Java – Aula 08

Rede:

Comunicação entre processos

31/10/2012

Celso Olivete Júnior

olivete@fct.unesp.br

Aula passada:

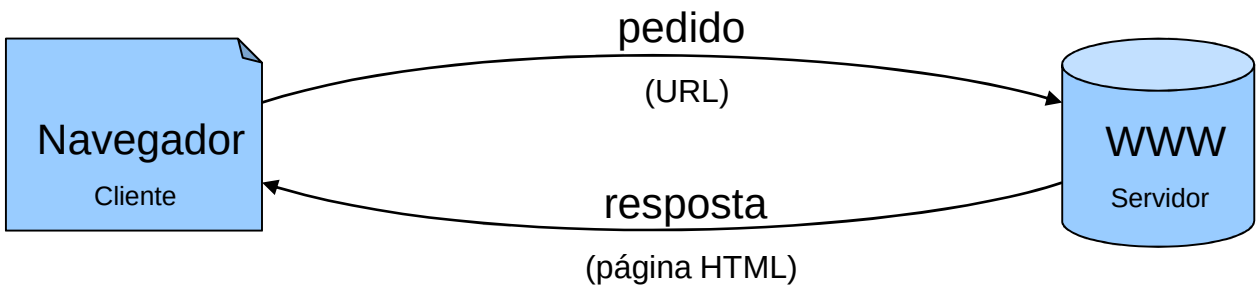
- Multithreading

Na aula de hoje:

- Introdução
- Manipulando URLs
- Lendo um arquivo de um Servidor Web
- Estabelecendo um simples Servidor usando Stream Sockets
- Estabelecendo um simples Cliente usando Stream Sockets
- Interação Cliente/Servidor com conexão Stream Socket

Introdução

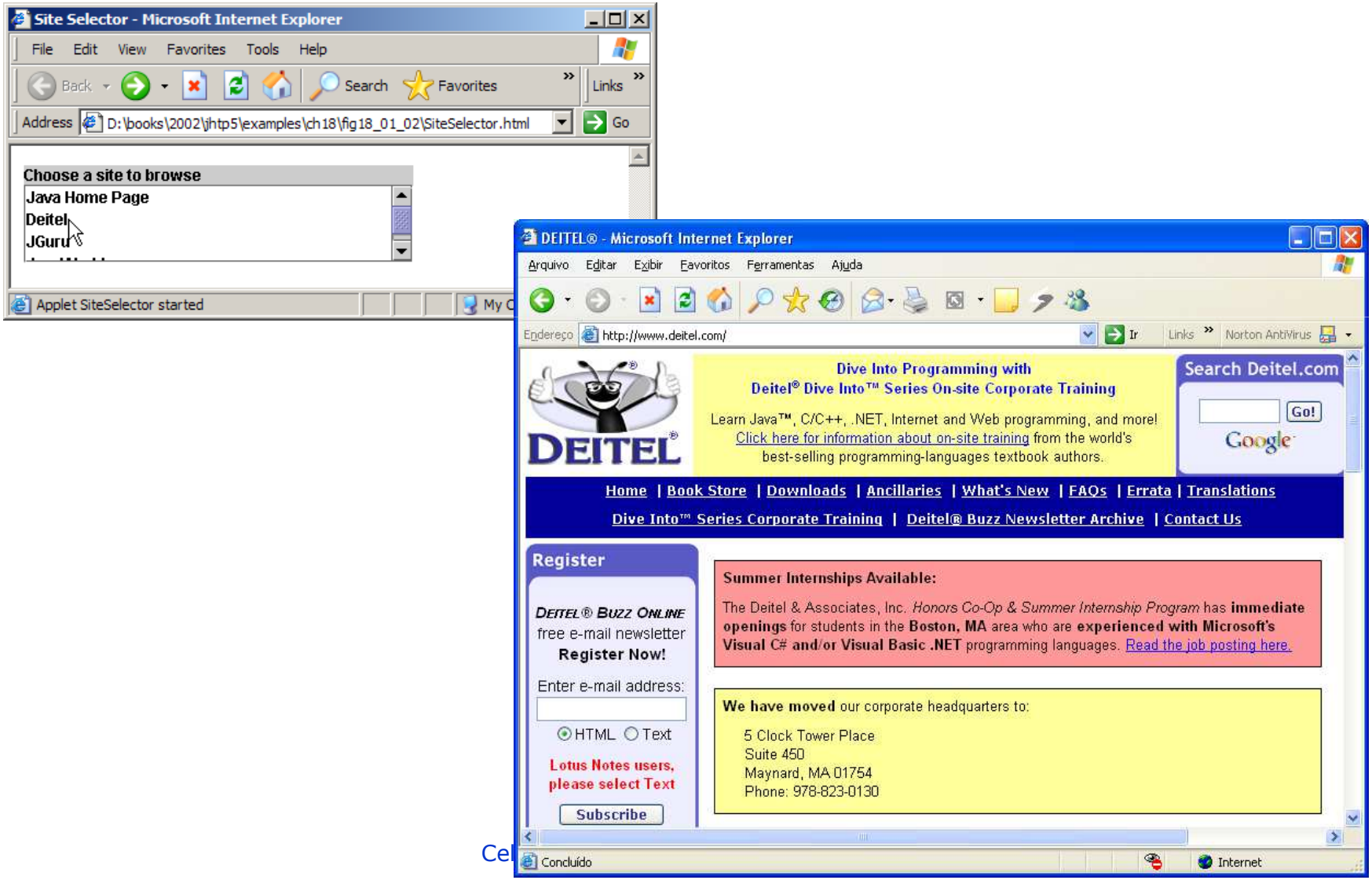
- O pacote de rede é java.net
  - Permite comunicação baseada em sockets;
  - Também permite a comunicação baseada em pacotes;
  - Será focalizado os dois lados de um relacionamento cliente/servidor;
    - O cliente pede que alguma ação seja executada; o servidor executa a ação e responde para o clientes.
    - Ex: Pedido/resposta efetuado pelos navegadores e os servidores WWW.



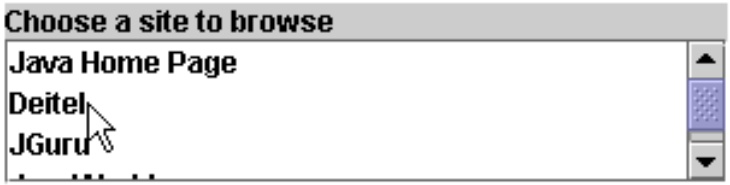
Introdução

- Java fornece dois tipos de sockets:
  - Sockets de Fluxo
    - O processo estabelece uma conexão com outro processo. Enquanto essa conexão estiver estabelecida, os dados fluem entre os processos em fluxos contínuos;
    - Chamamos de Serviço Orientado a Conexão;
    - Ex: TCP (Transmission Control Protocol)
  - Sockets de Datagrama
    - Transmite pacotes individuais de informação;
    - Serviço sem conexão;
    - Ex: UDP (User Datagram Protocol)
      - Os pacotes podem ser perdidos, duplicados e até chegar fora de sequência.

Exemplo Applets: seleciona uma página a partir de um Jlist e o navegador exibe o conteúdo

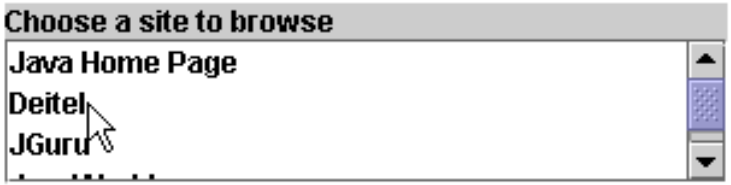


Manipulando URL's



- HyperText Transfer Protocol (HTTP)
  - Utiliza URIs (Uniform Resource Identifiers - identificador de dados) para localizar os dados:
    - URIs são frequentemente chamadas de URLs (Uniform Resource Locators)
    - Refere-se aos arquivos, diretórios e objetos complexos (ex: pesquisa no BD).

SiteSelector.html




```
1   <html>
2   <title>Site Selector</title>
3   <body>
4     <applet code = "SiteSelector.class" width = "300" height = "75">
5       <param name = "title0" value = "Java Home Page">
6       <param name = "location0" value = "http://java.sun.com/">
7       <param name = "title1" value = "Deitel">
8       <param name = "location1" value = "http://www.deitel.com/">
9       <param name = "title2" value = "JGuru">
10      <param name = "location2" value = "http://www.jGuru.com/">
11      <param name = "title3" value = "JavaWorld">
12      <param name = "location3" value = "http://www.javaworld.com/">
13    </applet>
14  </body>
15  </html>
```

Declara tags `param` para a applet

Servirão para carregar a página

Exemplo:

String titulo = getParameter(title0);

//recupera o valor associado, neste caso **Java Home Page**

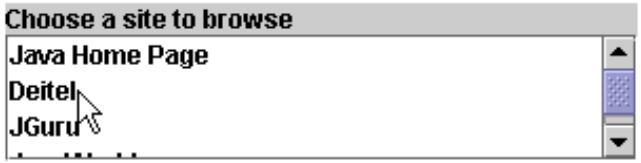
SiteSelector.java

```
1   // Fig. 18.2: SiteSelector.java
2   //Carrega um documento a partir da URL.
3   import java.net.*;
4   import java.util.*;
5   import java.awt.*;
6   import java.applet.AppletContext;
7   import javax.swing.*;
8   import javax.swing.event.*;
10  public class SiteSelector extends JApplet {
11    private HashMap sites;    // nomes e URLs dos sites
12    private Vector siteNames;   // nomes dos sites
13    private JList siteChooser;  // lista de sites a escolher
14
15    // lê parâmetros html e configura a GUI
16    public void init()
17    {
18      // cria HashMap e Vector
19      sites = new HashMap();
20      siteNames = new Vector();
21      // obtem parametros do documento HTML
22      // que irão compor o JList
23      getSitesFromHTMLParameters();
24
```

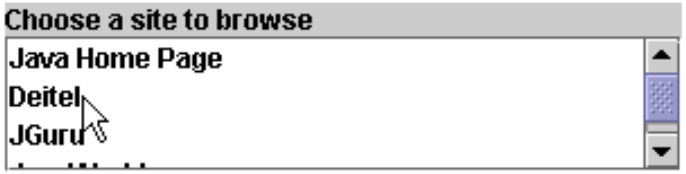
Cria um objeto `HashMap` e um objeto `Vector`

Chave é a String na JList (nome do site)

Valor é a URL



SiteSelector.java




```
25    // cria componentes GUI e o layout
26    Container container = getContentPane();
27    container.add( new JLabel( "Choose a site to browse" ),
28       BorderLayout.NORTH );
30    siteChooser = new JList( siteNames ); // preenche a Jlist com o Vector
31    siteChooser.addListSelectionListener(
32
33       new ListSelectionListener() {// classe anônima
34
35          // vai para o site selecionado pelo usuário
36          public void valueChanged( ListSelectionEvent event )
37          {
38             // recupera o nome do site selecionado
39             Object object = siteChooser.getSelectedValue();
41             // utiliza o nome do site para localizar a url
42             URL newDocument = ( URL ) sites.get( object );
44             // obtem o container de applets
45             AppletContext browser = getAppletContext();
47             // faz com que o container de applets mude de página
48             browser.showDocument( newDocument );
49          }
50
```

O método `valueChanged` vai para o Web site selecionado

Cria o documento

Mostra o documento no browser

SiteSelector.java

```
<param name = "title0" value = "Java Home Page">
<param name = "location0" value = "http://java.sun.com/">
```

```
51          } // fim da classe anônima
53        ); // fim da chamada para addListSelectionListener
54
55        container.add( new JScrollPane( siteChooser ),
56           BorderLayout.CENTER );
57
58     } // fim do método init
60     // obtem parametros do documento HTML
61     private void getSitesFromHTMLParameters()
62     {
63     // verifica os parametros applet no documento HTML e adiciona para HashMap
64        String title, location; // titulo e localização do site
65        URL url; //url da localização
66        int counter = 0; //conta numeros de sites
67
68        title = getParameter( "title" + counter ); // retorna o primeiro titulo do site
69
70        // faz um loop até que não haja mais parametros no documento HTML
71        while ( title != null ) {
73           // obtem localização do site
74           location = getParameter( "location" + counter );
75
```

Atribui o título do site

Atribui a localização do site

SiteSelector.java

```
76          // coloca title/URL no HashMap sites e title no Vector siteNames
77          try {
78             url = new URL( location ); // converte a localização em URL
79             sites.put( title, url );   // coloca title/URL no HashMap
80             siteNames.add( title );    // coloca title no Vector
81          }
82
83          // URL mal formada
84          catch ( MalformedURLException urlException ) {
85             urlException.printStackTrace();
86          }
87
88          ++counter;
89          title = getParameter( "title" + counter ); // obtem o proximo title
90
91       } // fim while
92
93    } // fim method getSitesFromHTMLParameters
94
95 } // fim class SiteSelector
```

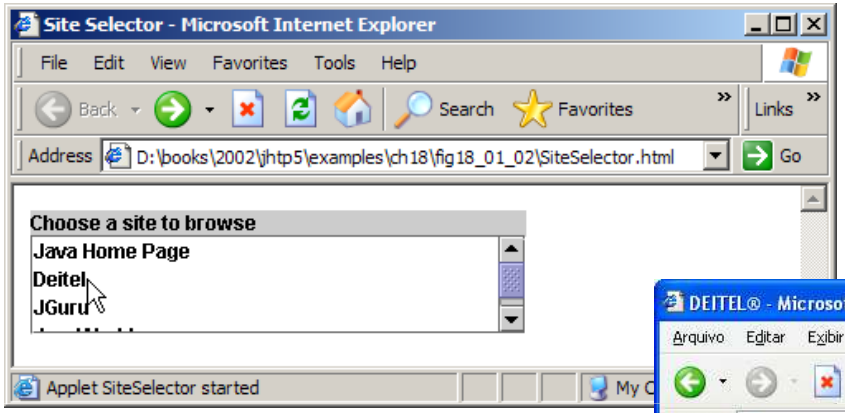
Cria a URL do local

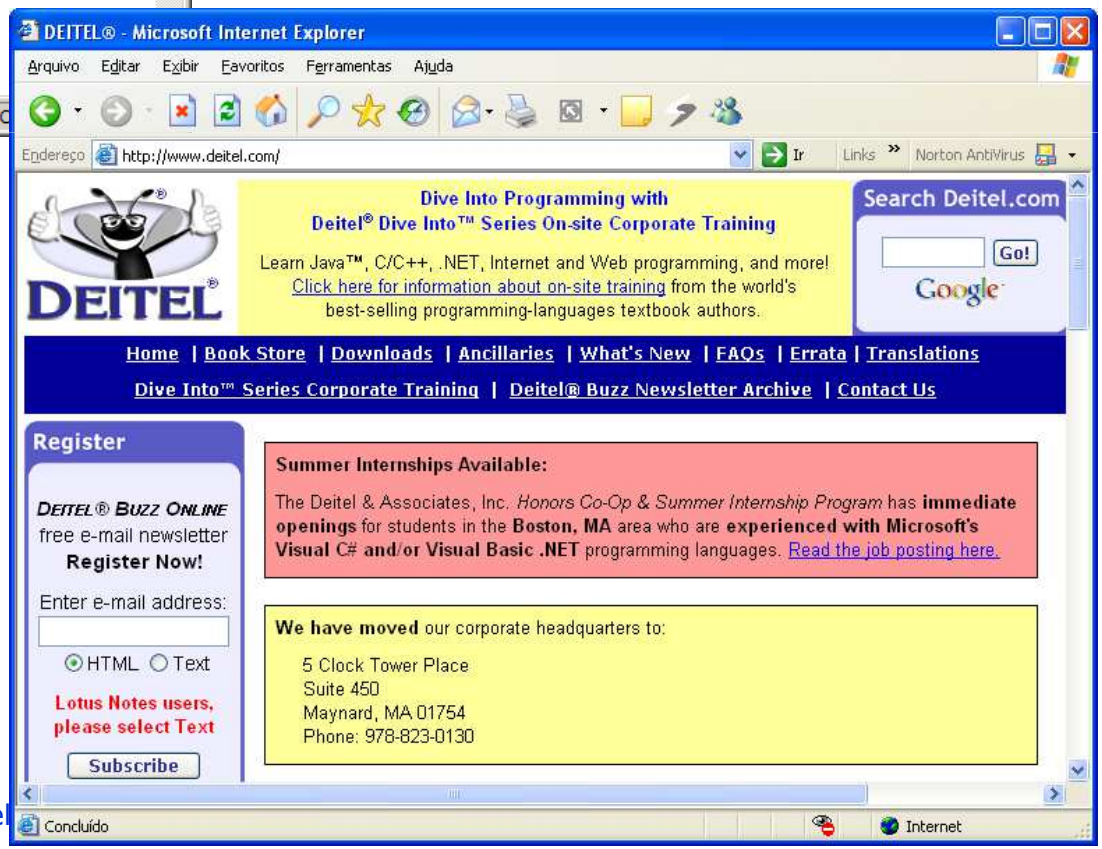
Adiciona a URL no `HashMap`

Adiciona o `title` no `Vector`

Busca o próximo título

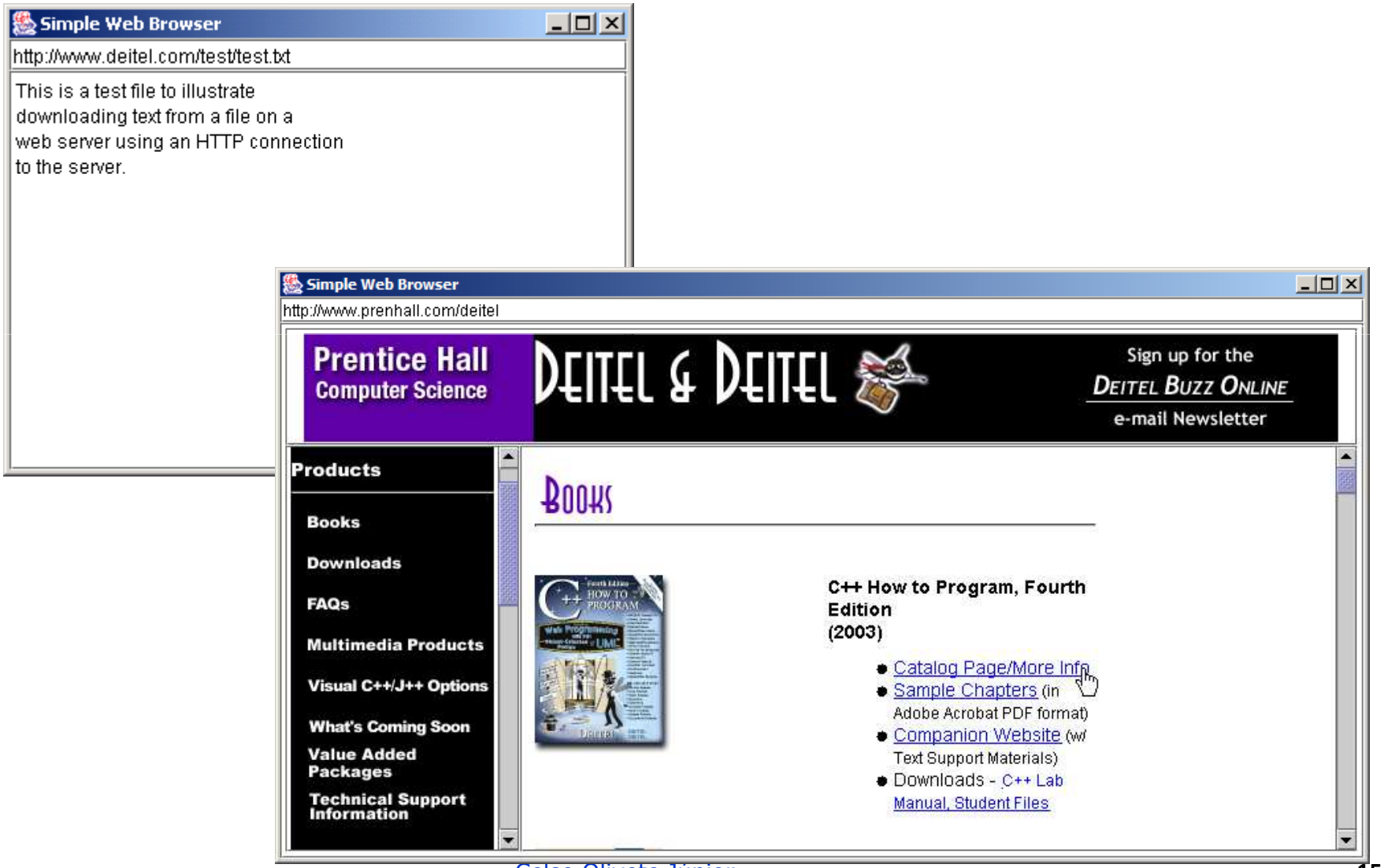
SiteSelector.java





Lendo um arquivo em um Servidor Web

- Utiliza-se o componente da Swing GUI JEditorPane
  - Ele pode mostrar simples textos e textos formatados em HTML;
  - Pode ser realizada navegação a partir de *links*
  - Pode ser usado como um simples navegador
    - Recupera arquivos do servidor Web para uma dada URI.


Simple Web Browser
http://www.deitel.com/test/test.txt
This is a test file to illustrate
downloading text from a file on a
web server using an HTTP connection
to the server.
Simple Web Browser
http://www.prenhall.com/deitel
Prentice Hall
Computer Science
DEITEL & DEITEL
Sign up for the
DEITEL BUZZ ONLINE
e-mail Newsletter
Products
Books
Downloads
FAQs
Multimedia Products
Visual C++/J++ Options
What's Coming Soon
Value Added Packages
Technical Support Information
BOOKS
C++ How to Program, Fourth Edition
(2003)
Catalog Page/More Info
Sample Chapters (in Adobe Acrobat PDF format)
Companion Website (w/ Text Support Materials)
Downloads - C++ Lab Manual, Student Files

ReadServerFile.java

```
1   // Fig. 18.3: ReadServerFile.java
2   // Usa um JEditorPane para mostrar o conteudo de um arquivo em um servidor Web.
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; // para inserir o endereço
12     private JEditorPane contentsArea; // para apresentar o conteúdo
13
14     // configura GUI
15     public ReadServerFile()
16     {
17        super( "Simple Web Browser" );
19        Container container = getContentPane();
21        // criar o enterField e registra seu listener
22        enterField = new JTextField( "Enter file URL here" );
23        enterField.addActionListener(// ao pressionar a tecla enter...
24           new ActionListener() {
```

O arquivo será mostrado no `JEditorPane`

ReadServerFile.java

```
26          // carrega documento especificado pelo usuário
27          public void actionPerformed( ActionEvent event )
28          {
29             getThePage( event.getActionCommand() );
30          }
31
32       } // fim da classe inner
34    ); // fim da chamada para addActionListener
36    container.add( enterField, BorderLayout.NORTH );
38    // cria contentsArea e registra HyperlinkEvent listener
39    contentsArea = new JEditorPane();
40    contentsArea.setEditable( false );
41    contentsArea.addHyperlinkListener(
42       new HyperlinkListener() {
43
44          // se usuário clicou no hyperlink, chama hyperlinkUpdate
45          public void hyperlinkUpdate( HyperlinkEvent event )
46          {
47             if ( event.getEventType() ==
48                HyperlinkEvent.EventType.ACTIVATED )
49                getThePage( event.getURL().toString() );
50          }
```

Registra uma HyperlinkListener para manipular HyperlinkEvents

O método hyperlinkUpdate é chamado quando se clica em um hyperlink

Determina o tipo de hyperlink

Busca a URL do hyperlink e recupera a página

ReadServerFile.java

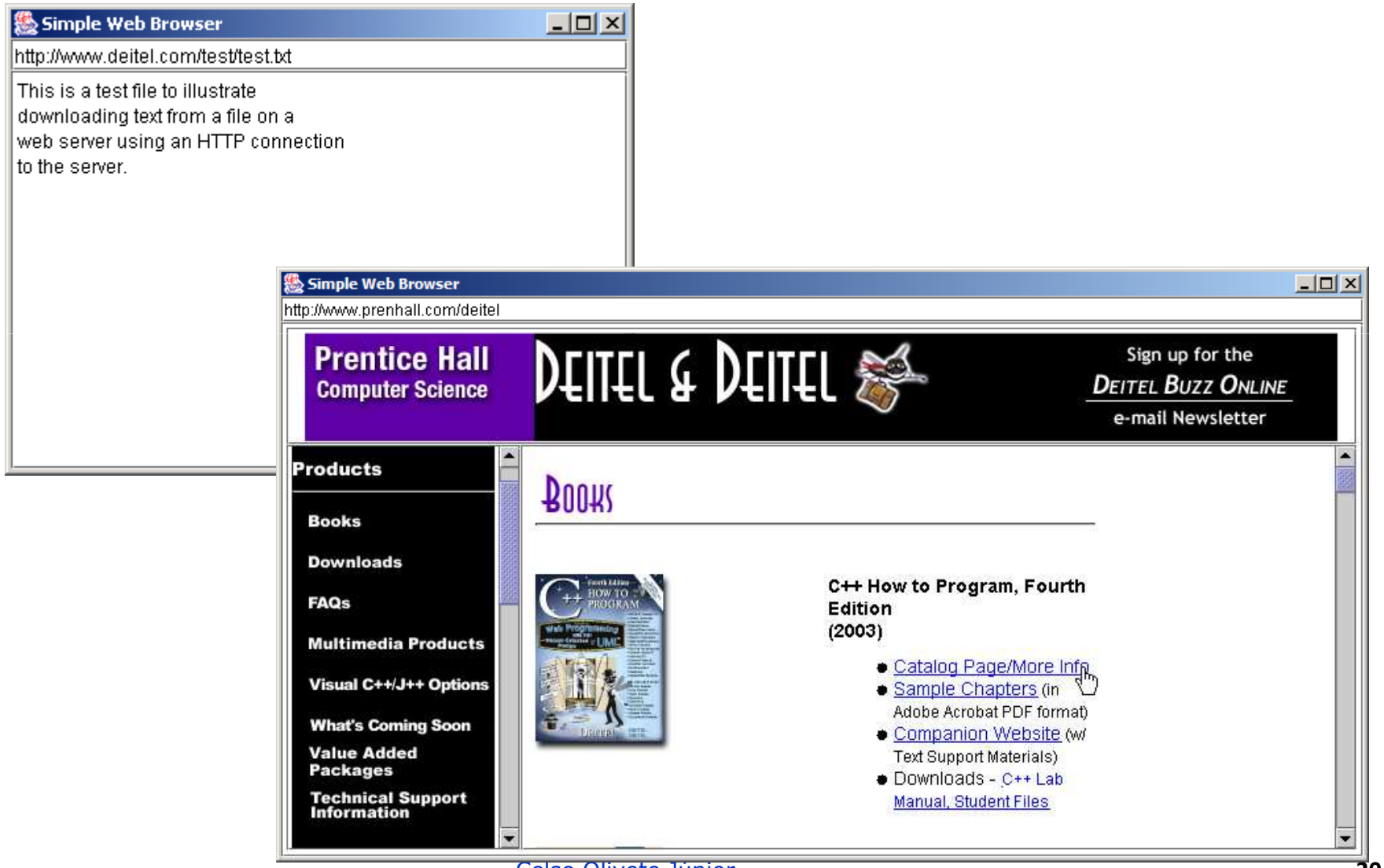
```
52          } // fim da classe inner
53
54       ); // fim da chamada para addHyperlinkListener
56       container.add( new JScrollPane( contentsArea ),
57          BorderLayout.CENTER );
58       setSize( 400, 300 );
59       setVisible( true );
61    } // fim do construtor ReadServerFile
63    // carrega documento
64    private void getThePage( String location )
65    {
66       // carrega documento e mostra a localização
67       try {
68          contentsArea.setPage( location ); //configura a página para JEditorPane
69          enterField.setText( location ); //configura o texto
70       }
71       catch ( IOException ioException ) {// se não for possível carregar a página ...
72          JOptionPane.showMessageDialog( this,
73             "Error retrieving specified URL", "Bad URL",
74             JOptionPane.ERROR_MESSAGE );
75       }
77    } // fim do metodo getThePage
```

O método `setPage` baixa o documento e mostra ele no `JEditorPane`

ReadServerFile.java

```
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
```

ReadServerFile.java



Estabelecendo um Servidor usando Sockets

Para criar um servidor simples são necessários 5 passos:

1. Criar um objeto ServerSocket

•Registra uma porta disponível e um número máximo de clientes;

```
ServerSocket server = new ServerSocket(porta, tamanhoDaFila);
```

2. Cada conexão cliente é controlada por um objeto Socket;

•O servidor espera indefinidamente (ou bloqueia) uma tentativa de conexão por parte do cliente;

•Para esperar o cliente, o programa chama o método **accept();**

```
Socket connection = server.accept();
```

3. Enviando e recebendo dados

•Objetos OutputStream (write - para enviar dados) e InputStream (read - para receber dados);

•Esses objetos enviam e recebem bytes isolados;

•Métodos getInputStream e getOutputstream

- •Usados no objeto Socket;

Estabelecendo um Servidor usando Sockets

3. Enviando e recebendo dados (cont.)

• É conveniente enviar dados primitivos (int e double) ou dados da classe Serializable (como String) ao invés de bytes;

```
ObjectInputStream input = new ObjectInputStream ( connection.getInputStream() );
```

```
ObjectOutputStream output = new ObjectOutputStream ( connection.getOutputStream() );
```

4. Processamento

•Servidor e Cliente se comunicam via os objetos OutputStream e InputStream;

5. Encerramento da transmissão

•Fecha a conexão (connection.close()) e os fluxos correspondentes.

Estabelecendo um Cliente usando Sockets

- São necessários 4 passos para criar um cliente em Java:

1. Criar um objeto Socket para conectar ao servidor;

```
Socket connection = new Socket(endereçoServidor, porta);
```

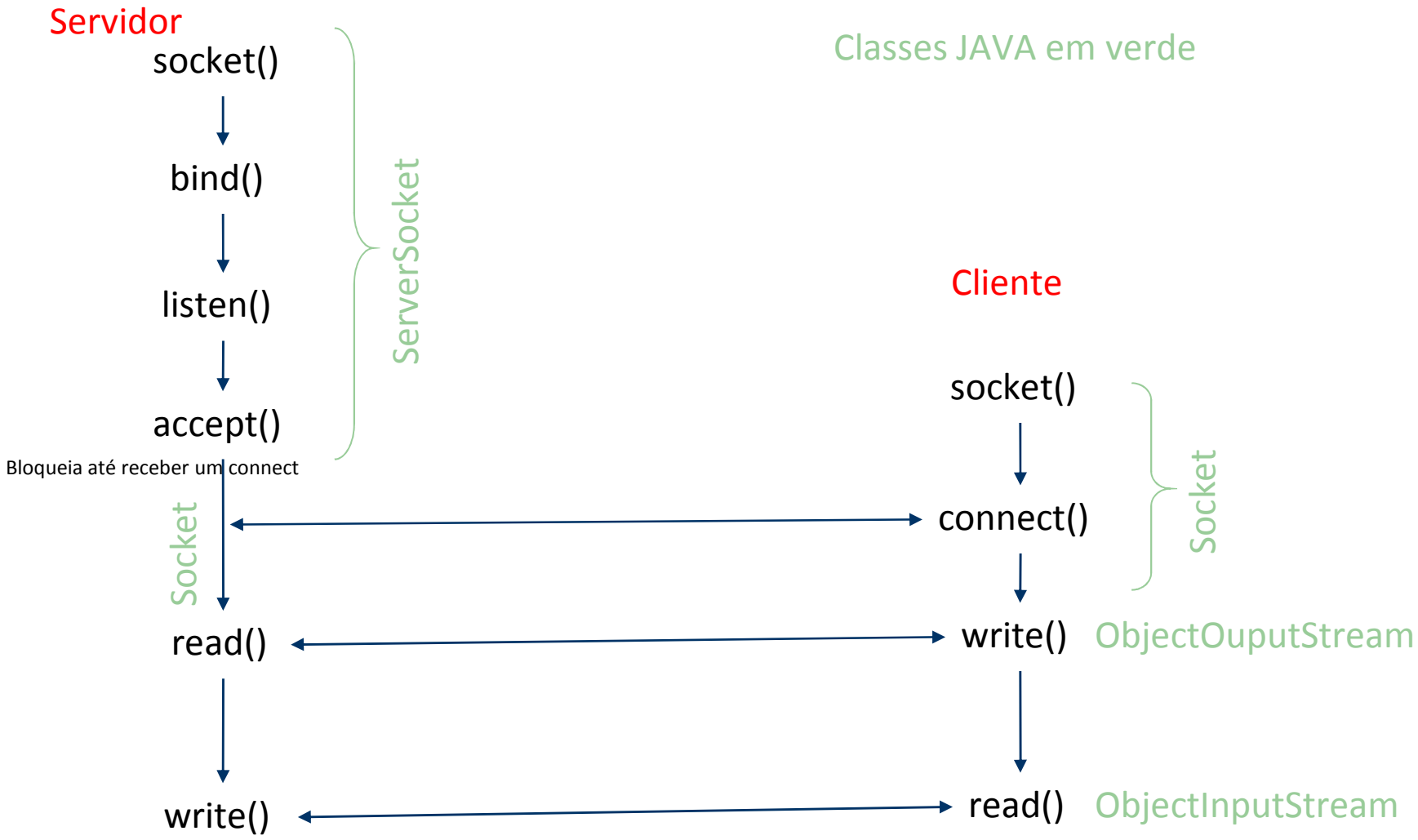
2. Obter os objetos InputStream e Outputstream do Socket para fazer referência aos objetos InputStream e Outputstream do servidor

3. Processar a informação comunicada;

4. Fechar a conexão e os fluxos correspondentes.

- O método read() do InputStream devolve –1 quando detecta o fim do fluxo (Também chamado de EOF);
- Se um objeto ObjectInputStream for usado para ler as informações do servidor, ocorre uma EOFException quando o cliente tenta ler um valor de um fluxo que foi finalizado.

Socket: Esquema (protocolo orientado a conexão)

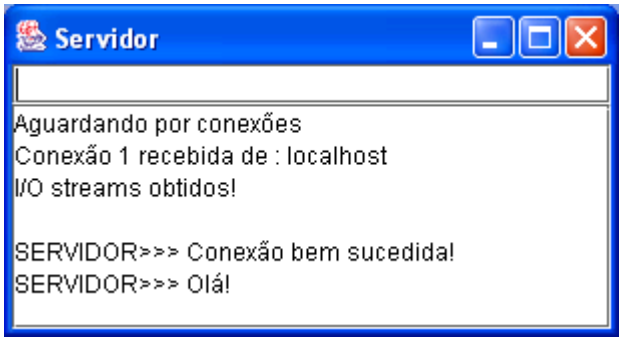


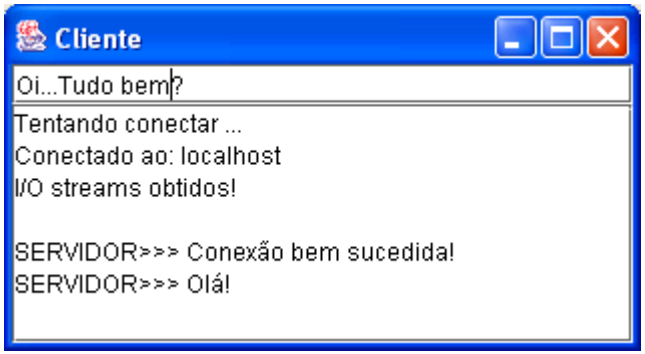
Interação Cliente/Servidor usando Conexões por meio de Sockets

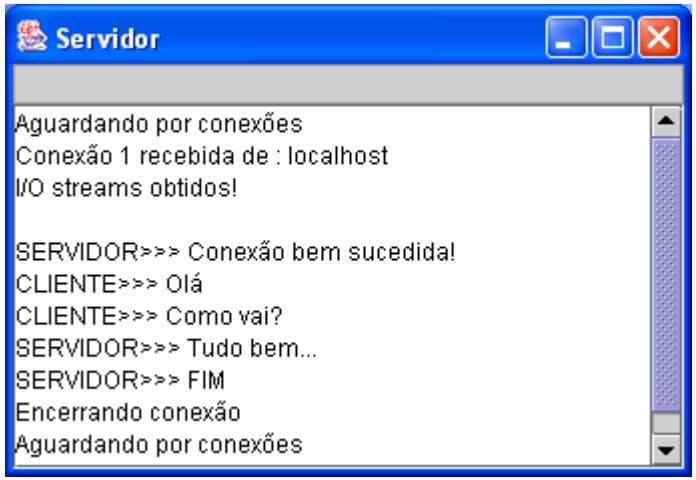
Aplicação: Chat entre Cliente/Servidor

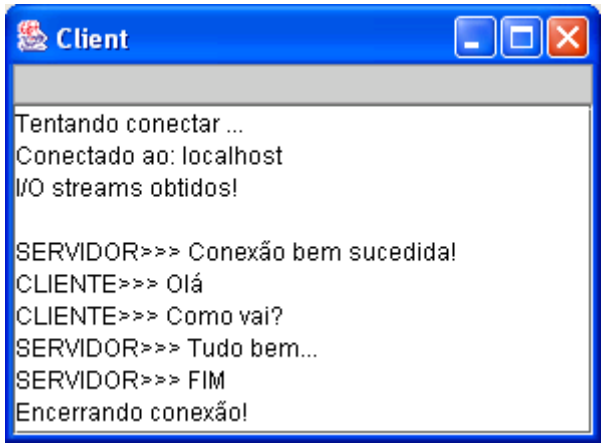
- Usa stream sockets como descrito nas seções anteriores.
- Detalhes do funcionamento:
  - Quando o cliente se conecta ao servidor, o servidor envia um objeto String indicando que a conexão foi bem-sucedida;
  - Ambos terminais possuem um JTextField que permite que mensagens possam ser trocadas entre eles, as quais são exibidas em um JTextArea;
  - Quando o cliente ou o servidor enviar a String "FIM", a conexão entre os dois é encerrada!

Aplicação: Chat entre Cliente/Servidor





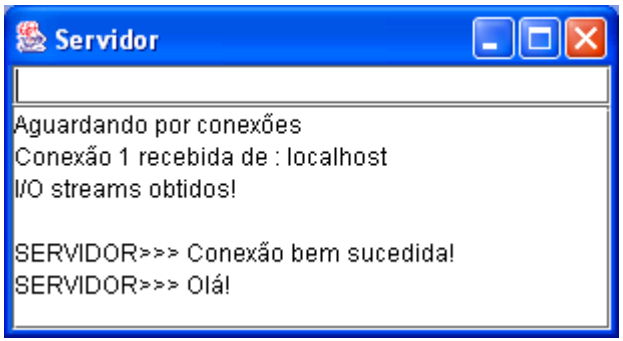




Server.java

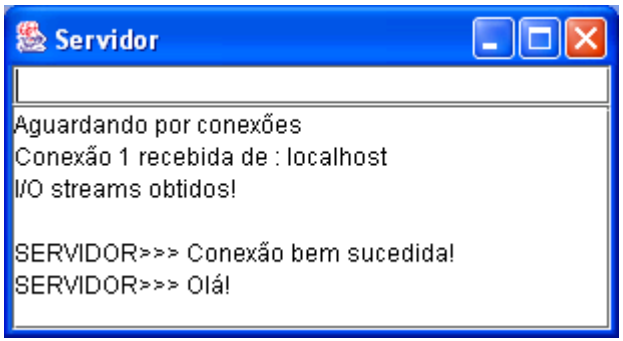
```
1   // Fig. 18.4: Server.java
2   // Configura um servidor que irá receber uma conexão de um cliente, enviar
3   // uma string e fechar a conexão.
4   import java.io.*;
5   import java.net.*;
6   import java.awt.*;
7   import java.awt.event.*;
8   import javax.swing.*;
10  public class Server extends 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;
19     // configura GUI
20     public Server()
21     {
22        super( "Servidor" );
24        Container container = getContentPane();
25
```

O *listen* está sobre o `ServerSocket`; a conexão é um `Socket`



Server.java

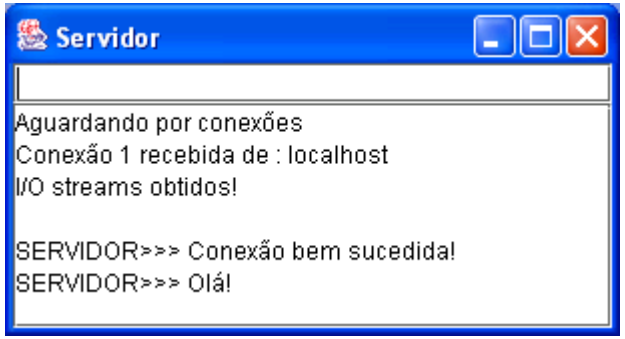
```
26      // cria campo de entrada e configura o listener (evento)
27      enterField = new JTextField();
28      enterField.setEditable( false );
29      enterField.addActionListener(
30        new ActionListener() {
32          // envia mensagem ao cliente
33          public void actionPerformed( ActionEvent event )
34          {
35            sendData( event.getActionCommand() );
36            enterField.setText( "" );
37          }
38        }
39      );
41      container.add( enterField, BorderLayout.NORTH );
43      // cria área de display
44      displayArea = new JTextArea();
45      container.add( new JScrollPane( displayArea ),
46        BorderLayout.CENTER );
48      setSize( 300, 150 );
49      setVisible( true );
51    } // fim Construtor
```



Server.java

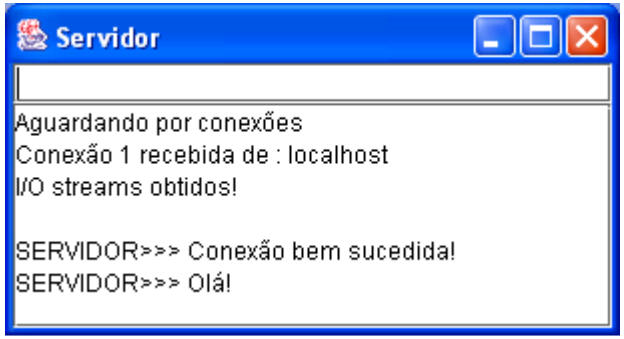
```
53  // configura e executa o servidor
54  public void runServer()
55  {
56    // configura o servidor para receber conexões; processa as conexões
57    try {
58
59      // Passo 1: Cria o ServerSocket.
60      server = new ServerSocket( 12345, 100 );
62      while ( true ) {
64        try {
65          waitForConnection(); // Passo 2: Aguarda conexão.
66          getStreams();       // Passo 3: Obtém input & output streams.
67          processConnection(); // Passo 4: Processa conexão.
68        }
70        // processa EOFException quando o cliente encerra a conexão
71        catch ( EOFException eofException ) {
72          System.err.println( "O Cliente encerrou a conexão!" );
73        }
75        finally {
76          closeConnection();   // Passo 5: Fecha a conexão.
77          ++counter;
78        }
```

Cria `ServerSocket` para a porta 12345 com fila de tamanho 100



Server.java

```
79
80          } // end while
81
82       } // end try
84       // processa problemas com I/O
85       catch ( IOException ioException ) {
86          ioException.printStackTrace();
87       }
89    } // end Método runServer
91    // PASSO 2: Aguarda os pedidos de conexão, e depois mostra a informação de conexão
92    private void waitForConnection() throws IOException
93    {
94       displayArea.append( "Aguardando por conexões\n" );
95       connection = server.accept(); //permite o servidor aceitar conexões
96       displayArea.append( "Conexão " + counter + " recebida de : " +
97          connection.getInetAddress().getHostName() );
98    }
100   // PASSO 3: Obtém streams para enviar e receber dados
101   private void getStreams() throws IOException
102   {
```



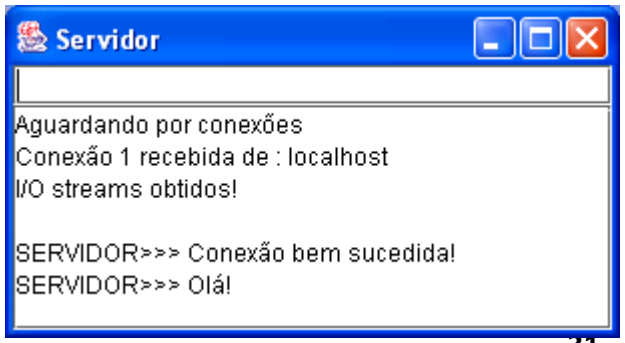
Método `accept` aguarda por conexão

Exibe o nome do computador que conectou

Server.java

```
103     // configura output stream
104     output = new ObjectOutputStream( connection.getOutputStream() );
105     output.flush(); // esvazia buffer de saída
106
107     // configura input stream
108     input = new ObjectInputStream( connection.getInputStream() );
109
110     displayArea.append( "\nI/O streams obtidos!\n" );
111  }
112
113  // PASSO 4: processa conexão com o cliente
114  private void processConnection() throws IOException
115  {
116     // envia mensagem de conexão bem-sucedida para o cliente
117     String message = "Conexão bem sucedida!";
118     sendData( message );
119
120     // habilita campo de entrada para o usuário do servidor
121     enterField.setEnabled( true );
122
123     do { // processa mensagens enviadas pelo cliente
124
```

Método `flush` descarrega buffer de saída para enviar cabeçalho de informação

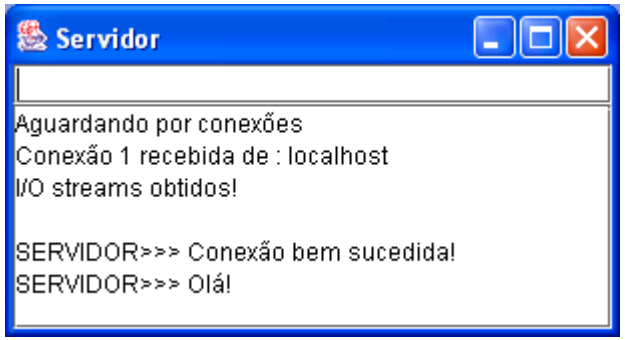


Server.java

```
125       // lê as mensagens a partir do objeto ObjectInputStream e as mostra
126       try {
127          message = ( String ) input.readObject();
128          displayArea.append( "\n" + message );
129       }
131       // captura problemas com as mensagens enviadas pelo cliente
132       catch ( ClassNotFoundException classNotFoundException ) {
133          displayArea.append( "\nO tipo de objeto é desconhecido!" );
134       }
136    } while ( !message.equals( "CLIENTE>>> FIM" ) );
138 } // end processConnection
139
140 //PASSO 5: Fecha streams e socket
141 private void closeConnection()
142 {
143    displayArea.append( "\nEncerrando conexão\n" );
144    enterField.setEnabled( false ); // desabilita campo de entrada
146    try {
147       output.close();
148       input.close();
149       connection.close();
150    }
```

Lê `String` do cliente e a mostra

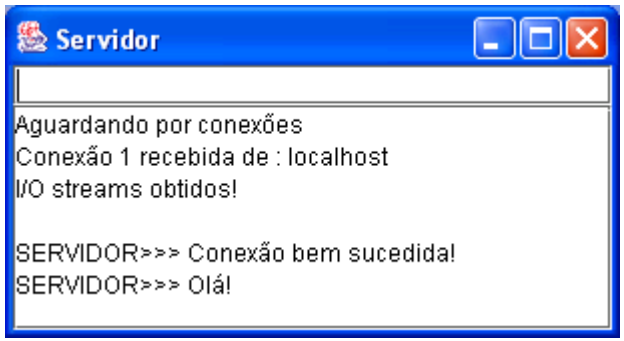
Método `closeConnection` encerra conexão



Server.java

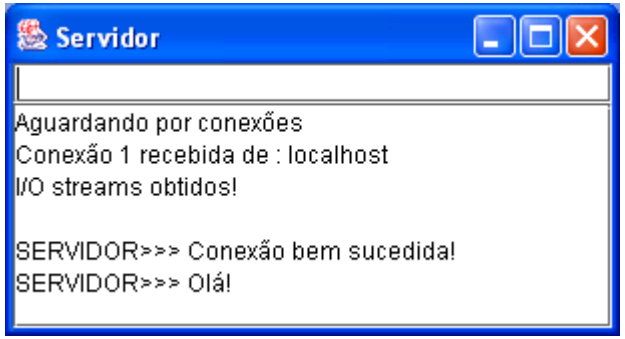
```
151     catch( IOException ioException ) {
152       ioException.printStackTrace();
153     }
154   }
155
156   // envia mensagem para o cliente
157   private void sendData( String message )
158   {
159
160     try {
161       output.writeObject( "SERVIDOR>>> " + message );
162       output.flush();
163       displayArea.append( "\nSERVIDOR>>> " + message );
164     }
165
166
167     catch ( IOException ioException ) {
168       displayArea.append( "\nErro ao enviar a mensagem!" );
169     }
170   }
171
172
```

Descarrega a saída



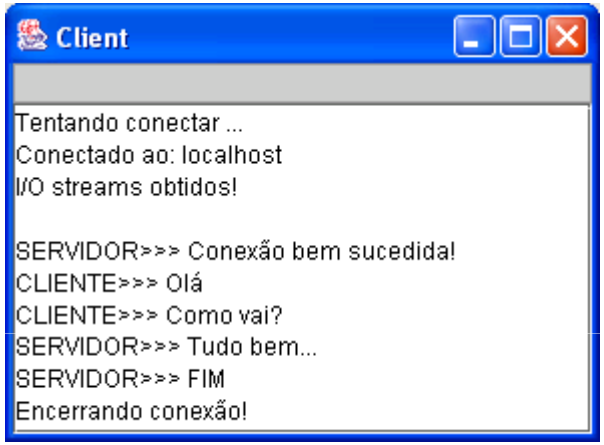
Server.java

```
173
174   public static void main( String args[] )
175   {
176      Server application = new Server();
177      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
178      application.runServer();
179   }
180
181 }  // end classe Server
```



Client.java

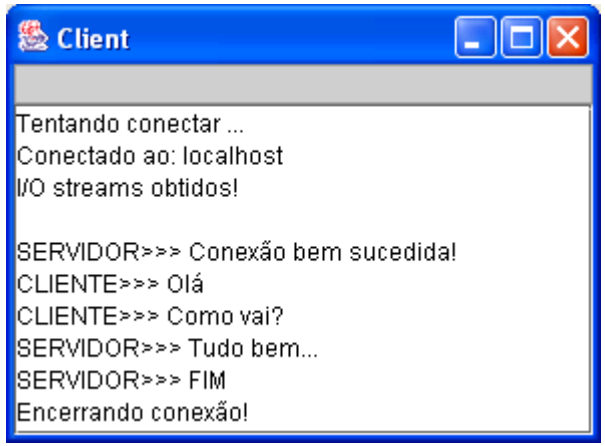
```
1   // Fig. 18.5: Client.java
2   // Cliente que lê e envia informação do/ao servidor.
3   import java.io.*;
4   import java.net.*;
5   import java.awt.*;
6   import java.awt.event.*;
7   import javax.swing.*;
8
9   public class Client extends JFrame {
10    private JTextField enterField;
11    private JTextArea displayArea;
12    private ObjectOutputStream output;
13    private ObjectInputStream input;
14    private String message = "";
15    private String chatServer;
16    private Socket client;
18    // inicializa chatServer e configura GUI
19    public Client( String host )
20    {
21       super( "Client" );
23       chatServer = host; // configura servidor a ser conectado
25       Container container = getContentPane();
```



O Cliente é um Socket

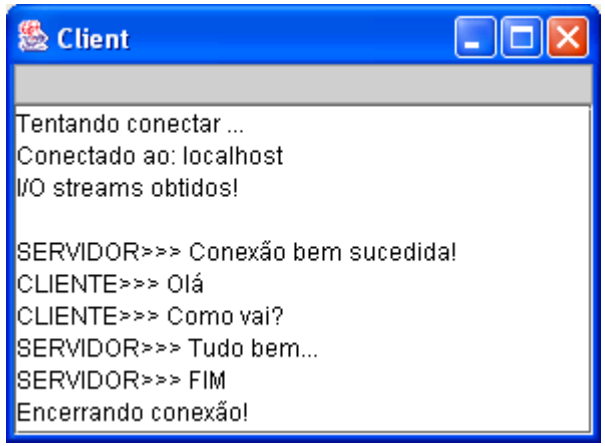
Client.java

```
27      // cria enterField e registra o listener (evento)
28      enterField = new JTextField();
29      enterField.setEditable( false );
30      enterField.addActionListener(
31        new ActionListener() {
33          // envia mensagem ao servidor
34          public void actionPerformed( ActionEvent event )
35          {
36             sendData( event.getActionCommand() );
37             enterField.setText( "" );
38          }
39        }
40      );
42      container.add( enterField, BorderLayout.NORTH );
43
44      // cria displayArea
45      displayArea = new JTextArea();
46      container.add( new JScrollPane( displayArea ),
47        BorderLayout.CENTER );
48
49      setSize( 300, 150 );
50      setVisible( true );
```

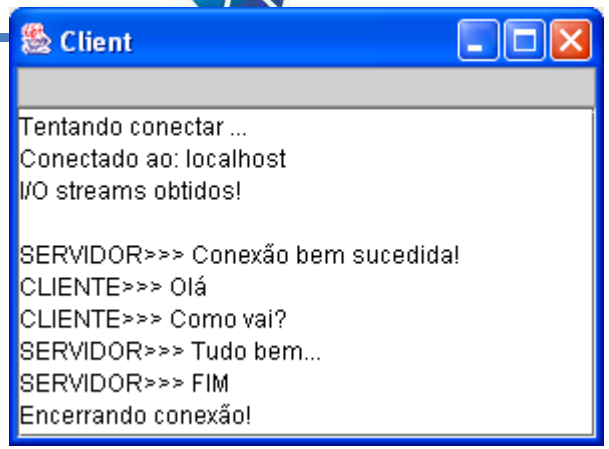


Client.java

```
52    } // end Client construtor
54    // conecta-se ao servidor e processa as mensagens
55    private void runClient()
56    {
57       // conecta ao servidor, obtém streams, processa conexão
58       try {
59          connectToServer(); // Passo 1: Cria o Socket para se conectar
60          getStreams();      // Passo 2: Obtêm input and output streams
61          processConnection(); // Passo 3: Processa conexão
62       }
64       // O Servidor encerra a conexão
65       catch ( EOFException eofException ) {
66          System.err.println( "O Servidor encerrou a conexão!" );
67       }
69       // processa problemas com a comunicação
70       catch ( IOException ioException ) {
71          ioException.printStackTrace();
72       }
74       finally {
75          closeConnection(); // Passo 4: Encerra a conexão
76       }
```



Client.java




```
78    } // end método runClient
80    // PASSO 1: Conecta ao servidor
81    private void connectToServer() throws IOException
82    {
83       displayArea.append( "Tentando conectar ... \n" );
85       // cria Socket para se conectar ao servidor – RETORNA O IP A PARTIR DO chatServer
86       client = new Socket( InetAddress.getByName( chatServer ), 12345 );
87
88       // mostra informações da conexão
89       displayArea.append( "Conectado ao: " +
90          client.getInetAddress().getHostName() );
91    }
92
93    // PASSO 2: Obtêm streams para enviar e receber dados
94    private void getStreams() throws IOException
95    {
96       // configura output stream
97       output = new ObjectOutputStream( client.getOutputStream() );
98       output.flush(); // esvazia output buffer
99
100      // configura input stream
101      input = new ObjectInputStream( client.getInputStream() );
```

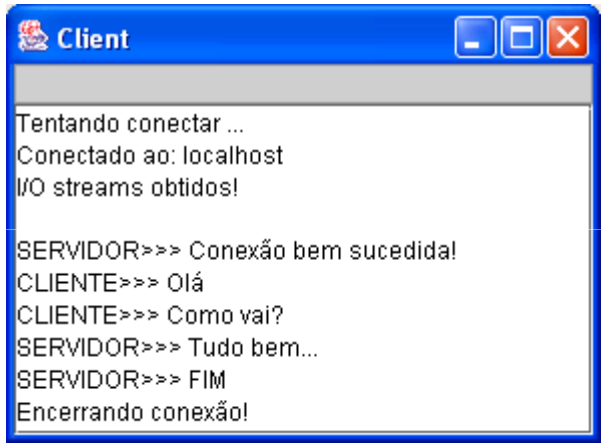
Cria um cliente que irá se conectar a porta 12345 do servidor

Notifica ao usuário que ele está conectado

Obtêm as streams para enviar e receber dados

Client.java

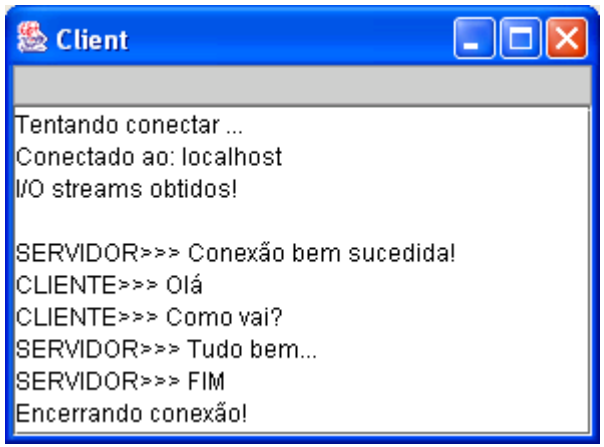
```
102
103     displayArea.append( "\nObtêm I/O streams\n" );
104   }
106   //  PASSO 3: processa conexão com o servidor
107   private void processConnection() throws IOException
108   {
109     // habilita enterField para o cliente enviar mensagens
110     enterField.setEnabled( true );
112     do { // processa mensagens enviadas pelo servidor
114       // lê a mensagem e a exibe
115       try {
116         message = ( String ) input.readObject();
117         displayArea.append( "\n" + message );
118       }
120       // captura problemas de leitura
121       catch ( ClassNotFoundException classNotFoundException ) {
122         displayArea.append( "\nObjeto recebido de tipo desconhecido!" );
123       }
125     } while ( !message.equals( "SERVIDOR>>> FIM" ) );
126
127   } // end método processConnection
```



Client.java

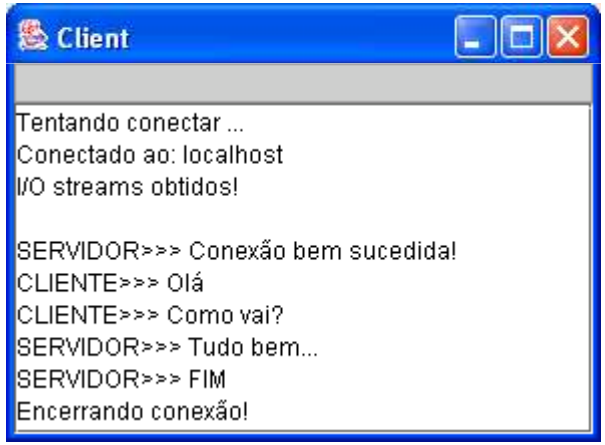
```
129  // PASSO 4: fecha streams e socket
130  private void closeConnection()
131  {
132     displayArea.append( "\nEncerrando conexão!" );
133     enterField.setEnabled( false ); // disable enterField
135     try {
136        output.close();
137        input.close();
138        client.close();
139     }
140     catch( IOException ioException ) {
141        ioException.printStackTrace();
142     }
143  }
145  // send message to server
146  private void sendData( String message )
147  {
148     // send object to server
149     try {
150        output.writeObject( "CLIENTE>>> " + message );
151        output.flush();
152        displayArea.append( "\nCLIENTE>>> " + message );
153     }
```

Método `closeConnection` encerra a conexão



Client.java

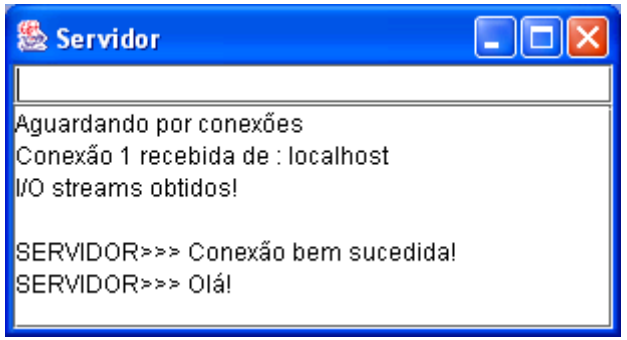
```
154
155     // process problems sending object
156     catch ( IOException ioException ) {
157        displayArea.append( "\nErro ao enviar a mensagem" );
158     }
159  }
160
199  public static void main( String args[] )
200  {
201     Client application;
202
203     if ( args.length == 0 )
204        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 classe Client
```

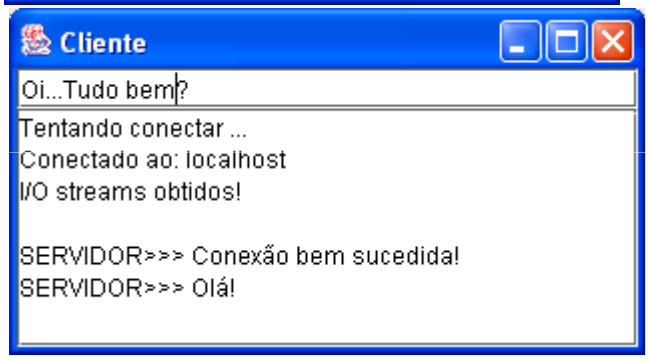


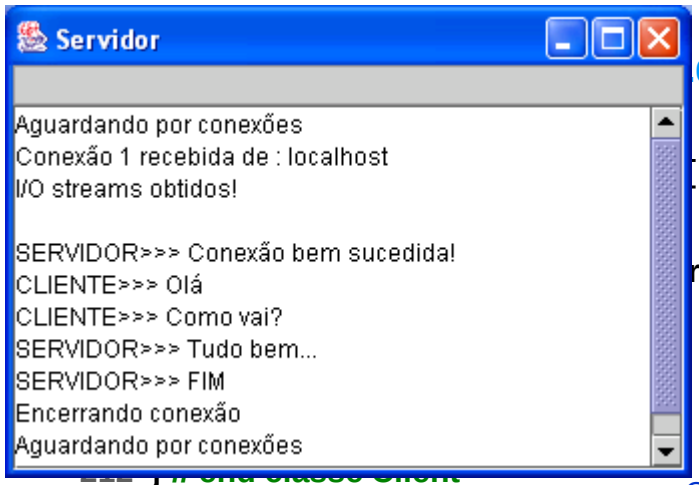
Client.java

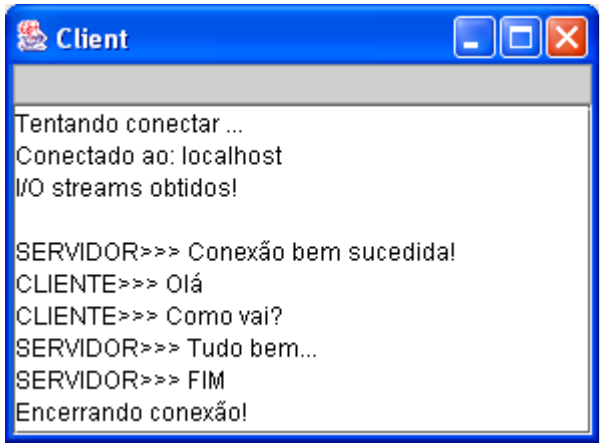
```
154
155     // process problems sending object
156     catch ( IOException ioException ) {
157        displayArea.append( "\nErro ao enviar a mensagem" );
158     }
159  }
160
199  public static void main( String args[] )
200  {
201     Client application;
202
```

```
0.0.1" );

 0 ] );

ration( JFrame.EXIT_ON_CLOSE );
```









Exercício 1: implemente os exemplos

Exercício 2: implemente o jogo-da-velha cliente/servidor que utiliza um servidor com multithread (exemplo do livro)