24

Redes

OBJETIVOS

- Neste capítulo, você aprenderá:
- Como entender a tecnologia de redes do Java com URLs, sockets e datagramas.
- Como implementar aplicativos de rede de Java utilizando sockets e datagramas.
- Como implementar clientes e servidores Java que se comunicam entre si.
- Como implementar aplicativos colaborativos baseados em rede.
- Como construir um servidor multiencadeado.

Sumário

24.1 Introdução

- **O pacote de rede é o `java.net`:**
  - **Comunicações baseadas em fluxo:**
    - **Os aplicativos visualizam a rede como fluxos de dados.**
    - **Protocolo baseado em conexão.**
    - **Utilizam o popular TCP (Transmission Control Protocol).**
  - **Comunicações baseadas em pacotes:**
    - **Transmitem pacotes individuais de informações.**
    - **Utilizam UDP (User Datagram Protocol) — que é um serviço sem conexão.**

24.1 Introdução (*Continuação*)

- **Relacionamento cliente-servidor:**
  - **O cliente solicita alguma ação a ser realizada.**
  - **O servidor realiza a ação e responde ao cliente.**
  - **Modelo de solicitação-resposta:**
    - **Implementação comum: Navegadores Web e servidores Web.**

Dica de desempenho 24.1

Serviços sem conexão geralmente oferecem desempenho maior, mas menos confiabilidade do que os serviços orientados a conexão.

Dica de portabilidade 24.1

O TCP, o UDP e os protocolos relacionados permitem que uma grande variedade de sistemas de computadores heterogêneos (isto é, sistemas de computadores com diferentes processadores e diferentes sistemas operacionais) se intercomuniquem.

24.2 Manipulando URLs

- **Hypertext Transfer Protocol (HTTP):**
  - **Utiliza URIs (Uniform Resource Identifier) para identificar os dados.**
    - **URLs (Uniform Resource Locators):**
      - **URIs que especificam as localizações dos documentos.**
      - **Referenciam arquivos, diretórios e objetos complexos.**
- **Documento HTML `SiteSelector.html` (Figura 24.1):**
  - **Elemento `applet`.**
  - **Tag `param`:**
    - **Atributo `name`.**
    - **Atributo `value`.**

Resumo

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; cada parâmetro tem um `name` e um `value`

Figura 24.1 | O documento HTML para carregar a applet `SiteSelector`.

Resumo

SiteSelector.java

Linhas 3-4

Linha 8

```
1  // Fig. 24.2: SiteSelector.java
2  // Esse programa carrega um documento de um URL.
3  import java.net.MalformedURLException;
4  import java.net.URL;
5  import java.util.HashMap;
6  import java.util.ArrayList;
7  import java.awt.BorderLayout;
8  import java.applet.AppletContext;
9  import javax.swing.JApplet;
10 import javax.swing.JLabel;
11 import javax.swing.JList;
12 import javax.swing.JScrollPane;
13 import javax.swing.event.ListSelectionEvent;
14 import javax.swing.event.ListSelectionListener;
15
16 public class SiteSelector extends JApplet
17 {
18    private HashMap< Object, URL > sites; // nomes e URLs de site
19    private ArrayList< String > siteNames; // nomes de site
20    private JList siteChooser; // lista de sites a escolher
21
22    // lê parâmetros de HTML e configura a GUI
23    public void init()
24    {
25       sites = new HashMap< Object, URL >(); // cria HashMap
26       siteNames = new ArrayList< String >(); // cria ArrayList
27
28       // obtém parâmetros do documento HTML
29       getSitesFromHTMLParameters();
30
```

Importa classes no pacote `java.net`

Importa a interface `AppletContext` do pacote `java.net`

Resumo

SiteSelector.java

(2 de 5)

Linhas 39-52

```
31       // cria componentes GUI e interface de layout
32       add( new JLabel( "Choose a site to browse" ), BorderLayout.NORTH );
33
34       siteChooser = new JList( siteNames.toArray() ); // preenche JList
35       siteChooser.addListSelectionListener(
36          new ListSelectionListener() // classe interna anônima
37          {
38             // vai ao site selecionado pelo usuário
39             public void valueChanged( ListSelectionEvent event )
40             {
41                // obtém o nome do site selecionado
42                Object object = siteChooser.getSelectedVal
43
44                // utiliza o nome do site para localizar c
45                URL newDocument = sites.get( object );
46
47                // obtém contêiner de applets
48                AppletContext browser = getAppletContext();
49
50                // instrui o contêiner de applets a mudar as págir
51                browser.showDocument( newDocument );
52             } // fim do método valueChanged
53          } // fim da classe interna anônima
54       ); // fim da chamada para addListSelectionListener
55
56       add( new JScrollPane( siteChooser ), BorderLayout.CENTER );
57    } // fim do método init
58
```

Quando o usuário seleciona um dos sites Web listados no `siteChooser`, o

Passa o nome do site selecionado (a chave) para o método `Hashtable`

Obtém uma referência a um objeto `AppletContext` que

Exibe na janela atual do navegador o recurso associado com o `URL` `newDocument`

Resumo

SiteSelector.java

(3 de 5)

Linha 67

Linha 73

```
59    // obtém parâmetros do documento HTML
60    private void getSitesFromHTMLParameters()
61    {
62       String title; // título do site
63       String location; // localização do site
64       URL url; // URL da localização
65       int counter = 0; // conta número de sites
66
67       title = getParameter( "title" + counter ); // obtém o primeiro título do site
68
69       // faz um loop até não haja mais parâmetros no documento HTML
70       while ( title != null )
71       {
72          // obtém a localização do site
73          location = getParameter( "locat
74
75          try // coloca título/URL no HashMap e título na ArrayList
76          {
77             url = new URL( location ); // converte a localização e
78             sites.put( title, url ); // coloca título/URL no HashMap
79             siteNames.add( title ); // coloca o título na ArrayList
80          } // fim do try
81          catch ( MalformedURLException urlException )
82          {
83             urlException.printStackTrace();
84          } // fim do catch
85
```

Utiliza o método `Applet` `getParameter` para obter o título do site Web

Utiliza o método `Applet` `getParameter` para obter a localização do site

Utiliza a `location` como o valor de um novo objeto `URL`

Se a `location` passada para o construtor de `IURL` for inválida, o construtor de `URL` lançará uma `MalformedURLException`

```
86          counter++;
87          title = getParameter( "title" + counter ); // obtém o próximo título do site
88       } // fim do while
89    } // fim do método getSitesFromHTMLParameters
90 } // fim da classe SiteSelector
```

Utiliza o método `Applet` `getParameter` para obter o título do próximo site

Resumo

`Selector.java`

5)

Linha 87

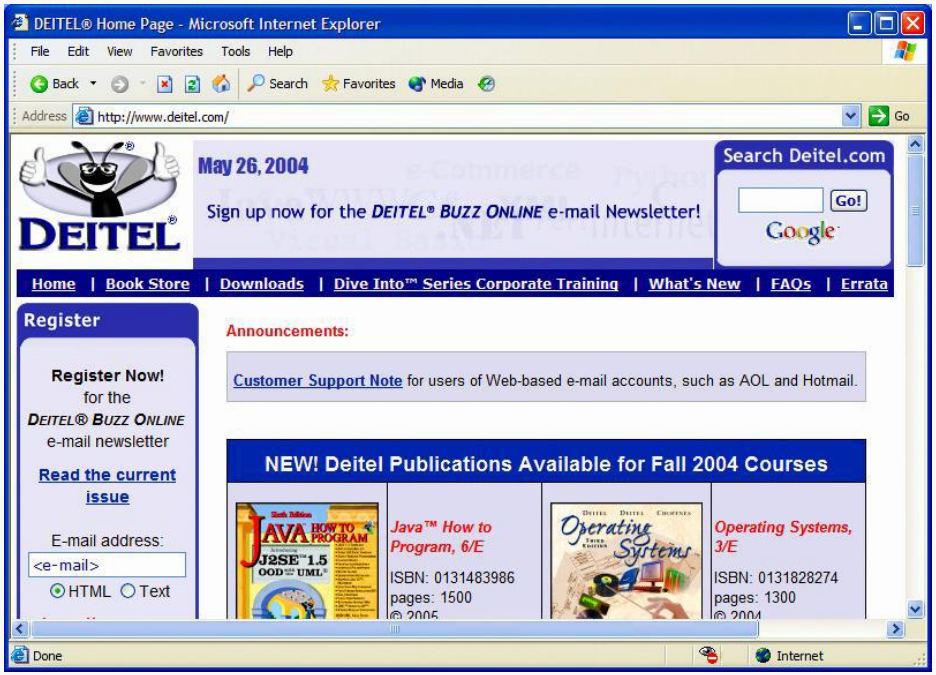
Saída do programa

Resumo

`SiteSelector.java`

(5 de 5)

Saída do programa



24.2 Manipulando URLs (*Continuação*)

- **Frames HTML:**
  - **Especifica o frame-alvo no método `showDocument`:**
    - `_blank`
    - `_self`
    - `_top`

Dica de prevenção de erro 24.1

A applet na Figura 24.2 deve ser executada a partir de um navegador Web, como o Mozilla ou o Microsoft Internet Explorer, para ver os resultados da exibição de uma outra página da Web. O `appletviewer` só é capaz de executar applets — ele ignora todas as outras tags HTML. Se os sites Web no programa contiverem applets Java, somente essas applets aparecerão no `appletviewer` quando o usuário selecionar um site Web. Cada applet executaria em uma janela `appletviewer` separada.

24.3 Lendo um arquivo em um servidor Web

- **Componente GUI Swing `JEditorPane`:**
  - **Renderiza tanto texto simples como texto formatado em HTML.**
  - **Atua como um navegador Web simples.**
    - **Recupera arquivos de um servidor Web em um dado URI.**
    - **`HyperlinkEvent`s:**
      - **Ocorrem quando o usuário clica em um hyperlink.**
      - **Três tipos de eventos:**
        - `HyperlinkEvent.EventType.ACTIVATED`
        - `HyperlinkEvent.EventType.ENTERED`
        - `HyperlinkEvent.EventType.EXITED`

Resumo

```
1  // Fig. 24.3: ReadServerFile.java
2  // Usa um JEditorPane para exibir o conteúdo de um arquivo em um servidor Web.
3  import java.awt.BorderLayout;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.io.IOException;
7  import javax.swing.JEditorPane;
8  import javax.swing.JFrame;
9  import javax.swing.JOptionPane;
10 import javax.swing.JScrollPane;
11 import javax.swing.JTextField;
12 import javax.swing.event.HyperlinkEvent;
13 import javax.swing.event.HyperlinkListener;
14
15 public class ReadServerFile extends JFrame
16 {
17    private JTextField enterField; // JTextField para inserir nome do site
18    private JEditorPane contentsArea; // para exibir site da Web
19
20    // configura a GUI
21    public ReadServerFile()
22    {
23       super( "Simple Web Browser" );
24
```

ReadServerFile

Importa `JEditPane` do pacote `javax.swing`, importa `HyperlinkEvent` e `HyperlinkListener` do pacote `javax.swing.event`

Linha 18

Declara o `JEditorPane` `contentsArea`, que será utilizado para exibir o conteúdo do arquivo

Resumo

ReadServerFile.java

(2 de 3)

Linha 40

Linha 41

```
25    // cria o enterField e registra seu ouvinte
26    enterField = new JTextField( "Enter file URL here" );
27    enterField.addActionListener(
28       new ActionListener()
29       {
30          // obtém o documento especificado pelo usuário
31          public void actionPerformed( ActionEvent event )
32          {
33             getThePage( event.getActionCommand() );
34          } // fim do método actionPerformed
35       } // fim da classe interna
36    ); // fim da chamada para addActionListener
37
38    add( enterField, BorderLayout.NORTH );
39
40    contentsArea = new JEditorPane(); // cria contentsAre
41    contentsArea.setEditable( false );
42    contentsArea.addHyperlinkListener(
43       new HyperlinkListener()
44       {
45          // se usuário clicou no hyperlink, vai para a página esp
46          public void hyperlinkUpdate( HyperlinkEvent event )
47          {
48             if ( event.getEventType() ==
49                HyperlinkEvent.EventType.ACTIVATED )
50                getThePage( event.getURL().toString() );
51          } // fim do método hyperlinkUpdate
52       } // fim da classe interna
53    ); // fim da chamada para addHyperlinkListener
54
```

Cria um

Configura a propriedade

Registra um `HyperlinkListener` para tratar `HyperlinkEvents` que ocorrem

O método `hyperlinkUpdate` é

Utiliza o método `HyperlinkEvent getEventType` para determinar o tipo do

Utiliza o método `HyperlinkEvent getURL` para obter o URL representado pelo hyperlink.

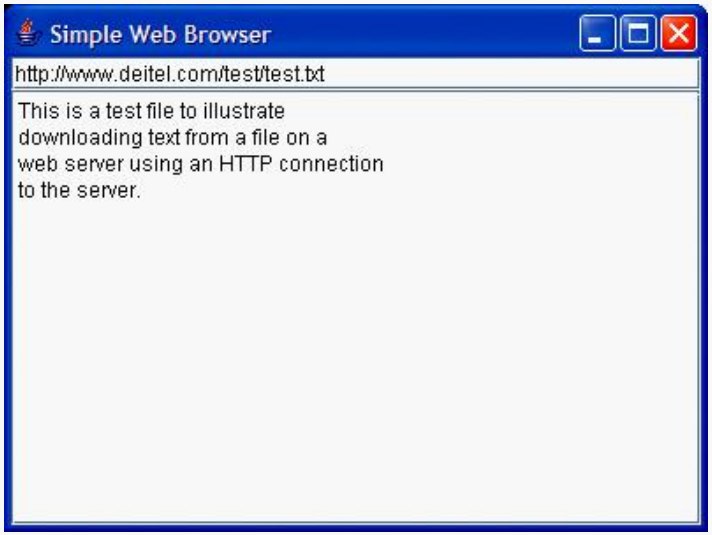
Resumo

ReadServerFile.java

```
55       add( new JScrollPane( contentsArea ), BorderLayout.CENTER );
56       setSize( 400, 300 ); // configura o tamanho da janela
57       setVisible( true ); // mostra a janela
58    } // fim do construtor ReadServerFile
59
60    // carrega o documento
61    private void getThePage( String location )
62    {
63       try // carrega o documento e exibe a localização
64       {
65          contentsArea.setPage( location ); // configura a página
66          enterField.setText( location ); // configura o texto
67       } // fim do try
68       catch ( IOException ioException )
69       {
70          JOptionPane.showMessageDialog( this,
71             "Error retrieving specified URL", "Bad URL",
72             JOptionPane.ERROR_MESSAGE );
73       } // fim do catch
74    } // fim do método getThePage
75 } // fim da classe ReadServerFile
```

Invoca o método `JEditorPane` `setPage` para fazer o download do documento especificado por `location` e exibi-lo no `JEditPane`

```
1  // Fig. 24.4: ReadServerFileTest.java
2  // Cria e inicia um ReadServerFile.
3  import javax.swing.JFrame;
4
5  public class ReadServerFileTest
6  {
7     public static void main( String args[] )
8     {
9        ReadServerFile application = new ReadServerFile();
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11    } // fim do main
12 } // fim da classe ReadServerFileTest
```



Resumo

ReadServerFileTest.java

(1 de 2)

Saída do programa

Resumo

`ReadServerFileTest.java`

(2 de 2)

Saída do programa

Observação sobre aparência e comportamento 24.1

Um `JEditorPane` gera `HyperlinkEvents` somente se eles forem não-editáveis.

24.4 Estabelecendo um servidor simples utilizando sockets de fluxo

- **Cinco passos para criar um servidor simples em Java:**
  - ***Passo 1*: Criar o objeto `ServerSocket`:**
    - `ServerSocket server = new ServerSocket( `*`númeroDaPorta, comprimentoDaFila`*` );`
    - **Registra uma porta disponível.**
    - **Especifica um número máximo de clientes.**
    - **Ponto de *handshake*.**
    - **Vincula o servidor a uma porta.**
    - **Somente um cliente pode ser vinculado a uma porta específica.**

Observação de engenharia de software 24.1

Os números de porta podem estar entre 0 e 65.535. A maioria dos sistemas operacionais reserva números de porta abaixo de 1.024 para serviços de sistema (por exemplo, correio eletrônico e servidores da World Wide Web). Geralmente, essas portas não devem ser especificadas como portas de conexão nos programas de usuário. De fato, alguns sistemas operacionais requerem privilégios especiais de acesso para que possam ser vinculados a números de porta abaixo de 1.024.

24.4 Estabelecendo um servidor simples utilizando sockets de fluxo (*Continuação*)

- **Cinco passos para criar um servidor simples em Java:**
  - ***Passo 2*: O servidor ouve a conexão cliente:**
    - **O servidor bloqueia até o cliente se conectar.**
    - `Socket connection = server.accept();`
  - ***Passo 3*: Enviar e receber dados:**
    - **`OutputStream` para enviar, e `InputStream` para receber dados:**
      - **O método `getOutputStream` retorna `OutputStream` do `Socket`.**
      - **O método `getInputstreams` retorna `InputStream` de `Socket`.**

24.4 Estabelecendo um servidor simples utilizando sockets de fluxo (*Continuação*)

- **Cinco passos para criar um servidor simples em Java:**
  - ***Passo 4*: Fase de processamento:**
    - **O servidor e o cliente se comunicam via fluxos.**
  - ***Passo 5*: Fechar fluxos e conexões.**
    - **Método `close`.**

Observação de engenharia de software 24.2

Com sockets, a E/S de rede aparece para programas Java como sendo similar à E/S de arquivo seqüencial. Os sockets ocultam muito da complexidade de programação de rede do programador.

Observação de engenharia de software 24.3

Com o multithreading do Java, podemos criar servidores com múltiplas threads que podem gerenciar várias conexões simultâneas com vários clientes. Essa arquitetura de servidor com múltiplos threads é precisamente a que é utilizada nos servidores de rede populares.

Observação de engenharia de software 24.4

Um servidor com múltiplas threads pode receber o `Socket` retornado por uma a uma das chamadas a `accept` e criar uma nova thread que gerencia a E/S de rede por meio desse `Socket`. Alternativamente, um servidor com múltiplas threads pode manter um pool de threads (um conjunto de threads existentes) pronto para gerenciar a E/S de rede por meio dos novos `Sockets` à medida que são criados. Consulte o Capítulo 23 para informações adicionais sobre multithreading.

Dica de desempenho 24.2

Em sistemas de alto desempenho em que a memória é abundante, um servidor multiencadeado pode ser implementado para criar um pool de threads que pode ser atribuído rapidamente para tratar E/S de rede através de cada novo `Socket` quando ele é criado. Portanto, quando o servidor recebe uma conexão, ele não precisa incorrer no overhead da criação de thread. Quando a conexão é fechada, a thread é retornada ao pool para reutilização.

24.5 Estabelecendo um cliente simples utilizando sockets de fluxo

- **Quatro passos para criar um cliente simples em Java:**
  - ***Passo 1*: Criar um `Socket` para conexão ao servidor:**
    - `Socket connection = new Socket ( endereçoDoServidor, porta );`
  - ***Passo 2*: Obter `InputStream` e `Outputstream` de `Socket`.**
  - ***Passo 3*: Processar informações comunicadas.**
  - ***Passo 4*: Fechar fluxos e conexão.**

24.6 Interação cliente/servidor com conexões de socket de fluxo

- **Aplicativo de bate-papo cliente/servidor simples:**
  - **Utiliza soquetes de fluxo.**
  - **O servidor espera uma tentativa de conexão do cliente.**
  - **O cliente conecta-se ao servidor:**
    - **Envia e receber mensagens.**
  - **O cliente ou o servidor fecha a conexão.**
  - **O servidor espera que o próximo cliente se conecte.**

Resumo

Server.java

Linha 8-9

Linha 25

Linha 26

```
1  // Fig. 24.5: Server.java
2  // Configura uma classe Server que receberá uma conexão de um cliente, envia
3  // uma string ao cliente e fecha a conexão.
4  import java.io.EOFException;
5  import java.io.IOException;
6  import java.io.ObjectInputStream;
7  import java.io.ObjectOutputStream;
8  import java.net.ServerSocket;
9  import java.net.Socket;
10 import java.awt.BorderLayout;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13 import javax.swing.JFrame;
14 import javax.swing.JScrollPane;
15 import javax.swing.JTextArea;
16 import javax.swing.JTextField;
17 import javax.swing.SwingUtilities;
18
19 public class Server extends JFrame
20 {
21    private JTextField enterField; // insere a mensagem do usuário
22    private JTextArea displayArea; // exibe informações para o usuário
23    private ObjectOutputStream output; //  gera fluxo de saída para o cliente
24    private ObjectInputStream input; // gera fluxo de entra
25    private ServerSocket server; // socket de servidor
26    private Socket connection; // conexão com o cliente
27    private int counter = 1; // contador do número de conexões
28
```

Importa `ServerSocket` e `Socket` do pacote `java.net`

Declara `ServerSocket server`

Declara `Socket connection` que se conecta ao cliente

Resumo

Server.java

(2 de 8)

```
29    // configura a GUI
30    public Server()
31    {
32       super( "Server" );
33
34       enterField = new JTextField(); // cria enterField
35       enterField.setEditable( false );
36       enterField.addActionListener(
37          new ActionListener()
38          {
39             // envia messagem para o cliente
40             public void actionPerformed( ActionEvent event )
41             {
42                sendData( event.getActionCommand() );
43                enterField.setText( "" );
44             } // fim do método actionPerformed
45          } // fim da classe interna anônima
46       ); // fim da chamada para addActionListener
47
48       add( enterField, BorderLayout.NORTH );
49
50       displayArea = new JTextArea(); // cria displayArea
51       add( new JScrollPane( displayArea ), BorderLayout.CENTER );
52
53       setSize( 300, 150 ); // configura o tamanho da janela
54       setVisible( true ); // mostra a janela
55    } // fim do construtor Server
56
```

Resumo

`Server.java`

```
57    // configura e executa o servidor
58    public void runServer()
59    {
60       try // configura o servidor para receber conexões; processa as conexões
61       {
62          server = new ServerSocket( 12345, 100 ); // cria ServerSocket
63
64          while ( true )
65          {
66             try
67             {
68                waitForConnection(); // espera uma conexão
69                getStreams(); // obtém fluxos de entrada e saída
70                processConnection(); // processa conexão
71             } // fim do try
72             catch ( EOFException eofException )
73             {
74                displayMessage( "\nServer terminated connection" );
75             } // fim do catch
```

Cria o `ServerSocket` na porta `12345` com uma fila de comprimento 100

Espera uma conexão

Depois de a conexão ser

Envia a mensagem de conexão inicial ao cliente e processa todas as mensagens recebidas do cliente

Resumo

Server.java

(4 de 8)

Linha 93

Linha 95

```
76            finally
77            {
78               closeConnection(); // fecha conexão
79               counter++;
80            } // fim do finally
81         } // fim do while
82      } // fim do try
83      catch ( IOException ioException )
84      {
85         ioException.printStackTrace();
86      } // fim do catch
87   } // fim do método runServer
88
89   // espera que a conexão chegue e, então,
90   private void waitForConnection() throws I
91   {
92      displayMessage( "Waiting for connection\n" );
93      connection = server.accept(); // permite que o servido aceite conexão
94      displayMessage( "Connection " + counter + " received from: " +
95         connection.getInetAddress().getHostName() );
96   } // fim do método waitForConnection
97
98   // obtém fluxos para enviar e r
99   private void getStreams() throw
100  {
101     // configura o fluxo de saíd
102     output = new ObjectOutputStream( connection.getOutputStream() );
103     output.flush(); // esvazia buffer de saída p/ enviar as informações de cabeçalho
104
```

Gera o nome de host do computador que cr
Soc

Utiliza o método `ServerSocket` `accept` para esperar uma conexão de um cliente

Obtém `OutputStream` de `Socket` e
ra inicializar
tputStream

O método `flush` esvazia o buffer de saída e envia as informações de cabeçalho

Resumo

```
105      // configura o fluxo de entrada para objetos
106      input = new ObjectInputStream( connection.getInputStream() );
107
108      displayMessage( "\nGot I/O streams\n" );
109   } // fim do método getStreams
110
111   // processa conexão com cliente
112   private void processConnection() throws IOException
113   {
114      String message = "Connection successful";
115      sendData( message ); // envia uma mensagem de conexão bem-sucedida
116
117      // ativa enterField de modo que usuário do servidor possa enviar mensagens
118      setTextFieldEditable( true );
119
120      do // processa mensagens enviadas do cliente
121      {
122         try // lê messagem e a exibe
123         {
124            message = ( String ) input.readObject(); // lê nova messagem
125            displayMessage( "\n" + message ); // exibe messagem
126         } // fim do try
127         catch ( ClassNotFoundException classNotFoundException )
128         {
129            displayMessage( "\nUnknown object type received" );
130         } // fim do catch
131
```

Obtém `InputStream` de `Socket` e utiliza-o para inicializar `ObjectInputStream`

(5 de 8)

Linha 106

Linha 124

Utiliza o método `ObjectInputStream` `readObject` para ler uma `String` do cliente

Resumo

Server.java

(6 de 8)

Linhas 136-151

Linha 145

```
132       } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
133    } // fim do método processConnection
134
135    // fecha os fluxos e o socket
136    private void closeConnection()
137    {
138       displayMessage( "\nTerminating connection\n" );
139       setTextFieldEditable( false ); // desativa enterField
140
141       try
142       {
143          output.close(); // fecha o fluxo de saída
144          input.close(); // fecha o fluxo de entrada
145          connection.close(); // fecha o socket
146       } // fim do try
147       catch ( IOException ioException )
148       {
149          ioException.printStackTrace();
150       } // fim do catch
151    } // fim do método closeConnection
153    // envia messagem para o cliente
154    private void sendData( String message )
155    {
156       try // envia objeto para o cliente
157       {
158          output.writeObject( "SERVER>>> " + message );
159          output.flush(); // esvazia saída para o cliente
160          displayMessage( "\nSERVER>>> " + message );
161       } // fim do try
```

O método `closeConnection` fecha fluxos e sockets

Invoca o método `Socket close` para fechar o socket

Utiliza o método `ObjectOutputStream writeObject` para enviar uma `String` ao cliente

Resumo

Server.java

(7 de 8)

```
162      catch ( IOException ioException )
163      {
164         displayArea.append( "\nError writing object" );
165      } // fim do catch
166   } // fim do método sendData
167
168   // manipula displayArea na thread de despacho de eventos
169   private void displayMessage( final String messageToDisplay )
170   {
171      SwingUtilities.invokeLater(
172         new Runnable()
173         {
174            public void run() // atualiza displayArea
175            {
176               displayArea.append( messageToDisplay ); // anexa messagem
177            } // fim do método run
178         } // fim da classe interna anônima
179      ); // fim da chamada para SwingUtilities.invokeLater
180   } // fim do método displayMessage
181
```

```
182   // manipula o enterField na thread de despacho de eventos
183   private void setTextFieldEditable( final boolean editable )
184   {
185      SwingUtilities.invokeLater(
186         new Runnable()
187         {
188            public void run() // configura a editabilidade do enterField
189            {
190               enterField.setEditable( editable );
191            } // fim do método run
192         }  // fim da classe inner
193      ); // fim da chamada para SwingUtilities.invokeLater
194   } // fim do método setTextFieldEditable
195} // fim da classe Server
```

Resumo

Server.java

(8 de 8)

Resumo

ServerTest.java

```
1  // Fig. 24.6: ServerTest.java
2  // Testa o aplicativo Server.
3  import javax.swing.JFrame;
4
5  public class ServerTest
6  {
7     public static void main( String args[] )
8     {
9        Server application = new Server(); // cria o servidor
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       application.runServer(); // executa o aplicativo do servidor
12    } // fim do main
13 } // fim da classe ServerTest
```

Erro comum de programação 24.1

Especificar uma porta que já está em utilização ou especificar um número de porta inválido ao criar um `ServerSocket` resulta em uma `BindException`.

Observação de engenharia de software 24.5

Ao utilizar um `ObjectOutputStream` e um `ObjectInputStream` para enviar e receber objetos em uma conexão de rede, sempre crie o `ObjectOutputStream` primeiro e esvazie (`flush`) o fluxo de modo que o `ObjectInputStream` do cliente possa se preparar para receber os dados. Isso só é necessário para aplicativos de rede que se comunicam utilizando `ObjectOutputStream` e `ObjectInputStream`.

Dica de desempenho 24.3

Os componentes de entrada e saída de um computador são, em geral, muito mais lentos que a memória do computador. Comumente, buffers de saída são utilizados para aumentar a eficiência de um aplicativo enviando volumes maiores de dados menos vezes, reduzindo, assim, o número de vezes que um aplicativo acessa os componentes de entrada e saída do computador.

Resumo

Client.java

(1 de 7)

```
1  // Fig. 24.7: Client.java
2  // Cliente que lê e exibe as informações enviadas a partir de um Servidor.
3  import java.io.EOFException;
4  import java.io.IOException;
5  import java.io.ObjectInputStream;
6  import java.io.ObjectOutputStream;
7  import java.net.InetAddress;
8  import java.net.Socket;
9  import java.awt.BorderLayout;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import javax.swing.JFrame;
13 import javax.swing.JScrollPane;
14 import javax.swing.JTextArea;
15 import javax.swing.JTextField;
16 import javax.swing.SwingUtilities;
17
18 public class Client extends JFrame
19 {
20    private JTextField enterField; // insere informações fornecidas pelo usuário
21    private JTextArea displayArea; // exibe informações para o usuário
22    private ObjectOutputStream output; // gera o fluxo de saída para o servidor
23    private ObjectInputStream input; // gera o fluxo de entrada do servidor
24    private String message = ""; // messagem do servidor
25    private String chatServer; // servidor de host para esse aplicativo
26    private Socket client; // socket para comunicação com o servidor
27
```

Resumo

Client.java

(2 de 7)

```
28    // inicializa chatServer e configura a GUI
29    public Client( String host )
30    {
31       super( "Client" );
32
33       chatServer = host; // configura o servidor ao qual esse cliente se conecta
34
35       enterField = new JTextField(); // cria enterField
36       enterField.setEditable( false );
37       enterField.addActionListener(
38          new ActionListener()
39          {
40             // envia mensagem para o servidor
41             public void actionPerformed( ActionEvent event )
42             {
43                sendData( event.getActionCommand() );
44                enterField.setText( "" );
45             } // fim do método actionPerformed
46          } // fim da classe interna anônima
47       ); // fim da chamada para addActionListener
48
49       add( enterField, BorderLayout.NORTH );
50
51       displayArea = new JTextArea(); // cria displayArea
52       add( new JScrollPane( displayArea ), BorderLayout.CENTER );
53
54       setSize( 300, 150 ); // configura o tamanho da janela
55       setVisible( true ); // mostra a janela
56    } // fim do construtor Client
57
```

Resumo

Client.java

(3 de 7)

```
58    // conecta-se ao servidor e processa as mensagens a partir do servidor
59    public void runClient()
60    {
61       try // conecta-se ao servidor, obtém fluxos, processa a conexão
62       {
63          connectToServer(); // cria um Socket para fazer a conexão
64          getStreams(); // obtém os fluxos de entrada e saída
65          processConnection(); // processa a conexão
66       } // fim do try
67       catch ( EOFException eofException )
68       {
69          displayMessage( "\nClient terminated connection" );
70       } // fim do catch
71       catch ( IOException ioException )
72       {
73          ioException.printStackTrace();
74       } // fim do catch
75       finally
76       {
77          closeConnection(); // fecha a conexão
78       } // fim do finally
79    } // fim do método runClient
80
81    // conectá-se ao servidor
82    private void connectToServer() throws IOException
83    {
84       displayMessage( "Attempting connection\n" );
85
```

```
86       // cria Socket fazer a conexão ao servidor
87       client = new Socket( InetAddress.getByName( chatServer ), 12345 );
88
89       // exibe informações sobre a conexão
90       displayMessage( "Connected to: " +
91          client.getInetAddress().getHostName() );
92    } // fim do método connectToServer
93
94    // obtém fluxos para enviar e re
95    private void getStreams() throws
96    {
97       // configura o fluxo de saída
98       output = new ObjectOutputStream( client.getOutputStream() );
99       output.flush(); // esvazia buffer de saída enviar as informações de cabeçalho
100
101      // configura o fluxo de entrada para objetos
102      input = new ObjectInputStream( client.getInputStream() );
103
104      displayMessage( "\nGot I/O streams\n" );
105   } // fim do método getStreams
106
107   // processa conexão com servidor
108   private void processConnection() throws IOException
109   {
110      // ativa enterField para que o usuário cliente possa enviar mensagens
111      setTextFieldEditable( true );
112
```

Resumo

Cria
será
12

Utiliza o método `static` `getByName` de para obter um objeto `InetAdress` que contém o endereço IP esp

Exibe uma mensagem que

dor

e

conectou

O método f
saída e en
cabeçalho

Obtém `OutputStream` de `Socket` e utiliza-o para inicializar `ObjectOutputStream`

Linha 98

Linha 99

Resumo

Client.java

Linha 117

Linha 138

```
113     do // processa mensagens enviadas do servidor
114     {
115        try // lê mensagem e a exibe
116        {
117           message = ( String ) input.readObject(); // lê nova mensagem
118           displayMessage( "\n" + message ); // exibe mensagem
119        } // fim do try
120        catch ( ClassNotFoundException classNotFoundException )
121        {
122           displayMessage( "\nUnknown object type received" );
123        } // fim do catch
124
125     } while ( !message.equals( "SERVER>>> TERMINATE" ) );
126  } // fim do método processConnection
127
128  // fecha fluxos e socket
129  private void closeConnection()
130  {
131     displayMessage( "\nClosing connection" );
132     setTextFieldEditable( false ); // disativa enterField
133
134     try
135     {
136        output.close(); // fecha fluxo de saída
137        input.close(); // fecha o fluxo de entrada  1
138        client.close(); // fecha socket
139     } // fim do try
```

Lê um objeto `String` no servidor

Invoca o método `Socket` `close` para fechar o socket

Resumo

Client.java

```
140      catch ( IOException ioException )
141      {
142         ioException.printStackTrace();
143      } // fim do catch
144   } // fim do método closeConnection
145
146   // envia mensagem para o servidor
147   private void sendData( String message )
148   {
149      try // envia objeto para o servidor
150      {
151         output.writeObject( "CLIENT>>> " + message );
152         output.flush(); // esvazia os dados para saída
153         displayMessage( "\nCLIENT>>> " + message );
154      } // fim do try
155      catch ( IOException ioException )
156      {
157         displayArea.append( "\nError writing object" );
158      } // fim do catch
159   } // fim do método sendData
160
```

Utiliza o método `ObjectOutputStream` `writeObject` para enviar uma `String` ao servidor

```
161   // manipula a displayArea na thread de despacho de eventos
162   private void displayMessage( final String messageToDisplay )
163   {
164      SwingUtilities.invokeLater(
165         new Runnable()
166         {
167            public void run() // atualiza displayArea
168            {
169               displayArea.append( messageToDisplay );
170            } // fim do método run
171         }  // fim da classe interna anônima
172      ); // fim da chamada para SwingUtilities.invokeLater
173   } // fim do método displayMessage
174
175   // manipula o enterField na thread de despacho de eventos
176   private void setTextFieldEditable( final boolean editable )
177   {
178      SwingUtilities.invokeLater(
179         new Runnable()
180         {
181            public void run() // configura a editabilidade do enterField
182            {
183               enterField.setEditable( editable );
184            } // fim do método run
185         } // fim da classe interna anônima
186      ); // fim da chamada para SwingUtilities.invokeLater
187   } // fim do método setTextFieldEditable
188} // fim da classe Client
```

Resumo

Client.java

(7 de 7)

Resumo

ClientTest.java

(1 de 2)

Saída do programa

```
1  // Fig. 24.8: ClientTest.java
2  // Testa a classe Client.
3  import javax.swing.JFrame;
4
5  public class ClientTest
6  {
7     public static void main( String args[] )
8     {
9        Client application; // declara o aplicativo cliente
10
11       // se não houver nenhum argumento de linha de comando
12       if ( args.length == 0 )
13          application = new Client( "127.0.0.1" ); // conecta-se ao host local
14       else
15          application = new Client( args[ 0 ] ); // usa argumentos p/ se conectar
16
17       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
18       application.runClient(); // executa o aplicativo cliente
19    } // fim do main
20 } // fim da classe ClientTest
```

Resumo

`ClientTest.java`

(2 de 2)

Saída do programa

24.7 Interação cliente/servidor sem conexão com datagramas

- **Transmissão sem conexão com datagramas:**
  - **Nenhuma conexão mantida com um outro computador.**
  - **Divide a mensagem em partes separadas e as envia como pacotes.**
  - **Uma mensagem pode chegar na ordem, fora da ordem ou simplesmente não chegar.**
  - **O receptor coloca mensagens na ordem e as lê.**

Resumo

Server.java

(1 de 4)

Linha 16

```
1  // Fig. 24.9: Server.java
2  // Servidor que recebe e envia pacotes de/para um cliente.
3  import java.io.IOException;
4  import java.net.DatagramPacket;
5  import java.net.DatagramSocket;
6  import java.net.SocketException;
7  import java.awt.BorderLayout;
8  import javax.swing.JFrame;
9  import javax.swing.JScrollPane;
10 import javax.swing.JTextArea;
11 import javax.swing.SwingUtilities;
12
13 public class Server extends JFrame
14 {
15    private JTextArea displayArea; // exibe pacotes recebidos
16    private DatagramSocket socket; // socket para se conectar ao cliente
17
18    // configura a GUI e o DatagramSocket
19    public Server()
20    {
21       super( "Server" );
22
23       displayArea = new JTextArea(); // cria displayArea
24       add( new JScrollPane( displayArea ), BorderLayout.CENTER );
25       setSize( 400, 300 ); // configura o tamanho da janela
26       setVisible( true ); // mostra a janela
27
```

Utiliza um `DatagramSocket` como nosso servidor

```
28       try // cria DatagramSocket para enviar e receber pacotes
29       {
30          socket = new DatagramSocket( 5000 );
31       } // fim do try
32       catch ( SocketException socketException )
33       {
34          socketException.printStackTrace();
35          System.exit( 1 );
36       } // fim do catch
37    } // fim do construtor Server
38
39    // espera pacotes chegarem, exibe dados e ecoa o pacote para o cliente
40    public void waitForPackets()
41    {
42       while ( true )
43       {
44          try // recebe pacote, exibe conteúdo, retorna cópia para o cliente
45          {
46             byte data[] = new byte[ 100 ]; // configura o pac
47             DatagramPacket receivePacket =
48                new DatagramPacket( data, data.length );
49
50             socket.receive( receivePacket ); // espera recebe
51
```

Utiliza o construtor `DatagramSocket` que recebe um argumento de número da porta inteiro para vincular o servidor a uma porta em que possa receber pacotes dos clientes

Linha 30

Linhas 47-48

Linha 50

Cria um `DatagramPacket` em que um pacote de informações recebido pode ser armazenado

Utiliza o método `DatagramSocket receive` para esperar um pacote chegar ao servidor

```
52          // exibe informações do pacote recebido
53          displayMessage( "\nPacket received:" +
54             "\nFrom host: " + receivePacket.getAddress() +
55             "\nHost port: " + receivePacket.getPort() +
56             "\nLength: " + receivePacket.getLength() +
57             "\nContaining:\n\t" + new String( receivePacket.get
58                0, receivePacket.getLength() ) );
59
60          sendPacketToClient( receivePacket ); // envia pacote p
61       } // fim do try
62       catch ( IOException ioException )
63       {
64          displayMessage( ioException.toString() +
65          ioException.printStackTrace();
66       } // fim do catch
67    } // fim do while
68 } // fim do método waitForPackets
69
70 // ecoa pacote para o cliente
71 private void sendPacketToClient( DatagramPacket receivePacket )
72    throws IOException
73 {
74    displayMessage( "\n\nEcho data to client..." )
75
76    // cria pacote a enviar
77    DatagramPacket sendPacket = new DatagramPacket(
78       receivePacket.getData(), receivePacket.getLength(),
79       receivePacket.getAddress(), receivePacket.getPort() );
80
```

Utiliza o método `DatagramPacket` `getLength` para obter o número de bytes dos dados enviado

Utiliza o método `DatagramPacket` `getData` para obter um array de bytes contendo os dados

Linha 57

Linhas 77-79

Cria um `DatagramPacket`, que especifica os dados a enviar, o número de bytes a enviar, o endereço Internet do computador cliente e a porta em que o cliente está esperando receber os pacotes

```
81       socket.send( sendPacket ); // envia o pacote para o cliente
82       displayMessage( "Packet sent\n" );
83    } // fim do método sendPacketToClient
84
85    // manipula a displayArea na thread de despach
86    private void displayMessage( final String messageToDisplay )
87    {
88       SwingUtilities.invokeLater(
89          new Runnable()
90          {
91             public void run() // atualiza displayArea
92             {
93                displayArea.append( messageToDisplay ); // exibe a mensagem
94             } // fim do método run
95          } // fim da classe interna anônima
96       ); // fim da chamada para SwingUtilities.invokeLater
97    } // fim do método displayMessage
98 } // fim da classe Server
```

Utiliza o método `DatagramSocket` `send` para enviar o pacote pela rede

Resumo

`Server.java`

(4 de 4)

Resumo

ServerTest.java

Saída de programa

```
1  // Fig. 24.10: ServerTest.java
2  // Testa a classe Server.
3  import javax.swing.JFrame;
4
5  public class ServerTest
6  {
7     public static void main( String args[] )
8     {
9        Server application = new Server(); // cria o servidor
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       application.waitForPackets(); // executa o aplicativo do servidor
12    } // fim do main
13 } // fim da classe ServerTest
```

Janela **Server** depois que o pacote de dados é recebido do **client**

Resumo

Client.java

(1 de 5)

```
1  // Fig. 24.11: Client.java
2  // Cliente que envia e recebe pacotes para/a partir de um servidor.
3  import java.io.IOException;
4  import java.net.DatagramPacket;
5  import java.net.DatagramSocket;
6  import java.net.InetAddress;
7  import java.net.SocketException;
8  import java.awt.BorderLayout;
9  import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import javax.swing.JFrame;
12 import javax.swing.JScrollPane;
13 import javax.swing.JTextArea;
14 import javax.swing.JTextField;
15 import javax.swing.SwingUtilities;
16
17 public class Client extends JFrame
18 {
19    private JTextField enterField; // para inserir mensagens
20    private JTextArea displayArea; // para exibir mensagens
21    private DatagramSocket socket; // socket para conectar-se ao servidor
22
23    // configura a GUI e DatagramSocket
24    public Client()
25    {
26       super( "Client" );
27
```

Resumo

Client.java

(2 de 5)

Linha 41

```
28       enterField = new JTextField( "Type message here" );
29       enterField.addActionListener(
30          new ActionListener()
31          {
32             public void actionPerformed( ActionEvent event )
33             {
34                try // cria e envia o pacote
35                {
36                   // obtém a mensagem no campo de texto
37                   String message = event.g
38                   displayArea.append( "\nS
39                      message + "\n" );
40
41                   byte data[] = message.ge
42
43                   // cria sendPacket
44                   DatagramPacket sendPacket = new DatagramPacket( data
45                      data.length, InetAddress.getLocalHost(), 5000 );
46
47                   socket.send( sendPacket ); // envia o pacote
48                   displayArea.append( "Packet sent\n" );
49                   displayArea.setCaretPosition(
50                      displayArea.getText().length() );
51                } // fim do try
```

Cria um `DatagramPacket` e o inicializa com o array de bytes, o comprimento da string que foi inserida pelo usuário, o endereço IP ao qual o pacote deve ser enviado e o número de porta em que o servidor está esperando

Utiliza o método `DatagramPacket send` para enviar o pacote

Resumo

Client.java

(3 de 5)

Linha 71

```
52              catch ( IOException ioException )
53              {
54                 displayMessage( ioException.toString() + "\n" );
55                 ioException.printStackTrace();
56              } // fim do catch
57           } // fim do actionPerformed
58        } // fim da classe interna anônima
59     ); // fim da chamada para addActionListener
60
61     add( enterField, BorderLayout.NORTH );
62
63     displayArea = new JTextArea();
64     add( new JScrollPane( displayArea ), BorderLayout.CENTER );
65
66     setSize( 400, 300 ); // configura o tamanho da janela
67     setVisible( true ); // mostra a janela
68
69     try // cria DatagramSocket para enviar e receber pacotes
70     {
71        socket = new DatagramSocket();
72     } // fim do try
73     catch ( SocketException socketException )
74     {
75        socketException.printStackTrace();
76        System.exit( 1 );
77     } // fim do catch
78  } // fim do construtor Client
79
```

Cria `DatagramSocket` para envio e recebimento de pacotes

Resumo

```
80    // espera que os pacotes cheguem do Server, exibe o conteúdo do pacote
81    public void waitForPackets()
82    {
83       while ( true )
84       {
85          try // recebe pacote e exibe conteúdo
86          {
87             byte data[] = new byte[ 100 ]; // configura o pacote
88             DatagramPacket receivePacket = new DatagramPacket(
89                data, data.length );
90
91             socket.receive( receivePacket ); // espera o pacote
92
93             // exibe o conteúdo do pacote
94             displayMessage( "\nPacket received:" +
95                "\nFrom host: " + receivePacket.getAddress() +
96                "\nHost port: " + receivePacket.getPort() +
97                "\nLength: " + receivePacket.getLength() +
98                "\nContaining:\n\t" + new String( receivePacket.getD
99                   0, receivePacket.getLength() ) );
100         } // fim do try
101         catch ( IOException exception )
102         {
103            displayMessage( exception.toString() + "\
104            exception.printStackTrace();
105         } // fim do catch
106      } // fim do while
107   } // fim do método waitForPackets
108
```

Client.java

Linha 96

Cria um `DatagramPacket` para armazenar informações recebidas

Linha 97

Utiliza o método `DatagramPacket` `getLength` para obter o número de bytes dos dados enviado

Utiliza o método `DatagramPacket` `getData` para obter um array de bytes contendo os dados

Resumo

Client.java

(5 de 5)

```
109   // manipula a displayArea na thread de despacho de eventos
110   private void displayMessage( final String messageToDisplay )
111   {
112      SwingUtilities.invokeLater(
113         new Runnable()
114         {
115            public void run() // atualiza displayArea
116            {
117               displayArea.append( messageToDisplay );
118            } // fim do método run
119         }  // fim da classe inner
120      ); // fim da chamada para SwingUtilities.invokeLater
121   } // fim do método displayMessage
122}  // fim da classe Client
```

Resumo

ClientTest.java

Saída do programa

```
1  // Fig. 24.12: ClientTest.java
2  // Testa a classe Client.
3  import javax.swing.JFrame;
4
5  public class ClientTest
6  {
7     public static void main( String args[] )
8     {
9        Client application = new Client(); // cria o cliente
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       application.waitForPackets(); // executa o aplicativo cliente
12    } // fim do main
13 }  // fim da classe ClientTest
```

Janela **Client** depois de enviar pacotes para **Server** e receber pacotes de volta de **Server**

Erro comum de programação 24.2

Especificar uma porta que já está em utilização ou especificar um número de porta inválido ao criar um `DatagramSocket` resulta em uma `SocketException`.

24.8 Jogo-da-velha cliente/servidor que utiliza um servidor com multithread

- **Múltiplas threads:**
  - **O servidor utiliza uma thread por jogador.**
    - **Permite que cada jogador jogue de maneira independente**

Resumo

TicTacToeServer.java

(1 de 12)

```
1  // Fig. 24.13: TicTacToeServer.java
2  // Essa classe mantém um jogo da velha para dois clientes.
3  import java.awt.BorderLayout;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6  import java.io.IOException;
7  import java.util.Formatter;
8  import java.util.Scanner;
9  import java.util.concurrent.ExecutorService;
10 import java.util.concurrent.Executors;
11 import java.util.concurrent.locks.Lock;
12 import java.util.concurrent.locks.ReentrantLock;
13 import java.util.concurrent.locks.Condition;
14 import javax.swing.JFrame;
15 import javax.swing.JTextArea;
16 import javax.swing.SwingUtilities;
17
```

Resumo

TicTacToeServer.java

(2 de 12)

```
18 public class TicTacToeServer extends JFrame
19 {
20    private String[] board = new String[ 9 ]; // tabuleiro do jogo-da-velha
21    private JTextArea outputArea; // para gerar saída das jogadas
22    private Player[] players; // array de Players
23    private ServerSocket server; // socket de servidor para conectar com clientes
24    private int currentPlayer; // monitora o jogador com a jogada atual
25    private final static int PLAYER_X = 0; // constante para o primeiro jogador
26    private final static int PLAYER_O = 1; // constante para o segundo jogador
27    private final static String[] MARKS = { "X", "O" }; // array de marcas
28    private ExecutorService runGame; // executará os jogadores
29    private Lock gameLock; // para bloquear a sincronização do jogo
30    private Condition otherPlayerConnected; // para esperar outro jogador
31    private Condition otherPlayerTurn; // para esperar a jogada do outro jogador
32
33    // configura o servidor de tic-tac-toe e a GUI que exibe as mensagens
34    public TicTacToeServer()
35    {
36       super( "Tic-Tac-Toe Server" ); // configura o título da janela
37
38       // cria ExecutorService com uma thread para cada jogador
39       runGame = Executors.newFixedThreadPool( 2 );
40       gameLock = new ReentrantLock(); // cria um bloqueio para o jogo
41
42       // variável de condição para os dois jogadores sendo conectados
43       otherPlayerConnected = gameLock.newCondition();
44
45       // variável de condição para a jogada do outro jogador
46       otherPlayerTurn = gameLock.newCondition();
47
```

```
48       for ( int i = 0; i < 9; i++ )
49          board[ i ] = new String( "" ); // cria o tabuleiro de jogo-da-velh
50       players = new Player[ 2 ]; // cria array de jogadores
51       currentPlayer = PLAYER_X; // configura o jogador atual como o primeiro joga
52
53       try
54       {
55          server = new ServerSocket( 12345, 2 ); // configura ServerSocket
56       } // fim do try
57       catch ( IOException ioException )
58       {
59          ioException.printStackTrace();
60          System.exit( 1 );
61       } // fim do catch
62
63       outputArea = new JTextArea(); // cria JTextArea para saída
64       add( outputArea, BorderLayout.CENTER );
65       outputArea.setText( "Server awaiting connections\n" );
66
67       setSize( 300, 300 ); // configura o tamanho da janela
68       setVisible( true ); // mostra a janela
69    } // fim do construtor TicTacToeServer
70
```

Cria um array `players` com 2 elementos

TicTacToeServer.java

Cria `ServerSocket` para ouvir na porta `12345`

Linha 55

```
71    // espera duas conexões para que o jogo possa ser jogado
72    public void execute()
73    {
74       // espera que cada cliente se conecte
75       for ( int i = 0; i < players.length; i++ )
76       {
77          try // espera a conexão, cria Player, inicia o executável
78          {
79             players[ i ] = new Player( server.accept(), i );
80             runGame.execute( players[ i ] ); // executa o executável de j
81          } // fim de try
82          catch ( IOException ioException )
83          {
84             ioException.printStackTrace();
85             System.exit( 1 );
86          } // fim do catch
87       } // fim do for
88
89       gameLock.lock(); // bloqueia o jogo para sinalizar a thread do jogador X
90
```

Faz um loop duas vezes, bloqueando na linha 79 toda vez enquanto espera a

Cria um novo objeto `Player` para gerenciar a como uma thread separada

Executa o `Player` no pool de threads de `runGame`

Resumo

Linha 79

Linha 80

Resumo

TicTacToeServer.java

(5 de 12)

```
91       try
92       {
93          players[ PLAYER_X ].setSuspended( false ); // retoma o jogador X
94          otherPlayerConnected.signal(); // acorda a thread do jogador X
95       } // fim do try
96       finally
97       {
98          gameLock.unlock(); // desbloqueia o jogo depois de sinalizar p/ o jogador X
99       } // fim do finally
100   } // fim do método execute
101
102   // exibe mensagem na outputArea
103   private void displayMessage( final String messageToDisplay )
104   {
105      // exibe uma mensagem da thread de despacho de eventos da execução
106      SwingUtilities.invokeLater(
107         new Runnable()
108         {
109            public void run() // atualiza outputArea
110            {
111               outputArea.append( messageToDisplay ); // adiciona mensagem
112            } // fim do método run
113         } // fim da classe inner
114      ); // fim da chamada para SwingUtilities.invokeLater
115   } // fim do método displayMessage
116
```

Resumo

TicTacToeServer.java

(6 de 12)

```
117   // determina se a jogada é válida
118   public boolean validateAndMove( int location, int player )
119   {
120      // enquanto não for o jogador atual, deve esperar a jogada
121      while ( player != currentPlayer )
122      {
123         gameLock.lock(); // bloqueia o jogo para que o outro jogador prossiga
124
125         try
126         {
127            otherPlayerTurn.await(); // espera a jogada do jogador
128         } // fim do try
129         catch ( InterruptedException exception )
130         {
131            exception.printStackTrace();
132         } // fim do catch
133         finally
134         {
135            gameLock.unlock(); // desbloqueia o jogo depois de esperar
136         } // fim do finally
137      } // fim do while
138
139      // se a posição não estiver ocupada, faz a jogada
140      if ( !isOccupied( location ) )
141      {
142         board[ location ] = MARKS[ currentPlayer ]; // configura uma jogada no tabuleiro
143         currentPlayer = ( currentPlayer + 1 ) % 2; // troca o jogador
144
```

Resumo

TicTacToeServer.java

(7 de 12)

```
145         // deixa que novo jogador atual saiba que a jogada ocorreu
146         players[ currentPlayer ].otherPlayerMoved( location );
147
148         gameLock.lock(); // bloqueia o jogo p/ sinalizar ao outro jogador a prosseguir
149
150         try
151         {
152            otherPlayerTurn.signal(); // sinaliza que o outro jogador continue
153         } // fim do try
154         finally
155         {
156            gameLock.unlock(); // desbloqueia jogo depois de sinalizar
157         } // fim do finally
158
159         return true; // notifica o jogador que a jogada foi válida
160      } // fim do if
161      else // jogada não era válida
162         return false; // notifica o jogador que a jogada foi inválida
163   } // fim do método validateAndMove
164
165   // determina se a posição está ocupada
166   public boolean isOccupied( int location )
167   {
168      if ( board[ location ].equals( MARKS[ PLAYER_X ] ) ||
169         board [ location ].equals( MARKS[ PLAYER_O ] ) )
170         return true; // posição é ocupada
171      else
172         return false; // posição não é ocupada
173   } // fim do método isOccupied
174
```

Resumo

TicTacToeServer.java

(8 de 12)

Linhas 200-201

```
175  // coloca o código nesse método para determinar se o jogo terminou
176  public boolean isGameOver()
177  {
178     return false; // isso é deixado como um exercício
179  } // fim do método isGameOver
180
181  // classe interna privada Player gerencia cada Player como um executável
182  private class Player implements Runnable
183  {
184     private Socket connection; // conexão com o cliente
185     private Scanner input; // entrada do cliente
186     private Formatter output; // saída para o client
187     private int playerNumber; // monitora que jogador é esse
188     private String mark; // marca para esse jogador
189     private boolean suspended = true; // se a thread for suspensa
190
191     // configura a thread Player
192     public Player( Socket socket, int number )
193     {
194        playerNumber = number; // armazena o número desse jogador
195        mark = MARKS[ playerNumber ]; // especifica a marca do jogador
196        connection = socket; // armazena o socket para o cliente
197
198        try // obtém fluxos a partir do Socket
199        {
200           input = new Scanner( connection.getInputStream() );
201           output = new Formatter( connection.getOutputStream() );
202        } // fim do try
```

Obtém os fluxos para enviar e receber dados

Resumo

TicTacToeServer.java

(9 de 12)

```
203        catch ( IOException ioException )
204        {
205           ioException.printStackTrace();
206           System.exit( 1 );
207        } // fim do catch
208     } // fim do construtor Player
209
210     // envia uma mensagem de que o outro jogador fez uma jogada
211     public void otherPlayerMoved( int location )
212     {
213        output.format( "Opponent moved\n" );
214        output.format( "%d\n", location ); // envia posição da
215        output.flush(); // esvazia a saída
216     } // fim do método otherPlayerMoved
217
218     // controla a execução da thread
219     public void run()
220     {
221        // envia ao cliente a marca (X ou O), processa as mensagens do cliente
222        try
223        {
224           displayMessage( "Player " + mark + " connected\n" );
225           output.format( "%s\n", mark ); // envia a marca do jogador
226           output.flush(); // esvazia a saída
227
```

Formata a saída, ...lo o outro ...la jogada

Chama o método `Formatter` `flush` para forçar a saída para o cliente

...25-226

Envia a marca do jogador

Resumo

```
228         // se for o jogador X, espera que o outro jogador chegue
229         if ( playerNumber == PLAYER_X )
230         {
231            output.format( "%s\n%s", "Player X connected",
232               "Waiting for another player\n" );
233            output.flush(); // esvazia a saída
234
235            gameLock.lock(); // bloqueia o jogo p/ esperar o segu
236
237            try
238            {
239               while( suspended )
240               {
241                  otherPlayerConnected.await(); // espera o jogador O
242               } // fim do while
243            } // fim do try
244            catch ( InterruptedException exception )
245            {
246               exception.printStackTrace();
247            } // fim do catch
248            finally
249            {
250               gameLock.unlock(); // desbloqueia o jogo depois do segundo jogador
251            } // fim do finally
252
253            // envia uma mensagem de que o outro jogador se conectou
254            output.format( "Other player connected. Your move.\n" );
255            output.flush(); // esvazia a saída
256         } // fim do if
```

Envia uma mensagem indicando que um dos jogadores está conectado e esperando um outro jogador chegar

Linhas 231-233

Linhas 254-255

Inicia o jogo

```
257         else
258         {
259            output.format( "Player O connected, please wait\n" );
260            output.flush(); // esvazia a saída
261         } // fim do else
262
263         // enquanto o jogo não terminou
264         while ( !isGameOver() )
265         {
266            int location = 0; // inicializa a posição da jogada
267
268            if ( input.hasNext() )
269               location = input.nextInt(); // obtém posição da jogada
270
271            // verifica jogada válida
272            if ( validateAndMove( location, playerNumber ) )
273            {
274               displayMessage( "\nlocation: " + location );
275               output.format( "Valid move.\n" ); // notifica o clien
276               output.flush(); // esvazia a saída
277            } // fim do if
```

Envia uma mensagem indicando que o jogador ‘O’ está conectado

Resumo

TicTacToeServer.java

(11 de 12)

Linhas 259-260

Lê uma jogada

Verifica a jogada

Linhas 275-276

Envia uma mensagem indicando que a jogada é válida

```
278             else // joga foi inválida
279             {
280                output.format( "Invalid move, try again\n" );
281                output.flush(); // esvazia a saída
282             } // fim do else
283          } // fim do while
284       } // fim do try
285       finally
286       {
287          try
288          {
289             connection.close(); // fecha conexão com o cliente
290          } // fim do try
291          catch ( IOException ioException )
292          {
293             ioException.printStackTrace();
294             System.exit( 1 );
295          } // fim do catch
296       } // fim do finally
297    } // fim do método run
298
299    // configura se a thread está ou não suspensa
300    public void setSuspended( boolean status )
301    {
302       suspended = status; // configura o valor do suspenso
303    } // fim do método setSuspended
304 } // fim da classe Player
305} // fim da classe TicTacToeServer
```

Envia uma mensagem indicando que a jogada é inválida

TicTacToeServer.java

(12 de 12)

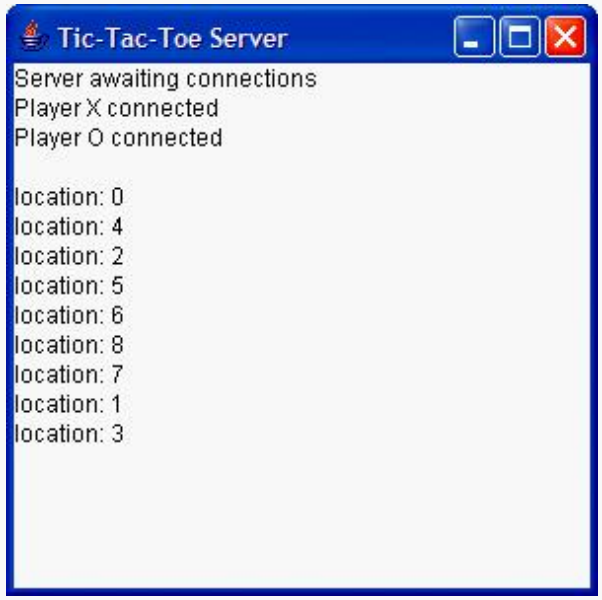
Linhas 280-281

Resumo

TicTacToeServer Test.java

Saída do programa

```
1  // Fig. 24.14: TicTacToeServerTest.java
2  // Testa a TicTacToeServer.
3  import javax.swing.JFrame;
4
5  public class TicTacToeServerTest
6  {
7     public static void main( String args[] )
8     {
9        TicTacToeServer application = new TicTacToeServer();
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       application.execute();
12    } // fim do main
13 } // fim da classe TicTacToeServerTest
```



Resumo

TicTacToeClient.java

(1 de 10)

```
1  // Fig. 24.15: TicTacToeClient.java
2  // Classe cliente p/ deixar um usuário jogar o jogo-da-velha c/ um outro usuário por uma rede.
3  import java.awt.BorderLayout;
4  import java.awt.Dimension;
5  import java.awt.Graphics;
6  import java.awt.GridLayout;
7  import java.awt.event.MouseAdapter;
8  import java.awt.event.MouseEvent;
9  import java.net.Socket;
10 import java.net.InetAddress;
11 import java.io.IOException;
12 import javax.swing.JFrame;
13 import javax.swing.JPanel;
14 import javax.swing.JScrollPane;
15 import javax.swing.JTextArea;
16 import javax.swing.JTextField;
17 import javax.swing.SwingUtilities;
18 import java.util.Formatter;
19 import java.util.Scanner;
20 import java.util.concurrent.Executors;
21 import java.util.concurrent.ExecutorService;
22
```

Resumo

TicTacToeClient.java

(2 de 10)

```
23 public class TicTacToeClient extends JFrame implements Runnable
24 {
25    private JTextField idField; // campo de texto para exibir a marca do jogador
26    private JTextArea displayArea; // JTextArea para exibir a saída
27    private JPanel boardPanel; // painel para o tabuleiro do jogo-da-velha
28    private JPanel panel2; // painel para conter o tabuleiro
29    private Square board[][]; // tabuleiro do jogo-da-velha
30    private Square currentSquare; // quadrado atual
31    private Socket connection; // conexão com o servidor
32    private Scanner input; // entrada a partir do servidor
33    private Formatter output; // saída para o server
34    private String ticTacToeHost; // nome do host para o servidor
35    private String myMark; // marca desse cliente
36    private boolean myTurn; // determina de qual cliente é a vez
37    private final String X_MARK = "X"; // marca para o primeiro cliente
38    private final String O_MARK = "O"; // marca para o segundo cliente
39
40    // configura a interface com o usuário e o tabuleiro
41    public TicTacToeClient( String host )
42    {
43       ticTacToeHost = host; // configura o nome do servidor
44       displayArea = new JTextArea( 4, 30 ); // configura JTextArea
45       displayArea.setEditable( false );
46       add( new JScrollPane( displayArea ), BorderLayout.SOUTH );
47
48       boardPanel = new JPanel(); // configura painel para quadrados no tabuleiro
49       boardPanel.setLayout( new GridLayout( 3, 3, 0, 0 ) );
50
```

Resumo

TicTacToeClient.java

(3 de 10)

```
51       board = new Square[ 3 ][ 3 ]; // cria o tabuleiro
52
53       // loop over the rows in the board
54       for ( int row = 0; row < board.length; row++ )
55       {
56          // faz um loop pelas linhas no tabuleiro
57          for ( int column = 0; column < board[ row ].length; column++ )
58          {
59             // cria o quadrado
60             board[ row ][ column ] = new Square( ' ', row * 3 + column );
61             boardPanel.add( board[ row ][ column ] ); // adiciona o quadrado
62          } // fim do for interno
63       } // fim da for externo
64
65       idField = new JTextField(); // configura o campo de texto
66       idField.setEditable( false );
67       add( idField, BorderLayout.NORTH );
68
69       panel2 = new JPanel(); // configura o painel para conter boardPanel
70       panel2.add( boardPanel, BorderLayout.CENTER ); // adiciona o painel do tabuleiro
71       add( panel2, BorderLayout.CENTER ); // adiciona o painel contêiner
72
73       setSize( 300, 225 ); // configura o tamanho da janela
74       setVisible( true ); // mostra a janela
75
76       startClient();
77    } // fim do construtor TicTacToeClient
78
```

Resumo

```
79    // inicia a thread cliente
80    public void startClient()
81    {
82       try // conecta-se ao servidor, obtém os fluxos e inicia o ou
83       {
84          // faz uma conexão com o servidor
85          connection = new Socket(
86             InetAddress.getByName( ticTacToeHost ), 12345 );
87
88          // obtém os fluxos de entrada e saída
89          input = new Scanner( connection.getInputStream() );
90          output = new Formatter( connection.getOutputStream() );
91       } // fim do try
92       catch ( IOException ioException )
93       {
94          ioException.printStackTrace();
95       } // fim do catch
96
97       // cria e inicia thread trabalhadora para esse cliente
98       ExecutorService worker = Executors.newFixedThreadPool( 1 );
99       worker.execute( this ); // executa o cliente
100   } // fim do método startClient
101
102   // controla a thread que permite atualização contínua da displayArea
103   public void run()
104   {
105      myMark = input.nextLine(); // obtém a marca do jogador (X ou O)
106
```

Conecta-se ao servidor

Obtém os fluxos para enviar e receber dados

Lê o caractere de marca no servidor

TicTacToeClient.java

(4 de 10)

Linhas 89-90

Linha 105

Resumo

TicTacToeClient.java

(5 de 10)

Linhas 121-125

Linhas 132-136

```
107     SwingUtilities.invokeLater(
108        new Runnable()
109        {
110           public void run()
111           {
112              // exibe a marca do jogador
113              idField.setText( "You are player \"" + myMark + "\"" );
114           } // fim do método run
115        } // fim da classe interna anônima
116     ); // fim da chamada para SwingUtilities.invokeLater
117
118     myTurn = ( myMark.equals( X_MARK ) ); // determina se a vez do cliente
119
120     // recebe mensagens enviadas para o cliente e gera sua saída
121     while ( true )
122     {
123        if ( input.hasNextLine() )
124           processMessage( input.nextLine() );
125     } // fim do while
126  } // fim do método run
127
128  // processa mensagens recebidas pelo cliente
129  private void processMessage( String message )
130  {
131     // ocorreu jogada válida
132     if ( message.equals( "Valid move." ) )
133     {
134        displayMessage( "Valid move, please wait.\n" );
135        setMark( currentSquare, myMark ); // configura marca no qua
136     } // fim do if
```

Faz um loop continuamente

Lê e processa as mensagens no servidor

Se a jogada é válida, grava a mensagem e configura a marca no quadrado

```
137     else if ( message.equals( "Invalid move, try again" ) )
138     {
139        displayMessage( message + "\n" ); // exibe jogada inválida
140        myTurn = true; // ainda é a vez desse cliente
141     } // fim do else if
142     else if ( message.equals( "Opponent moved" ) )
143     {
144        int location = input.nextInt(); // obtém a posição da jogada
145        input.nextLine(); // pula uma nova linha depois da posição de int
146        int row = location / 3; // calcula linha
147        int column = location % 3; // calcula coluna
148
149        setMark(  board[ row ][ column ],
150           ( myMark.equals( X_MARK ) ? O_MARK : X_MARK ) ); // marca a jogada
151        displayMessage( "Opponent moved. Your turn.\n" );
152        myTurn = true; // agora é a vez desse cliente
153     } // fim de else if
154     else
155        displayMessage( message + "\n" ); // exibe a mensagem
156  } // fim do método processMessage
157
```

Se o oponente jogar, configura a marca no quadrado

Resumo

(6 de 10)

Linhas 137-141

Linhas 142-153

Resumo

TicTacToeClient.java

(7 de 10)

```
158   // manipula outputArea na thread de despacho do evento
159   private void displayMessage( final String messageToDisplay )
160   {
161      SwingUtilities.invokeLater(
162         new Runnable()
163         {
164            public void run()
165            {
166               displayArea.append( messageToDisplay ); // atualiza saída
167            } // fim do método run
168         }  // fim da classe inner
169      ); // fim da chamada para SwingUtilities.invokeLater
170   } // fim do método displayMessage
171
172   // método utilitário p/ configurar a marca sobre o tabuleiro na thread de despacho de eventos
173   private void setMark( final Square squareToMark, final String mark )
174   {
175      SwingUtilities.invokeLater(
176         new Runnable()
177         {
178            public void run()
179            {
180               squareToMark.setMark( mark ); // configura a marca no quadrado
181            } // fim do método run
182         } // fim da classe interna anônima
183      ); // fim da chamada para SwingUtilities.invokeLater
184   } // fim do método setMark
185
```

Resumo

TicTacToeClient.java

```
186   // envia mensagem para o servidor indicando o quadrado clicado
187   public void sendClickedSquare( int location )
188   {
189      // se for minha vez
190      if ( myTurn )
191      {
192         output.format( "%d\n", location ); // envia a posição ao servidor
193         output.flush();
194         myTurn = false; // não é minha vez
195      } // fim do if
196   } // fim do método sendClickedSquare
197
198   // configura quadrado atual
199   public void setCurrentSquare( Square square )
200   {
201      currentSquare = square; // configura quadrado atual para o argumento
202   } // fim do método setCurrentSquare
203
204   // classe interna privada para os quadrados no tabuleiro
205   private class Square extends JPanel
206   {
207      private String mark; // marca a ser desenhada nesse quadrado
208      private int location; // posição do quadrado
209
210      public Square( String squareMark, int squareLocation )
211      {
212         mark = squareMark; // configura marca para esse quadrado
213         location = squareLocation; // configura posição desse quadrado
214
```

Linhas 192-193

Envia a jogada ao servidor

Resumo

TicTacToeClient.java

(9 de 10)

```
215       addMouseListener(
216          new MouseAdapter()
217          {
218             public void mouseReleased( MouseEvent e )
219             {
220                setCurrentSquare( Square.this ); // configura o quadrado atual
221
222                // envia a posição desse quadrado
223                sendClickedSquare( getSquareLocation() );
224             } // fim do método mouseReleased
225          } // fim da classe interna anônima
226       ); // fim da chamada para addMouseListener
227    } // fim do construtor Square
228
229    // retorna o tamanho preferido de Square
230    public Dimension getPreferredSize()
231    {
232       return new Dimension( 30, 30 ); // retorna o tamanho preferido
233    } // fim do método getPreferredSize
234
235    // retorna o tamanho mínimo de Square
236    public Dimension getMinimumSize()
237    {
238       return getPreferredSize(); // retorna o tamanho preferido
239    } // fim do método getMinimumSize
240
```

Resumo

TicTacToeClient.java

(10 de 10)

```
241      // configura marca para Square
242      public void setMark( String newMark )
243      {
244         mark = newMark; // configura a marca do quadrado
245         repaint(); // pinta o quadrado novamente
246      } // fim do método setMark
247
248      // retorna a posição de Square
249      public int getSquareLocation()
250      {
251         return location; // retorna a posição do quadrado
252      } // fim do método getSquareLocation
253
254      // desenha Square
255      public void paintComponent( Graphics g )
256      {
257         super.paintComponent( g );
258
259         g.drawRect( 0, 0, 29, 29 ); // desenha o quadrado
260         g.drawString( mark, 11, 20 ); // desenha a marca
261      } // fim do método paintComponent
262   } // fim da classe interna Square
263} // fim da classe TicTacToeClient
```

Resumo

TicTacToeClient Test.java

```
1  // Fig. 24.16: TicTacToeClientTest.java
2  // Testa a classe TicTacToeClient.
3  import javax.swing.JFrame;
4
5  public class TicTacToeClientTest
6  {
7     public static void main( String args[] )
8     {
9        TicTacToeClient application; // declara o aplicativo cliente
10
11       // se não houver nenhum argumento de linha de comando
12       if ( args.length == 0 )
13          application = new TicTacToeClient( "127.0.0.1" ); // host local
14       else
15          application = new TicTacToeClient( args[ 0 ] ); // usa argumentos
16
17       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
18    } // fim do main
19 } // fim da classe TicTacToeClientTest
```

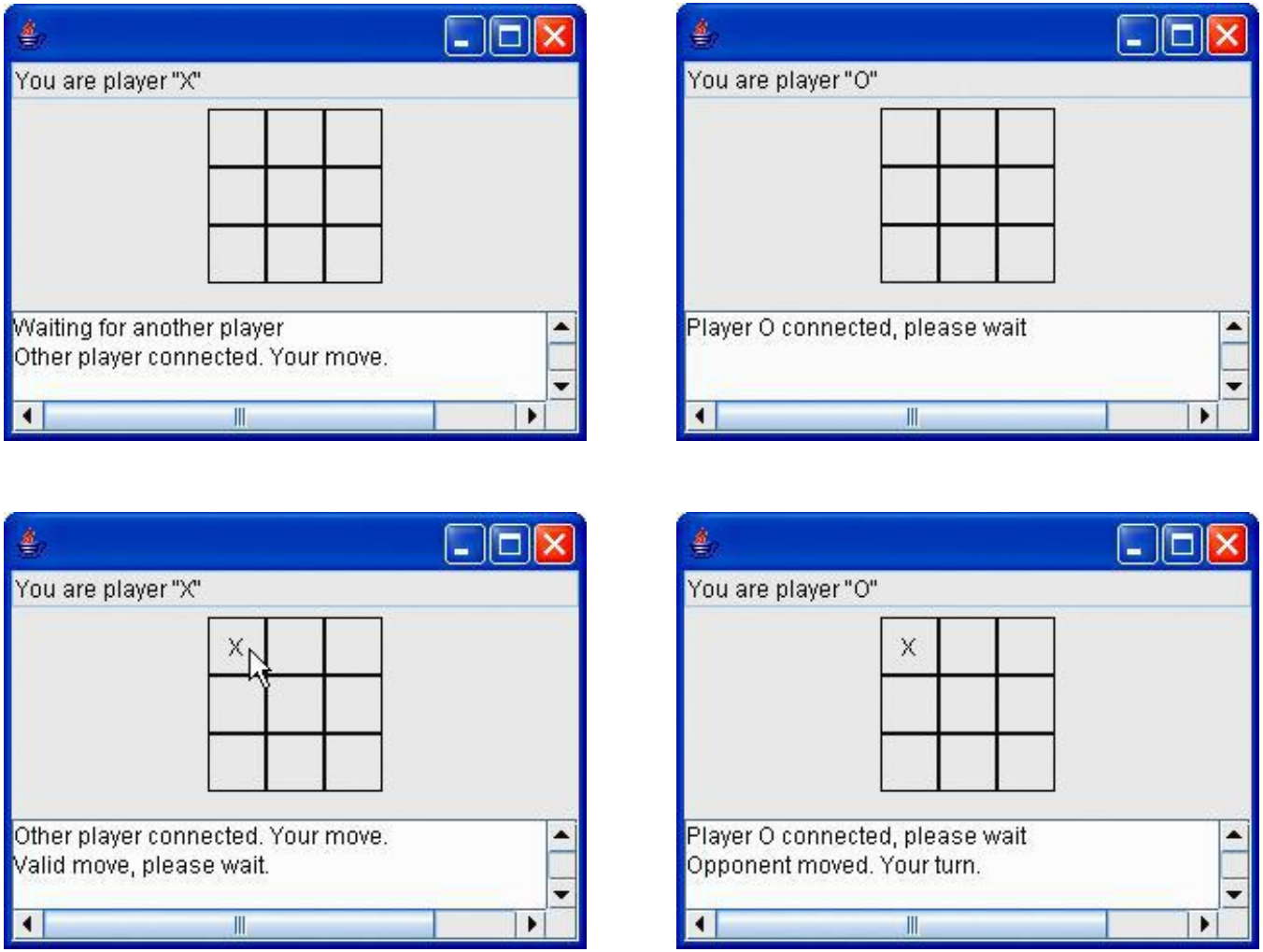


Figura 24.17 | Saídas de exemplo do programa cliente/servidor Tic-Tac-Toe. (Parte 1 de 2.)

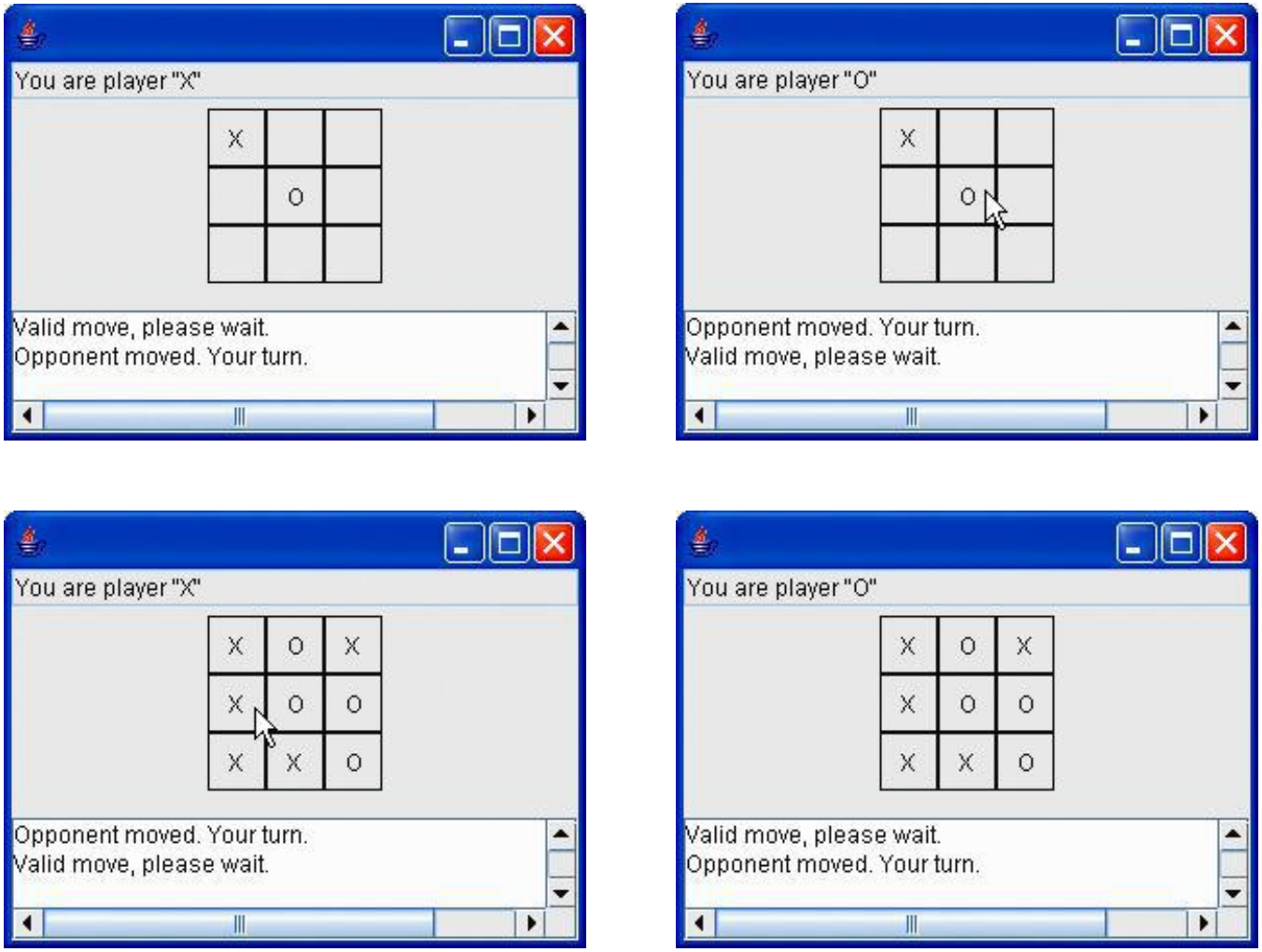


Figura 24.17 | Saídas de exemplo do programa cliente/servidor Tic-Tac-Toe. (Parte 2 de 2.)

24.9 Segurança e redes

- **Por padrão, applets não podem realizar o processamento de arquivo.**
- **Em geral, as applets são limitadas quanto ao acesso à máquina.**
  - **Applets podem se comunicar somente com a máquina a partir da qual foram originalmente baixadas.**
- **API do Java Security:**
  - **Applets digitalmente assinadas.**
  - **Applets com mais privilégios se provenientes de uma fonte confiável.**

24.10 Estudo de caso: Servidor e cliente `DeitelMessenger`

- **Salas de bate-papo:**
  - **Cada usuário pode postar uma mensagem e ler todas as outras mensagens.**
  - **Multicast:**
    - **Envia pacotes aos grupos de clientes.**

24.10.1 DeitelMessengerServer e classes de suporte

- DeitelMessengerServer:
  - Sistema de bate-papo on-line.
  - Classes:
    - DeitelMessengerServer
      - Cliente conecta-se a esse servidor.
    - Interface SocketMessengerConstants
      - Define constantes para números de porta.
    - Interface MessageListener
      - Define o método para receber novas mensagens de bate-papo.
    - Classe MessageReceiver
      - Thread separada ouve as mensagens dos clientes.
    - Classe MulticastSender
      - Thread separada entrega as mensagens enviadas aos clientes.

Resumo

DeiterMessenger Server.java

(1 de 3)

Linha 15

```
1  // Fig. 24.18: DeitelMessengerServer.java
2  // DeitelMessengerServer é um servidor de bate-papo com múltiplas threads,
3  // baseado em socket e em pacotes.
4  package com.deitel.messenger.sockets.server;
5
6  import java.net.ServerSocket;
7  import java.net.Socket;
8  import java.io.IOException;
9  import java.util.concurrent.Executors;
10 import java.util.concurrent.ExecutorService;
11
12 import com.deitel.messenger.MessageListener;
13 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
14
15 public class DeitelMessengerServer implements MessageListener
16 {
17    private ExecutorService serverExecutor; // executor para o servidor
18
19    // inicia o servidor de bate-papo
20    public void startServer()
21    {
22       // cria o executor para executáveis do servidor
23       serverExecutor = Executors.newCachedThreadPool();
24
25       try // cria o servidor e gerencia novos clientes
26       {
27          // cria ServerSocket para conexões entrantes
28          ServerSocket serverSocket =
29             new ServerSocket( SERVER_PORT, 100 );
30
```

Implementa a interface `MessageListener`

Cria um `ServerSocket` para aceitar conexões de rede entrantes

Resumo

```
31          System.out.printf( "%s%d%s", "Server listening on port ",
32             SERVER_PORT, " ..." );
33
34          // ouve clientes constantemente
35          while ( true )
36          {
37             // aceita nova conexão de cliente
38             Socket clientSocket = serverSocket.accept();
39
40             // cria MessageReceiver para receber mensagens do cli
41             serverExecutor.execute(
42                new MessageReceiver( this, clientSocket ) );
43
44             // imprime informações de conexão
45             System.out.println( "Connection received from: " +
46                clientSocket.getInetAddress() );
47          } // fim do while
48       } // fim do try
```

DeiterMessenger

Invoca o método

Cria e inicia um novo `MessageReceiver` para o cliente

Linhas 41-42

Resumo

DeiterMessenger Server.java

(3 de 3)

Linhas 62-63

```
49       catch ( IOException ioException )
50       {
51          ioException.printStackTrace();
52       } // fim do catch
53    } // fim do método startServer
54
55    // quando nova mensagem é recebida, transmite a mensagem para os clientes
56    public void messageReceived( String from, String message )
57    {
58       // cria String contendo mensagem inteira
59       String completeMessage = from + MESSAGE_SEPARATOR + message;
60
61       // cria e inicia MulticastSender para transmitir mensagens
62       serverExecutor.execute(
63          new MulticastSender( completeMessage.getBytes() ) );
64    } // fim do método messageReceived
65 } // fim da classe DeitelMessengerServer
```

Cria e inicia um novo **MulticastSender** para entregar **completeMessage** a todos os clientes

Resumo

DeitelMessenger ServerTest.java

```
1  // Fig. 24.19: DeitelMessengerServerTest.java
2  // Testa a classe DeitelMessengerServer.
3  package com.deitel.messenger.sockets.server;
4
5  public class DeitelMessengerServerTest
6  {
7     public static void main ( String args[] )
8     {
9        DeitelMessengerServer application = new DeitelMessengerServer();
10       application.startServer(); // inicia o servidor
11    } // fim do main
12 } // fim da classe DeitelMessengerServerTest
```

Saída do programa

```
Server listening on port 12345 ...
Connection received from: /127.0.0.1
Connection received from: /127.0.0.1
Connection received from: /127.0.0.1
```

Resumo

```
1  // Fig. 24.20: SocketMessengerConstants.java
2  // SocketMessengerConstants define constantes para números de porta
3  // e o endereço de multicast em DeitelMessenger
4  package com.deitel.messenger.sockets;
5
6  public interface SocketMessengerConstants
7  {
8     // endereço para datagramas de multicast
9     public static final String MULTICAST_ADDRESS = "239.0.0.1";
10
11    // porta para ouvir datagramas de multicast
12    public static final int MULTICAST_LISTENING_PORT = 5555;
13
14    // porta para enviar datagramas de multicast
15    public static final int MULTICAST_SENDING_PORT = 5554;
16
17    // porta para conexões de Socket para DeitelMessengerServer
18    public static final int SERVER_PORT = 12345;
19
20    // String que indica desconexão
21    public static final String DISCONNECT_STRING = "DISCONNECT";
22
23    // String que separa o nome de usuário do corpo da mensagem
24    public static final String MESSAGE_SEPARATOR = ">>>";
25
26    // tamanho da mensagem (em bytes)
27    public static final int MESSAGE_SIZE = 512;
28 } // fim da interface SocketMessengerConstants
```

SocketMessenger Constants.java

Endereço para enviar datagramas de multicast

Porta que ouve os datagramas de multicast

Porta para enviar datagramas de multicast

Porta para conexões de socket ao servidor

`String` que indica desconexão

`String` que separa o nome do

Tamanho máximo da mensagem (em bytes)

Resumo

```
1  // Fig. 24.21: MessageListener.java
2  // MessageListener é uma interface para classes que desejam
3  // receber novas mensagens de bate-papo.
4  package com.deitel.messenger;
5
6  public interface MessageListener
7  {
8     // recebe uma nova mensagem de bate-papo
9     public void messageReceived( String from, String message );
10 } // fim da interface MessageListener
```

MessageListener.java

O método `messageReceived` permite que uma classe implementadora receba mensagens de bate-papo

Resumo

MessageReceiver.java

(1 de 5)

```
1  // Fig. 24.22: MessageReceiver.java
2  // MessageReceiver é um Runnable que ouve mensagens de um
3  // cliente particular e entrega mensagens para um MessageListener.
4  package com.deitel.messenger.sockets.server;
5
6  import java.io.BufferedReader;
7  import java.io.IOException;
8  import java.io.InputStreamReader;
9  import java.net.Socket;
10 import java.net.SocketTimeoutException;
11 import java.util.StringTokenizer;
12
13 import com.deitel.messenger.MessageListener;
14 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
15
16 public class MessageReceiver implements Runnable
17 {
18    private BufferedReader input; // fluxo de entrada
19    private MessageListener messageListener; // ouviente da mensagem
20    private boolean keepListening = true; // quando falsa, termina executável
21
22    // construtor MessageReceiver
23    public MessageReceiver( MessageListener listener, Socket clientSocket )
24    {
25       // configura o ouvinte ao qual novas mensagens devem ser enviadas
26       messageListener = listener;
27
```

```
28       try
29       {
30          // configura o tempo limite para leitura do cliente
31          clientSocket.setSoTimeout( 5000 ); // cinco segundos
32
33          // cria BufferedReader para ler mensagens entrantes
34          input = new BufferedReader( new InputStreamReader(
35             clientSocket.getInputStream() ) );
36       } // fim do try
37       catch ( IOException ioException )
38       {
39          ioException.printStackTrace();
40       } // fim do catch
41    } // fim do construtor MessageReceiver
42
43    // ouve novas mensagens e as envia para o MessageListener
44    public void run()
45    {
46       String message; // String para mensagens entrantes
47
```

Tenta ler por cinco segundos

Resumo

MessageReceiver.java

(2 de 5)

Linha 31

Resumo

```
48    // ouve mensagens até ser parado
49    while ( keepListening )
50    {
51       try
52       {
53          message = input.readLine(); // lê a mensagem do cliente
54       } // fim do try
55       catch ( SocketTimeoutException socketTimeoutException )
56       {
57          continue; // continua para a próxima iteração para se
58       } // fim do catch
59       catch ( IOException ioException )
60       {
61          ioException.printStackTrace();
62          break;
63       } // fim do catch
64
65       // assegura que a mensagem não seja nula
66       if ( message != null )
67       {
68        // divide a mensagem em tokens p/ recuperar o nome do usuário e corpo da m
69          StringTokenizer tokenizer = new StringTokenizer(
70             message, MESSAGE_SEPARATOR );
71
```

Lê a linha dos dados no cliente

.java

Uma `SocketTimeOutException` é lançada se a leitura expirar

Linha 55

Linhas 69-70

Mensagem separada em dois tokens delimitados pelo `Message_SEPARATOR`

Resumo

```
72          // ignora as mensagens que não contêm um nome de usuário
73          // e um corpo de mensagem
74          if ( tokenizer.countTokens() == 2 )
75          {
76             // envia a mensagem para MessageListener
77             messageListener.messageReceived(
78                tokenizer.nextToken(), // nome de usuário
79                tokenizer.nextToken() ); // corpo de mensagem
80          } // fim do if
81          else
82          {
83             // se recebeu mensagem de desconexão, pára de ouvir
84             if ( message.equalsIgnoreCase(
85                MESSAGE_SEPARATOR + DISCONNECT_STRING ) )
86                stopListening();
87          } // fim do else
88       } // fim do if
89    } // fim do while
90
```

Invoca o método `messageReceived` da interface `MessageListener` para entregar a nova mensagem para o `MessageListener` registrado

Determina se a mensagem indica se o usuário deseja sair da sala de bate-papo

```
91       try
92       {
93          input.close(); // fecha o BufferedReader (também fecha o Socket)
94       } // fim do try
95       catch ( IOException ioException )
96       {
97          ioException.printStackTrace();
98       } // fim do catch
99    } // fim do método run
100
101   // pára de ouvir mensagens entrantes
102   public void stopListening()
103   {
104      keepListening = false;
105   } // fim do método stopListening
106} // fim da classe MessageReceiver
```

Resumo

MessageReceiver.java

(5 de 5)

Resumo

MulticastSender.java

(1 de 2)

Linhas 27-28

```
1  // Fig. 24.23: MulticastSender.java
2  // MulticastSender transmite uma mensagem de bate-papo usando um datagrama de multicast.
3  package com.deitel.messenger.sockets.server;
4
5  import java.io.IOException;
6  import java.net.DatagramPacket;
7  import java.net.DatagramSocket;
8  import java.net.InetAddress;
9
10 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
11
12 public class MulticastSender implements Runnable
13 {
14    private byte[] messageBytes; // dados de mensagem
15
16    public MulticastSender( byte[] bytes )
17    {
18       messageBytes = bytes; // cria a mensagem
19    } // fim do construtor MulticastSender
20
21    // entrega a mensagem para o MULTICAST_ADDRESS via um DatagramSocket
22    public void run()
23    {
24       try // entrega a mensagem
25       {
26          // cria DatagramSocket para enviar a mensagem
27          DatagramSocket socket =
28             new DatagramSocket( MULTICAST_SENDING_PORT );
29
```

Cria um `DatagramSocket` para entregar `DatagramPacket`s via multicast

Resumo

```
30          // utiliza o InetAddress reservado para grupo de multicast
31          InetAddress group = InetAddress.getByName( MULTICAST_ADDRESS );
32
33          // cria o DatagramPacket contendo a mensagem
34          DatagramPacket packet = new DatagramPacket( messageBytes,
35             messageBytes.length, group, MULTICAST_LISTENING_PORT );
36
37          socket.send( packet ); // envia o pacote para
38          socket.close(); // fecha o socket
39       } // fim do try
40       catch ( IOException ioException )
41       {
42          ioException.printStackTrace();
43       } // fim do catch
44    } // fim do método run
45 } // fim da classe MulticastSender
```

Cria um objeto `InetAddress` para o endereço de multicast

Fecha o `DatagramSocket` e o método `run` retorna, terminando o `MulticastSender`

`DatagramSocket` `send`

`cket` que
gem

31

Linhas 34-35

Linha 37

Linha 38

24.10.1 `DeitelMessengerServer` e classes de suporte (*Continuação*)

- **Executa `DeitelMessengerServerTest`:**
  - **Altera os diretórios para a localização adequada.**
  - **Digite o comando:**
  - `java com.deitel.mesenger.sockets.server.DeitelMessengerServerTest`

24.10.2 Cliente DeitelMessenger e classes de suporte

- Cliente DeitelMessengerServer:
  - Consiste em vários componentes:
    - Interface MessageManager.
    - A classe que implementa a interface MessageManager:
      - Gerencia a comunicação com o servidor.
    - Subclasse Runnable:
      - Ouve mensagens no endereço de multicast do servidor.
    - Uma outra subclasse Runnable:
      - Envia mensagens do cliente para o servidor.
    - Subclasse JFrame:
      - Fornece a GUI do cliente.

Resumo

MessageManager.java

```
1  // Fig. 24.24: MessageManager.java
2  // MessageManeger é uma interface para objetos capazes de gerenciar
3  // comunicações com um servidor de mensagens.
4  package com.deitel.messenger;
5
6  public interface MessageManager
7  {
8     // conecta-se ao servidor de mensagens e roteia as mensagens
9     // para um dado MessageListener
10    public void connect( MessageListener listener );
11
12    // desconecta-se de um servidor de mensagens e pára de rotea
13    // mensagens entrantes para um dado MessageListener
14    public void disconnect( MessageListener listener );
15
16    // envia a mensagem para o servidor de mensagens
17    public void sendMessage( String from, String message );
18 } // fim da interface MessageManager
```

Conecta `MessageManager` ao `DeitelMessengerServer` e roteia mensagens entrantes para o

Desconecta `MessageManager` do `DeitelMessengerServer` e pára de entregar mensagens ao

Envia uma nova mensagem ao `DeitelMessengerServer`

Resumo

SocketMessage Manager.java

(1 de 4)

Linha 20

Linha 22

```
1  // Fig. 24.25: SocketMessageManager.java
2  // SocketMessageManager se comunica com um DeitelMessengerServer utilizando
3  // Sockets e MulticastSockets.
4  package com.deitel.messenger.sockets.client;
5
6  import java.net.InetAddress;
7  import java.net.Socket;
8  import java.io.IOException;
9  import java.util.concurrent.Executors;
10 import java.util.concurrent.ExecutorService;
11 import java.util.concurrent.ExecutionException;
12 import java.util.concurrent.Future;
13
14 import com.deitel.messenger.MessageListener;
15 import com.deitel.messenger.MessageManager;
16 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
17
18 public class SocketMessageManager implements MessageManager
19 {
20    private Socket clientSocket; // Socket para mensagens enviadas
21    private String serverAddress; // Endereço do DeitelMessengerServer
22    private PacketReceiver receiver; // recebe mensagens de multicast
23    private boolean connected = false; // status da conexão
24    private ExecutorService serverExecutor; // executor para o servidor
25
```

Soquete para conectar e enviar mensagens ao

`Deite`

`Runnable` ouve as mensagens entrantes

Resumo

SocketMessageManager.java

(2 de 4)

Linhas 40-41

```
26  public SocketMessageManager( String address )
27  {
28     serverAddress = address; // armazena o endereço de servidor
29     serverExecutor = Executors.newCachedThreadPool();
30  } // fim do construtor SocketMessageManager
31
32  // conecta-se ao servidor e envia mensagens para um dado MessageListener
33  public void connect( MessageListener listener )
34  {
35     if ( connected )
36        return; // se já conectado, retorna imediatamente
37
38     try // abre a conexão de Socket ao DeitelMessengerServer co
39     {
40        clientSocket = new Socket(
41           InetAddress.getByName( serverAddress ), SERVER_PORT );
42
43        // cria um executável para receber as mensagens en
44        receiver = new PacketReceiver( listener );
45        serverExecutor.execute( receiver ); // executa o e
46        connected = true; // atualiza o flag conectado
47     } // fim do try
48     catch ( IOException ioException )
49     {
50        ioException.printStackTrace();
51     } // fim do catch
52  } // fim do método connect
53
```

Cria `Socket` para se comunicar com o `DeitelMessengerServer`

Cria um novo `packetReceiver`, que ouve as mensagens de multicast enti

Executa `Runnable`

```
54  // desconecta-se do servidor e remove o registro de um dado MessageListener
55  public void disconnect( MessageListener listener )
56  {
57     if ( !connected )
58        return; // se não conectado, retorna imediatamente
59
60
61
62
63        Runnable disconnecter = new MessageSender( clientSocket, "",
64           DISCONNECT_STRING );
65        Future disconnecting = serverExecutor.submit( disconnecter );
66        disconnecting.get(); // espera que a mensagem de desconexão seja enviada
67        receiver.stopListening(); // pára o receptor
68        clientSocket.close(); // fecha o Socket sainte
69     } // fim do try
70     catch ( ExecutionException exception )
71     {
72        exception.printStackTrace();
73     } // fim do catch
74     catch ( InterruptedException exception )
75     {
76        exception.printStackTrace();
77     } // fim do catch
78     catch ( IOException ioException )
79     {
80        ioException.printStackTrace();
81     } // fim do catch
82
```

Cria um novo `MessageSender` para enviar `DISCONNECT_STRING` ao servidor

`MessageSender` para entregar a
n utilizando o método `submit` de
`ExecutorService`

Invoca o método `Future get` para esperar que a mensagem de desconexão seja entregue e que `Runnable` termine

Resumo

(3 de 4)

Linhas 63-64

Linha 65

Linha 66

Resumo

SocketMessageManager.java

(4 de 4)

```
83       connected = false; // atualiza o flag conectado
84    } // fim do método disconnect
85
86    // envia mensagem ao servidor
87    public void sendMessage( String from, String message )
88    {
89       if ( !connected )
90          return; // se não conectado, retorna imediatamente
91
92       // cria e inicia um novo MessageSender para entregar a mensagem
93       serverExecutor.execute(
94          new MessageSender( clientSocket, from, message) );
95    } // fim do método sendMessage
96 } // fim do método SocketMessageManager
```

Cria e inicia um novo `MessageSender` para entregar a nova mensagem em uma thread separada da execução

Resumo

MessageSender.java

(1 de 2)

```
1  // Fig. 24.26: MessageSender.java
2  // Envia uma mensagem para o servidor de bate-papo em um executável separado.
3  package com.deitel.messenger.sockets.client;
4
5  import java.io.IOException;
6  import java.util.Formatter;
7  import java.net.Socket;
8
9  import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
10
11 public class MessageSender implements Runnable
12 {
13    private Socket clientSocket; // Socket pelo qual enviar a mensagem
14    private String messageToSend; // mensagem a enviar
15
16    public MessageSender( Socket socket, String userName, String message )
17    {
18       clientSocket = socket; // armazena o socket para o cliente
19
20       // cria uma mensagem a ser enviada
21       messageToSend = userName + MESSAGE_SEPARATOR + message;
22    } // fim do construtor MessageSender
23
```

Resumo

```
24    // envia a mensagem e finaliza
25    public void run()
26    {
27       try // envia a mensagem e esvazia PrintWriter
28       {
29          Formatter output =
30             new Formatter( clientSocket.getOutputStream
31          output.format( "%s\n", messageToSend ); // env
32          output.flush(); // esvazia saída
33       } // fim do try
34       catch ( IOException ioException )
35       {
36          ioException.printStackTrace();
37       } // fim do catch
38    } // fim do método run
39 } // fim da classe MessageSender
```

Cria um novo `Formatter` para o `Outp`

Invoca o método `Formatter flush` para assegurar que a mensagem seja enviada imediatamente

Invoca o método `rmatter format` para formatar a mensagem

Linha 31

Linha 32

Resumo

PacketReceiver.java

(1 de 4)

```
1  // Fig. 24.27: PacketReceiver.java
2  // PacketReceiver ouve DatagramPackets contendo
3  // mensagens de um DeitelMessengerServer.
4  package com.deitel.messenger.sockets.client;
5
6  import java.io.IOException;
7  import java.net.InetAddress;
8  import java.net.MulticastSocket;
9  import java.net.DatagramPacket;
10 import java.net.SocketTimeoutException;
11 import java.util.StringTokenizer;
12
13 import com.deitel.messenger.MessageListener;
14 import static com.deitel.messenger.sockets.SocketMessengerConstants.*;
15
16 public class PacketReceiver implements Runnable
17 {
18    private MessageListener messageListener; // recebe as mensagens
19    private MulticastSocket multicastSocket; // recebe as mensagens de broadcast
20    private InetAddress multicastGroup; // InetAddress do grupo de multicast
21    private boolean keepListening = true; // termina o PacketReceiver
22
23    public PacketReceiver( MessageListener listener )
24    {
25       messageListener = listener; // configura MessageListener
26
```

```
27       try // conecta o MulticastSocket ao endereço de multicast e à porta
28       {
29          // cria novo MulticastSocket
30          multicastSocket = new MulticastSocket(
31             MULTICAST_LISTENING_PORT );
32
33          // usa InetAddress para obter grupo de multicast
34          multicastGroup = InetAddress.getByName( MULTICAST_ADDRESS );
35
36          // associa-se ao grupo de multicast para receber mens
37          multicastSocket.joinGroup( multicastGroup );
38
39          // configura um tempo limite de 5 segundos ao esper
40          multicastSocket.setSoTimeout( 5000 );
41       } // fim do try
42       catch ( IOException ioException )
43       {
44          ioException.printStackTrace();
45       } // fim do catch
46    } // fim do construtor PacketReceiver
47
48    // ouve mensagens do grupo de multicast
49    public void run()
50    {
51       // ouve mensagens até ser parado
52       while ( keepListening )
53       {
54          // cria buffer para mensagem entrante
55          byte[] buffer = new byte[ MESSAGE_SIZE ];
56
```

`MulticastSocket` ouve mensagens de bate-papo entrantes na porta `MULTICAST_LISTENING_PORT`

`.java`

(2 de 4)

Objeto `InetAddress` para o qual

Registra o `MulticastSocket` para

Invoca o método `MulticastSocket` `setSoTimeout` para especificar que, se nenhum dado for recebido em 5000 milissegundos, o `MulticastSocket` deve emitir uma `InterruptedIOException`

Cria o array de `byte`s para armazenar o `DatagramPacket`

```
57          // cria DatagramPacket para mensagem entrante
58          DatagramPacket packet = new DatagramPacket( buffer,
59             MESSAGE_SIZE );
60
61          try // recebe um novo DatagramPacket (chamada bloqueadora)
62          {
63             multicastSocket.receive( packet );
64          } // fim do try
65          catch ( SocketTimeoutException socketTimeoutException )
66          {
67             continue; // continua para a próxima iteração para se manter ouvindo
68          } // fim do catch
69          catch ( IOException ioException )
70          {
71             ioException.printStackTrace();
72             break;
73          } // fim do catch
74
75          // coloca os dados da mensagem em uma String
76          String message = new String( packet.getData() );
77
78          message = message.trim(); // apara o espaço em branco na mensagem
79
80       // divide a mensagem em tokens p/ recuperar o nome do usuário e corpo da mensagem
81          StringTokenizer tokenizer = new StringTokenizer(
82             message, MESSAGE_SEPARATOR );
83
```

Cria um `DatagramPacket` para armazenar mensagens

Recebe o pacote entrante do endereço de multicast

Invoca o método `trim` da classe `String` para

`Cria` um `StringTokenizer` para separar o corpo da mensagem do nome do usuário que enviou a mensagem

(3 de 4)

Linhas 58-59

Linha 63

Linha 78

Linhas 81-82

Resumo

```
84          // ignora as mensagens que não contêm um nome de usuário
85          // e um corpo de mensagem
86          if ( tokenizer.countTokens() == 2 )
87          {
88             // envia a mensagem para MessageListener
89             messageListener.messageReceived(
90                tokenizer.nextToken(), // nome de usuário
91                tokenizer.nextToken() ); // corpo de mens
92          } // fim do if
93       } // fim do while
94
95       try
96       {
97          multicastSocket.leaveGroup( multicastGroup ); // deixa o gr
98          multicastSocket.close(); // fecha MulticastSocket
99       } // fim do try
100      catch ( IOException ioException )
101      {
102         ioException.printStackTrace();
103      } // fim do catch
104   } // fim do método run
105
106   // pára de ouvir novas mensagens
107   public void stopListening()
108   {
109      keepListening = false;
110   } // fim do método stopListening
111} // fim da classe PacketReceiver
```

Depois de analisar sintaticamente a mensagem, entrega a mensagem para `MessageListener` do `PacketReceiver`

Invoca o método `MulicastSocket leaveGroup` para parar o recebimento de mensagens do

Invoca o método `MulticastSocket close` para fechar o `MulticastSocket`

Resumo

ClientGUI.java

(1 de 10)

```
1  // Fig. 24.28: ClientGUI.java
2  // A ClientGUI fornece uma interface com o usuário para enviar e receber
3  // mensagens para e do DeitelMessengerServer.
4  package com.deitel.messenger;
5
6  import java.awt.BorderLayout;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import java.awt.event.WindowAdapter;
10 import java.awt.event.WindowEvent;
11 import javax.swing.Box;
12 import javax.swing.BoxLayout;
13 import javax.swing.Icon;
14 import javax.swing.ImageIcon;
15 import javax.swing.JButton;
16 import javax.swing.JFrame;
17 import javax.swing.JLabel;
18 import javax.swing.JMenu;
19 import javax.swing.JMenuBar;
20 import javax.swing.JMenuItem;
21 import javax.swing.JOptionPane;
22 import javax.swing.JPanel;
23 import javax.swing.JScrollPane;
24 import javax.swing.JTextArea;
25 import javax.swing.SwingUtilities;
26 import javax.swing.border.BevelBorder;
27
```

Resumo

ClientGUI.java

(2 de 10)

Linha 40

Linha 41

```
28 public class ClientGUI extends JFrame
29 {
30    private JMenu serverMenu; // para conectar/desconectar o servidor
31    private JTextArea messageArea; // exibe as mensagens
32    private JTextArea inputArea; // insere mensagens
33    private JButton connectButton; // botão para conectar
34    private JMenuItem connectMenuItem; // item de menu para conectar
35    private JButton disconnectButton; // botão para desconectar
36    private JMenuItem disconnectMenuItem; // item de menu para desconectar
37    private JButton sendButton; // envia mensagens
38    private JLabel statusBar; // rótulo para o status da conexão
39    private String userName; // userName para adicionar a mensagens enviadas
40    private MessageManager messageManager; // comunica-se com o servidor
41    private MessageListener messageListener; // recebe mensagens
42
43    // construtor ClientGUI
44    public ClientGUI( MessageManager manager )
45    {
46       super( "Deitel Messenger" );
47
48       messageManager = manager; // configura MessageManager
49
```

`MessageListener` recebe mensagens entrantes de `MessageManager`

```
50       // cria MyMessageListener para receber mensagens
51       messageListener = new MyMessageListener();
52
53       serverMenu = new JMenu ( "Server" ); // cria o JMenu do servido
54       serverMenu.setMnemonic( 'S' ); // configura o mnemônico p/ o menu d
55       JMenuBar menuBar = new JMenuBar(); // croa o JMenuBar
56       menuBar.add( serverMenu ); // adiciona o menu de servidor à barra de menus
57       setJMenuBar( menuBar ); // adiciona JMenuBar ao aplicativo
58
59       // cria ImageIcon para botões de conexão
60       Icon connectIcon = new ImageIcon(
61          lus().getResource( "images/Connect.gif" ) );
62
63       // cria connectButton e connectMenuItem
64       connectButton = new JButton( "Connect", connectIcon );
65       connectMenuItem = new JMenuItem( "Connect", connectIcon );
66       connectMenuItem.setMnemonic( 'C' );
67
68       // cria ConnectListener botões de conexão
69       ActionListener connectListener = new ConnectListener();
70       connectButton.addActionListener( connectListener );
71       connectMenuItem.addActionListener( connectListener );
72
73       // cria ImageIcon para botões de desconexão
74       Icon disconnectIcon = new ImageIcon(
75          getClass().getResource( "images/Disconnect.gif" ) );
76
```

Cria uma instância de `MyMessageListener`, que implementa a interface `MessageListener`

(3 de 10)

Linha 51

Resumo

ClientGUI.java

(4 de 10)

```
77      // cria disconnectButton e disconnectMenuItem
78      disconnectButton = new JButton( "Disconnect", disconnectIcon );
79      disconnectMenuItem = new JMenuItem( "Disconnect", disconnectIcon );
80      disconnectMenuItem.setMnemonic( 'D' );
81
82      // desativa o botão de desconexão e o item de menu
83      disconnectButton.setEnabled( false );
84      disconnectMenuItem.setEnabled( false );
85
86      // cria DisconnectListener para botões de desconexão
87      ActionListener disconnectListener = new DisconnectListener();
88      disconnectButton.addActionListener( disconnectListener );
89      disconnectMenuItem.addActionListener( disconnectListener );
90
91      // adiciona os JMenuItems de conexão e desconexão ao fileMenu
92      serverMenu.add( connectMenuItem );
93      serverMenu.add( disconnectMenuItem );
94
95      // adiciona JButtons de conexão e conexão ao buttonPanel
96      JPanel buttonPanel = new JPanel();
97      buttonPanel.add( connectButton );
98      buttonPanel.add( disconnectButton );
99
100     messageArea = new JTextArea(); // exibe mensagens
101     messageArea.setEditable( false ); // desativa edição
102     messageArea.setWrapStyleWord( true ); // configura o estilo de quebra de linha
103     messageArea.setLineWrap( true ); // ativa a quebra de linha
104
```

Resumo

ClientGUI.java

(5 de 10)

Linhas 128-129

```
105    // coloca a messageArea no JScrollPane para permitir rolagem
106    JPanel messagePanel = new JPanel();
107    messagePanel.setLayout( new BorderLayout( 10, 10 ) );
108    messagePanel.add( new JScrollPane( messageArea ),
109       BorderLayout.CENTER );
110
111    inputArea = new JTextArea( 4, 20 ); // para inserir novas mensagens
112    inputArea.setWrapStyleWord( true ); // configura o estilo de quebra de linha
113    inputArea.setLineWrap( true ); // ativa a quebra de linha
114    inputArea.setEditable( false ); // desativa a edição
115
116    // cria Icon para sendButton
117    Icon sendIcon = new ImageIcon(
118       getClass().getResource( "images/Send.gif" ) );
119
120    sendButton = new JButton( "Send", sendIcon ); // cria o botão de enviar
121    sendButton.setEnabled( false ); // desativa o botão de enviar
122    sendButton.addActionListener(
123       new ActionListener()
124       {
125          // envia uma nova mensagem quando usuário ativa o sendButton
126          public void actionPerformed( ActionEvent event )
127          {
128             messageManager.sendMessage( userName,
129                inputArea.getText() ); // envia mensagem
130             inputArea.setText( "" ); // limpa inputArea
131          } // fim do método actionPerformed
132       } // fim da classe interna anônima
133    ); // fim da chamada para addActionListener
134
```

Envia o nome do usuário e o texto da `inputArea` a `DeitelMessengerServer` como uma mensagem de bate-papo

Resumo

ClientGUI.java

(6 de 10)

Linha 155

```
135     Box box = new Box( BoxLayout.X_AXIS ); // cria uma nova caixa para layout
136     box.add( new JScrollPane( inputArea ) ); // adiciona a área de entrada à caixa
137     box.add( sendButton ); // adiciona o botão Enviar à caixa
138     messagePanel.add( box, BorderLayout.SOUTH ); // adiciona a caixa ao painel
139
140     // cria um JLabel para statusBar com uma borda reentrante
141     statusBar = new JLabel( "Not Connected" );
142     statusBar.setBorder( new BevelBorder( BevelBorder.LOWERED ) );
143
144     add( buttonPanel, BorderLayout.NORTH ); // adiciona painel de botão
145     add( messagePanel, BorderLayout.CENTER ); // adiciona painel de mensagem
146     add( statusBar, BorderLayout.SOUTH ); // adiciona barra de status
147
148     // adiciona WindowListener para desconectar quando o usuário sair
149     addWindowListener (
150        new WindowAdapter ()
151        {
152           // desconecta-se do servidor e encerra o aplicativo
153           public void windowClosing ( WindowEvent event )
154           {
155              messageManager.disconnect( messageListener );
156              System.exit( 0 );
157           } // fim do método windowClosing
158        } // fim da classe interna anônima
159     ); // fim da chamada para addWindowListener
160  } // fim do construtor ClientGUI
161
```

Desconecta-se do servidor de chat quando o usuário fecha a aplicação cliente

Resumo

```
162 // ConnectListener ouve solicitações do usuário para conectar ao servidor
163 private class ConnectListener implements ActionListener
164 {
165    // conecta-se ao servidor e ativa/desativa componentes GUI
166    public void actionPerformed( ActionEvent event )
167    {
168       // conecta-se ao servidor e roteia mensagens para
169       messageManager.connect( messageListener );
170
171       // solicita o userName
172       userName = JOptionPane.showInputDialog(
173          ClientGUI.this, "Enter user name:" );
174
175       messageArea.setText( "" ); // limpa a messageArea
176       connectButton.setEnabled( false ); // desativa conectar
177       connectMenuItem.setEnabled( false ); // desativa conectar
178       disconnectButton.setEnabled( true ); // ativa desconectar
179       disconnectMenuItem.setEnabled( true ); // ativa desconectar
180       sendButton.setEnabled( true ); // ativa o botão Enviar
181       inputArea.setEditable( true ); // ativa a edição para a área de entrada
182       inputArea.requestFocus(); // configura foco para a área de entrada
183       statusBar.setText( "Connected: " + userName ); // configura texto
184    } // fim do método actionPerformed
185 } // fim da classe interna ConnectListener
186
```

Quando o usuário acessa o menu **Connect**, conecta-se ao servidor de bate-papo

Solicita ao usuário um nome de usuário

Resumo

ClientGUI.java

(8 de 10)

Linha 195

```
187   // DisconnectListener ouve solicitações do usuário para desconectar-se
188   // de DeitelMessengerServer
189   private class DisconnectListener implements ActionListener
190   {
191      // desconecta-se do servidor e ativa/desativa componentes GUI
192      public void actionPerformed( ActionEvent event )
193      {
194         // desconecta-se do servidor e pára de rotear mensagens
195         messageManager.disconnect( messageListener );
196         sendButton.setEnabled( false ); // desativa o botão de enviar
197         disconnectButton.setEnabled( false ); // desativa desconectar
198         disconnectMenuItem.setEnabled( false ); // d
199         inputArea.setEditable( false ); // desativa
200         connectButton.setEnabled( true ); // ativa a
201         connectMenuItem.setEnabled( true ); // ativa
202         statusBar.setText( "Not Connected" ); // configura a barra de status
203      } // fim do método actionPerformed
204   } // fim da classe interna DisconnectListener
205
```

Invoca o método `MessageManager` `disconnect` para se desconectar do servidor de bate-papo

Resumo

ClientGUI.java

(9 de 10)

```
206   // MyMessageListener ouve novas mensagens do MessageManager e
207   // exibe as mensagens na messageArea utilizando o MessageDisplayer.
208   private class MyMessageListener implements MessageListener
209   {
210      // quando recebida, exibe novas mensagens na messageArea
211      public void messageReceived( String from, String message )
212      {
213         // acrescenta a mensagem utilizando o MessageDisplayer
214         SwingUtilities.invokeLater(
215            new MessageDisplayer( from, message ) );
216      } // fim do método messageReceived
217   } // fim da classe interna MyMessageListener
218
219   // Exibe nova mensagem acrescentando a mensagem à JTextArea.  Deve
220   // ser executado somente na thread Event; modifica componente Swing ativo
221   private class MessageDisplayer implements Runnable
222   {
223      private String fromUser; // usuário do qual a mensagem veio
224      private String messageBody; // corpo da mensagem
225
```

Exibe a mensagem quando `MessageListener` detecta que a mensagem foi recebida

Resumo

ClientGUI.java

(10 de 10)

Linha 237

```
226      // construtor MessageDisplayer
227      public MessageDisplayer( String from, String body )
228      {
229         fromUser = from; // armazena o usuário que origina a mensagem
230         messageBody = body; // armazena o corpo da mensagem
231      } // fim do construtor MessageDisplayer
232
233      // exibe nova mensagem na messageArea
234      public void run()
235      {
236         // acrescenta nova mensagem
237         messageArea.append( "\n" + fromUser + "> " + messageBody );
238      } // fim do método run
239   } // fim da classe interna MessageDisplayer
240} // fim da classe ClientGUI
```

Acrescenta o nome de usuário, “> ” e `messageBody` à `messageArea`

Resumo

DeitelMessenger.java

(1 de 3)

Linha 17

Linha 20

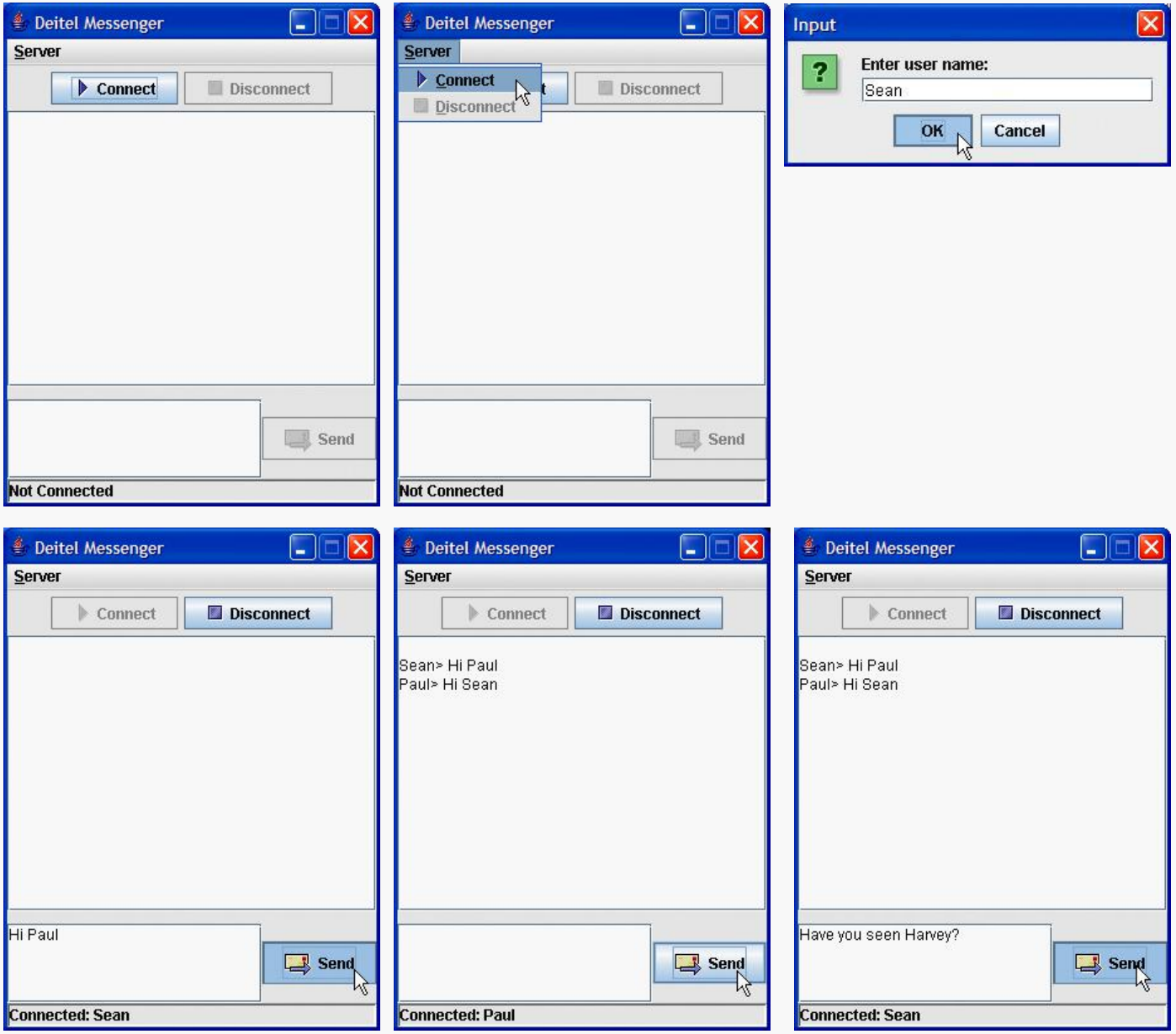
```
1  // Fig. 24.29: DeitelMessenger.java
2  // DeitelMessenger é um aplicativo de bate-papo que utiliza um ClientGUI
3  // e um SocketMessageManager para se comunicar com o DeitelMessengerServer.
4  package com.deitel.messenger.sockets.client;
5
6  import com.deitel.messenger.MessageManager;
7  import com.deitel.messenger.ClientGUI;
8
9  public class DeitelMessenger
10 {
11    public static void main( String args[] )
12    {
13       MessageManager messageManager; // declara MessageManager
14
15       if ( args.length == 0 )
16          // conecta-se ao host local
17          messageManager = new SocketMessageManager( "localhost" );
18       else
19          // conecta-se utilizando o argumento de linha de comando
20          messageManager = new SocketMessageManager( args[ 0 ] );
21
22       // cria a GUI para SocketMessageManager
23       ClientGUI clientGUI = new ClientGUI( messageManager );
24       clientGUI.setSize( 300, 400 ); // configura o tamanho da janela
25       clientGUI.setResizable( false ); // desativa o redimensionamento
26       clientGUI.setVisible( true ); // mostra a janela
27    } // fim do main
28 } // fim da classe DeitelMessenger
```

Cria um cliente para se conectar ao host local

Conecta-se a um host fornecido pelo usuário

Cria uma `ClientGUI` para o `MessageManager`

Resumo



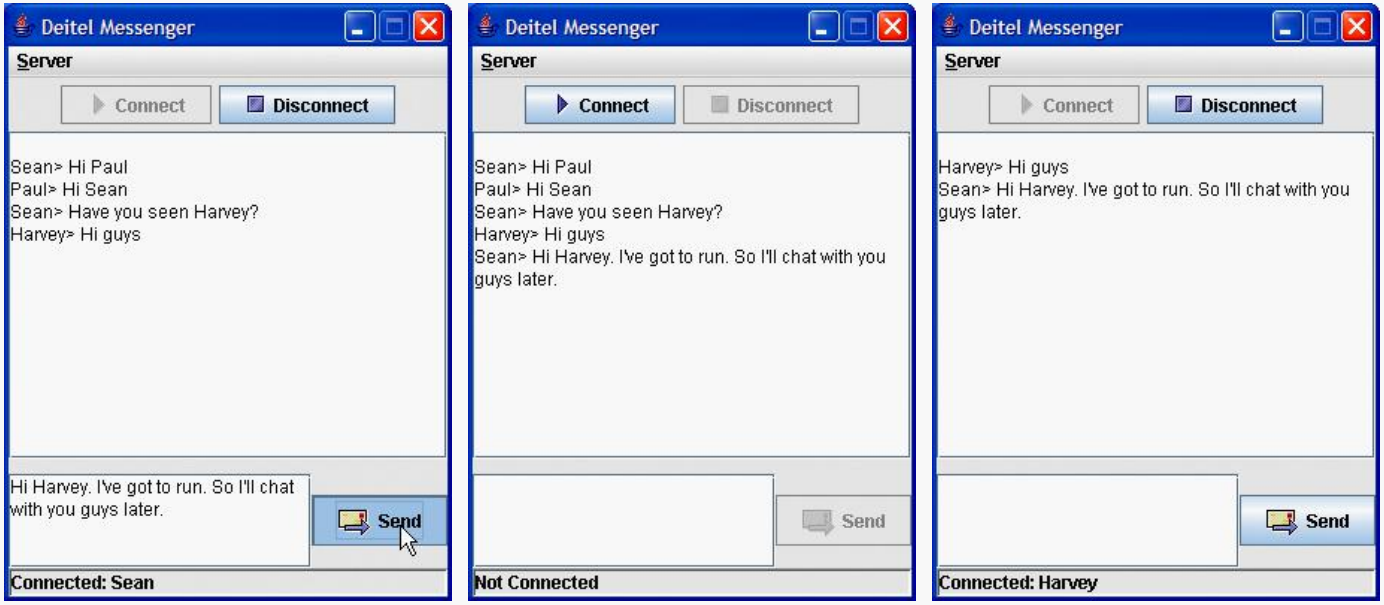
`DeitelMessenger.java`

(2 de 3)

Resumo

DeitelMessenger.java

(3 de 3)



24.10.2 Cliente DeitelMessenger e classes de suporte (*Continuação*)

- **Executa o aplicativo cliente `DeitelMessenger`:**
  - **Altera os diretórios para a localização adequada.**
  - **Digite o comando:**
  - `java com.deitel.messenger.sockets.client.DeitelMessenger`
  - **Que é equivalente a:**
  - `java com.deitel.messenger.sockets.client.DeitelMessenger localhost`
  - **ou**
  - `java com.deitel.messenger.sockets.client.DeitelMessenger 127.0.0.1`