displayArea
45          displayArea = new JTextArea();
46          container.add( new JScrollPane( displayArea ),
47             BorderLayout.CENTER );
48
49          setSize( 300, 150 );
50          setVisible( true );
```

```
51
52     } // end Client constructor
53
54     // connect to server and process messages from server
55     private void runClient()
56     {
57        // connect to server, get streams, process connection
58        try {
59           connectToServer(); // Step 1: Create a Socket to make connection
60           getStreams();      // Step 2: Get the input and output streams
61           processConnection(); // Step 3: Process connection
62        }
63
64        // server closed connection
65        catch ( EOFException eofException ) {
66           System.err.println( "Client terminated connection" );
67        }
68
69        // process problems communicating with server
70        catch ( IOException ioException ) {
71           ioException.printStackTrace();
72        }
73
74        finally {
75           closeConnection(); // Step 4: Close connection
76        }
```

```
77
78      } // end method runClient
79
80      // connect to server
81      private void connectToServer() throws IOException
82      {
83         displayMessage( "Attempting connection\n" );
84
85         // create Socket to make connection to server
86         client = new Socket( InetAddress.getByName( chatServer ), 12345 );
87
88         // display connection information
89         displayMessage( "Connected to: " +
90            client.getInetAddress().getHostName() );
91      }
92
93      // get streams to send and receive data
94      private void getStreams() throws IOException
95      {
96         // set up output stream for objects
97         output = new ObjectOutputStream( client.getOutputStream() );
98         output.flush(); // flush output buffer to send header information
99
100        // set up input stream for objects
101        input = new ObjectInputStream( client.getInputStream() );
```

Create a client that will connect with port 12345 on the server

Notify the user that we have connected

Get the streams to send and receive data

```
102
103        displayMessage( "\nGot I/O streams\n" );
104     }
105
106     // process connection with server
107     private void processConnection() throws IOException
108     {
109        // enable enterField so client user can send messages
110        setTextFieldEditable( true );
111
112        do { // process messages sent from server
113
114           // read message and display it
115           try {
116              message = ( String ) input.readObject();
117              displayMessage( "\n" + message );
118           }
119
120           // catch problems reading from server
121           catch ( ClassNotFoundException classNotFoundException ) {
122              displayMessage( "\nUnknown object type received" );
123           }
124
125        } while ( !message.equals( "SERVER>>> TERMINATE" ) );
126
127     } // end method processConnection
```

Read String from client and display it

```
128
129    // close streams and socket
130    private void closeConnection()
131    {
132       displayMessage( "\nClosing connection" );
133       setTextFieldEditable( false ); // disable enterField
134
135       try {
136          output.close();
137          input.close();
138          client.close();
139       }
140       catch( IOException ioException ) {
141          ioException.printStackTrace();
142       }
143    }
144
145    // send message to server
146    private void sendData( String message )
147    {
148       // send object to server
149       try {
150          output.writeObject( "CLIENT>>> " + message );
151          output.flush();
152          displayMessage( "\nCLIENT>>> " + message );
153       }
```

Method closeConnection closes streams and sockets

Method flush empties output buffer and sends header information

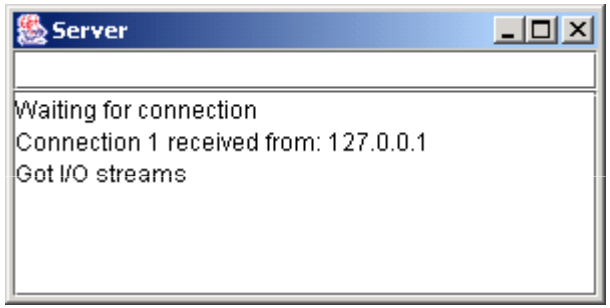
```
154
155         // process problems sending object
156         catch ( IOException ioException ) {
157            displayArea.append( "\nError writing object" );
158         }
159      }
160
161      // utility method called from other threads to manipulate
162      // displayArea in the event-dispatch thread
163      private void displayMessage( final String messageToDisplay )
164      {
165         // display message from GUI thread of execution
166         SwingUtilities.invokeLater(
167            new Runnable() {  // inner class to ensure GUI updates properly
168
169               public void run() // updates displayArea
170               {
171                  displayArea.append( messageToDisplay );
172                  displayArea.setCaretPosition(
173                     displayArea.getText().length() );
174               }
175
176            }  // end inner class
177
178         ); // end call to SwingUtilities.invokeLater
179      }
```

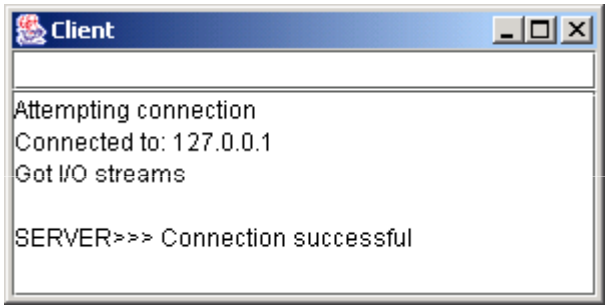
```
180
181    // utility method called from other threads to manipulate
182    // enterField in the event-dispatch thread
183    private void setTextFieldEditable( final boolean editable )
184    {
185       // display message from GUI thread of execution
186       SwingUtilities.invokeLater(
187          new Runnable() {  // inner class to ensure GUI updates properly
188
189             public void run()  // sets enterField's editability
190             {
191                enterField.setEditable( editable );
192             }
193
194          }  // end inner class
195
196       ); // end call to SwingUtilities.invokeLater
197    }
198
199    public static void main( String args[] )
200    {
201       Client application;
202
```

```
203         if ( args.length == 0 )
204            application = new Client( "127.0.0.1" );
205         else
206            application = new Client( args[ 0 ] );
207
208         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
209         application.runClient();
210      }
211
212 } // end class Client
```

Create a client to connect to the localhost

Connect to a host supplied by the user



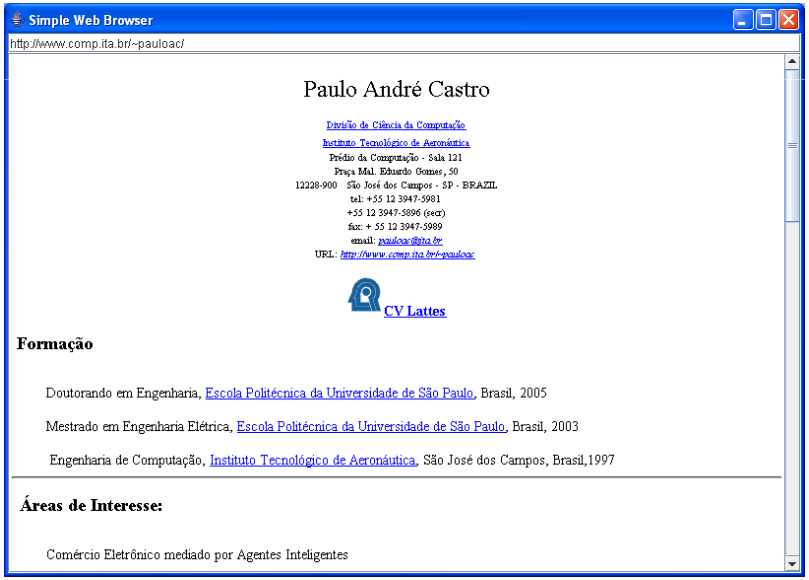



# Exercício

- Estabeleça um chat com um colega através da porta 8189.

# 6.3 Reading a File on a Web Server

- Swing GUI component `JEditorPane`
  - Can display simple text and HTML formatted text
  - Can be used as a simple Web browser
    - Retrieves files from a Web server at a given URI



```
1  // ReadServerFile.java
2  // Use a JEditorPane to display the contents of a file on a Web server.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.net.*;
6  import java.io.*;
7  import javax.swing.*;
8  import javax.swing.event.*;
9
10 public class ReadServerFile extends JFrame {
11    private JTextField enterField;
12    private JEditorPane contentsArea;
13
14    // set up GUI
15    public ReadServerFile()
16    {
17       super( "Simple Web Browser" );
18
19       Container container = getContentPane();
20
21       // create enterField and register its listener
22       enterField = new JTextField( "Enter file URL here" );
23       enterField.addActionListener(
24          new ActionListener() {
25
```

File displayed in JEditorPane

```
26              // get document specified by user
27              public void actionPerformed( ActionEvent event )
28              {
29                 getThePage( event.getActionCommand() );
30              }
31
32           } // end inner class
33
34        ); // end call to addActionListener
35
36        container.add( enterField, BorderLayout.NORTH );
37
38        // create contentsArea and register HyperlinkEvent liste
39        contentsArea = new JEditorPane();
40        contentsArea.setEditable( false );
41        contentsArea.addHyperlinkListener(
42           new HyperlinkListener() {
43
44              // if user clicked hyperlink, go to specified page
45              public void hyperlinkUpdate( HyperlinkEvent event )
46              {
47                 if ( event.getEventType() ==
48                      HyperlinkEvent.EventType.ACTIVATED )
49                    getThePage( event.getURL().toString() );
50              }
51
```

Register a HyperlinkListener to handle HyperlinkEvents

Method hyperlinkUpdate called when hyperlink clicked

Determine type of hyperlink

Get URL of hyperlink and retrieve page

```
52          } // end inner class
53
54       ); // end call to addHyperlinkListener
55
56       container.add( new JScrollPane( contentsArea ),
57          BorderLayout.CENTER );
58       setSize( 400, 300 );
59       setVisible( true );
60
61    } // end constructor ReadServerFile
62
63    // load document
64    private void getThePage( String location )
65    {
66       // load document and display location
67       try {
68          contentsArea.setPage( location );
69          enterField.setText( location );
70       }
71       catch ( IOException ioException ) {
72          JOptionPane.showMessageDialog( this,
73             "Error retrieving specified URL", "Bad URL",
74             JOptionPane.ERROR_MESSAGE );
75       }
76
77    } // end method getThePage
```

Method setPage downloads document and displays it in JEditorPane

```
78
79     public static void main( String args[] )
80     {
81        ReadServerFile application = new ReadServerFile();
82        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
83     }
84
85  } // end class ReadServerFile
```

# Exercício

- Incluir um barra de ferramentas acima do campo de edição da URL com os botões “Home” (Página Inicial) e “Atualizar” e seus respectivos comportamentos
- Criar botão “Voltar” e seu respectivo comportamento

# Desafio

- Criar novas versões das classes de Client e Server do package ClientServer tais que seja possível estabelecer um chat com múltiplos clientes. Cada cliente deve ser identificado Client1, client2,etc. de acordo com a ordem de conexão ao servidor.
- Ao ser enviado mensagem de um cliente qualquer todos os clientes e servidor devem receber tal mensagem com a identificação ClientX>>>
- O servidor não precisa ter uma interface gráfica